4th NASA Symposium on VLSI Design

University of Idaho

Moscow, Idaho

October 29-30, 1992

Cover design by Peter Vincent

Welcome to the fourth annual NASA Symposium on VLSI Design, co-sponsored by the IEEE. Each year this symposium is organized by the NASA Space Engineering Research Center (SERC) at the University of Idaho and is held in conjunction with a quarterly meeting of the NASA Data System Technology Working Group (DSTWG). One task of the DSTWG is to develop new electronic technologies that will meet next generation electronic data system needs. The symposium provides insights into developments in VLSI and digital systems which can be used to increase data systems performance.

The NASA SERC is proud to offer, at its fourth symposium on VLSI design, presentations by an outstanding set of individuals from national laboratories, the electronics industry and universities. These speakers share insights into next generation advances that will serve as a basis for future VLSI design.

In spite of the present economic conditions, interest in the conference has remained steady with 42 papers in 10 categories included in this years proceedings. National Laboratories are represented by Rockwell International Science Center, NCCOSC Research and Development, Jet Propulsion Laboratory and the Johns Hopkins University Applied Physics Laboratory. Private industry is represented by Texas Instruments, Advanced Hardware Architectures and Spacebourne Inc. Universities are represented by Cornell University, Utah State University, University of Alabama, Washington State University, University of Calgary, University of Southwestern Louisiana , Nanyang Technological University, Syracuse University, Concordia University, University of California at Davis and Irving, California State University at Fullerton Colorado State University, University of Toledo, University of Peradeniya, University of New Mexico and the University of Idaho. In addition we are happy to welcome a number of papers presented by international authors.

There are individuals whose assistance was critical to the success of this symposium. Barbara Martin worked long hours to assemble the conference proceedings. The efforts of these professionals were vital and are greatly appreciated. I hope you enjoy your stay in Coeur d'Alene, Idaho.

Sterling R. Whitaker

Session 1 – Featured Presentations I

Chairman: Gary Maki

Session 2 – VLSI Architectures

Chairman: Kelly Cameron

Session 3 – Design Synthesis

Chairman: James Frenzel

Session 4 – Featured Presentations II

Chairman: Sterling Whitaker

Session 5 – VLSI Design

Chairman: Kel Winters

Session 6 – Verification

Chairman: Phil Windley

Session 7 – Tools and Methods

Chairman: Jonathan Gibson

Session 8 – VLSI Design

Chairman: Jody Gambles

Session 9 – Cache Architectures

Chairman: Joseph Feeley

Session 10 – Correlators

Chairman: John Canaris

List of Authors

Session 1
Featured Presentations I

Chairman: Gary Maki

Applications of Correlator Chips in Radio Science

Jon B. Hagen
Cornell University
NAIC Lab 124
124 Maple Ave
Ithaca NY 14850
607-255-5274

Abstract - **Spectral line observations in radio astronomy require simultaneous power estimation in many (often hundreds to thousands) of frequency bins. Digital autocorrelation spectrometers, which appeared thirty years ago, are now being implemented in VLSI. The same architecture can be used to implement transversal digital filters. This was done at the Arecibo Observatory for pulse compression in radar observations of Venus.**

1 Introduction

This paper reviews applications and implementations of the digital correlator, a signal processor long used for radio spectrometry and increasingly used for digital filtering. Most radio engineers intuitively define points on the power spectrum (power density v.s. frequency) of a signal to be the average voltages from a set of square-law detectors following a bank of bandpass filters. A formal definition is provided by the Wiener-Khinchin theorem: the power spectrum of a signal is the Fourier transform of the signal's autocorrelation function (ACF). In autocorrelation spectrometry the intensive on-line averaging is done in a correlator; the Fourier transform is a one-time operation done prior to further data analysis. (Occasionally experimental ACFs are compared to theoretical ACFs without ever transforming from the time domain to the frequency domain).

2 Basic Correlator Architecture

The architecture in question is extremely simple; the chip contains N identical arithmetic modules or "lags" which operate in parallel, independently forming products from two data streams and accumulating the sum of these products. One of the data streams, the "immediate" data is common to every lag. The other data stream, the "delayed" data, is supplied from successive taps of a shift register contained on the chip. Each clock pulse advances the shift register data and adds the new lagged products to the contents of each accumulator. Each module thus calculates a point on the autocorrelation function, i.e. the average product of the signal voltage at time T and the signal voltage at time (t-T). An output multiplexer provides access to the individual accumulators. The basic block diagram is shown in Figure 1.

To measure a power spectrum via the autocorrelation function, the immediate and the delayed inputs are both tied to the same signal (the undetected baseband output voltage of the radio astronomy receiver). This type of correlator was introduced in radio astronomy by

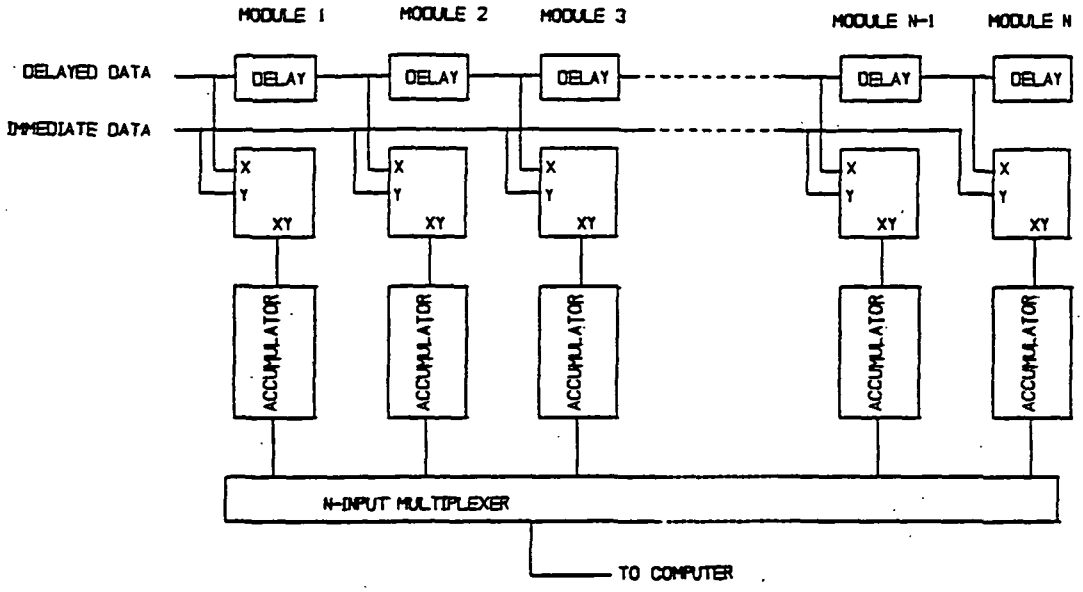


Figure 1: Basic Correlator Architecture

Sander Weinreb (1) in 1963 and has been the workhorse for spectral analysis. It is discussed further in Section 4 after a review of the basic principles of spectrometry.

3 Spectrometry in Radio Astronomy

Though a radio astronomer occasionally deals with objects that are sources of spectrally flat white noise, the more interesting and challenging sources have considerable spectral structure. Such sources include molecular clouds with emission or absorption lines, natural masers, and dispersed pulses from rapidly rotating neutron stars (pulsars). The spectrum can have enough structure to warrant hundreds to thousands of points of resolution. Each point requires its own radiometer, i.e. bandpass filter plus square-law detector or an equivalent such as one module of a correlator. This kind of **multiplex** spectrometer (meaning that all points are measured simultaneously) is always necessary in radio astronomy because the received signals are noiselike random processes and time averaging is needed at each filter/detector or correlator module. If the required accuracy is, say, 1% or one part in one hundred, it is necessary to average about 1002 or 10,000 samples of V^2. Sampling theory shows that the time needed to gather the required number of independent samples will be inversely proportional to the bandwidth. These radiometry considerations show why an ordinary laboratory spectrum analyzer - a scanning receiver with a single bandpass filter - is not suitable for this work as it would increase the required observation time in proportion to the number of frequency channels. Such increases in observing time are simply not available; radio astronomers are usually trying to dig small spectral features out of background noise (the sum of cosmic noise, atmospheric noise, antenna noise, and amplifier noise) and, even with multiplex spectrometers, integration times run to dozens or even hundreds of hours since the signal-to-noise ratios (before averaging) are typically in the -30 to -60 dB range.

4 Digital Correlators

The basic correlator module can be implemented partly or wholly in analog circuitry but all-digital implementations have been favored from the start. Autocorrelation is not the only way to do digital spectrometry; the Fast Fourier Transform (FFT), familiar since the late sixties, is, in principle, a more appealing algorithm since it ultimately must be more efficient - arithmetic operations are done at a rate proportional to N log(N) rather than N^2 where N is the number of points on the spectrum. Surely, for N large enough, the FFT will be better. Yet for N smaller than, say, 10,000, and at sampling rates, i.e. bandwidths, high enough that the digital circuitry is running at full speed, it seems that the correlator architecture has two advantages.

The first advantage comes from the simplicity of the architecture. The modules run independently. Apart from the shift register that distributes the delayed data, there are no data interconnections between modules. The correlator is easily expandable. To get more points on the spectrum one cascades more of the same modules. Conversely, a machine with many modules can be subdivided into several smaller correlators which can independently analyze several different signals. The simplicity of the basic module thus provides flexibility at the system level.

The second advantage is that correlators can use very coarse quantization of the input signal. Instead of digitizing the signal with 8 to 16 bits, as is often required in digital signal processing, it has been common to digitize to only one bit; the amplitude is ignored and the signal's polarity determines whether the value of the bit is a 1 or a -1. As one would expect, such a distorted representation of the signal produces a distorted version of the desired correlation function, but, as long as the signal is a Gaussian random process, the distortion can be removed perfectly after the integration is finished. The correction procedure is very simple: divide all the points (lags) by the first point (zero lag), multiply them by $\pi/2$ radians, and then replace them by their trigonometric sines. This relationship between the correlation function of the input signal and the correlation of the 1-bit representation was first described by J.H. VanVleck in a wartime study of overmodulation (clipping) for radar jammers.

Obviously 1-bit arithmetic greatly simplifies the multipliers and accumulators that form the correlator modules. Each multiplier reduces to a single exclusive-or gate and each accumulator reduces to a counter. (The same simplification doesn't work for an FFT processor; while the data could be represented with one bit, the trigonometric coefficients still need to be multibit and, except for the first stage, the butterfly operations still use multibit $\times$ multibit multiplication). Simple one-bit correlation, however, does have its price: more integration time, by a factor of $\pi^2/4$ or 2.47, is required if the averaging is to be as effective as if the signal had been multibit. Going from a 1-bit digitizer to a 2-bit digitizer reduces the 2.47 integration time penalty to only 1.29 but obviously complicates the multiplier/accumulator circuitry. A three-level digitizer ("1.6 bits") is often used; it permits multiplier/accumulator circuitry almost as simple as the 1-bit case and yields an integration time penalty of 1.51. In all of these quantization schemes the distortion of the ACF can be corrected [2].

Some of the integration time penalty can be recovered. When the signal is oversampled by a factor of 2, i.e. sampling at a rate of 4 times the bandwidth, the factors of 2.47, 1.29, and 1.51 reduce to 1.82, 1.14, and 1.26, respectively. Very little is gained by going to still

faster sampling rates.

5 Digital Filtering

The canonical transversal digital filter uses a tapped delay line (shift register). Successively delayed samples of the signal are taken from these taps and multiplied by constant weighting coefficients. The weighted signals are combined in a single multi-input adder. Each new input sample pushes data down the shift register to produce a new output value from the adder. The correlator architecture described above provides another way to implement the transversal filter. The input signal provides the immediate data to an N-lag correlator. The filter coefficients are generated serially (e.g. from a ROM addressed by a counter) and form the delayed "data." After each input sample one correlator module will contain the desired convolution of the last N data values and the N coefficients. Its accumulator contents are latched into an output register and its accumulator is zeroed. After the next input sample, the output register is loaded from the adjacent correlator module. The process continues indefinitely, with the correlator modules being read out and rezeroed cyclicly.

At the Arecibo Observatory an 8-bit $\times$ 1-bit correlator was used this way as a transversal filter to produce radar maps of the surface of Venus. The c.w. transmitted signal is phase-coded with a long (4095, say) pseudo-noise sequence to get the bandwidth needed for range resolution. The received signal (radar echo) is decoded by the transversal filter. The filter coefficients are the plus ones and minus ones of the code so the code generator itself provides these coefficients in the required serial sequence. With the code properly phased, the first lag module of the correlator produces the signal reflected from the front cap of the planet. The next lag module produces the signal corresponding to the next range ring, etc [3]. The lags are read in order and the resulting data sequence is identical to what would have been obtained by transmitting a powerful short pulse (pulse length equal to the baud length) and using no decoder. Good range resolution, essentially equal to the baud length, is a consequence of the very sharp peak resulting when the pseudo-noise code is convolved with itself. The code sequence must be long enough to avoid aliasing other ranges in each decoder channel, i.e. the code length is deeper than the radius of the planet. In this application our correlator had fewer lag modules than the number of bauds in the code sequence - an economic constraint. The code was phased such that the ranges of interest were decoded by the implemented modules; other ranges landed in absent modules (which were, of course, just ignored). This trade-off of ranges for less hardware would not have been possible with the canonical delay line transversal filter.

6 VLSI Implementations

Correlators were implemented first with discrete transistor logic and then with all the standard families of integrated logic elements. A gate-array correlator chip designed by Albert Bos (4) has 16 lags running at 50 MHz. More recently, Brian Von Herzen (5) produced a VLSI chip with 320 lags that runs at 250 MHz. John Canaris (6), of the VLSI design group at the University of Idaho (now at the University of New Mexico), is designing a 1024-lag

correlator chip with 3-level quantization to run at 100 MHz. A commercial product, the Zoran ZR33891 Digital Filter Processor is an 8-lag correlator chip that does 9-bit × 9-bit multiplications at a 20 MHz rate and has 26-bit accumulators. As explained above, radio astronomy applications favor designs use much coarser quantization in order to have many more lags and higher speed.

7 Multi-chip Expansions

The basic correlator chip shown in Figure 1 can be used in a variety of ways to build expanded systems. Perhaps the most obvious is series expansion (simple cascading), which only requires that the chip have an output pin from its last delay element to provide the delayed data for the following chip. Time multiplexing is a parallel expansion technique that provides both higher speed and more lags. For example, suppose our N-lag correlator chip is able to accept data at only half the required rate. We can split the data stream into two half-speed streams, the even samples and the odd samples, and then use four chips to calculate the two possible autocorrelation functions (ACFs) and two possible cross-correlation functions (CCFs). When the integrated results are dumped to the computer, the sum of the ACFs gives us N even lags and the sum of the CCFs gives us N odd lags. By going from one to four chips we've doubled the speed and doubled the number of lags. Generalizing, we can divide data into N streams and use N^2 correlators to form all the possible lagged products. Another interesting form of parallelism, distributed arithmetic (from the distributive law of multiplication), was described by Zohar (7) as a way by which signals with many bits can by correlated using simple 1-bit, 2-bit, etc, correlator chips. Since signed numbers complicate matters, consider first a digitizer that encodes positive integers into, say, standard 8- bit binary representation. Group the eight bits into four 2-bit digits. We could multiply two such 4-digit numbers by separately multiplying each digit of one by each of the digits of the other. This could be done in parallel by 4^2 or sixteen 2-bit multipliers. Sixteen 2-bit correlators could independently form sub-products and accumulate them during an integration cycle. The results would be read out, appropriately weighted, and added to accomplish 8-bit × 8-bit correlation. This technique will work with signed numbers if we have either

1. a number system where the place weights carry signs, or

2. a number system where the digit values possible for each place are signed.

An example of the first case is the negabinary system (radix -2) where the digit values are just 0 and 1 but the place weights are 1, -2, 4, -8, etc. An example of the second case is the balanced ternary system where the values are -1, 0 and 1 and the place weights are 1,3,9,27, etc. For the negabinary system we would need 1-bit correlators with AND gates as multipliers rather than the EX-OR gates of the standard 1-bit correlator. For the balanced ternary system the popular 3-level correlator is just the right correlator element. Four 3-level (1.6-bit) correlators can run in parallel to correlate 9-level (3.2-bit) numbers. Correlators using many bits provide wide instantaneous dynamic range, especially if the quantization levels are exponentially spaced rather than uniformly spaced.

8 Summary

Digital VLSI correlator chips using coarse quantization and having many lag units are simple building blocks for spectrometers, convolvers, and digital filters. Chips can be paralleled to increase the number of channels, speed, and dynamic range.

References

[1] Weinreb, Sander, "A Digital Spectral Analysis Technique and its Application to Radio Astronomy," Technical Report 412, MIT Research Laboratory of Electronics, August, 1963.

[2] Hagen, J.B. and D.T. Farley, "Digital-correlation techniques in radio science," Radio Science, Vol. 8, Numbers 9.9, pp 775- 784, August-September 1973.

[3] Hagen, Jon B. "Communication Techniques in Radio Physics and Astronomy," IEEE Communications Magazine, October 1986 - Vol. 24, No. 10, pp 16-20.

[4] Bos, Albert, "A High Speed 2-Bit Correlator Chip for Radio Astronomy," IEEE Transactions on Instrumentation and Measurement, Vol. 40, No. 3, June 1991.

[5] Von Herzen, Brian, "VLSI Partitioning of a 2-Gs/s Digital Spectrometer," IEEE Journal of Solid-State Circuits, Vol. 26, No. 5, May 1991.

[6] Canaris, John, "NASA SERC Digital Correlator Projects" Fourth NASA Symposium on VLSI Design, October 1992.

[7] Zohar, Shalhav, "The Role of the 1 Number System in Multibit Hardware Correlators," IEEE Transactions on Acoustics, Speech and Signal Processing, Vol. 37, No. 10, October 1989.

Single Event Phenomena: Testing and Prediction

James D. Kinnison
The Johns Hopkins University
Applied Physics Laboratory
Laurel, Maryland 20723
jdk@aplcomm.jhuapl.edu

Abstract - **Highly integrated microelectronic devices are often used to increase the performance of satellite systems while reducing the system power dissipation, size, and weight. However, these devices are usually more susceptible to radiation than less integrated devices. In particular, the problem of sensitivity to single event upset and latchup is greatly increased as the integration level is increased. Therefore, a method for accurately evaluating the susceptibility of new devices to single event phenomena is critical to qualifying new components for use in space systems. This evaluation includes testing devices for upset or latchup and extrapolating the results of these tests to the orbital environment. Current methods for testing devices for single event effects are reviewed, and methods for upset rate prediction, including a new technique based on Monte Carlo simulation, are presented.**

1 Introduction

Satellite system designers are constantly seeking new technology to increase the performance of their systems. Many new VLSI devices are well suited to space applications. For example, telescopes flown on spacecraft have greatly benefited from the use of on-board digital signal processors. However, space contains many hazards for which most VLSI devices have no inherent immunity. To be used in a satellite application, the response of these integrated circuits to the various space hazards must be measured, and some estimate of the survivability of these devices must be made.

One of the most difficult problems facing integrated circuits is that of exposure to charged particles. As these particles pass through a material, they lose energy by ionizing the material, inducing electron-hole pairs along the path of the particle. The interaction of electrons, protons, and heavy ions with integrated circuits leads to two different kinds of problems. The first is total dose degradation, in which the cumulative effect of many particles passing through an integrated circuit causes parameter degradation and functional failure, and will not be considered here. In addition to this problem, single particles passing through a device may cause a variety of effects, collectively known as single event phenomena (SEP).

In digital devices, the most common phenomenon is single event upset, in which the state of a bit or an active node is reversed. This erroneous state remains until that bit or node is reset. CMOS devices are susceptible to ion-induced latchup, similar to input latchup commonly observed on the workbench. In this case, a charged particle traverses a p-n-p-n region, and the induced current pulse injects enough minority carriers into the parasitic transistors present in the structure to cause a low impedance path from supply to ground.

Latchup is usually destructive. Analog or mixed analog-digital devices are susceptible to current pulses injected by charged particles. These pulses are short-term transients which can propagate to other devices and cause a bewildering array of system-level effects. Finally, power MOSFETs are susceptible to burnout and gate rupture due to charged particles, described first in [1].

2 General Considerations

The effects discussed above are nominally different, but have a core of similarity. In each case, the effect is initiated by a single charged particle traversing a sensitive region of a device. In the simplest case, a monoenergetic beam of particles irradiating a device, some particles deposit enough charge in a sensitive region to cause an event (i.e., an upset, latchup, or transient). The number of events occurring, N, is given by

$$N = \rho * n \tag{1}$$

where ρ is the probability that a particle causes an event, and n is the number of particles passing through the device. Normally, the number of interactions is given in terms of the particle fluence, the number of particles per unit area, instead of the number of particles. If we multiply ρ and divide n by the area of the device, the previous equation becomes

$$N = \sigma * \Psi \tag{2}$$

where σ is the event cross-section, with units of area, and Ψ is the particle fluence.

In principle, the cross-section contains all the information about an integrated circuit necessary to estimate the event rate in any environment. For each type of event, the cross-section depends on many factors, including angle of incidence, the type and energy of the incoming particle, the temperature, and the electric fields present near the sensitive regions. In some cases, it is possible to assign a physical meaning to the cross-section. For example, in a static RAM, each cell is nearly identical, and contains a nearly identical sensitive volume. If a particle induces enough charge in one of these volumes, an upset occurs in that cell. The cross-section is just the area of the sensitive volume normal to the particle path.

In the case of heavy ions, it is convenient to describe the event cross-section as a function of the incident particle linear energy transfer (LET), rather than particle type and energy. The linear energy transfer is the amount of energy transferred to the target material along the particle path, and is the derivative of the energy with respect to distance along the track. LET is usually given in units of MeV*cm^2/mg, and is determined by the incident particle type, charge, and energy, and by the target material density. The cross-section for a device is zero below some LET normally referred to as the threshold LET, and rapidly increases to a value called the asymptotic cross-section. For SEP due to protons, the cross-section is determined as a function of incident proton energy rather than LET, but the form of the cross-section remains the same.

For many devices, the angular dependence of the cross-section is easily expressed. A particle which travels through a device at some oblique angle has a longer path through the device than a particle which travels normal to the device; therefore, more energy is

transferred to the target material. The result is a particle which appears to have a larger LET than the same particle at normal incidence. Therefore, the LET of the particle is expressed as an effective LET given by

$$L_{eff} = \frac{L}{cos\theta} \tag{3}$$

Note that this does not account for azimuthal asymmetry of the device, and so does not give the azimuthal dependence of the cross-section. This dependence is measured by irradiating the device at several azimuthal angles.

The cross-section of a device can sometimes be estimated using a computer model. This obviously requires in-depth knowledge of the device layout which is not usually available to the user. In most cases, the cross-section must be measured. These tests are usually performed at a particle accelerator laboratory; one of the most commonly used sites is the Tandem Van de Graaff accelerator at Brookhaven National Laboratory. No matter where the test is performed, the layout is much the same. The accelerator provides a beam to a target area with the same LET (or in the case of protons, energy) as particles found in space. Since the path length of heavy ions in air is very short, the target area is usually a vacuum chamber connected to an evacuated beam line. The device under test (DUT) is mounted in the target area, irradiated with particles, and tested for events. Since the beam is monoenergetic, and the particle fluence over the exposure is measured, equation (2) can be used to calculate the cross-section. This procedure is repeated for a variety of test conditions, until the cross-section is completely characterized.

3 Test Methods

Nearly all SEP cross-section measurements are performed in the manner outlined above. The main differences occur in the ways that logistical problems for a target area are solved. For example, nearly all SEP tests are computer controlled, but the tests must be performed in a vacuum chamber. Therefore, the problem of communicating with a DUT or test hardware through vacuum feedthroughs must be solved. Similarly, DUTs dissipate heat when operated; in a vacuum chamber, good thermal control is essential.

The integrated circuits under consideration are usually quite complex, and the problem of exercising the device while looking for errors is challenging, to say the least. While there are many ways to implement a test for SEP events, they all fall into four categories, the squirt, golden-chip, pseudo-golden-chip, and loosely coupled methods. There are benefits to each method, but none is appropriate for all device types. Therefore, an understanding of all four is essential to planning a good SEP test [2].

3.1 Squirt Method

The squirt method is the simplest way to test for SEP. In this type of test, a device is prepared for exposure, irradiated, and then interrogated for the presence of events. In general, the DUT is under static bias, and only SEP which are not time dependent are considered.

As an example, consider upsets in a static RAM. The test hardware consists of a circuit which loads a fixed pattern into the memory, then reads back the contents of the memory and compares this to the original. The cross-section is calculated by counting the total number of upset bits and dividing by the particle fluence over that exposure. This sequence is repeated for a variety of particles with different LET and incident angle, and for a number of other device conditions such as bit pattern and temperature.

Latchup or burnout tests are often performed using this method. In this case a device is irradiated under static bias; after exposure, the device is examined for latchup or burnout. Since the latchup could have occurred during any part of the irradiation, the results of this test can not be used to calculate the latchup cross-section, but the susceptibility of a device to a particle of a particular LET is determined.

Obviously, this type of test is easy to implement, and for many devices, is quite adequate as a screen for destructive phenomena. However, this type of test is usually incomplete in the sense that only static errors are counted. In the case of the static RAM, for instance, those portions of the device which provide control and address decoding are never tested. Since one error in control logic may upset many bits at once, this oversight may gloss over grossly undesirable behavior. Other significant behavior, such as the device operating frequency dependence of the event cross-section, may also be missed. Finally, sensitive bits may flip more than once during an exposure. If this occurs, an error may never be counted, or may only be counted once, and so the cross-section may be too l ow by a factor of two or three. Despite these drawbacks, there may be times when this type of test is appropriate. The upset cross-section of many memories is still measured this way, since cell upsets dominate the cross-section.

3.2 Golden-Chip Method

Since static tests can miss some types of events, dynamic testing is preferred for SEP measurements. The golden chip method is a way to implement these dynamic tests without building a large VLSI integrated circuit tester that fits in a vacuum chamber. The test hardware is built around two identical devices which are mounted so that one is irradiated while the other is not. If the devices are given the same inputs, they should have the same outputs, unless an event occurs. Therefore, the error detection consists of a set of level comparators, one for each output, and a counter which keeps track of the number of events observed. In addition, the control hardware must be able to simultaneously reset the test device and the golden device when an error occurs.

Golden-chip tests offer one important advantage over the previous type; the test is dynamic, not static. Therefore, all portions of a circuit can be exercised. In addition, events are counted dynamically, so missed multiple events are much less likely. The golden-chip method can be used to test many different kinds of logic devices which operate at relatively low frequency and with a small number of inputs. For example, latchup tests can be performed by comparing the supply current and outputs of an irradiated device to those of a golden device. When a large difference in the supply current is observed, a latchup has occurred.

However, there are limitations to the usefulness of this method. First, the test hardware

must provide inputs to the devices simultaneously. Second, the number of input states in a test cycle determines the amount of data storage needed in the test control hardware; for some devices, the number of inputs needed to effectively exercise all portions of the DUT can be quite large. Third, the devices should not be overly sensitive to noise, which on an output could be mistaken for an event. Finally, the device outputs, including timing parameters, must be identical to within the resolution of the tester comparators. In most cases, this restriction imposes an upper limit to the DUT operating frequency during the test.

3.3 Pseudo-Golden-Chip Method

The pseudo-golden-chip method seeks to avoid the limitations of the golden-chip method by comparing the output sequence of the DUT to some standard sequence. In essence, we replace the golden device with a computer. As the DUT is irradiated, each output is compared to an expected value. Any discrepancy is counted as an event.

A good example of a device suited to this test method is a FIFO memory. A known bit pattern is repeatedly applied to the FIFO, and after each write, a word is read. If the correct pattern does not appear, then a single event upset has occurred in either the control logic or the circular buffer in the FIFO.

This type of test is frequently used in SEP testing. Most groups who regularly test devices for SEP susceptibility have developed some type of computer-based test hardware which can rapidly be tailored for this kind of test. As with the golden-chip method, the number of different states in the input sequence limits the usefulness of this tests. The operating frequency of the DUT is also limited by the time taken to compare all the DUT outputs to the expected values.

3.4 Loosely Coupled Systems Method

All three methods discussed above are based on some set of test hardware closely controlling and examining a device. This is, in principle, the best way to test for SEP. In practice, however, an event in a DUT can cause the test hardware to malfunction. This is disastrous in an SEP test; any dead time, in which the beam is on but events cannot be counted, artificially reduces the accuracy of the cross-section measurement. Therefore, it is essential that the test system is able to quickly detect and correct any events observed in the DUT.

The loosely coupled systems method is built around the interaction of two systems. The first system, which I will call the DUT test system is used to exercise the DUT. The second system, called the test controller, is used to monitor for events reported by the DUT test system, and to reset the DUT test system when an event is observed. The two systems communicate asynchronously.

This method is the most effective way to test microprocessors and related devices. In this case, the DUT test system would be a single board computer which executes some fixed test routine on bootup. This routine is written so that a known status word, which indicates whether the DUT test routine was completed successfully or an event occurred, is sent to the test controller computer at regular intervals via a serial link or some similar communication protocol. The test controller receives the status word, examines it, and forces a reset in the

DUT test computer if an event is detected. In addition, the test controller operates as a watchdog so that if an event in the DUT stops the operation of the DUT test system, a reset is forced.

In spite of the benefits discussed above, there are problems with this method. First, some information about the distribution and effect of event in the DUT is lost. We can determine that an event occurred, but not where or exactly what the consequences of that event are in the DUT. Second, the cross-section measured is really the cross-section of the DUT test system, not the DUT; if only the DUT is irradiated and all events which occur in the DUT are observed, then the system event cross-section is a close approximation to the DUT cross-section. Third, because the devices tested in this manner are complex, the cross-section may vary significantly with test program. Because of these problems, the DUT test system should be as close to the target application as possible.

4 Rate Prediction

The cross-section for some event is a parameter used to characterize the susceptibility of a device to that event. It contains all the information needed to estimate the event rate in an arbitrary charged particle environment. The methods by which these estimates are calculated, however, are not simple, and much research into event rate prediction methods has been centered around simplifying the calculation of event rate estimates.

Unfortunately, some methods developed for one type of event may not be applicable to other types of events. Most of the effort has been directed toward developing single event upset prediction methods. Only recently have researchers started examining other effects and the relationship between the cross-sections for different effects. The most important event rate prediction methods are discussed below.

4.1 The Path-Length Distribution Model

The path-length distribution model assumes that every upsetable cell in a device is identical, and that each contains one thin rectangular parallelepiped region, called the sensitive volume, such that a particle which deposits charge in this region may cause an upset. If this is true, the upset rate, N, is given by

$$N = 22.5\pi A Q_C \int_0^\infty \frac{D[p(L)]}{L^2}\psi(L)dL \tag{4}$$

where A is the area of the sensitive volume, Q_C is the smallest deposited charge that will cause an upset (called the critical charge), L is LET, D[p(L)] is the differential distribution of path lengths over which a particle of LET, L, will produce a charge greater than the critical charge, and ψ is the integral flux spectrum given as a function of LET. Pickel and Blanford [3], in 1980, originally proposed an integral of this form for the error rate.

All of the parameters in (4) are given by the cross -section except the particle flux. The critical charge is just the threshold LET in units of pC/ μm(which is equivalent to MeV*cm^2/mg) times the minimum thickness of the sensitive volume (in μm). A is the asymptotic cross-section. The differential path length distribution depends on the shape of

the sensitive volume, and is not simply expressed. In most cases, the sensitive volume is assumed to be a rectangular parallelepiped, and the appropriate path-length distribution is used to evaluate (4)

The path-length distribution model was developed to predict single event upset rates in memories. The assumptions used to write (4) approximate most static RAMs to first order. However, many devices are not so simply described. For instance, a microprocessor may contain many different upsetable nodes, each with a unique sensitive volume. The result is that each node may have a different threshold and asymptotic value. The cross-section, as measured in SEP tests, is the sum of the responses of each individual sensitive volume. The integral in (4) does not model these situations well, but it is possible to increase the accuracy of the calculation for some devices which do not have identical sensitive volumes. If the individual sensitive volumes can be separated into a small number of groups with nearly identical volumes in each group, error rates can be calculated for each group. The total device error rate is just the sum of the group error rates.

Only single event upsets are considered in the path-length distribution model; the assumptions concerning the shape and uniformity of the sensitive volume only apply to single event upset. In other cases, particularly latchup, the concept of a sensitive volume is not meaningful, and so an equation like (4) cannot be written.

Several computer programs have been written to estimate the single event upset rate of a device using the path-length distribution model. The first of these was *Cosmic Ray Induced Error Rate Analysis* (CRIER) by Pickel and Blandford [4]. In 1981, Adams, Silverberg, and Tsao published the first part of a new model of the galactic cosmic ray environment. Over the next three years, they added the effects of the Earth's magnetic field to the model, and included a program to calculate the single event upset rate of a device in an arbitrary environment by evaluating (4). This model is called *Cosmic Ray Effects in Microelectronics*, also known as CREME [5]. CREME is commonly used today for single event upset rate estimations.

4.2 The Bendel Proton Upset Models

It is commonly known that galactic cosmic rays cause single event upsets. Less commonly known is that high energy protons found in the Van Allen radiation belts can also cause single event upsets. These protons do not produce enough ionization to upset a device directly, but secondary particles created in nuclear reactions with the target material can produce sufficiently high ionization to cause upset. The Bendel models provide an estimate of the upset rate due to proton nuclear reactions in a variety of low Earth orbits as a function of semi-empirical fit parameters.

The first model is called the one-parameter Bendel model [6]. The proton upset cross-section is measured at a few proton energies, and then fitted to the following equations to solve for A:

$$\sigma = (24/A)^{14}[1 - exp(-0.18Y^{1/2})]^4 \tag{5}$$

$$Y = (18/A)^{1/2}(E - A) \tag{6}$$

where E is the proton energy and σ is in units of 10^{-12} upsets per proton/cm^2 per bit. An estimate of the proton upset rate as a function of orbit altitude and inclination, and as a

function of the parameter A is given in [6]. This semi-empirical model works reasonably well for older devices, but needs improvement for newer, more complex technology.

To increase the accuracy of the method, Stapor, et al. [7]
have recast (5) as a two parameter model. The equation becomes

$$\sigma = (B/A)^{14}[1 - exp(-0.18Y^{1/2})]^4 \tag{7}$$

where Y and σ are defined as in (5)-(6). As before, an estimate of the proton upset rate as a function of altitude and inclination, and as a function of both parameters, A and B, is given in [7].

Estimates made with the two parameter Bendel model have been compared to flight data taken on the CRRES satellite. These estimates are in reasonably good agreement with the flight data [8]. However, the estimates provided by the Bendel models are averages over a very dynamic environment, and the short term error rate may be much higher than the estimate. Therefore, systems with devices susceptible to proton upset must be able to withstand short periods of greatly increased error rate.

4.3 Probabilistic Model

The definition of a cross-section given in (1)-(2) is based on a probabilistic interpretation of the interaction which causes SEP. Consider a monoenergetic beam of particles with direction normal to the face of a device. If the beam is uniform, the probability that a particle intersects the device at a particular point is constant across the face of the device. Assume that some regions of the device are sensitive to an event; any particle passing through those regions will cause an upset. The probability that a particle passes through these regions is just the total area of the sensitive regions divided by the area of the device. The total area of the sensitive regions is given by the cross-section.

In space, the particles interacting with the device are not monoenergetic, and so not all have the same LET. In addition, the particle flux is omnidirectional; the particles arrive from every direction at the same rate. So, for a particle of a given LET and direction, the probability that an event occurs is just the cross-section for that LET and direction divided by the projected area of the device normal to the particle direction.

The probabilistic nature of the cross-section can be exploited to calculate an event rate for a device in a particular environment using a Monte Carlo simulation [9]. For a calculation of this nature, only the event cross-section as a function of LET and incident direction and the fluence as a function of LET are needed. The flux spectrum is divided into narrow LET bins. For a particle in one of the bins, the basic algorithm is as follows:

1. Pick a random LET within the bin.
2. Pick a direction.
3. Calculate the cross-section based on the LET and direction.
4. Generate a random number.
5. If the random number is less than the cross-section, increment the event count.

The event rate estimate is given by the event count which results after repeating the algorithm for the total number of particles in each LET bin of the flux spectrum.

The algorithm given above is applicable to any SEP for which a cross-section can be measured. Event rates generated by this type of computer simulation compare well with CREME and Bendel model estimates and with flight data. The advantage of this method is that no information about the geometry or architecture of the device is necessary to generate a good estimate of the event rate.

5 Current Research

Several areas of current research in SEP may lead to new test and estimation techniques which will simplify the problem of qualifying new devices for use in space. These new techniques include new ways to stimulate devices to produce SEP, methods of estimating the cross-section for on type of event from the cross-section of another type, and new models for estimating event rates from cross-section data.

Accelerator beams are not cheap, and it can be difficult to schedule an experiment in a timely fashion. In addition, few researchers are located near an accelerator; experiments must be transported to the accelerator facility. Because of these difficulties, some researchers are using radioactive isotopes to provide high LET, low energy particles in a small vacuum chamber as an alternative stimulus for SEP. The most commonly used isotope for SEP testing is ^{252}Cf, which produces fission fragments with a mean LET of 43 MeV*cm^2/mg. Unfortunately, these fragments have a limited range in silicon; they lose a significant fraction of their total energy in about 25 μm. Therefore, this type of testing works reasonably well for single event upset testing, but is not useful for latchup, which requires deeper penetration into the device. In addition, as the particle travels deeper into the device, the LET changes as energy is lost. This effect must be taken into account when assigning an LET to the fission fragments. In spite of the disadvantages, ^{252}Cf is commonly used to compare the relative sensitivity of devices, and to supplement accelerator tests.

Several groups are examining the use of pulsed lasers to simulate the effects produced by charged particles in devices [10]. A narrowly focussed laser beam produces localized electron-hole pairs in the target material like an ion. The effective LET of the laser beam can be varied by changing the intensity of the laser, so a wide LET range can be simulated. The primary advantage of using a laser beam to stimulate SEP is that the spot size is of the same order as the feature size. Therefore, sensitive regions of the device can be mapped accurately, and geometrical assumptions about the sensitive volume are not necessary. Unfortunately, metal layers reflect the laser light, and so regions under metal can not be tested; the cross-section given by laser exposures is not complete, and can not be used in event rate calculations. However, the information gathered using a laser can be used to identify SEP sensitive regions so that device designers can correct the problem.

Solid-state detectors have been used in nuclear and radiation physics experiments for some time. In many ways, an integrated circuit may be viewed as a collection of solid state detectors. McNulty, et al, [11] have characterized the charge collection properties of many integrated circuits, and have shown that these measurements can be used to calculate the single event upset cross-section of a device. In a test of this type, the DUT is biased such

that the inputs and output are floating, supply pins are grounded, and the ground pins are connected to pulse height analysis system. When the device is exposed to a particle beam, charge is collected; the passage of a single particle is seen as a current pulse. These pulses are collected and sorted by the pulse height analyzer into a pulse spectrum. Peaks in this spectrum represent sensitive regions in the device, and the area under a peak divided by the total fluence over the exposure is the cross-section of the sensitive region which generated that peak. This technique may be used to map sensitive volumes, so geometrical assumptions are not necessary to calculate an event rate. To date, this technique has only been used to measure single event upset cross-sections in regular structures, like static RAMs. A complex device such as a microprocessor may not be tested well with charge collection techniques. In addition, this is a static test, which can not be used while dynamically exercising a device. It is not clear that this test will accurately measure a cross-section that depends on frequency, program, or other dynamic variables.

A group led by Newberry has recently begun examining the relationship between device upset rates and system-level upset rates [12]. She has shown that, for some systems, single event upsets which occur in a device in that system are not propagated to the system output, and that, for other systems, device upsets are multiplied and spread throughout the system. This indicates that the upset rate of a system may not be the sum of the upset rates for each component of the system. If this is true, methods for simulating the effect of upsets on system behavior must be developed. In addition, this behavior may force experimenters to test for events at the system level, instead of the device level; entirely new test methods will have to be developed.

6 Summary

Satellite systems will continue to grow more complex, and the problem of SEP susceptibility will not go away. Therefore, it is critical that space systems designers understand the behavior of new integrated circuits in the space environment. I have discussed the methods which are commonly used to assess SEP vulnerability and to estimate the rate at which SEP occur in an orbital environment. SEP tests can be grouped in four classes of test types. Each of these test types have benefits and drawbacks, and an understanding of each test type is necessary to effectively gather the data necessary to predict the rate at which SEP occur in space.

In addition to the test types, I have discussed the most commonly used methods used to calculate estimates of the SEP rate in any environment. The assumptions upon which each of the methods are based are given, and the range of applicability is discussed.

Finally, some of the newest areas of SEP research are discussed. These research areas give some indication of the direction future SEP tests will take. Of critical importance will be the way system-level SEP tests are performed, and how these tests will be used to estimate the performance of a complex system in space.

References

[1] A. E. Waskiewicz, J. W. Groninger, V. H. Strahan, and D. M. Long, "Burnout of Power MOS Transistors with Heavy Ions of Californium-252," *IEEE Trans. Nucl. Sci.* NS-33, 1710-1713 (1986).

[2] R. Koga, W. A. Kolasinski, M. T. Marra, and W. A. Hanna, "Techniques of Microprocessor Testing and SEU Rate Prediction,UU *IEEE Trans. Nucl. Sci.* NS-32, 4219-4224 (1985).

[3] J. C. Pickel and J. T. Blanford, "Cosmic Ray Induced Errors in MOS Devices," *IEEE Trans. Nucl. Sci.* NS-27, 1006-1015 (1980).

[4] J. C. Pickel and J. T. Blanford, "CMOS RAM Cosmic Ray Induced Error Rate Analysis," *IEEE Trans. Nucl. Sci.* NS-28, 3962-3967 (1981).

[5] J. H. Adams, Jr., *Cosmic Ray Effects on Microelectronics*, MR 5901, Naval Research Laboratory, Washington DC (Dec 1986).

[6] W. L. Bendel and E. L. Petersen, "Proton Upsets in Orbit," *IEEE Trans. Nucl. Sci.* NS-30, 4481-4485 (1983).

[7] W. J. Stapor, J. P. Meyers, J. B. Langworthy, and E. L. Petersen, "Two Parameter Bendel Model Calculations for Predicting Proton Induced Upset," *IEEE Trans. Nucl. Sci.* NS-37, 1966-1973 (1990).

[8] P. J. McNulty, W. J. Beauvais, W. G. Abdel-Kader, S. S. El-Teleaty, E. G. Mullen, and K. P. Ray, "Test of SEU Algorithms Against Preliminary CRRES Data," *IEEE Trans. Nucl. Sci.* NS-38, 1642-1646 (1991).

[9] J. D. Kinnison, R. H. Maurer, and T. M. Jordan, "Estimation of the Charged Particle Environment for Earth Orbits," *Johns Hopkins APL Technical Digest* 11, 300-310 (1990).

[10] S. Buchner, K. Kang, W. J. Stapor, A. B. Campbell, A. R. Knudson, P. McDonald, and S. Rivet, "Pulsed Laser Induced SEU in Integrated Circuits: A Practical Method for Hardness Assurance Testing," *IEEE Trans. Nucl. Sci.* NS-37, 1825-1831 (1990).

[11] P. J. McNulty, "Predicting Single Event Phenomena in Space," 27th IEEE Nuclear and Space Radiation Effects Conference Short Course, Reno, Nevada (July 16, 1990).

[12] D. M. Newberry, D. H. Kaye, and G. A. Soli, "Single Event Induced Transients in I/O Devices: A Characterization," *IEEE Trans. Nucl. Sci.* NS-37, 1974-1980 (1990).

Heterojunction Bipolar Transistor Technology for Data Acquisition and Communication

C. Wang, M. Chang, S. Beccue, R. Nubling, P. Zampardi, N. Sheng and R. Pierson
Rockwell International Science Center
Camino Dos Rios
Thousand Oaks, CA 91360
kcw@jupiter.risc.rockwell.com, 805-373-4143

Abstract - **Heterojunction Bipolar Transistor (HBT) technology has emerged as one of the most promising technologies for ultrahigh-speed integrated circuits. HBT circuits for digital and analog applications, data conversion, and power amplification have been realized, with speed performance well above 20 GHz. At Rockwell, a baseline AlGaAs/GaAs HBT technology has been established in a manufacturing facility. This paper describes the HBT technology, transistor characteristics, and HBT circuits for data acquisition and communication.**

1 Introduction

Heterojunction Bipolar Transistor (HBT) technology has shown great capabilities for the realization of high performance analog, digital, and microwave circuits [1-2]. To date, most of the HBT circuits demonstrated are based on AlGaAs/GaAs material system with wide-bandgap AlGaAs emitters and heavily doped GaAs bases. The AlGaAs/GaAs HBT technology offers a number of intrinsic advantages. High f_t and f_{max} (above 55 GHz), as well as high transconductance (above 10000 mS/mm), can be realized. The threshold voltages (Vth) are highly uniform; matching of Vth's of HBTs in differential pairs of about 1 mV has been measured. The intrinsic junctions of HBTs are well shielded from substrate and surface. Trap induced hysteresis effects are absent and $1/f$ noise is low. AlGaAs/GaAs HBTs also offer high breakdown voltages. The HBTs are fabricated on semi-insulating GaAs substrate, which reduces parasitic capacitances of transistors and interconnect lines and allows integration of multifunctional circuits. Rockwell pioneered the research of AlGaAs/GaAs HBT technology and developed a baseline technology with high current gain for analog, digital, and A/D conversion applications. The technology is now established in a manufacturing facility (Rockwell's Microelectronics Technology Center-MTC). High performance HBT circuits can be realized with high yields. As a result, recently Rockwell announced its initial HBT products. This paper presents Rockwell's baseline HBT technology and its applications in data acquisition and communication. Development of HBT-based technologies will also be described.

2 Rockwell's Baseline HBT Technology

The HBT technology features emitter-up/single-heterojunction bipolar transistors, monolithically integrated Schottky diodes, NiCr thin film resistors, MIM capacitors, and up to

three levels of metal interconnect. AlGaAs/GaAs HBTs are fabricated on MBE or MOCVD grown epitaxial wafers. The epitaxial layer structure and a schematic cross section of an integrated HBT and a Schottky diode are shown in Fig. 1. The minimum geometry device used for the circuits has an emitter area of $1.4\mu m \times 3\mu m$, defined by projection optical lithography. The measured Gummel Plot of this device as shown in Fig. 2(a) illustrates current gain of about 100 at $Ic = 1mA$. The measured RF characteristics of the transistor at $Vce = 2\ V$ and $Jc = 8 \times 10^4 A/cm^2$ are shown in Fig. 3(b). F_t and f_{max} above 55 GHz are obtained.

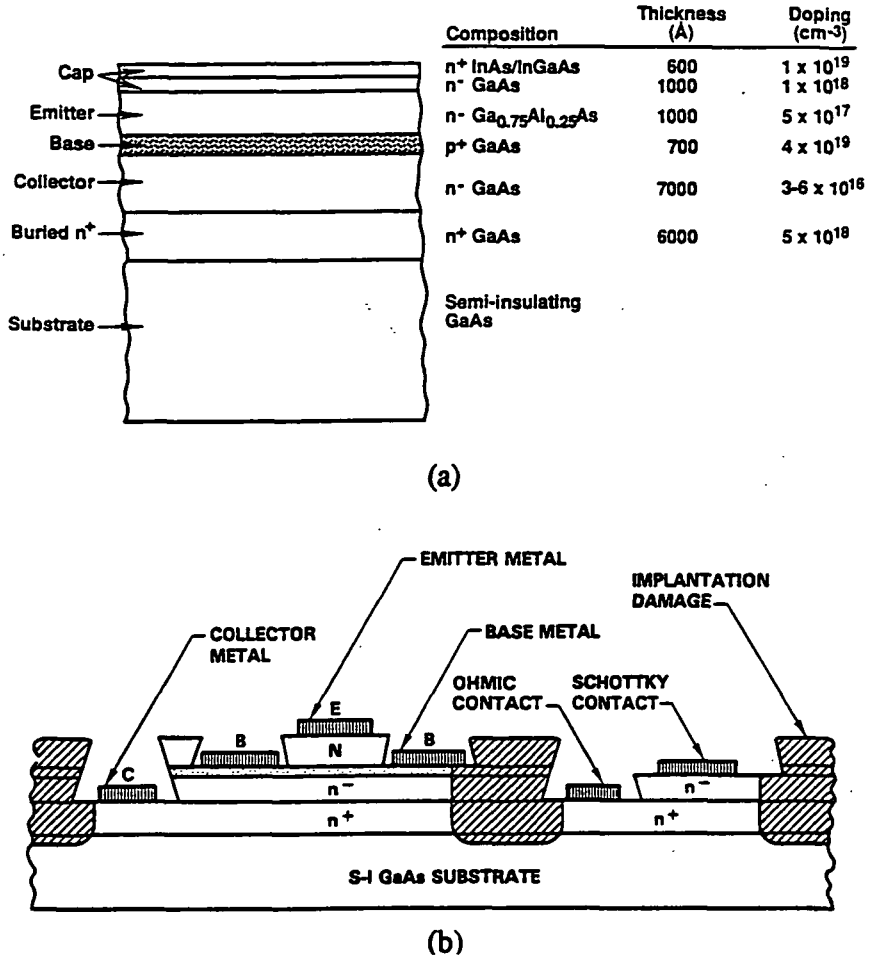


Figure 1: (a) HBT epitaxtal layer structure (b) Simplified cross section of an HBT and a Schottky diode

The diodes are realized using same layer structure and process as the HBTs. Typical series resistance and parasitic capacitance are 40Ω and $12fF$ for a $2\mu m \times 4\mu m$ diode.

Many high-performance digital and analog circuits have been realized with the Rockwell baseline HBT technology. Frequency dividers have been demonstrated to work above 25 GHz [3]. 8-GHz 1000-gate gate arrays and 15-GHz 500-gate gate arrays were also realized

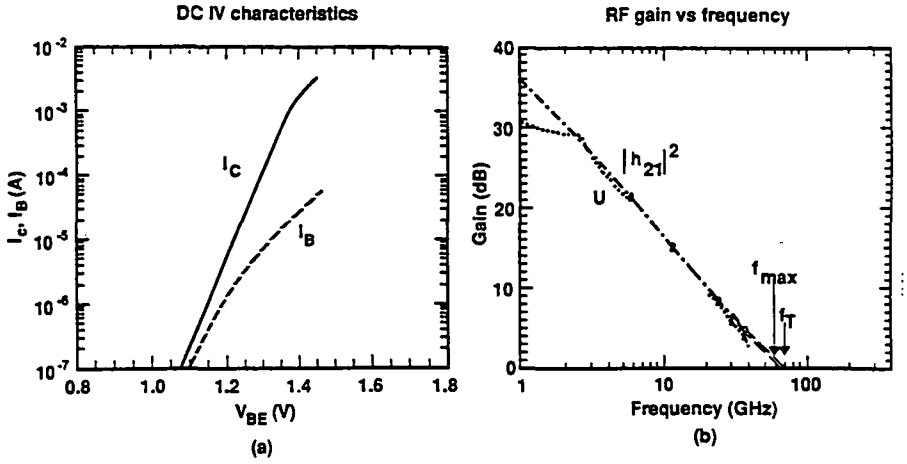


Figure 2: (a) Gummel Plot and (b) RF characteristics of an HBT that has an emitter area of $1.4\mu m \times 3\mu m$

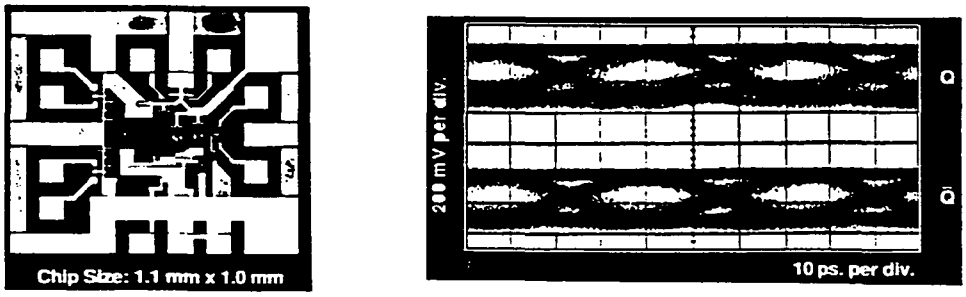


Figure 3: (a) Microphotograph of a fabricated HBT 2:1 mux. (b) Eye pattern of the mux output at 30 Gb/s operation.

(jointly developed with IBM) [4-5]. We have demonstrated a number of ultra-high speed HBT circuits for lightwave communication, jointly with Bellcore. These include a 30 Gb/s 2:1 mux, a 27 Gb/s 1:2 demux, a 27 Gb/s 4-bit mux/demux, and a 7 Gb/s 8-bit mux/demux [6-7]. The microphotograph and operation of the 30 Gb/s 2:1 mux are shown in Fig. 3.

In the A/D conversion area, we have realized HBT voltage comparators that operate up to 20 GSps [8], as shown in Fig. 4. We have demonstrated 2 GSps sample-and-hold (S/H) circuits with less than -40 dB distortion (jointly with HP Labs.), and Multi-GSps 4-bit and 6-bit quantizers [9-10]. We are developing high performance ADCs and DACs with the HBT technology. These include: a 6-GSps 6-bit ADC (also with HP Labs.), a 1.5-GSps 8-bit ADC, a 100-MSps 12-14 bit sigma-delta ADC, and a 1.2-GSps 12-bit DAC.

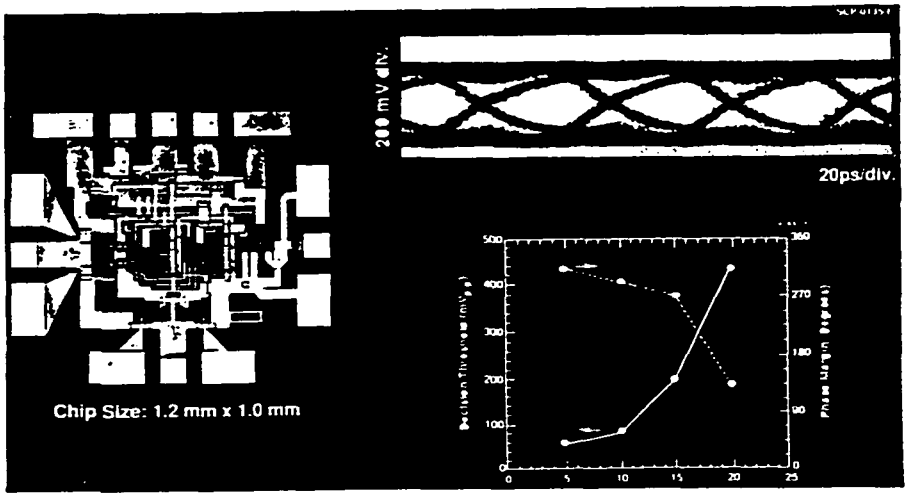


Figure 4: (a) Microphotograph of a fabricated HBT voltage comparator. (b) Output eye pattern of the comparator at 20 GSps operation. (c) Measured sensitivity and phase margin.

3 An HBT Based Data Acquisition System

We are developing a 1.5-GHz 8-bit data acquisition system (DAS), under a NASA/ONR contract. The primary goal is for laser altimeter applications, although the system can be used for general purpose ultra-high speed data acquisition. The system specifications are shown in Fig. 5. The system includes an HBT ADC, an HBT clock driver, 16 HFET (heterojunction FET) 1K memory circuits, and Si CMOS interface circuits. A schematic block diagram is also shown in Fig. 5.

The key component of the system is an HBT 8-bit 1.5-GSps ADC. The ADC employs a folding and interpolating scheme [11] that features flash ADC speed at significant savings in device counts and power. The block diagram of the ADC design is shown in Fig. 6.

Sampling rate	1.5	GSps
Resolution	8	Bits
Data demultiplexing	8:16	
Memory size	2048	Bytes
Data acquisition period	1.36	μs
Power dissipation	15	W
Operating temperature range	-15 to 75	°C

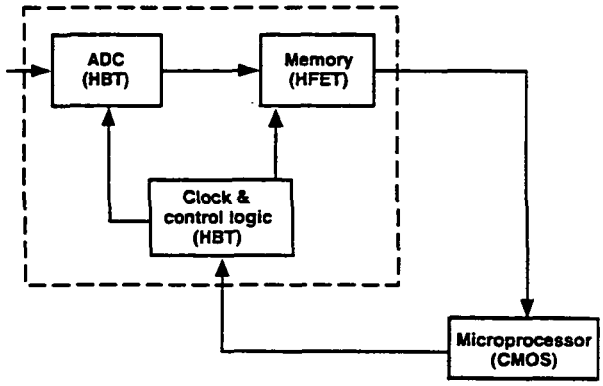


Figure 5: System specifications and Block diagram of an HBT based DAS.

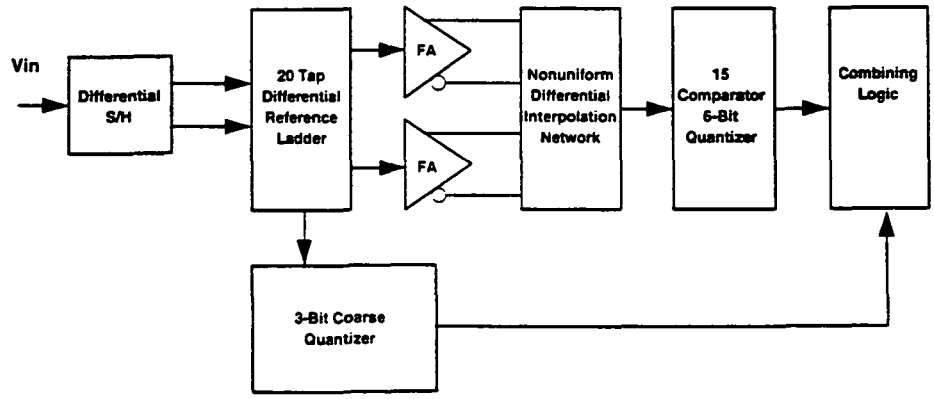


Figure 6: Block diagram of the 8-bit 1.5-GHz ADC design.

It contains an on-chip S/H circuit, a 3-bit coarse quantizer, a 6-bit fine quantizer, and a combining logic. We chose an all-differential architecture to reduce common-mode errors. An on-chip 1:2 demultiplexer for each output bit reduces the output data rate. The circuit contains about 1700 HBTs (950 in the A/D and 750 in the demux). The estimated power consumption is 3 W. Figure 7 shows a fabricated 8-bit 1.5 GSps HBT ADC.

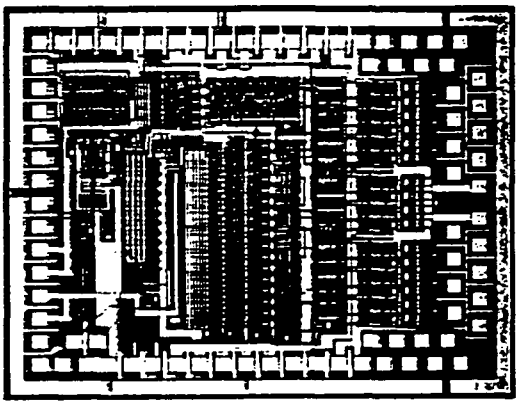


Figure 7: Microphotograph of a fabricated 8-bit 1.5 GSps HBT ADC.

To evaluate the expected dynamic accuracy of the HBT 8-bit ADC design, we have simulated the performance of the whole ADC circuit (except the data demuxing) with HSPICE, and analyzed the results. The digitized input waveform for a 1.6 MHz input at 1.5 GSps operation was reconstructed as shown in Fig. 8. The vertical scale of the signal waveform (shown on the right side of the figure) is in units of LSBs (least-significant bits). The distortions and noises were calculated, with much finer vertical scale (on left side). The calculated signal to noise + distortion ratio (S/N) is about 47 dB, which corresponds to 7.5 effective bits. Operations of the ADC at other sampling rates and input frequencies were similarly simulated and analyzed. Good S/N ratios were obtained. Initial fabricated ADCs are being evaluated.

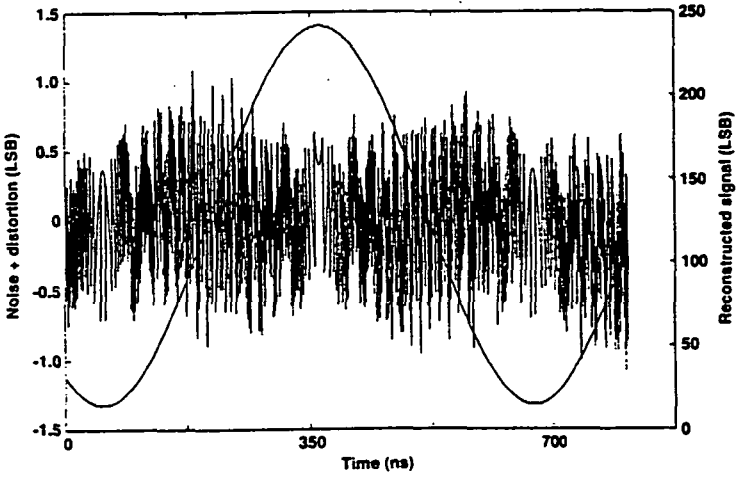


Figure 8: Simulated output waveform and noise + distortion of the 8-bit HBT ADC operation at 1.5 GSps and with a 1.6 MHz input.

The HBT clock driver circuit provides control logic and synchronized complimentary clocks to the 16 memory chips. It also contains a high speed frequency divider as a part of a phase locked loop for a clock generator. Other important components of the DAS are the 16 1K-bit FIFOs (first-in first-out memory). They are implemented with Rockwell's HMESFET which has a large noise margin.

The access time is 1.3 ns, with an estimated power consumption of 0.8 W per chip. Design of the FIFO allows convenient cascading of a series of chips for memory extension. The HBT clock driver and HMESFET circuits are also in fabrication now. We used an off-the-shelf CMOS transceiver chip to interface the GaAs chips with a personal computer.

4 Packaging Considerations

The HBT ADC and clock driver will be packaged with commercially available multi-layer ceramic packages, such as the TriQuint MLC132/64. They can be tested by direct capture of output data using logic analyzers (e.g. HP 16500) up to 1.5 GSps operation (with the help of on-chip demux). The memory chips will be packaged in MSI 3H32CM chip carriers. The data acquisition system will be assembled on a printed-circuit board (PCB). Figure 9 shows the design of the eight-layer PCB. The outer two layers will be implemented with polymide for high performance microstrips. The inner six power and ground layers will be realized with FR4 material for strength and low cost. The layout of one outer layer is shown in Fig. 10(a). The PCB measures $20cm \times 15cm$. The ADC and clock driver will be mounted back-to-back, so that the ADC output and clock signals can be routed to each FIFO identically. The PCB provides a clock synthesizer, a DMA (direct memory access) interface to computer, and SMA connectors for analog and clock inputs.

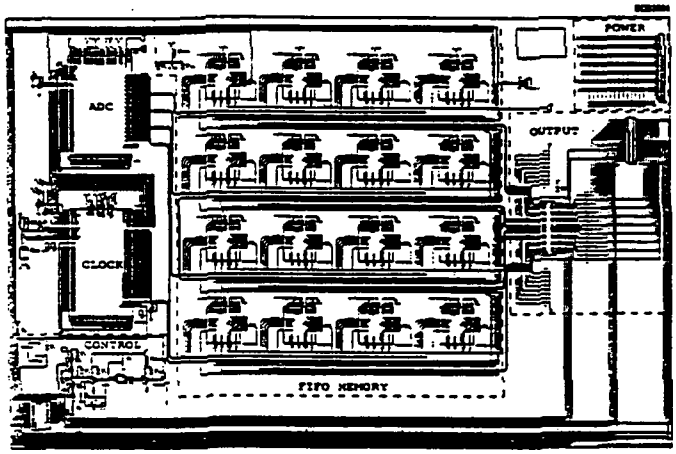


Figure 9: PCB design of the data acquisition system.

At Rockwell Science Center, advanced packaging technologies are being developed. One idea is to use Total Substrate Removal (also known as epitaxial layer lift-off or peeled film process) of the GaAs circuits, and attach them on a substrate of high thermal conductivity (e.g. diamond or AlN) in high density. The thin (about $5\mu m$) circuits and high thermal

conductivity of the substrate allow extra-dense chip packaging. This shortens the metal interconnect, thus increasing the allowed signal rates. A conventional Multi-Chip Module (MCM), using polyimide and Au/Cu wires, can be used for signal interconnect and power supply. The circuits on diamond or AlN substrate will be flip-chip bonded to the MCM. We estimated that $15cm \times 10cm$ of PC board area can be reduced to $2cm \times 1.1cm$ of the new MCM are a, as shown in Fig. 10(b). The packaging concept is illustrated in Fig.11. In addition to the size and speed benefits, the new packaging allows for increased ADC accuracy due to lower temperature variation over the chip, and improved lifetime/reliability due to improved thermal management. Furthermore, the HBT clock driver circuit may not be needed since the clocks to the memory chips may be distributed with a common bus in the new packaging. This will result in about 20% reduction in power consumption.

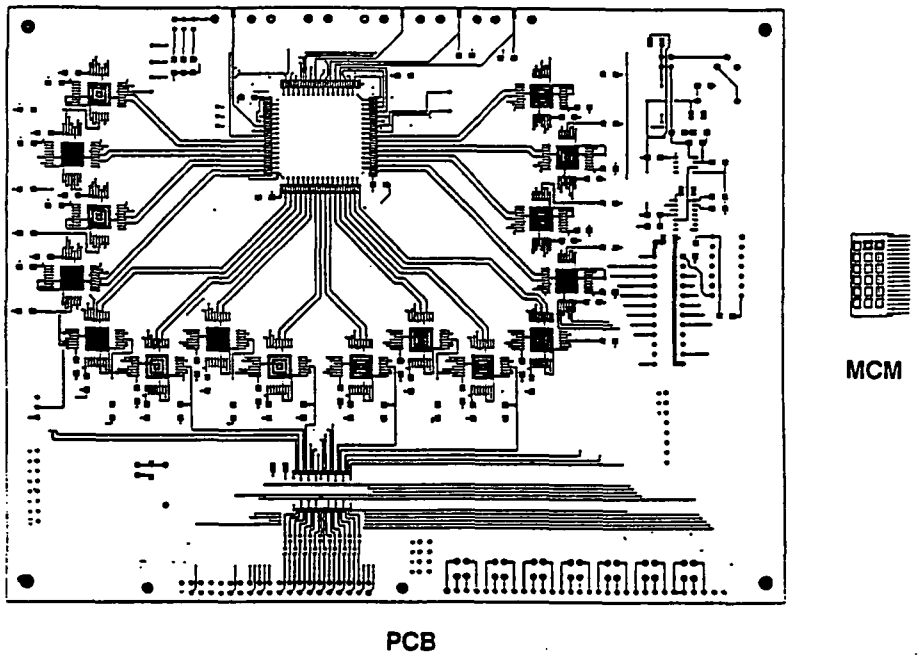


Figure 10: (a) Layout of one outer layer of the PCB. (b) Schematic layout of an advanced MCM, that contains the two HBT circuits and sixteen HFET memory chips.

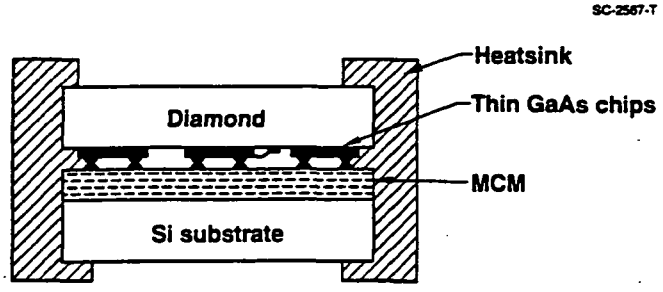


Figure 11: Advanced MCM packaging concept.

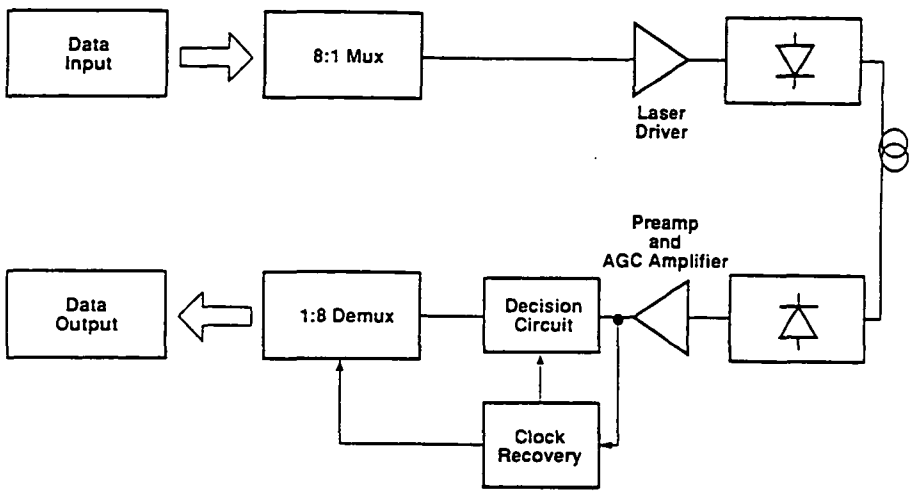


Figure 12: A simplified lightwave communication system.

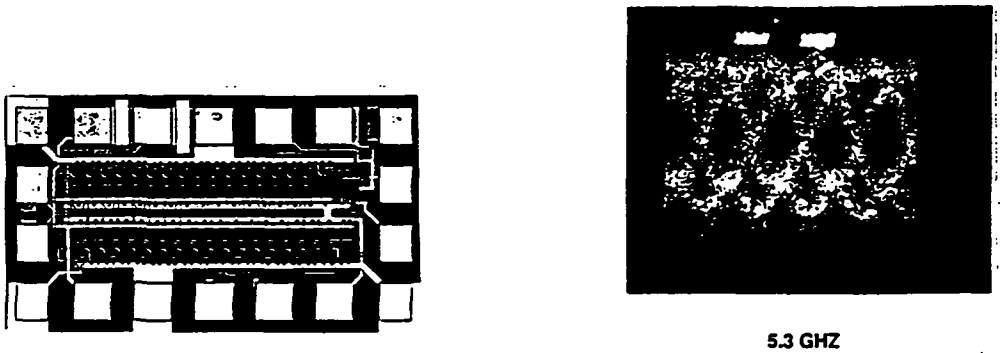
5 HBT Circuits for Communications

HBT technology offers great capabilities to meet speed, accuracy, and power requirements of many communication systems. High-performance HBT circuits for lightwave and wireless communications have been demonstrated and many more are being developed.

Figure 12 shows a simplified lightwave communication system. Parallel digital data are serialized by a mux into a bit string of high data rate. A laser driver receives the bit string and generates current pulses to modulate a laser diode. The light signal is transmitted to a photo-detector through an optical fiber. The detector converts the light energy back to current pulses that are then translated into voltage pulses by a transimpedance amplifier. The signal is amplified, and a clock recovery circuit is used to regenerate synchronized clocks for the decision circuit and demux. Finally, the demux de-serializes the data into parallel outputs. As mentioned in Section 2, we have demonstrated ultra-high speed mux and demux circuits for this application. The 20 GSps voltage comparator can serve as the decision circuit to distinguish data 1's from 0's. In addition, we have realized an 11 Gb/s HBT laser driver that can modulate more than 50 mA of AC current into a 50 ohm load [6]. For the receiver side, a D C coupled, differential transimpedance amplifier has been demonstrated [12]. We have also realized DC coupled amplifiers of 10-dB gain and 14 GHz bandwidth. An HBT phase detector for clock recovery operated at 5 GHz. We are now developing complete clock recovery circuits for 2.5 and 10 Gb/s operation. These building blocks will be integrated monolithically into transmitter and receiver circuits, and in 4- to 8-channel arrays, for various fiber optic applications.

Pseudo-random bit generators (PRBS) are used for scrambling and descrambling data and for testing communication modules (as part of a bit-error-rate detector). They can also be used in pseudo-random code modulated laser altimeters for NASA [13]. We have realized an HBT PRBS (with a 15-stage shift register) and tested it up to 5.3 GHz, as shown in Fig. 13.

For wireless communication, power HBTs implemented with thebaseline technology have



(a) (b)

Figure 13: (a) Microphotograph of a fabricated HBT PRBS. (b) Output eye patterns at 1 and 5.3 GHz.

shown > 60% of power added efficiency at 825-850 MHz. High performance HBT mixers were realized. We are developing an HBT based DDS (direct digital synthesizer) for I and Q demodulation. The DDS includes a DAC (digital to analog converter), an accumulator, and a ROM for sine/cosine look-up table. In addition, we are designing high-resolution sigma-delta ADCs with high speed HBT modulators. These ADCs are aimed at 12- to 14-bit 100 MSps for digital radios.

6 Development of Advanced Technologies Involving HBTs

HBT technologies with material systems other than GaAs are being developed at many research laboratories. InP based HBTs and integrated circuits have been demonstrated [14-16]. The technology offers low power, high speed, and OEIC compatibility with 1.3 and 1.55μm lights. It does not have good Schottky diodes, however. The breakdown voltages of simple single-heterojunction transistors with InGaAs collectors are low, compared with those of GaAs HBTs. High-performance Si/Ge HBTs have also been realized, with ft's up to 75 GHz [17]. Si/Ge HBT technology allows use of many established techniques of Si transistor technologies. It suffers, however, because of conductive Si substrate. This leads to finite collector to substrate capacitance and prohibits integration of analog, digital, and microwave circuits. Other III-V HBTs being investigated include GaInP/GaAs HBTs (for wider bandgap and less DX problems than AlGaAs), InAs and GaSb based HBTs (for low power), and GaP/AlGaP HBTs (for high temperature electronics).

Monolithic integration of III-V HBTs with field-effect transistors (FETs) are being pursued. FETs extend the capability of HBTs by providing high input impedance, low noise, active loads, and current sources and sinks. A simple manufacturable process of GaAs HBT and MESFET process has been proposed by a UCSD/Rockwell collaboration team [18]. High performance MESFETs (with 300 mS/mm transconductance, 11.5 GHz f_t, and 16 GHz f_{max}) have been demonstrated on the baseline HBT material. Integration of GaAs HBTs and HEMTs are being developed at Rockwell, supported by Air Force Wright Laboratory. This development involves regrowth of HEMT layers on HBT wafers, followed by

etching away HEMT layers from areas devoted to HBT. SPICE simulations predict that sub-0.5 ns 1K SRAMs and 7-bit 4-GSps sample-and-hold circuits can be implemented with this technology. Monolithic optoelectronic receiver circuits have been fabricated with the baseline HBT technology [19]. The base-collector junction of HBT layers was used for the photodetector. The OEICs have measured bandwidths as high as 13 GHz for optical signals in the $0.8\mu m$ band. This implies suitability for 17 Gb/s digital transmission with a -12 dBm sensitivity for 10^{-9} bit error rate. Furthermore, Rockwell has also demonstrated integration of complementary Pnp and Npn HBTs, under an ARO contract [20]. The Pnp transistors exhibit $f_t = 20$ GHz and $f_{max} = 19$ GHz. Gain blocks implemented with the Pnp HBTs showed a gain of 8 dB and a 3 dB bandwidth of 6 GHz.

A Rockwell/Lincoln Laboratory/UCSD collaboration team has proposed integration of resonant tunneling diodes (RTDs) and HBTs [21]. A schematic cross section of the integrated RTD/HBT technology is shown in Fig. 14. RTD layers will be grown on top of HBT layers, and RTDs will be fabricated in series with HBTs. RTD/HBT integration achieves several objectives. When RTDs are integrated on the emitter side of HBTs, circuits with high functional density can be obtained, similar to that of RHETs-resonant tunneling hot electron transistors [22]. Active device counts can be reduced by about a factor of four for same logic functions. RTDs can be integrated on the collector sides of HBTs as loads with differential negative resistance (NDR) to realize circuits of minimum static power consumption. Power reduction of about a factor of three is achievable. The high density and low power features can be combined on the same circuits. We have experimentally verified the operation of a basic RTD/HBT gate with a discrete RTD and a discrete HBT packaged together. Subnanosecond operation of RTD/HBT circuits has been demonstrated with SPICE simulations.

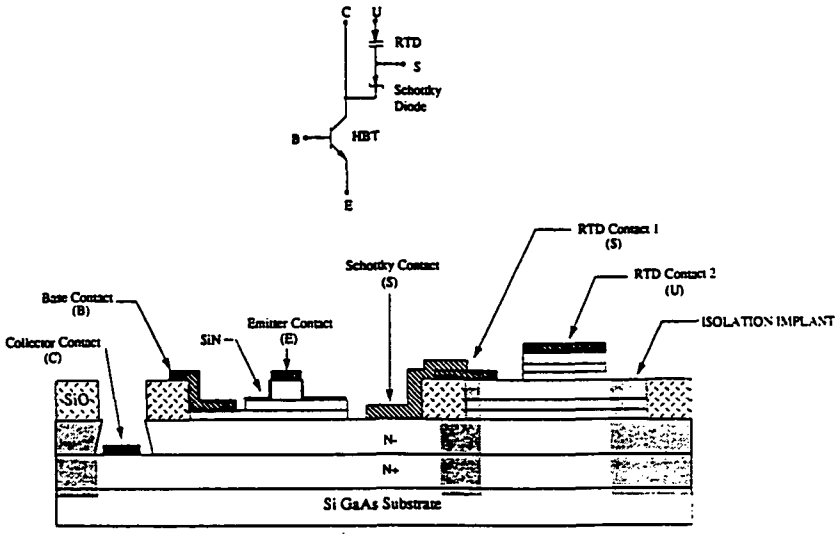


Figure 14: Schematic cross section of integrated RTD/HBT technology

7 Summary

Rockwell has established a manufacturable AlGaAs/GaAs HBT technology in a production facility. The technology features emitter-up single heterojunction Npn transistors with integrated Schottky diodes, NiCr resistors, MIM capacitors, and three levels of metal interconnect. The HBT has a current gain of about 100, and f_t and f_{max} of about 55 GHz. Rockwell has announced initial HBT products based on this technology.

In this paper, we have described a 1.5 GSps 8-bit data acquisition system. The system contains an HBT ADC, an HBT clock driver, 16 HMESFET 1K FIFO memory chips, and Si interface ICs. The GaAs circuits are in fabrication. The system will be assembled on a printed circuit board, with DMA interface to a personal computer. We have presented advanced packaging ideas for reduction in size and power, and for improved performance. Ultra-high performance HBT circuits for lightwave and wireless communications have been demonstrated or are being developed. Integration of these building block circuits into transceiver modules and arrays is being pursued. HBT related technologies being developed at Rockwell and its collaborating institutes include InP based HBTs, GaAs HBT/FET integration, OEICs, and RTD/HBT integration. With successful realization of these technologies, we will be able to support many system innovations and improvements for NASA.

8 Acknowledgments

The work was supported in part by a NASA/ONR contract, N00014-83-C-0347. The authors acknowledge Warner Miller of NASA/GODDARD and Yoon Soo Park of ONR for their support. We thank D.T. Cheung for support and encouragement, William Colleran of UCLA for contributions to HBT ADC design, and Alfred Yen for PCB layout. We are g rateful to the HBT processing staff and material growth staff at Rockwell Science Center and Microelectronics Technology Center for their dedicated work.

References

[1] P.M. Asbeck, et al., "GaAlAs/GaAs Heterojunction Bipolar Transistors: Issues and Prospects for Application," IEEE Trans. Elec. Dev., Vol. 36, No. 10, 1989, p. 2032.

[2] M.F. Chang, et al., "Heterojunction Bipolar Transistor Technology for Analog and Digital Applications," Dig. of GOMAC, 1990, p. 29.

[3] R.B. Nubling, N.H. Sheng, K.C. Wang, M.F. Chang, W.J. Ho, G.J. Sullivan, C.W. Farley, and P.M. Asbeck, "25 GHz HBT Frequency Dividers," Tech. Dig. 1989 GaAs IC Symp. pp. 125-128.

[4] J.D. George et al., "A High-Speed Gate Array Implemented with AlGaAs/GaAs Heterojunction Bipolar Transistors," ISSCC Dig. Tech. Papers, 1989, pp. 186-187.

[5] K.C. Wang et al., "A 15-GHz Gate Array Implemented with AlGaAs/GaAs Heterojunction Bipolar Transistors," IEEE J. of Sol. Sta. Cir., Vol. 26, No. 11, November 1991, pp. 1669-1672.

[6] K. Runge, R.D. Standley, D. Daniel, J.L. Gimlett, R.B. Nubling, R.L. Pierson, S. Beccue, K.C. Wang, N.H. Sheng, M.F. Chang, D.M. Chen, and P.M. Asbeck, "AlGaAs/GaAs HBT ICs for High Speed Lightwave Transmission Systems," to be published in IEEE J. of Solid-State Circuits, Oct. 1992.

[7] R.B. Nubling, J. Yu, K.C. Wang, P.M. Asbeck, N.H. Sheng, M.A. McDonald, A.J. Price, and D.M. Chen, "High-Speed 8:1 Multiplexer and 1:8 Demultiplexer Implemented with AlGaAs/GaAs HBTs," IEEE J. of Solid State Circuits, Vol 26, No. 10, Oct. 1991, pp. 1354-1361.

[8] K. Runge, J.L. Gimlett, R.B. Nubling, K.C. Wang, M.F. Chang, R.L. Pierson, and P.M. Asbeck, "20 Gbit/s AlGaAs/GaAs HBT Decision Circuit IC," Elec. Letters, Vol. 27, No. 25, Dec. 1991, pp. 2376-2378.

[9] K. Poulton, et al., "A 2 Gs/s HBT Sample and Hold," Tech. Dig. of GaAs IC Symp., 1988, p. 88.

[10] K.C. Wang, P.M. Asbeck, M.F. Chang, R.B. Nubling, R.L. Pierson, D.M. Chen, J.J. Corcoran, K. Poulton, and K.L. Knudsen. "Heterojunction Bipolar Technology for Analog to Digital Conversion," 1991 GOMAC Dig. of Papers, pp. 75-79.

[11] R.J. van der Plassche, et al., "An 8-Bit 100 MHz Full Nyquist Analog-to-Digital Converter," IEEE J. of Solid-State Circuits, Vol 23, 1988, p. 1334.

[12] G.-K. Chang, T.P. Liu, J.L. Gimlett, H. Shirokmann, M.Z. Iqbal, J.R. Hayes, and K.C. Wang, "A Direct-Current Coupled, All Differential Optical Receiver for High-Bit-Rate Sonet Systems," IEEE Photonics Tech. Letts., Vol. 4, No. 4, April 1992, pp. 384-386.

[13] James B. Abshire, "Pseudo-Random Code Modulated AlGaAs Laser Altimeter," private communication.

[14] C.W. Farley, K.C. Wang, M.F. Chang, P.M. Asbeck, R.B. Nubling, N. H. Sheng, R. Pierson and G. J. Sullivan, "A High Speed Low-Power Divide-by-4 Frequency Divider Implemented with AlInAs/GaInAs HBTs," IEEE Elec. Dev. Letts. Vol. 10, No. 8, August 1989, pp. 377-379.

[15] J.F. Jensen, W.E. Stanchina, R.A. Metzger, D.B. Rensch, Y.K. Allen, M.W. Pierce, and T.V. Kargodorian, "High Speed Dual Modulus Dividers using AlInAs-GaI nAs HBT IC Technology," Tech. Dig. 1990 GaAs IC Symp. pp. 41-44.

[16] R.N. Nottenburg, A.F.J. Levi, Y.K. Chen, B. Jalali, M.B. Panish, and A.Y. Cho, "InP-Based Heterostructure Bipolar Transistors," Tech. Dig. 1989 GaAs IC Symp. pp. 135-138.

[17] C.T. Chuang, K. Chin, J.M.C. Stork, G.L. Patton, E.F. Crabbe, and J.H. Comfort, "On the Leverage of High-ft Transistors for Advanced High-Speed Bipolar Circuits," IEEE J. of Solid State Circuits, Vol. 27, No. 2, February 1992, pp. 225-228.

[18] D. Cheskis, C.E. Chang, P.M. Asbeck, M.F. Chang, R.L. Pierson, and A. Sailer, "Co-Integration of GaAlAs/GaAs HBTs and GaAs FETs with a Simple, Manufacturable Process," to be presented at 1992 IEDM.

[19] K.D. Pedrotti, N.H. Sheng, R.L. Pierson, C.W. Farley, M.J. Rosker, and M.F. Chang, "Monolithic Ultrahigh-Speed GaAs HBT Optical Integrated Receivers," Tech. Dig. 1991 GaAs IC Symp., pp. 205-208.

[20] C.W. Farley, R.J. Anderson, R.B. Bernescut, R.W. Grant, M.F. Chang, K.C. Wang, R.B. Nubling and N.H. Sheng, " High Speed AlGaAs/GaAs Complementary HBT Technology Realized by Multiple MBE Growth and Merged Processing," Tech. Dig. 1991 IEDM, pp. 36.3.1-4.

[21] E.R. Brown, M.A. Hollis, F.W. Smith, K.C. Wang and P.M. Asbeck, "Resonant-Tunneling-Diode Loads: Speed Limits and Applications in Fast Logic Circuits," Tech. Dig. 1992 ISSCC, Paper TP8.6.

[22] M. Takatsu, K. Imamura, H. Ohnishi, T. Mori, T. Adachihara, S. Muto and N. Yokoyama, "Logic Circuits Using Resonant-Tunneling Hot Electron Transistors (RHETs)," Tech. Dig. 1991 GaAs IC Symp., pp. 95-98.

Session 2
VLSI Architectures

Chairman: Kelly Cameron

A New Eddy Current Model for Magnetic Bearing Control System Design [1]

Joseph J. Feeley and Daniel J. Ahlstrom
Department of Electrical Engineering
Microelectronics Research Center
NASA Space Engineering Research Center
University of Idaho
Moscow, Idaho 83843

Abstract - **This paper describes a new VLSI-based controller for the implementation of a Linear-Quadratic-Gaussian (LQG) theory-based control system. Use of the controller is demonstrated by design of a controller for a magnetic bearing and its performance is evaluated by computer simulation.**

1 Introduction

Magnetic levitation is being used in an increasing number of applications to support rotating or reciprocating shafts. In this setting, the assemblage of power supplies, control circuits, actuator coils, pole-pieces, amplifiers, and position sensors is termed a magnetic bearing. The control of magnetic bearings is discussed in [1] and the references cited therein. Traditional control system design studies require, first of all, a simple and accurate model of the system to be controlled. The lack of such a model for magnetic actuators containing significant eddy currents has hampered recent control system design efforts [1, 2]. The purpose of this paper is to introduce a new mathematical model of eddy currents in a magnetic actuator. The model is relatively simple, appears reasonably accurate, and is convenient to use in control system design. The model is based on Maxwell's electromagnetic equations and is simplified using reasonable, ad hoc assumptions consistent with a magnetic bearing application.

A complete set of equations for the mathematical model of a magnetic bearing is presented in Section 2, and conclusions and directions of future work are discussed in Section 3.

2 Mathematical Model

For convenience in developing the mathematical model, the magnetic bearing system is separated into three subsystems: a) the magnetic actuator, b) the magnetic force production mechanism, and c) the shaft. Each of these subsystems is treated separately in the following subsections. For the sake of specificity, the model development is applied to specific magnetic bearing of interest to NASA. Figure 1 is a schematic cross-sectional view of a magnetic bearing used in a cryogenic refrigerator developed by NASA for certain space applications. The refrigerator, magnetic bearings, and associated control systems were designed and constructed by Phillips Laboratories [2].

[1]This research was supported in part by NASA under Space Engineering Research Grant NAGW-1406 and by the NSF under Research Initiation Grant MIP-9109618.

2.1.2

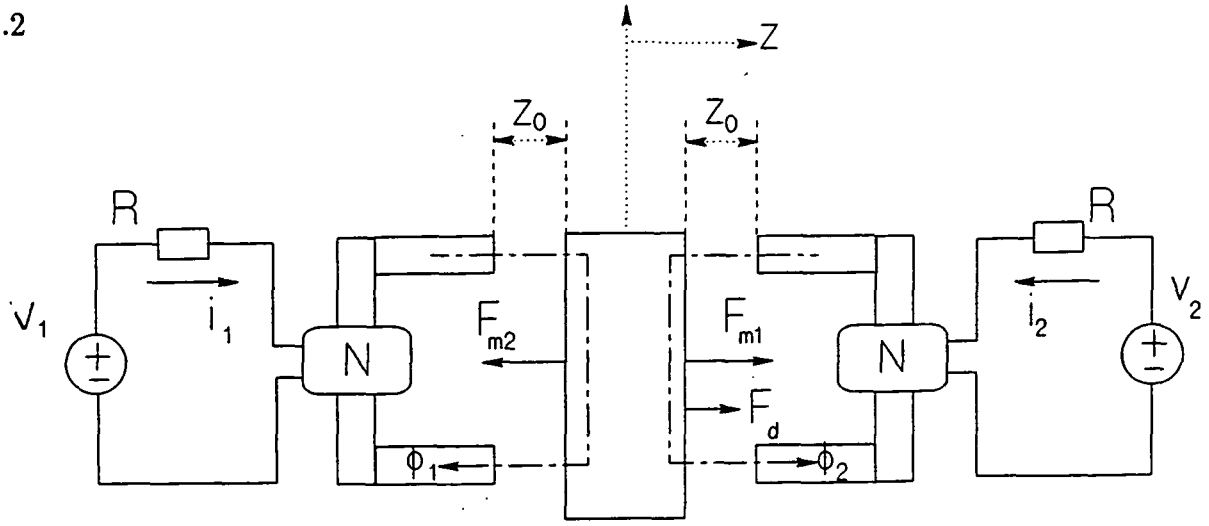


Figure 1: Cross-sectional view of magnetic bearing.

Figure 1 shows one end of a shaft centered between two diametrically opposed magnetic actuators. The actuators are activated by separate control circuits energized by voltage sources v_1 and v_2. The voltage sources are controlled in a coordinated way by a feedback controller to produce the forces required to maintain the shaft in the desired position in spite of disturbance forces. Another pair of actuators and control circuits are used to control the position of the shaft in the orthogonal plane containing the axis of the shaft. No coupling of forces between the two planes is considered and the two actuator pairs are controlled independently.

2.1 Magnetic Actuator

The purpose of this section is to develop a mathematical model relating the voltage applied to the actuator circuit to the current developed in that circuit. Applying Kirchoff's voltage law to the circuit on the left in Fig. 1 gives

$$v_1 = Ri_1 + N\frac{d\Phi_1}{dt} \tag{1}$$

where R is the resistance of the actuator circuit, i_1 is the coil current, N is the number of turns in the actuator coil, and Φ_1 is the flux in the magnetic circuit. The flux is related to the current by the magneto motive force (2) relationship

$$\Phi_1 = \frac{Ni_1}{\mathcal{R}} \tag{2}$$

where $\mathcal{R}$ is the reluctance of the magnetic circuit. Assuming the reluctance of the two air gaps is much larger than that of the magnetic material in the pole-piece, $\mathcal{R}$ can be approximated by

$$\mathcal{R} = 2\left(\frac{z_0 + z}{\mu_0 A}\right) \tag{3}$$

where $z_0 + z$ is the length of one air gap, μ_0 is the permeability of free space, and A is the area of pole-piece normal to the flux direction. Substituting (3) and (2) in (1) and carrying

out the indicated differentiation gives

$$v_1 = Ri_1 + \frac{N^2\mu_0 A}{2}\left[\frac{1}{(z_0+z)}\frac{di_1}{dt} - \frac{i_1}{(z_0+z)^2}\frac{dz}{dt}\right] \tag{4}$$

Similar analysis of the circuit on the right leads to the following expression for v_2

$$v_2 = Ri_2 + \frac{N^2\mu_0 A}{2}\left[\frac{1}{(z_0-z)}\frac{di_2}{dt} + \frac{i_2}{(z_0-z)^2}\frac{dz}{dt}\right] \tag{5}$$

Equations (4) and (5) are the desired expressions showing the relationships among control voltage, actuator current, shaft position, and shaft velocity.

2.2 Magnetic Force Production Mechanism

The purpose of this section is to develop an expression relating the current in the actuator coil to the electromagnetic force exerted on the shaft. The differential electromagnetic energy stored in an air gap as depicted in Fig. 1 is given as [3]

$$dw = \frac{1}{2}\frac{B^2 A}{\mu_0}dz \tag{6}$$

where $B = \Phi/A$ is the flux density. From Newton's second law the differential energy can also be expressed as a force acting through a differential distance

$$dw = F_m dz \tag{7}$$

Combining (6) and (7) and accounting for the presence of two air gaps, the attractive forces shown in Fig. 1 are

$$F_{m1} = \frac{B_1^2 A}{\mu_0} \tag{8}$$

and

$$F_{m2} = \frac{B_2^2 A}{\mu_0} \tag{9}$$

If the current in the actuator coil is constant, the flux it produces is constant, in both space and time, and no eddy currents are induced in the pole-piece. Further, if the majority of the mmf drop occurs in the air gaps, the mmf relationship can be used to express the flux density in terms of the coil current. The flux densities in the two magnetic circuits can then be written as

$$B_1 = \frac{\mu_0 N i_1}{2(z_0+z)} \tag{10}$$

and

$$B_2 = \frac{\mu_0 N i_2}{2(z_0 - z)} \tag{11}$$

Substituting (10) and (11) in (8) and (9) yields the usual [2] expressions for electromagnetic force in terms of the coil current and the length of the air gap

$$F_{m1} = \frac{\mu_0 N^2 A}{4}\left(\frac{i_1}{z_0 + z}\right)^2 \tag{12}$$

and

$$F_{m2} = \frac{\mu_0 N^2 A}{4}\left(\frac{i_2}{z_0 - z}\right)^2 \tag{13}$$

If, however, the current in the actuator coil changes with time, eddy currents are induced in the pole-piece. The eddy currents, in turn, produce a reactive flux in opposition to the original flux. The result of the superposition of the two effects is a reduced net flux in the pole-piece. The resultant spatial flux distribution in the pole-piece in a plane orthogonal to the flux direction is governed by the diffusion-type equation [4]

$$\frac{\partial^2 B(y,x,t)}{\partial x^2} + \frac{\partial^2 B(x,y,t)}{\partial y^2} = \sigma\mu\frac{\partial B(x,y,t)}{\partial t} \tag{14}$$

where σ and μ are the conductivity and permeability of the pole-piece material.

Equation (14) can be solved analytically for a bar whose length is long relative to its cross-sectional dimensions, $2a$ and $2b$. When excited by a sinusoidally varying actuator coil current

$$i_1(t) = I_1 \cos ax, \tag{15}$$

(10) and (15) can be used to develop the boundary conditions

$$B_1(x,y)|x \pm a, y \pm b = \frac{\mu_0 N I_1}{2(z_0 + z)} \tag{16}$$

Subject to this boundary condition, the sinusoidal steady state solution of (14) is [4]

$$B_1(x,y,z,\omega) = \frac{\mu_0 N I_1}{2(z_0 + z)}\left[\frac{\cosh(\alpha y)}{\cosh(\alpha b)} + \sum_{k=1,3,5}^{\infty} P_k \cos\left(\frac{k\pi y}{2b}\right)\cosh\left(x\sqrt{\beta_k}\right)\right] \tag{17}$$

where:

$$P_k = \frac{4\alpha^2}{k\pi\beta_k}\frac{\sin\left(\frac{k\pi}{2}\right)}{\cosh\left(a\sqrt{\beta_k}\right)} \tag{18}$$

$$\alpha = \sqrt{j\omega\sigma\mu} \tag{19}$$

and

$$\beta_k = \alpha^2 + \left(\frac{k\pi}{2b}\right)^2 \tag{20}$$

$B_1(x,y,z,\omega)$ is a complex number representing the magnitude and phase of the flux density in the pole-piece. The normalized magnitude of $B_1(x,y,z,\omega)$ is plotted in Fig. 2 with $z = 0$ and $\omega = 10$ rad/sec. The figure clearly shows the effect of eddy currents in depressing the magnitude of the flux in the central regions of the pole-piece.

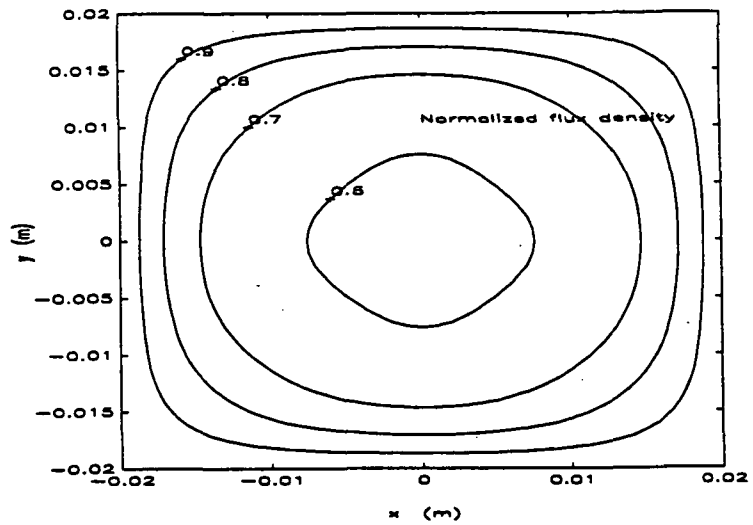


Figure 2: Contour plot of normalized magnitude of flux density.

In order to develop a simple, frequency dependent model of the force produced by the actuator current, the flux density as given by (17) is averaged over pole-piece face area yielding

$$\overline{B_1}(z,\omega) = \frac{\mu_0 N I_1}{4ab(z_0+z)}\left[\frac{1}{\alpha}tanh(\alpha b) + \left(\frac{4\alpha}{\pi}\right)^2 b \sum_{k=1,3,5}^{\infty} \frac{\sqrt{\beta_k}}{k^2\beta_k^2} tanh\left(a\sqrt{\beta_k}\right)\right] \tag{21}$$

The frequency response of $\overline{B_1}(z,\omega)$ is dominated by the $\frac{tanh(\alpha b)}{\alpha}$ term in (21), and log-magnitude and phase plots would reveal the familiar [2] high frequency roll-off of approximately 10 db/decade and 46 deg phase lag at high frequencies due to the $\sqrt{j\omega}$ factor in the α term. Substituting the spatially averaged flux density, as given by (21), for the constant flux density,B_1, in the force equation, (8), yields the following frequency dependent approximation for the magnetic force produced by a time-varying current

$$F_{m1}(z,\omega) = \frac{\mu_0 N^2}{A}\left(\frac{I_1}{z_0+z}\right)^2\left[\frac{1}{\alpha}tanh(\alpha b) + \left(\frac{4\alpha}{\pi}\right)^2 b \sum_{k=1,3,5}^{\infty} \frac{\sqrt{\beta_k}}{k^2\beta_k^2} \tanh\left(a\sqrt{\beta_k}\right)\right]^2 \tag{22}$$

Similar analysis yields a similar expression for the force produced by the other actuator

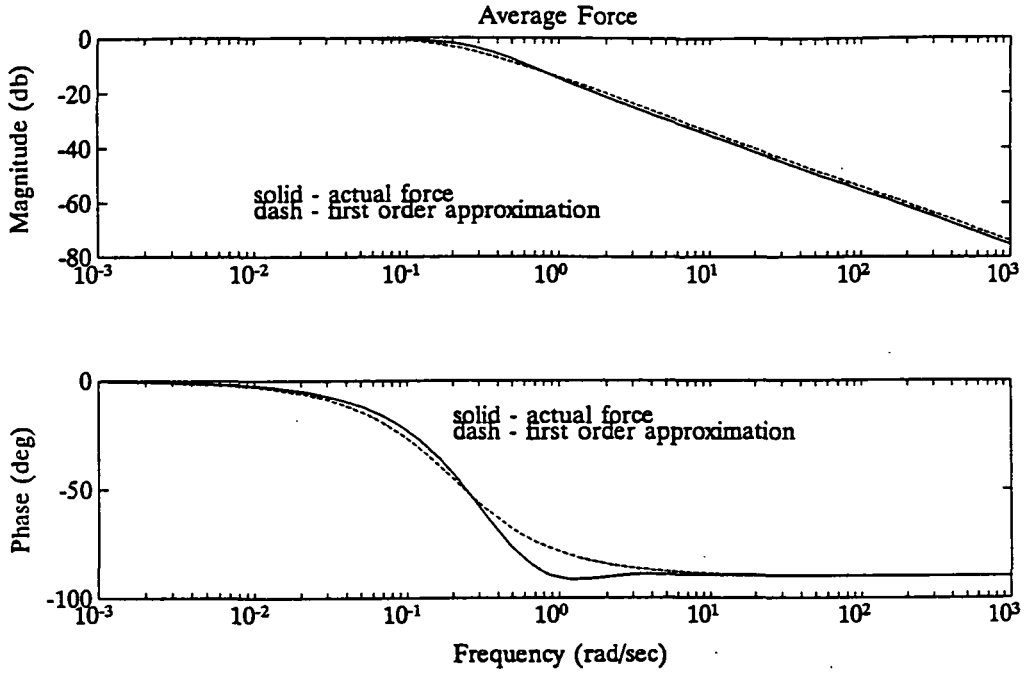


Figure 3: Normalized magnitude and phase of average electromagnetic force

$$F_{m2}(z,\omega) = \frac{\mu_0 N^2}{A}\left(\frac{I_2}{z_0 - z}\right)^2 \left[\frac{1}{\alpha} tanh(\alpha b) + \left(\frac{4\alpha}{\pi}\right)^2 b \sum_{k=1,3,5}^{\infty} \frac{\sqrt{\beta_k}}{k^2 \beta_k^2} \tanh\left(a\sqrt{\beta_k}\right)\right]^2 \qquad (23)$$

The normalized magnitude and phase of (22) are plotted in Fig. 3 and clearly show a first-order type response with a high frequency roll-off of approximately 20 db/decade and a phase lag at high frequencies of about 90 deg. The response of a first-order system of the form

$$\hat{F}(j\omega) = \frac{1}{1 + j\frac{\omega}{\omega_b}} \qquad (24)$$

is also plotted in Fig. 3. Comparison of the responses of the two functions indicates that, at least with respect to sinusoidal steady state conditions, the exact expression of (22) can be conveniently and accurately represented by the approximate expression of (24). While the break frequency ω_b is closely related to the $\sigma\mu$ product, an optimal value in a particular application is easily found numerically. The final forms of the frequency domain approximations for the magnetic forces are then

$$F_{m1}(j\omega) = \frac{\mu_0 N^2}{A}\left(\frac{I_1}{z_0 + z}\right)^2 \frac{1}{i + j\frac{\omega}{\omega_b}} \qquad (25)$$

and

$$F_{m2}(z\omega) = \frac{\mu_0 N^2}{A}\left(\frac{I_2}{z_0 - z}\right)^2 \frac{1}{i + j\frac{\omega}{\omega_b}} \tag{26}$$

While (25) and (26) are convenient frequency domain models of the response of magnetic force to sinusoidal steady state actuator current, it is also useful, for many design and analysis tools, to have equivalent time domain expressions. This can be accomplished with little justification, other than it produces an expression that appears to have the right form, by replacing $j\omega$ with s, holding i and z constant, and taking inverse Laplace transforms. The results are the first order differential equations

$$\frac{dF_{m1}}{dt} = -\omega_0 F_{m1} + \left(\frac{\mu_0 N^2 A \omega_0}{4}\right)\left(\frac{i_1}{z_0 + z}\right)^2 \tag{27}$$

and

$$\frac{dF_{m2}}{dt} = -\omega_0 F_{m2} + \left(\frac{\mu_0 N^2 A \omega_0}{4}\right)\left(\frac{i_2}{z_0 - z}\right)^2 \tag{28}$$

2.3 The Shaft

The objective of this section is to develop the equations of motion describing the dynamic response of the shaft to the forces imposed upon it. Referring to Fig. 1, application of Newton's second law gives rise to the following equations of motion

$$\frac{dz}{dt} = v \tag{29}$$

$$\frac{dv}{dt} = \frac{1}{M}\left[F_{m2} - F_{m1} - K_f v|v + F_d\right] \tag{30}$$

where v is the velocity of the shaft, M is its mass, K_f is a coefficient of viscous friction, and F_d is a disturbance force.

3 Conclusions and Recommendations

A new nonlinear dynamic model for use in the design and analysis of control systems for magnetic bearings has been presented. The time domain representation of the model is given by Equations (4), (5), (27), (28), (29), and (30). These equations are summarized below as a set of first order, nonlinear differential equations in state-variable form.

$$\frac{di_1}{dt} = -\frac{2(z_0 + z)Ri_1}{N^2\mu_0 A} + \frac{i_1 v}{z_0 + z} + \frac{2(z_0 + z)v_1}{N^2\mu_0 A} \tag{31}$$

$$\frac{di_2}{dt} = -\frac{2(z_0 - z)Ri_2}{N^2\mu_0 A} + \frac{i_2 v}{z_0 - z} + \frac{2(z_0 - z)v_2}{N^2\mu_0 A} \tag{32}$$

$$\frac{dF_{m1}}{dt} = -\omega_0 F_{m1} + \left(\frac{\mu_0 N^2 A \omega_0}{4}\right)\left(\frac{i_1}{z_0 + z}\right)^2 \tag{27}$$

$$\frac{dF_{m2}}{dt} = -\omega_0 F_{m2} + \left(\frac{\mu_0 N^2 A \omega_0}{4}\right)\left(\frac{i_2}{z_0 - z}\right)^2 \tag{28}$$

$$\frac{dz}{dt} = v \tag{29}$$

$$\frac{dz}{dt} = \frac{1}{M}\left[F_{m2} - F_{m1} - K_f v|v + F_d\right] \tag{30}$$

These six equations, and their linearized counterparts, form a convenient basis form magnetic bearing control system analysis and design. Research continues in two areas. First, to validate the proposed model with experimental data and second, to develop alternative control system designs compatible with VLSI implementation. Control system designs currently under investigation are based on the following approaches: a) classical frequency domain methods, b) modern H_2 and H_∞ methods, and c) neural network/fuzzy control methods. Results of these efforts will be published in future papers.

References

[1] J. J. Feeley, G. M. Niederauer, and D. J. Ahlstrom, "Fuzzy Control of Magnetic Bearings", NASA SERC Symposium on VLSI Design, 1991.

[2] F. Stolfi et. al., "Design and Fabrication of a Long-Life Stirling Cycle Cooler for Space Application", Phillips Laboratories report, March, 1983.

[3] Sadiku, M. N. O., "Elements of Electromagnetics", Saunders College Publishing, 1989.

[4] Stoll, R. L., "The Analysis of Eddy Currents", Clarendon Press, 1974.

Design of a New Squaring Function for the Viterbi Algorithm

Aria Eshraghi, Terri Fiez and Thomas Fischer
Washington State University
Pullman WA

Kel Winters
Advanced Hardware Architectures
Moscow ID 83843

Abstract- **A new algorithm and hardware implementation of the Viterbi squaring function has been developed. The use of an approximation squaring technique preserves the Viterbi performance as is demonstrated by Monte-Carlo simulations. Additionally, the 16-bit approximate squaring implementation is expected to require one-fourth the area and operate at three times the speed of the conventional squaring implementation.**

1 Introduction

The Viterbi algorithm [1] is used to encode and decode data in communication systems. This algorithm has been utilized in trellis coded modulation (TCM), trellis shaping (TS), and trellis coded quantization (TCQ). Use of the Viterbi algorithm at the TCM receiver results in 3-6 dB improvement in signal-to-noise ratio (SNR) for a given error rate [2]. In TS, the use of the Viterbi algorithm at the transmitter results in a reduction of 0.9 dB in the transmitted energy [3]. In TCQ, the Viterbi algorithm is used as a means of data compression, showing better performance than realizable vector quantizers [4] for the encoding of a memoryless source. The recent use of the Viterbi algorithm in communication products such as disk drives, tape drives, and modems has triggered the development of a single chip Viterbi processor [5].

This paper focuses on an efficient algorithm for performing the squaring function in the Viterbi processor. The new squaring function uses significantly less area than the conventional squaring techniques and, at the same time, boosts the speed of the processor without reducing its accuracy. In the first portion of this paper, the algorithm for the approximated squaring (APSQR) function is presented. Through Monte-Carlo simulations, it is shown in section two that this new squaring function results in accurate performance of the Viterbi algorithm. Section three covers the hardware implementation of the APSQR and compares the APSQR with a conventional squaring scheme.

2 Approximated Squaring Function

The Viterbi algorithm also known as forward dynamic programming [6], is an efficient search technique for determining the minimum cost sequence of states in a finite state machine. The

state transitional behavior of the finite state machine in time is mapped into a digraph known as the trellis diagram. As a result, only the path in the trellis diagram that agrees most closely with the received sequence is retained. The optimum path is determined by the path with the minimum mean square distance in the trellis diagram. Thus, the core computation in the Viterbi algorithm is:

$$\lambda_{t+1,j} = \min(\alpha_{k1,j} + \lambda_{t,k1} \ or \ \alpha_{k2,j} + \lambda_{t,k2}). \tag{1}$$

Where λ is the *path length*, α is the *path metric*, and $\lambda_{t+1,j}$ is the optimum path terminated at state j. Computing the path metric requires a dedicated squaring function ($\alpha_{k,j} = \beta_{k,j}^2$) which occupies approximately 30% of the total Viterbi processor chip area using conventional circuit techniques.

The conventional squaring technique relies on decomposing a number into the sum of the least significant bit and the remaining bits. The square is calculated using $(x+y)^2 = x^2 + 2 \cdot x \cdot y + y^2$ and applying it recursively to a shifted version of the remaining bits. By decomposing the number into the sum of the most significant bit and the remaining bits, it is possible to approximate the square of the number without degrading the accuracy of a Viterbi algorithm.

We will now describe the approximated squaring algorithm. Let $A = a_n a_{n-1} a_{n-2}..a_4 a_3 a_2 a_1$ be the number to be squared. Next, decompose A into the sum of the most significant bit and the remaining bits:

$$A = a_n 0\ 0..0\ 0\ 0\ 0 + a_{n-1} a_{n-2}..a_4 a_3 a_2 a_1. \tag{2}$$

We expand A^2, or rather A^{10} in binary, into:

$$A^{10} = (a_n 0\ 0\ ..0\ 0\ 0\ 0 + a_{n-1} a_{n-2}..a_4 a_3 a_2 a_1)^{10}. \tag{3}$$

Now applying $(x+y)^2 = x^2 + 2 \cdot x \cdot y + y^2$, A^{10} becomes:

$$A^{10} = a_n 0\ 0..0\ 0\ 0\ 0^{10} + (10)(a_{n-1} a_{n-2}..a_4 a_3 a_2 a_1)(a_n 0\ 0..0\ 0\ 0\ 0) + (a_{n-1} a_{n-2}..a_4 a_3 a_2 a_1)^{10}. \tag{4}$$

Neglecting the last term, A^{10} is approximated as:

$$A^{10} \simeq a_n 0\ 0..0\ 0\ 0\ 0^{10} + (10)(a_{n-1} a_{n-2}..a_4 a_3 a_2 a_1)(a_n 0\ 0..0\ 0\ 0\ 0). \tag{5}$$

This can be rewritten as:

$$A^{10} \simeq (a_n 0\ 0..0\ 0\ 0\ 0 + a_{n-1} a_{n-2}..a_4 a_3 a_2 a_1 0)(a_n 0\ 0..0\ 0\ 0\ 0). \tag{6}$$

The approximate squaring of A requires summing the two most significant bits (first term) and a left shift of $(a_n 0\ 0..0\ 0\ 0\ 0 + a_{n-1} a_{n-2}..a_4 a_3 a_2 a_1 0)$ by n-1 bits.

Fig.1 shows a plot of the output versus the input for both the approximated and the actual squaring functions. The maximum error is 25% and corresponds to input amplitudes of $2^n - 1$ where n is an integer. Using the approximated squaring function, the average error is approximately 10%.

3 Simulation Results

Although the average error due to the APSQR function is relatively high, the Viterbi algorithm inherently compensates for noise (or errors) in the data. To illustrate this property, Monte-Carlo simulations have been used to demonstrate the effect of approximating the squaring function on the TCQ.

The performance of the TCQ was measured for a Memoryless uniform source with a uniformly distributed codebook. The simulation results were based on encoding 1,000 different blocks of length 10,000 random data samples. The resolution for the source was selected to be 15 bits. This reduces the effect of finite resolution used in TCQ which results in better measurement of the APSQR function performance. The simulation results are shown in Table 1.

Table 1
The performance versus the bit rate
for the conventional and the approximated
squaring function.

Rate (Bits)	*Conventional* SNR *dB*	*Approximated* SNR *dB*
3	18.776	18.748
4	24.940	24.912
5	31.029	31.002
6	37.085	37.058

The bit rate is the number of bits per sampled input data. SNR is the expected signal to noise ratio, and it is given in dB. The variance was less than 0.006 dB in each case. As one can see, the degradation in performance of the TCQ due to utilization of APSQR is less than 0.03 dB for the expected value of the SNR. Thus, approximating the squared numbers produces a negligible error in the output. In the next section, it is shown that there is a significant saving in circuit area and increased speed with this implementation.

4 Hardware Implementation

To illustrate the efficiency of the APSQR function, the conventional squaring function is first described. The conventional implementation of the squaring function has a cellular architecture such that a single block is repetitively used in the design[7, 8]. Fig. 2 shows a cellular implementation of a 7-bit squaring function of [7]. Each cell contains a full adder and a multiplexer with connections shown. The input signals, $a_1..a_7$, enter at the top of Fig. 2 and propagate vertically through the cells. Simultaneously, the carry bits, C_i and C_o, propagate from right to left. This implementation uses $\sum_{i=3}^{n} i$ full adders and multiplexers for an n-bit squaring function. For example, a 7-bit squaring function requires:

$$\sum_{i=3}^{7} i = 3 + 4 + 5 + 6 + 7 = 25 \tag{7}$$

or 25 adders and 25 multiplexers. The number of adders and multiplexers increases quadratically with the number of input bits, i.e. $\sum_{i=3}^{n} i = (n^2 + n + 6)/2$. The increased hardware resulting from the increased number of input bits consumes excessive area and dynamic power, and significantly reduces the speed compared to the APSQR function.

The speed of the cellular squaring function is limited by the propagation of the carry bit through the last chain of adders. In Fig. 2 the worst case delay occurs as the carry bit propagates from cell 1 through cell 7. Note that the delay increases linearly as the number of input bits increases. Additionally, this design is not suitable for pipelining because of the two dimensional signal flow.

The proposed hardware implementation for a 7-bit APSQR is shown in Fig. 3. The controller circuit is responsible for detecting the most significant bit, and controlling the multiplexers for the proper number of left-shift operations. Three layers of multiplexers pass or shift their input by one, two, and three bits to the left. The final layer consists of modifier cells which sum the two most significant bits. The modifier cell becomes active if its input corresponds to the most significant bit. The detection of the most significant bits by the modifier cell is accomplished through observing the output of the controller.

The number of multiplexers used in this architecture with n input bits is upper bound by:

$$\text{Number of Multiplexers} = (n-1)\log_2(4 \cdot n) \tag{8}$$

As an example, a 7-bit input requires 35 multiplexers. Based on this equation, the number of transistors used by an n-bit APSQR function is:

$$\text{Number of Transistors} = 4 \cdot (n-1)\log_2(4 \cdot n) + 28 \cdot n + \delta \tag{9}$$

The first term corresponds to the number of transistors in each multiplexer and in this implementation, the multiplexers are composed of two bilateral switches (4 transistors). The term $28 \cdot n$ represents the number of transistors used in the design of modifier cells, and δ represents the number of transistors used in the controller design.

The number of transistors used in the cellular design is estimated as 4 transistors per multiplexer and 26 transistors per adder [9]. Thus, the number of transistors used in each cell of the conventional squaring function is 30, and the number of transistors used in an n-bit cellular squaring function is approximated as:

$$\text{Number of transistors} = (n^2 + n + 6)15 \tag{10}$$

Figure 4 shows the comparison between the two designs. A worst case number of transistors is estimated for the APSQR controller, $\delta = 200$. The dashed line represents the number of transistors used in the cellular design of the squaring function, and the solid line represents the number of transistors used in the APSQR function. With a 7-bit input, the APSQR is approximately 50% more area efficient than the cellular squaring function. As the number of input bits increases, the APSQR function becomes significantly more efficient. With a 16-bit input, the APSQR requires less than one-fourth the area of the conventional design.

The delay through the APSQR function is the sum of the delay through the controller, the delay through the multiplexers, and the delay through the modifier cell. Assuming the

controller is designed in two layers of logic, and the modifier cells are designed in three layers of logic, then the speed of the APSQR function can be approximated as 5-gate delays (plus a small delay through the tapered buffer at the output stage of the controller). This delay remains nearly constant despite the size of the input. For this reason, the speed of the APSQR is almost independent of the number of input bits. This is not the case with the conventional squaring scheme. The worst case delay is the propagation of the carry bit through the last chain of adders. Each adder introduces 2 gate delays. Thus for the number of input bits greater than 3, the APSQR function is faster than the conventional scheme. Additionally the APSQR architecture is inherently pipelinable.

5 Conclusion

The APSQR function is an appropriate squaring function for the Viterbi algorithm. The APSQR requires less hardware, and at same time, due to low input capacitance, its dynamic power consumption is less than the conventional squaring scheme. In addition, the APSQR function provides an improvement in the speed due to the shorter critical paths. Monte-Carlo simulations have shown negligible degradation in the performance of a Viterbi processor which utilizes the APSQR function.

References

[1] G.D. Forney,"The Viterbi algorithm," *Proc. of the IEEE*, vol. 61, pp. 268-276, March 1973.

[2] G. Ungerboeck,"Trellis-Coded modulation with redundant signal sets; part i: introduction," *IEEE Communications Magazine*, vol. 25, no.2, pp. 5-21, February 1987.

[3] G.D. Forney,"Trellis shaping," *IEEE Trans. on Information Theory*, vol.38, no.2, pp. 281-300, March 1992.

[4] M.W. Marcellin and T.R. Fisher,"Trellis coded quantization of memoryless and gauss-Markov source," *IEEE Trans. on Communication*, vol.38, no.1, pp. 82-93, January 1990.

[5] G. Fettweis and H. Meyr,"High-speed parallel decoding: algorithm and VLSI-architecture," *IEEE Communications Magazine*, pp.46-55, May 199.

[6] Bellman, R.E., and Dreyfus, S.E., "Applied dynamic programming," *Princeton University Press*, 1962.

[7] M. Shammanna, S. Whitaker and J. Canaris,"Cellular logic array for computation of squares," *3rd NASA Symposium on VLSI Design 1991*, pp. 2.4.1-2.4.7.

[8] K. Hwang, Computer Arithmetic: Principle, Architecture and Design, John Wiley and Sons, 1979.

[9] N. Zhuang and H. Wu,"A new design of the CMOS full adder,"*IEEE J. Solid-State Circuits*, vol. 27, no.5, pp. 840-844, May 1992.

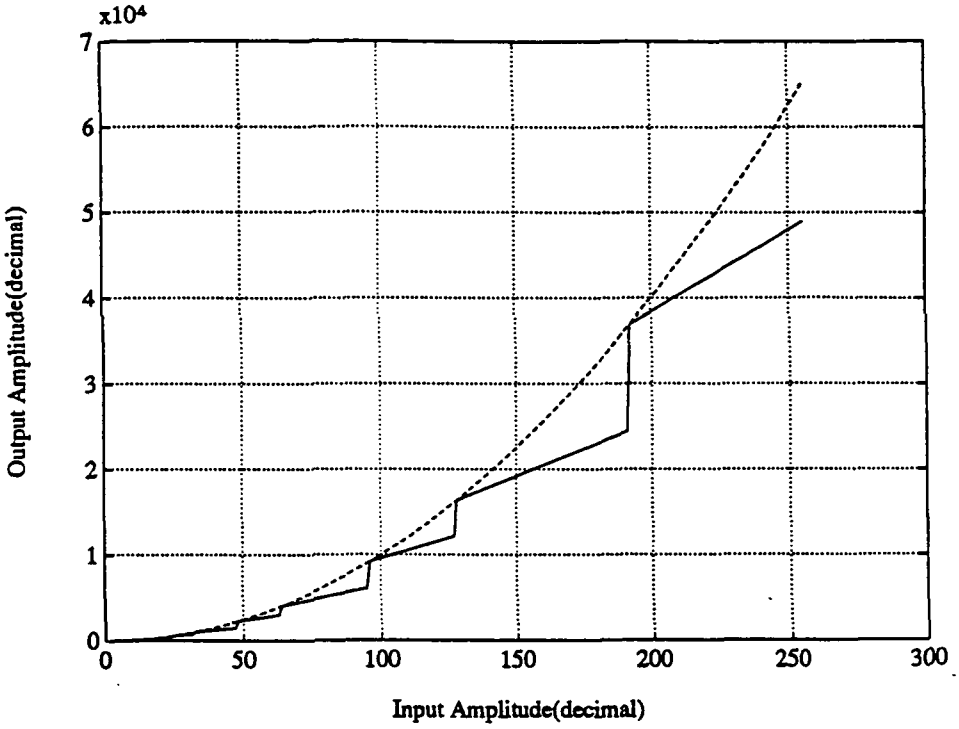


Figure 1: Simulation of approximated squaring function (solid) versus the actual squaring function (dashed).

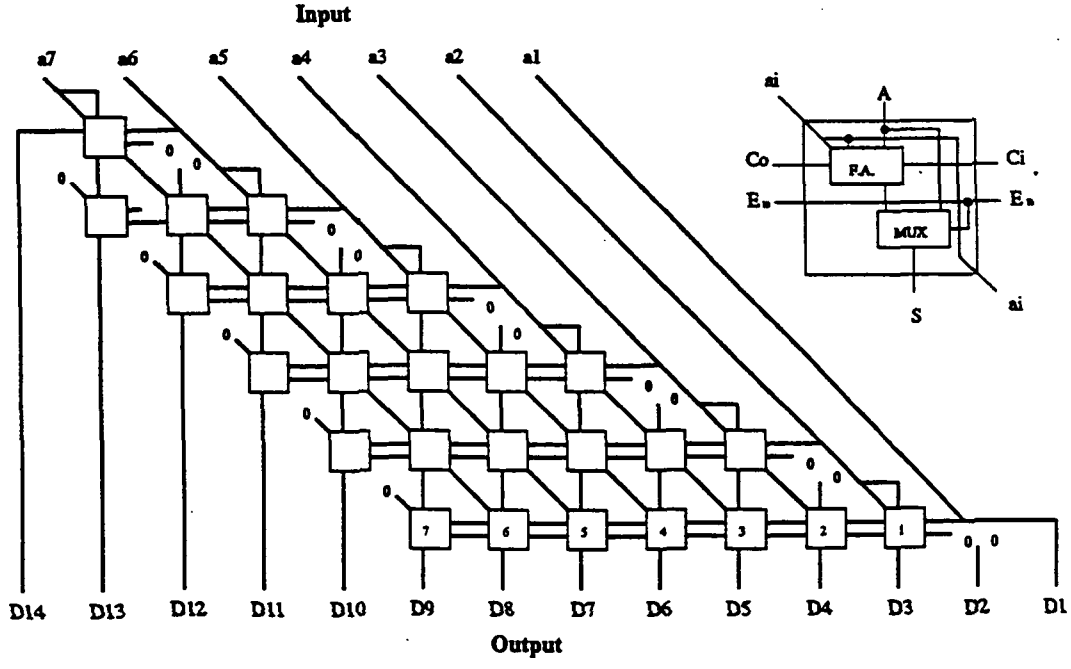


Figure 2: The block diagram of the conventional 7-bit squaring function.

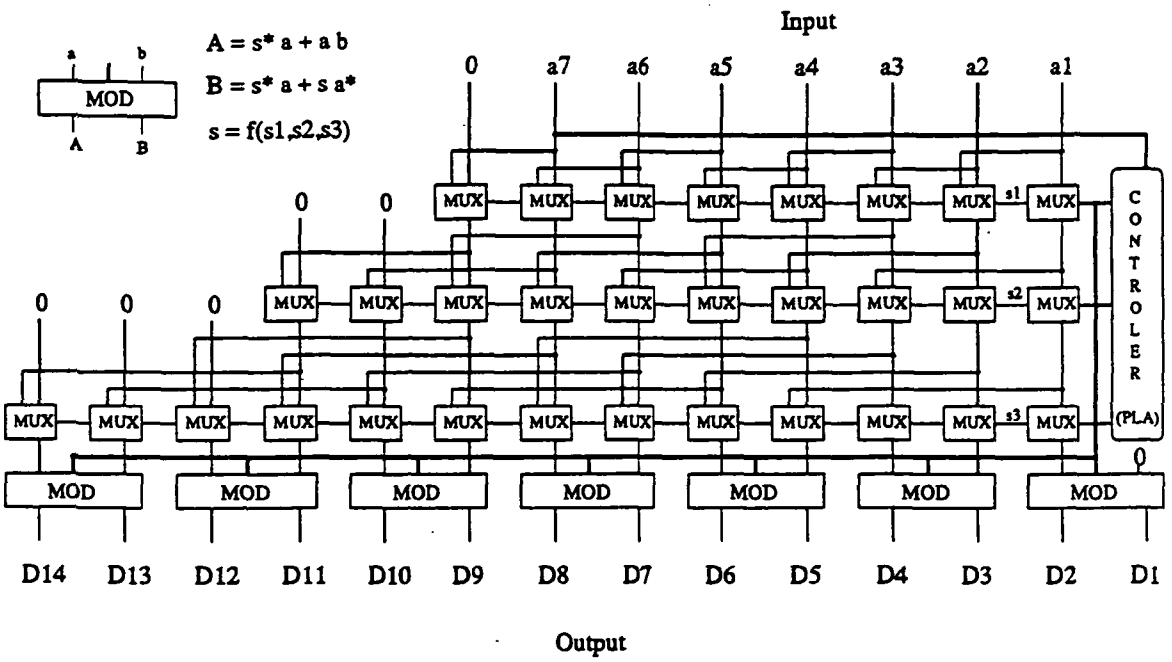


Figure 3: The block diagram of 7-bit approximated squaring function.

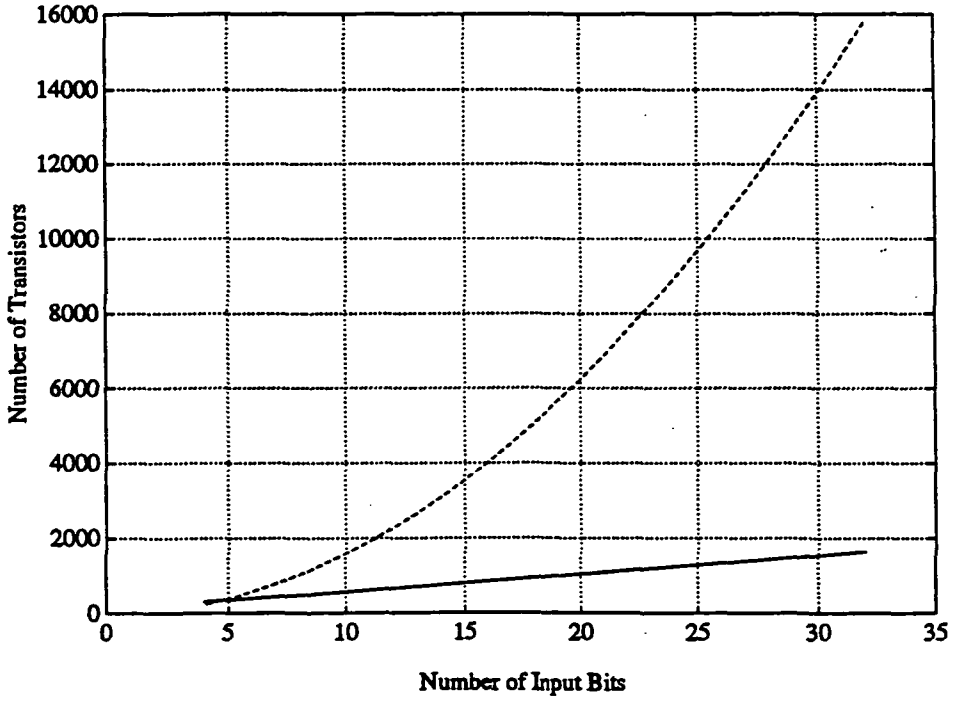


Figure 4: The numbers of transistors used in conventional technique (dashed line) and APSQR (solid line).

A 20MHz CMOS Reorder Buffer for a Superscalar Microprocessor

John Lenell
Standard Microsystems

Steve Wallace and Nader Bagherzadeh
Department of Electrical and Computer Engineering
University of California, Irvine
Irvine CA 92717

Abstract- **Superscalar processors can achieve increased performance by issuing instructions out-of-order from the original sequential instruction stream. Implementing an out-of-order instruction issue policy requires a hardware mechanism to prevent incorrectly executed instructions from updating register values. A reorder buffer can be used to allow a superscalar processor to issue instructions out-of-order, and maintain program correctness. This paper describes the design and implementation of a 20Mhz CMOS reorder buffer for superscalar processors. The reorder buffer is designed to accept and retire two instructions per cycle. A full-custom layout in 1.2 micron has been implemented, measuring 1.1058 mm by 1.3542 mm.**

1 Introduction

A superscalar processor can improve performance by looking ahead into the sequential instruction stream, and executing independent instructions out-of-order. However, issuing instructions out-of-order can cause the processor to execute instructions which should not have been executed. For example, a branch instruction in the instruction stream is predicted by the processor to be taken. The processor begins to execute instructions from the target of the branch before the actual direction of the branch is determined. If the branch is determined to be not taken, then the instructions from the target of the branch have been executed incorrectly. For the program to complete correctly, the results of the instructions which have been executed incorrectly must not write results to the register file. Program correctness can be maintained by only allowing instructions to update the register file in the original program order. A reorder buffer allows a superscalar processor to execute instructions out-of-order by allowing the register file to maintain the in-order state of the processor[?, ?]. The reorder buffer temporarily holds all results computed by instructions after execution regardless of the order of execution, and then updates the register file with the results in the original program order. Results are written to the register file in-order by operating the reorder buffer in FIFO fashion. As entries in the reorder buffer reach the bottom of the FIFO, the completed results are written to the register file[?].

The organization of a reorder buffer in a superscalar processor is shown in Figure 1. Each decoded instruction is allocated an entry at the top of the reorder buffer. During allocation, the instruction's destination register identifier and a unique tag identifier for the instruction

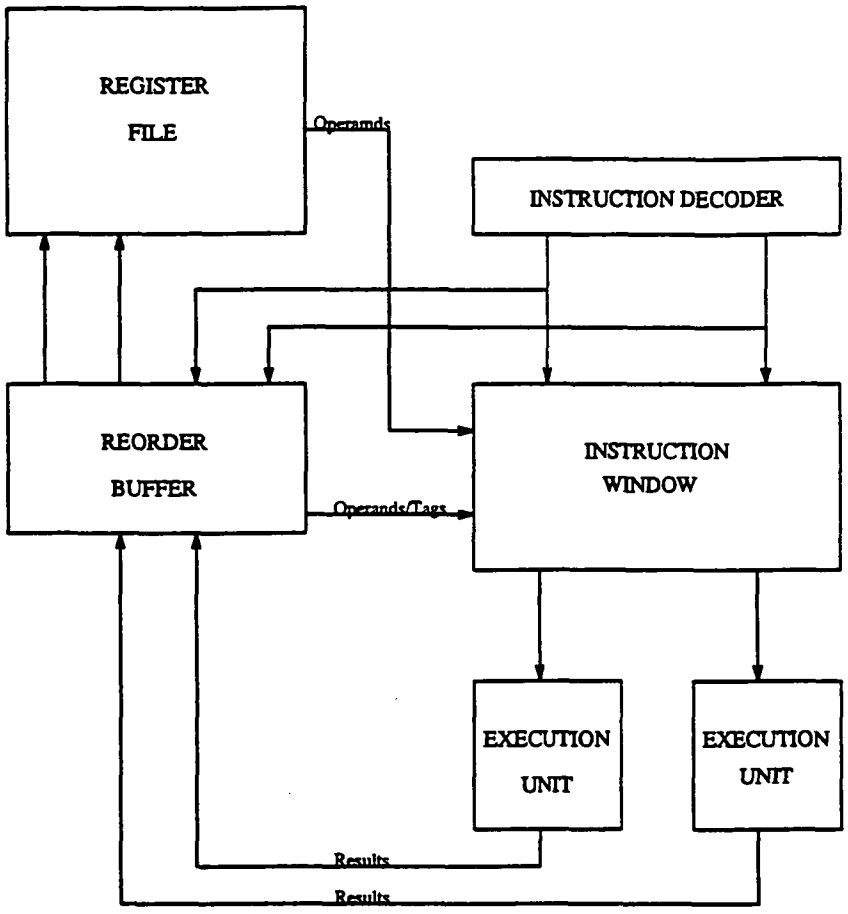


Figure 1: Location of a Reorder Buffer in a Superscalar Processor

the operand value, or neither, indicating that the value must be read from the register file. After the functional unit receives its operands and executes, the result is stored in the result field of the reorder buffer. When a completed instruction reaches the bot

2 Design

The reorder buffer presented here has been designed for a superscalar processor with a two instruction fetch unit, and two execution units which execute concurrently. The reorder buffer allows the instruction decoder to enter two instructions into the buffer, read the tag or data for four source registers specified by the instructions in the decoder, and write the results for two functional units each cycle. In terms of processor pipeline stages, the reorder buffer supports the instruction decode, write back, and register update stages.

2.1 Operation

During instruction decode, the reorder buffer is associatively searched with the source register identifiers of each instruction in the decoder. The source register identifiers are compared with a field of the reorder buffer which holds the destination register identifier of preceding instructions. If a match is found, three control fields of the reorder buffer are used to further

search of the completed instructions tag identifier with the tag field of the buffer. A match enables a write into the data field of the corresponding buffer entry, and sets the Ready bit.

As instructions reach the bottom of the reorder buffer, they are written to the global register file during the register update stage. Register update is controlled by the Ready and Valid bits of the entries at the bottom. If both bits are set, the contents of the data field can be written to the register file to the location specified by the destination register field.

2.2 Configuration

A block diagram for the reorder buffer is shown in Figure 2. The reorder buffer is split into two banks. Each bank contains four reorder buffer entries. Each entry has the following fields:

Destination Register Identifier This field specifies the true destination register of the instruction before register renaming takes place.

Destination Register Tag This field holds the renamed value of the destination register. Each Tag in the reorder buffer is unique so that every decoded instruction can be located by its tag.

Data This field temporarily holds the results of the corresponding instruction which have been computed by the functional units.

Valid This is a single bit field that defines whether the associated reorder buffer entry contains valid information. This field is set when instructions are entered into the reorder buffer, and cleared if the entire reorder buffer needs to be invalidated (ie. mispredicted branch).

Current This is a single bit field which tracks the most recent instruction to update a destination register. As each destination register is entered into the reorder buffer, this bit is set for the entry, and the current bit of any other entries with the same destination register are reset.

Ready This is a single bit field that is set when an instruction entry in the reorder buffer has completed execution by a functional unit. It signifies that the result of the corresponding instruction is ready to write back to the register file.

One entry from the instruction decoder can be written to each bank every cycle. The first instruction in the decoder is always written to the first reorder buffer bank, and the second instruction is written to the second reorder buffer bank. One result can be read from each bank every cycle from the bottom of the reorder buffer. and the entries will be shifted out of the reorder buffer if a shift is requested by the instruction decoder for incoming instructions. FIFO operation of the reorder buffer is achieved by shifting all the fields of each reorder buffer entry down one row. Both of the banks shift together so that when entries reach the bottom, they are paired with the same entry with which they entered the reorder buffer.

Phase	Function	Description
1	Precharge	The match lines of the content addressable memory cells are precharged prior to evaluation.
	Evaluate	A self timed signal turns off precharge mid-phase, and enables evaluation of the cam cells.
2	Read	Valid, Ready, and Tag or Data is read for each source register identifier in the decoder after match lines have evaluated. Data is read from the bottom of the reorder buffer and written to the register file if the Ready bit is set.
3	Precharge	The match lines of the CAM cells are precharged prior to evaluation.
	Shift	The entire reorder buffer shifts in a FIFO fashion. New entries are shifted into the top of the FIFO.
	Evaluate	Evaluation follows precharge.
4	Write	Results are written from the functional units to the reorder buffer for the entries whose tags match the tags of the completed instructions. Ready bits are set for entries which receive results. Current bits are reset for entries which no longer contain the most recent update to a destination register.

Table 1: Functions of Timing Phases

2.3 Timing

A four phase clock is used to trigger the multiple functions that the reorder buffer must perform each cycle. The functions performed during the four phases are given in Table 1. Figure 3 is the timing diagram of the four phases. An additional timing signal is generated by the circuitry to trigger two events in a single phase. This self-timed signal is generated by the precharging match lines of the content addressable memory cells. Precharge begins at the start of phase one and three. When precharge has completed, the signal turns off the precharge drivers and enables evaluation of the CAM cells. This self-timed signal is important since precharging the match lines is completed under a nanosecond. Otherwise, if two more phases are used, the cycle time would increase dramatically.

3 Implementation

The regular structure of the reorder buffer is ideal for a full custom implementation. Custom cells were designed for each field of the reorder buffer described in the previous section. A total of six custom cells were used with minor variations occurring within cells depending on the placement of the cell in the buffer. Additionally, a self timed signal is generated by a custom circuit to enable evaluation after precharge in phases one and three. A universal shift cell was designed to allow data to shift between memory elements.

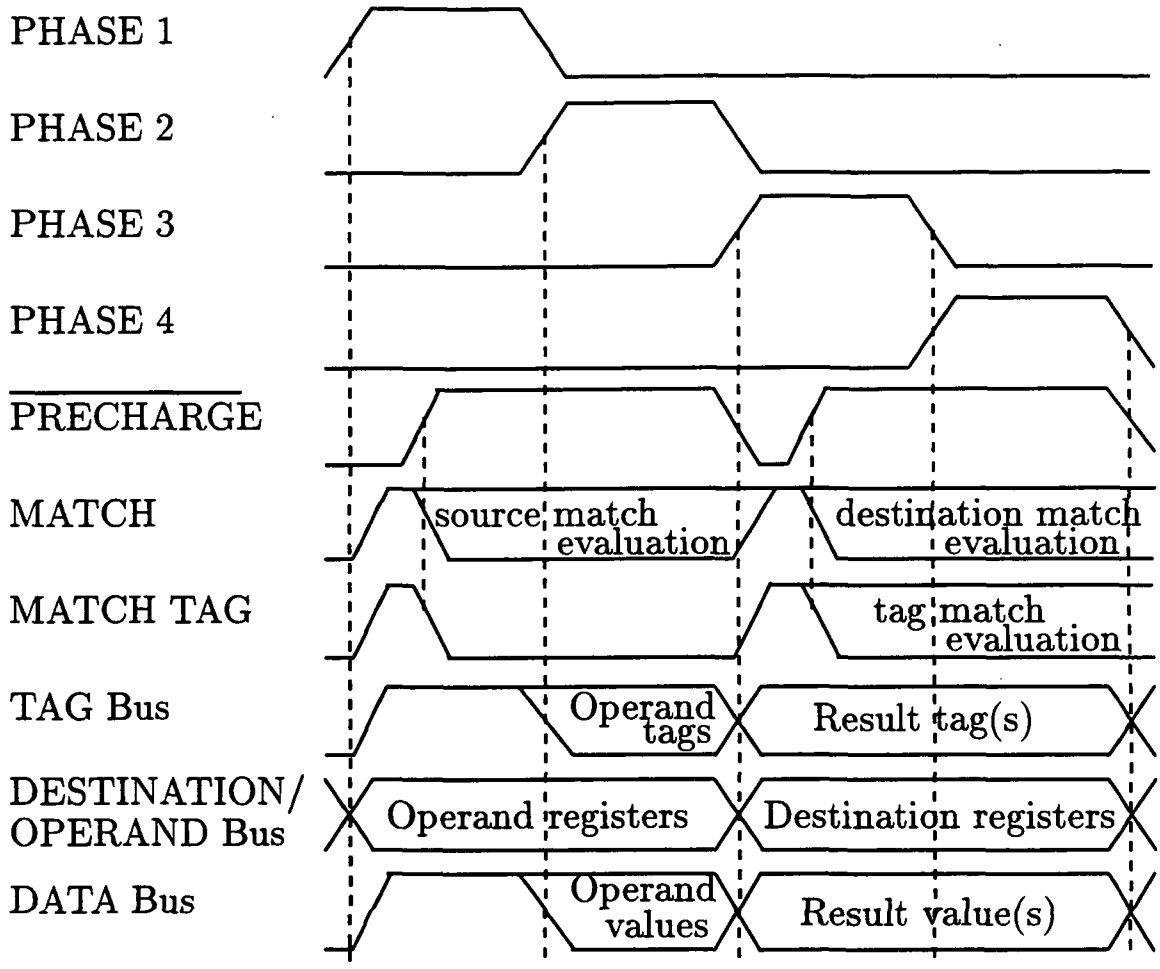


Figure 3: Timing Diagram of Reorder Buffer

3.1 Destination Identifier Cell

The destination identifier field of the reorder buffer is implemented with four-port CAM cells[4]. The four port CAM cell is able to evaluate four match signals simultaneously. The schematic of the circuit are shown in Figure 4(a). Operating the four port CAM cell involves precharging the four match lines and placing the value to be addressed on the bit lines. During precharge, the discharge paths for the match lines are off to prevent evaluation before the bit lines have settled. After precharge, if the value on a bit line does not match the content of the cell, the corresponding match line is discharged. The match lines of the CAM cell in the reorder buffer are precharged and evaluated twice each cycle. During the evaluate phase, the four source register values from the instruction decoder are placed on the bit lines of the CAM cell, and the match lines select the data value to be read from the reorder buffer. Then, in the write phase, the destination register of bo

3.2 Current Cell

The current cell is placed physically adjacent to the four-port CAM cell and shares the four match lines of the CAM cell. The current cell performs three functions each cycle: read, write, and reset. During the read phase, the current cell discharges all of the match lines passing through the cell if the value of the cell is logically zero. This prevents matches of duplicate destination registers to a single source register by allowing only the most recent destination register entry in the reorder buffer to match. Write access is provided to the current cell to provide shifting capabilities between reorder buffer rows. Reset of the current cell occurs during the write phase if match0 or match1 stay high. The schematic of the current cell is shown in Figure 4(b).

3.3 Tag Identifier Cell

The tag identifier field is implemented with a single memory cell with two-port CAM and four port RAM capabilities. Figure 5(a) shows the schematic of the tag cell. During the read phase, the cell allows four port read access on the bit lines. The corresponding word lines are enabled by the destination match lines which pass through the cell. The two-port CAM is implemented in the same way as the four-four port CAM, and provides two match lines for signaling matches between the reorder buffer tag and the tag of instructions completing execution in the functional units. Evaluation of the tag match lines is enabled in the write phase, during which, the tag match lines control the word lines of the data field.

3.4 Ready Cell

The ready cell provides a control signal from the reorder buffer during the read phase for each of the four source registers whose value is being read from the reorder buffer. The signal indicates whether the tag or data value read from the reorder buffer should be used for the corresponding source register. Logically, the ready signal will be the select signal for a multiplexor which chooses between the tag and data value. For this function, the ready signal is designed as a four-port RAM cell. The destination match lines, which pass through

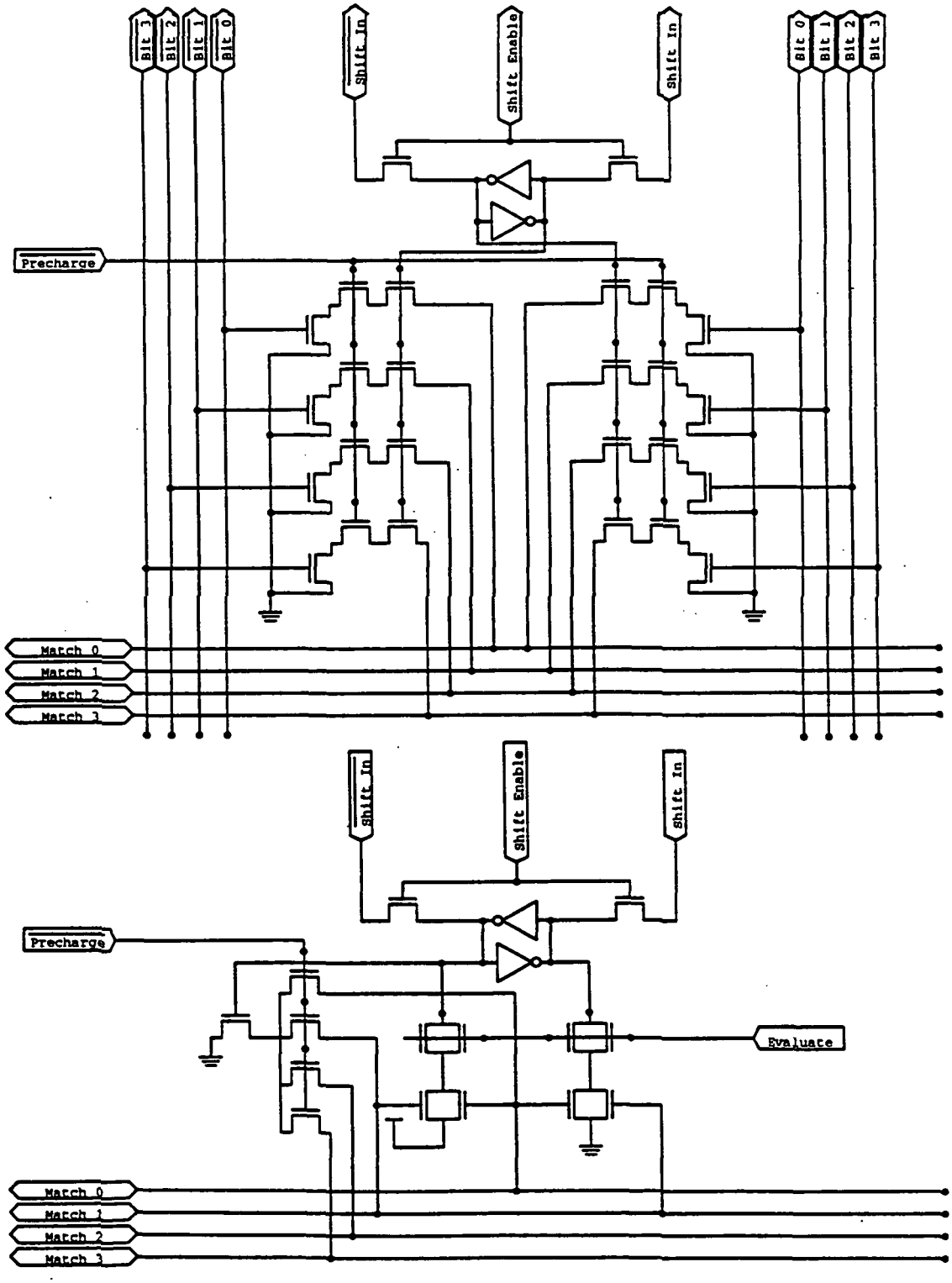


Figure 4: Schematic of a) Destination Iddentifier Cell (top) and b) Current Cell (bottom)

the cell, enable word access to the cell, allowing the value of the cell to be placed on the bit lines.

The tag match lines also pass through the cell as shown in the schematic of the cell in Figure 5(b), and they enable a set function during the write phase. When the tag for a reorder buffer entry matches during the write phase, the data from a functional unit is being written to the entry, and the ready bit is set.

3.5 Valid Cell

The function of the valid cell is to provide reset capabilities to the reorder buffer, and control the use of tags and data read from the reorder buffer for source operands. The cell is implemented as a four-port resetable RAM cell. The word lines of the cell are controlled by the destination match lines which pass through the cell. The bit lines of the valid cell are used to select between the value obtained from the reorder buffer, or from the register file for each source operand.

Reseting the reorder buffer can be achieved by clearing all of the valid cells in the buffer. The reset line of the valid cell asynchronously resets the cell, and indicates that the values in the reorder buffer should not be used. Typical uses of the reset capabilities are for reset of the microprocessor or to recover from mispredicted branches. The valid bit is set as each instruction enters the reorder buffer from the instruction decoder.

3.6 Data Cell

The data cell is a modified four-port RAM cell. The cell has two sets of bit lines for data access controlled by four word line enables. The word lines of the cell are controlled by the destination match lines during the read phase, allowing four values to be read from the data field. Then, during the write phase, the tag match lines are muxed onto the word lines. The tag match lines enable writes to the data field from functional units who have completed execution. Additional read and write control inputs to the cell, enable the path between the cross-coupled inverter pair and the pass transistors to the bit lines. These enables prevent the reading and writing of data during the evaluate phases of the cycle. Read is enabled by phase two, and write is enabled by phase four.

3.7 Shift Operation

The FIFO operation of this reorder buffer is achieved by shifting data between buffer entries. The shift is accomplished with a row of shift cells between each entry of the buffer, and with shift inputs and outputs on each of the cells of the buffer. During a shift phase, the data content of each cell in a column is shifted down one row or entry in the buffer, and the top entry is loaded with new instruction data. The shift cell performs two functions. First, it isolates entries of the buffer during shift with a transmission gate to prevent shift data from a single cell from propagating to other cells in the same column except for the target cell. Secondly, it uses the gate capacitance of inverters to store the data to be shifted, and then drive the shift inputs of the target cell. The transmission gate is enabled during phase 1

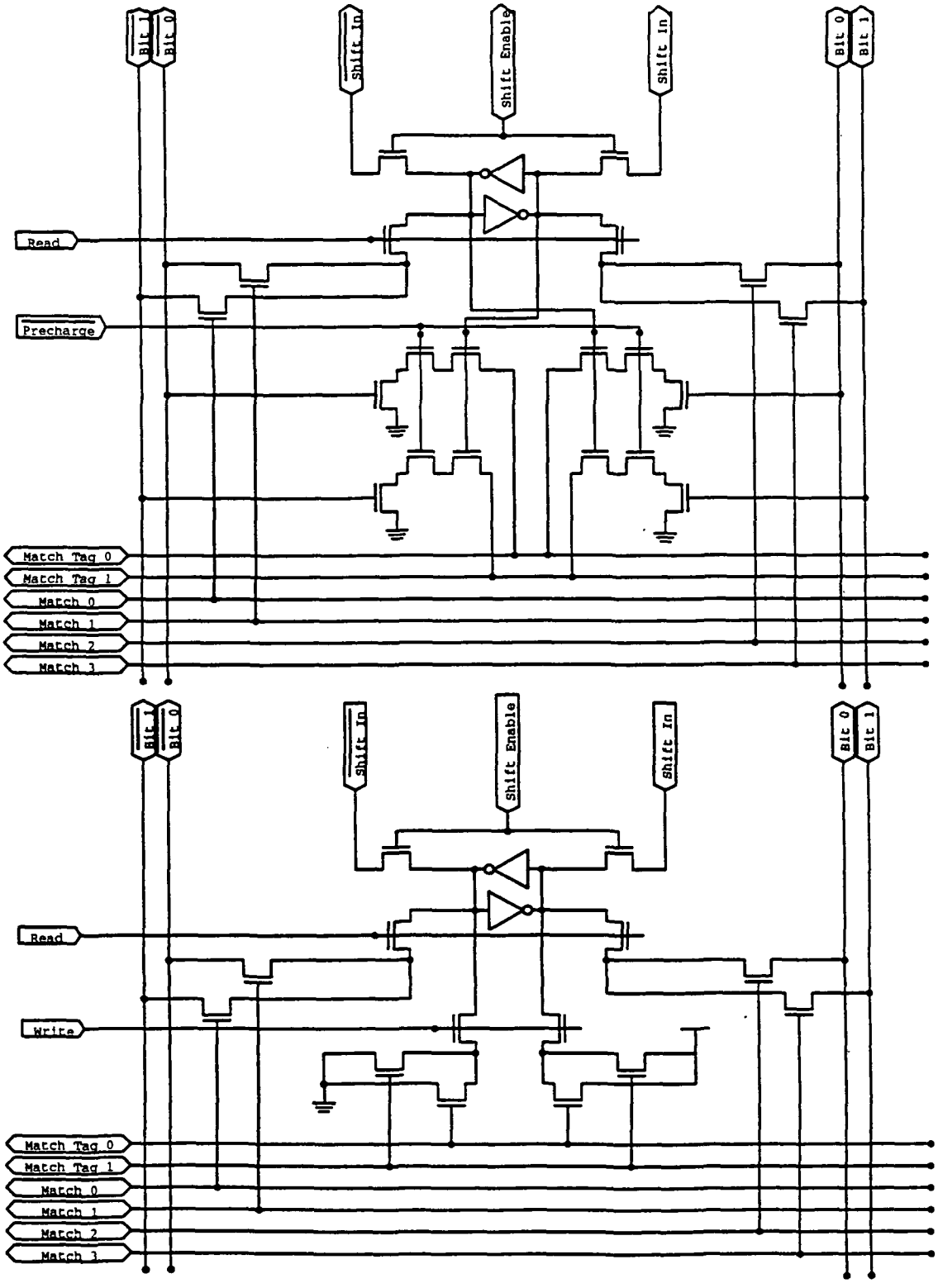


Figure 5: Schematic of a) Tag Cell (top) and b) Ready Cell (bottom)

to charge the shift cell with the data on the input. After phase 1, the transmission gate is disabled, and the data is shifted into the target cell during phase 3.

4 Simulation

A simulator program in C++ was developed to accurately represent a superscalar reorder buffer. It was used to verify the functionality of the extracted layout. The simulator is represented at the register transfer level and updates the contents of each cell at the end of each phase. Every type of cell in the design is defined as a class in C++. The simulator reads instruction information, generates an output file of the state of each cell at the end of phases 2 and 4, and generates a command (script) file. IRSIM uses that command file on the extracted layout to generate output which can then be compared against the simulated output. Optionally, the simulator has an user interactive feature whereby the complete state of the reorder buffer is represented in a user-friendly way.

The simulator reads in two (optionally one) instruction at a time from a .asm file and cycles through four phases until there are no more instructions left to execute. The .asm file input provides two source registers, a destination register, the latency involved in the operation, and the expected output for this virtual instruction. Note that no real operations are performed, the simulator behaves as if these values were passed to it from the instruction window.

As the simulator executes through each phase of a cycle, it generates appropriate commands that IRSIM should execute into a .cmd file. For instance, before phase 2 is executed, the source registers must be placed on the Destination Register Identifier bus for match comparison. This is done by the "set" vector command. At the end of phase 2, IRSIM is told to display the Valid, Ready, Tag, and Data values read from the reorder buffer, while at the end of phase 4, IRSIM is told to display the contents of the reorder buffer. Since the simulator knows the theoretical contents of the reorder buffer at that point, it outputs the expected value into a .out file, in a format identical to the output of IRSIM. After both the simulation and the layout simulation (IRSIM) have been executed, another program "cleans" the output of IRSIM, compares both outputs, and indicates any differences in a separate file.

4.1 Results

Simulation could be performed at 10 ns for each phase (25 MHz). During phase 1 and 3, precharging of the match lines took approximately 1 ns. The shifting during phase 3 took up to 3 ns to complete. Thus the self-timed circuit must be wary of this and not should not start before either condition is completed to avoid corruption of data or premature discharge of the match lines. During phase 1 evaluation of the match lines, if all the bits of the source operands compared the Destination Identifier Cell match except one, then the corresponding match line will have to discharge through a single transistor. This worst case took approximately 7 ns. Reading the reorder buffer during phase 2 lasted 3 ns for data bus,

6 ns for ready and valid bits, and up to 7.5 ns for the tag bus. The tag takes longer to read because it has additional internal load on it. During phase 3 evaluation of the match tag lines, if all the bits of the result tag compared to the Tag Cell match except one, then the corresponding ma

The limiting phase is phase 2. Since no external loads were used during simulation, the actual read time will take longer than stated. Since the tag bus has internal load, with additional external load, the worst case discharge time will take longer than 7.5 ns and probably longer than 10 ns. In addition, since no dead time was used, this will further increase the effective cycle time. In reality, taking in consideration for self-timed precautions, external load, and dead time, we can expect our 1.2 micron implementation would operate at 20 MHz.

5 Conclusion

The reorder buffer is an important part of an advanced computer architecture design that relies on run-time analysis of the concurrency at the instruction-level. This paper described the design and VLSI implementation of a reorder buffer. The results of this paper could be used to estimate and evaluate the design of superscalar microprocessors.

References

[1] Mike Johnson. *Superscalar Microprocessor Design.* Prentice Hall, Englewood Cliffs, 1991.

[2] J. Lenell. Superscalar and VLIW processors: A performance analysis. Master's thesis, University of California, Irvine, 1992.

[3] James Smith and Andrew Pleszkun. Implementation of precise interrupts in pipelined processors. In *Proceedings of the 12th Annual International Symposium on Computer Architecture*, pages 36–44, June 1985.

[4] Neil Weste and Kamran Eshraghian. *Principles of CMOS VLSI Design: A Systems Perspective.* Addison Wesley, 1985.

Analog/Digital pH Meter System I.C.

Paul Vincent and Jea Park
Department of Electrical Engineering
California State University
Fullerton, California 92634

Abstract - **The project utilizes design automation software tools to design, simulate and fabricate a pH meter I.C. system including a successive approximation type seven-bit analog to digital converter circuits using a $1.25\mu m$ N-Well CMOS MOSIS process. The input voltage ranges from 0.5V to 1.0V derived from a special type pH sensor and the output is a three-digit decimal number display of pH with one decimal point.**

1 Introduction

The recent availability of automated integrated circuit design software tools and the dramatic increases in the hardware tool performances make it feasible to design a VLSI circuits and systems on a personal computer. Also the unavailability of a commercial, small, inexpensive digital and single chip-electronic pH measuring system prompted authors to make use of these tools for creating a pencil-size pH measuring system. Since Tanner tools provided only digital cell libraries and no extensive analog cell libraries at present, additional libraries cells were created. Also other integrated circuit design tools were added to complete the system design on a personal computer.

2 System Operation and Features

A sensor which outputs a linear voltage proportional to pH value drives an analog input to the device. When a user depresses the "sample" momentary switch (see Figure 3), then the system displays a pH value between 1.1 and 13.8. The device is powered by a 3 volt battery. An internal timing is used to start and stop the system clock to conserve battery life. The device is intended to be used with three seven-segment displays. A seven-bit conversion was chosen, because the number of increments of tenths (131 step3). Since solutions with pHs at the extreme ends of the range are fairly exotic, the range of displays was set at between 1.1 and 13.8. These values are attained by adding a value of 11 to a seven-bit digital representation of the linear input analog voltage.

2.1 Successive Approximation

This method was chosen for it's simplicity of implementation (Figure 4). The seven-bit register holds an approximation of the input analog voltage. The approximation takes place in

seven clock cycles, one for each bit.

A seven-bit shift register, one bit for each of the approximation bits, controls the approximation. Approximation takes place one bit at a time, beginning with the most significant bit (msb). All of the control register bits are set to zero, except the bit that is currently being sampled. In effect a "1" is shifted through a field of zeros in the control register. The "1" enables loading of a mux-flipflop for the bit of the approximation register being sampled. The sampled bit of the register is set to a "1", and the output of the register is fed to an analog comparator via a D/A converter. If the approximation value is less than the input value, then that bit of the register is left at a "1." If making the sample bit a "1" causes the approximation to exceed the value of the input voltage, then that bit is set to a "0." Sampling begins at the msb and proceeds "SUCCESSIVELY" to the least significant bit (lsb).

2.2 D/A Conversion

A straight voltage divider resistors requires 2^n resistors for n-bits. A capacitor-based (charge scaling) approach requires capacitors which increase in area exponentially, both of the above methods were not chosen because of their large chip size [1].

A voltage-scaling R-2-R ladder method (Figure 5) was selected [2] because of it's simplicity and good area utilization. With the R-2-R ladder, 2n resistors are required for an n-bit conversion. The output voltage accuracy depends on the accuracy of the resistances and the ratio of the "2R" resistor values to the "R" resistor values. The ratio can be improved in layout by orientating the "R" and "2R" resistors in the same direction so that processing variations will track between them, and the effects on their resistance ratio can be minimized.

N-Well (R=2, 4KOhm/Square) is used for the resistors. Resistance Values of 30K ohms and 60K ohms are used.

2.3 Reference Voltages

The output of the R-2-R ladder is from 0.5 to 1 volt. For this work, the bottom of the ladder must be referenced at 0.5 volts. The input to the ladder must be referenced to 0.5 volts for a logic "0" and 1.0 volts for a logic "1" (Figure 5). Diodes are used here as voltage references with a current limiting MOS transistor controlled by an external voltage for this device. This approach is used, because of the single 3 volt power source and the closeness of the reference to the threshold voltage for this technology.

2.4 Analog Comparator

A two-stage comparator (Figure 6) comprising a differential stage and an inverting stage was selected. The poor gain of the differential stage is augmented by the inverting stage and problems controlling the trip point of the "current-sink" inverter stage are reduced by the differential stage [3].

2.5 Digital Decoding and Display Logic

As mentioned previously, the decoding was somewhat simplified over a pure 7-bit-binary-to-bcd decoder, by adding a decimal value of 11 to the output of the approximation register, and then converting that binary value to bcd. The bcd conversion method is that of a standard 74HC185 [4]. The BCD-to-7-segment decoder/drivers are derived from equation generated using Altera's Maxplus EPLD synthesis tool driven by a tabular ASCII input. Tabular input implementations of the binary-to-BCD decoding were also made, but proved to be much less area-efficient than the 74HC185 method. Self test is centered around signature generation and the output of a "go-no-go" indication. Self testing is initiated by setting an external test-node signal active. In the self test mode (Figure 7), the analog circuit feedback to the approximation register is tapped out and driven off-chip. A digital comparator is connected in its place, with the difference that the test now becomes greater than or equal to, instead of greater than. An on-chip counter, driven by the clock generator circuit provides exhaustive inputs to the approximation register. The decoded display outputs are compressed into a signature register, and a "go-no-go" indicator is set based on hard-wired comparison to the known good signature.

3 Simulation and Design Tools

This design required an IBM PC with at least 4 Mb of RAM. The design files all fit onto a 1.44 Mb floppy diskette. OrCAD was used for schematic entry. An OrCAD library was replaced by that of Tanner Research. This library includes ASCII "macro" files describing ports on the physical cell designs for the MOSIS library parts in order to tie the library into the place-and-route tool. Also it has a collection of macros for the logic simulator, GateSim.

OrCAD's annotation utility is used to flatten the design, annotate the cell references (instance numbers) and generate a flattened OrCAD wirelist. Two passes are made with the annotator. The first pass, with a switch on, causes edits to the schematic files, changing all the "reference" designators. Values for multiple instances of a work sheet are accounted for in the renumbering of references. The second pass, with difference switches on, flatten the design, elaborates references for the multiply instantiated sheets and generates a flattened wirelist. Tanner's netlist tool, NetTran, is run next, again in a two-pass process.

The first pass translates the wirelist to a silos format which includes the entire cell library of MOSIS cell macros. The second pass prunes the unused cell macros form the netlist.

3.1 Logic Simulation

Simulation was based on cell/primitive. The top-level of the design contained 407 schematic components (about 1250 two-input gate equivalents). The top-level functional simulation of all 128 input steps ran in 1 minute to 2 minutes on a 486/PC clone with "limitless" memory. The simulation comprised a "test-bench", where the digital portion of the chip was instantiated, which was connected to a counter and a comparator. The digital successive approximation register outputs were run into the digital comparator, whose output was fed back to the digital portion of the chip. Therefore, the test patterns for the chip consisted of a reset pulse for the counter and clear pulse for the chip.

3.2 Static Timing Verification

GateSim includes a static timing verification routine where a delay window must be specified for the verifier reports on all paths within the window. No delays above 200 ns were found. Since the target speed is more a function of the very slow R2R ladder, whose response time is in ms, timing in the digital logic did not give a problem.

3.3 Analog Simulation

Hand designs were done for the analog circuits and manually entered as netlist for P-SPICE. Based on these approximations, analog cells were laid out. Parametric information from the layout is then extracted using L-Edit. P-SPICE simulations were based on parametric from the actual layout.

3.4 Mixed Signal Simulation

A P-SPICE netlist is generated from the combined analog and digital schematics. P-SPICE digital primitives are used for the digital cells. Extracted parametric from analog cells are used for the analog devices.

3.5 Auto Place and Route

The Place-and-Route tool run off on a Tanner "tpr" netlist, translated from OrCAD wirelist. The Place-and-Route is a two step operation. In the first step, the interior core of the chip is routed using the standard cells from the cell library. In the second step, the pads are routed. The tool provides annealing algorithm also in order to optimize the chip area. The core placement and routing took about 2 hours an a 486/PC clone with 32 Mb of RAM.

3.6 Design Rule Check

DRC is built into the L-Edit. The MOSIS library from which the chip is built from contains the design rule for the process. This tool is run because the analog and wiring portions are added.

3.7 Layout vs Schematic Capture

As a verification of layout, LVS was used and compared SPICE files.

3.8 Fault Grading

GateSim contains a fault-grading utility. Fault detection is based on user-provided non-faulted simulation results.

4 Summary

The I.C. comprises a seven-bit successive approximation register, seven-bit digital-to-analog converter (R2R) ladder, an analog comparator and digital logic for display decoding and driving. The successive approximation register is clocked by an on-chip oscillator at a reasonably slow speed.

The chip is intended to be operated at 3 Vdc. The Ph reading is made by depressing a momentary "sample" button. This causes a reset pulse to the chip, which in turn will start the on-chip clock oscillator until the next sampling, the chip will shut off its clock oscillator until the next sampling is initiated by the user.

The I.C. design methodology includes the use of schematic capture, logic simulation, auto-placement-and-routing and layout-vs-schematic verification. Both the high-level and low-level simulations were performed. The fault grading and built-in self-test circuitry are included. Tanner Research's GateSim is used for the logic simulation and OrCAD is used for schematic capture. Tanner netlister and OrCAD library link the two different tools. The additional manual creations of the insufficient device library parts to the Tanner's cell library was added. The P-SPICE program was also merged with other software tools. The layout verification was performed using Tanners LVS tool.

References

[1] Phillip Allen and Douglas Holberg, "CMOS Analog Circuit Design," Holt, Reinhart and Winston, Inc.

[2] Private communication.

[3] Phillip Allen and Douglas Holberg, "CMOS Analog Circuit Design," Holt, Reinhart and Winston, Inc., "Two-Stage Comparator," pp 333-349, 1987.

[4] HCMOS data book.

- Synopsis/general description of design and methodology
- Circuit description, block diagram, hierarchy partitioning
- Simulation
- Custom design
- Layout, parasitic extraction, back annotation

Figure 1: Outline of Presentation

- 1.25 um N-Well Cmos
- 7-bit Successive Approximation Register
- D/A Converter
- Analog Comparator
- Input Voltage from Sensor (0.5v to 1v)
- 3V Battery Powered
- Digital Decoding - Display of pH from 1.1 to 13.8
- Built-in Self Test

Figure 2: Design Features

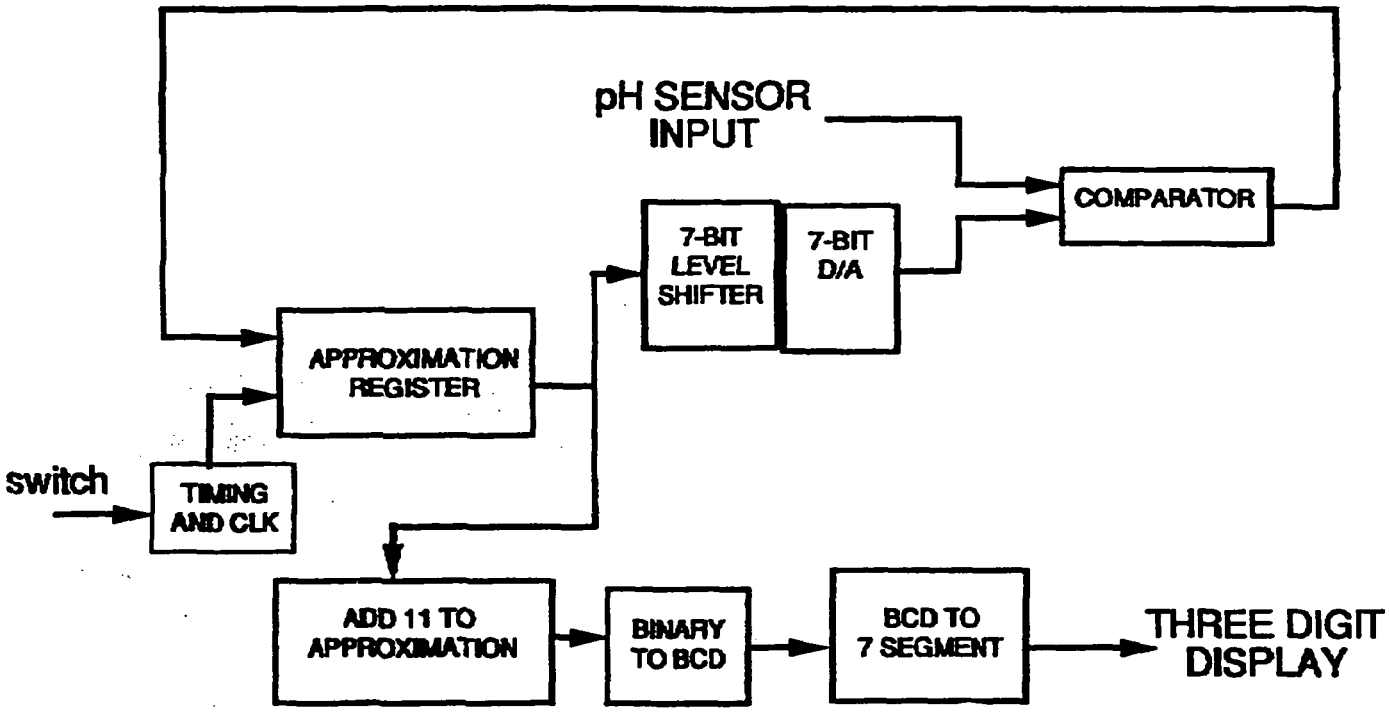


Figure 3: Block Diagram

```
approximation := 0;
approximation_temp := 0;

for n in 6 downto 0 loop
 approximation_temp := (approximation +2^n);
 if          (approximation_temp < analog_voltage)
             then approximation := approximation_temp;
end loop;
```

Figure 4: Successive Approximation

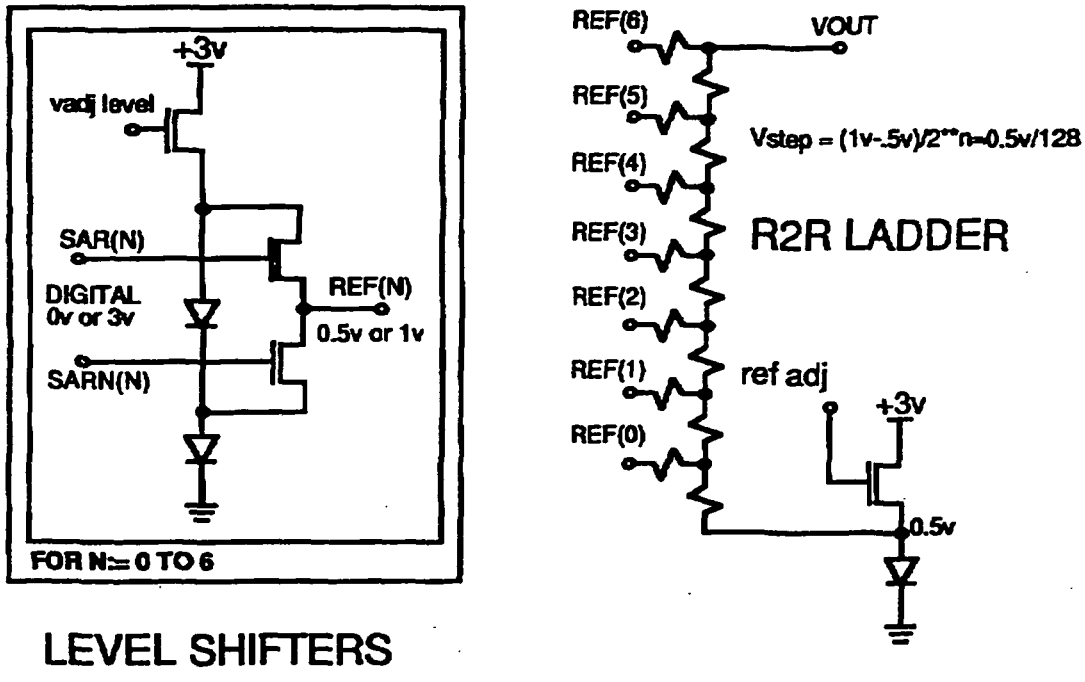


Figure 5: A/D Converter

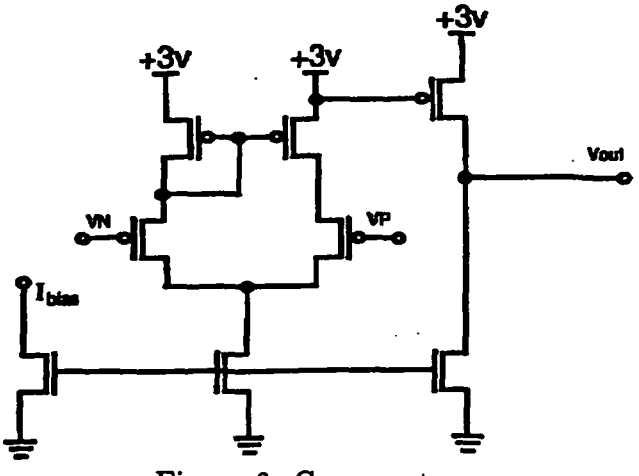


Figure 6: Comparator

schematic capture → logic simulation / analog simulation → place and route → layout vs schematic → mask data generation

Figure 7: CAE Flow

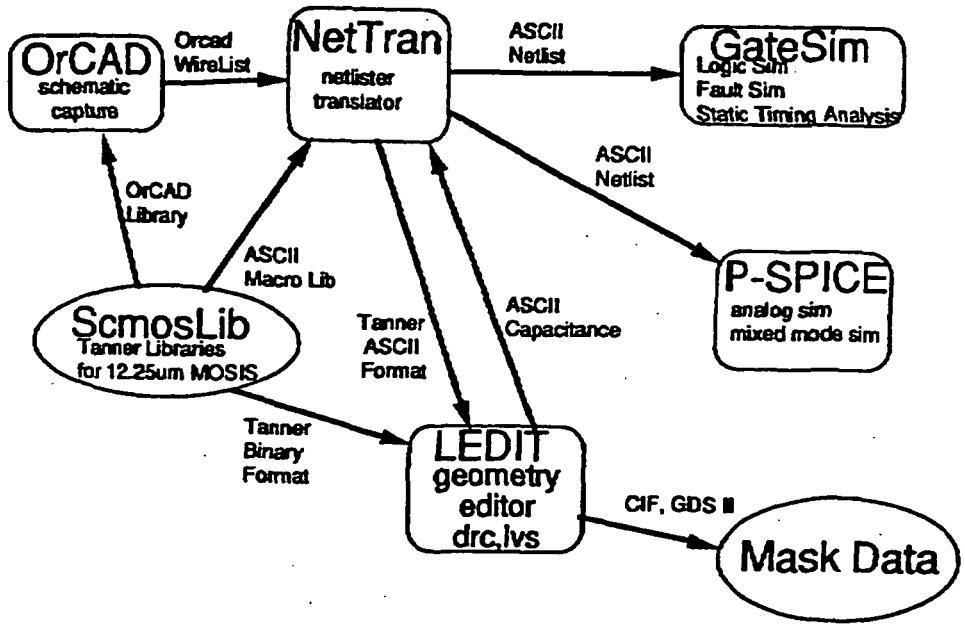


Figure 8: CAE Data Generation

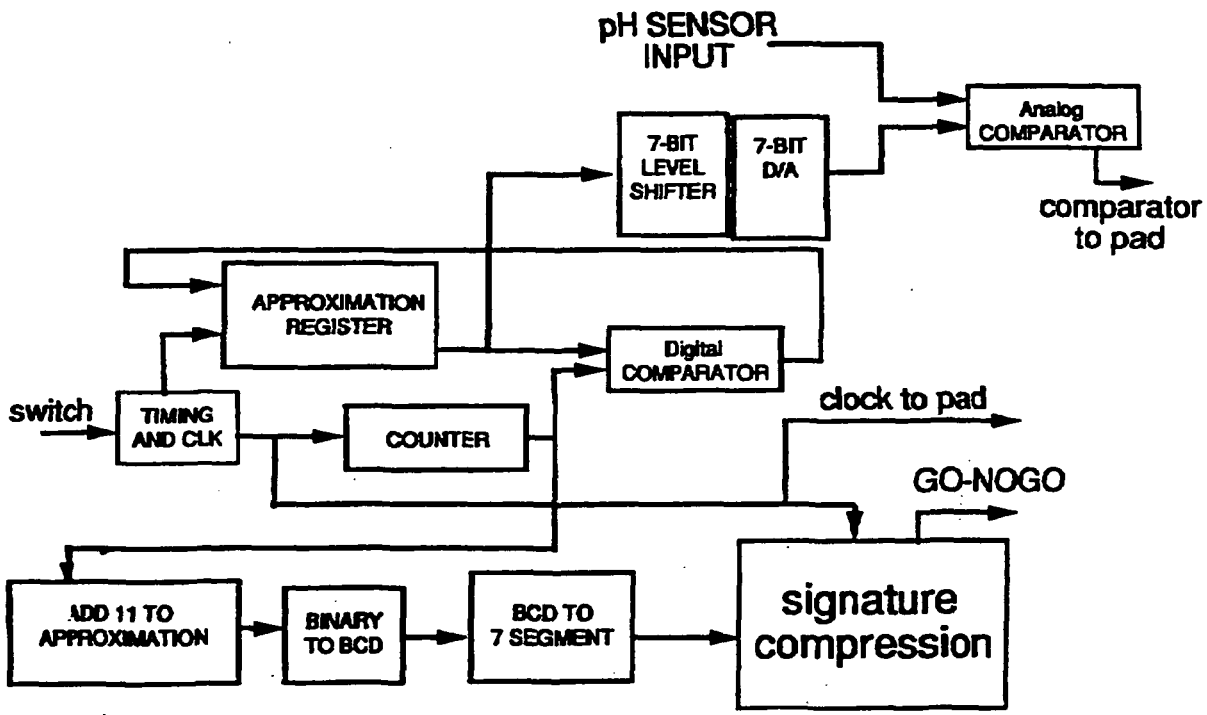


Figure 9: Built-In Self Test

- Analog symbols hand-made for OrCAD (LIBEDIT)
- "macro" definition (Tanner ascii format) for NetTran
- L-Edit cell created to match i/o port definition
- Capacitance information added to macro definition

Figure 10: Gluing the Analog Parts into NetTran/L-Edit

- OrCAD annotation two passes
  - 1st pass updates schematic references
  - 2nd pass generates annotation file
- OrCAD cleanup graphical rule checking
- OrCAD erc - electrical rule checking
- OrCAD Annotator (run twice ... explain details) – question - is this step superfluous?
- Tanner NetTran (run twice ... 2nd time to prune macros)

Figure 11: Net List Generation

An 18 Bit 50 kHz ADC for Low Earth Orbit

Donald C. Thelen
NASA Space Engineering Research Center for VLSI System Design
University of Idaho
Moscow, Idaho 83843

Abstract - **A fourth order incremental analog to digital converter (ADC) is proposed which performs 18 bit conversions at a 50 kHz rate on sampled and held data. A new self calibration scheme is presented which eases the matching requirements of capacitors, and the performance of the operational amplifiers in the ADC by changing coefficients in the digital postprocessing.**

1 Introduction

Dynamic range in charge coupled device (CCD) image sensors is measured by the maximum number of electrons in a well divided by the minimum number of usable electrons in the well. This dynamic range can be as high as 18 bits for today's CCD detectors, where each increment corresponds to two electrons. The analog outputs of the CCD detector must be converted to digital words before they can be manipulated by a microprocessor, or stored in memory. With a readout rate of 50,000 pixels per second, CCD sensors are pushing the limits of analog to digital converter (ADC) performance.

Using voltage division techniques such as R-2R ladders or charge redistribution capacitor arrays to build an 18 bit ADC would require component matching on the order of 4ppm to get good linearity. The best matching achievable on a silicon chip without expensive trimming is about 1000ppm. Sophisticated techniques are clearly needed to fabricate an 18 bit ADC on a silicon chip, without increasing the complexity of the semiconductor processing.

Satellite applications impose the additional requirement that an ADC perform properly when exposed to the radiation in space. Proton radiation found in low earth orbits may cause an ADC chip to latch up or loose data due to single event upsets. Radiation can also generate noise [1, 2] in analog circuits which must be filtered out, or compensated for. This paper proposes an architecture for a 18 bit, 50kHz (ADC) for low earth orbit.

2 Architecture

Self calibrating, oversampling, and integrating techniques are effective ways to overcome basic process limitations to get high resolution. These techniques are reviewed here to determine which one will most likely meet the speed requirements, and reject noise caused by proton radiation.

Charge redistribution successive approximation ADC's can achieve 18 bit resolution at 50kHz when self calibration techniques are used to overcome the matching problems of on chip capacitors [3, 4]. The combination of resolution and speed is achieved by performing one comparison per bit (for an 18 bit conversion, the comparator must settle 18 times). While this leads to good conversion speed, there is no inherent rejection of noise in the ADC. A

noise spike caused by the arrival of an energetic proton could lead to an erroneous value for one of the more significant bits, which would lead to a large error in the final answer. The remaining conversion cycles can not change a bit which has been corrupted by noise. Algorithmic [5], and pipelined [6] ADC's are also one comparison per bit architectures, and will be plagued by the same noise rejection problem. A redundant ADC [7] could overcome this problem, but the self calibration routine becomes more complex. The redundancy adds to the conversion time, while decreasing the importance of each individual comparison in the final answer.

Integrating ADC's such as dual slope ADC's exhibit excellent linearity and rejection of noise added to the input signal, and can have very high resolution [8, 9]. Noise on the input is integrated along with the desired signal, so only the average value of the noise, which is usually zero, corrupts the accuracy. The main disadvantage of integrating ADC's is the notoriously slow conversion rate. One analog to digital conversion consumes 2^n comparison cycles, so to perform an 18 bit analog to digital conversion at a 50kHz rate would require a 13GHz comparator clock rate. This fast comparator would be required to compare signals whose difference is only about $20\mu V$. This is clearly beyond the capability of inexpensive integrated circuit technologies.

The most popular oversampled data conversion technique in use today is the delta sigma ADC [10]. Delta sigma ADC's convert an analog input into a pulse density modulated signal at a frequency well above the Nyquist rate, then smooth the modulated signal with a digital low pass filter. The pulse density modulated signal is viewed as the original signal plus quantization noise. The digital filter gets rid of the quantization noise, and any other noise on the input which is outside the bandwidth of the ADC. Delta sigma ADC's work well on continuous signals such as voice or music, but are not intended for multiplexed sampled and held data [11]. A charge coupled device (CCD) image detector is one example where an ADC is multiplexed between a large number of pixels.

The incremental ADC [11, 12] is also an oversampled ADC, but it is intended for sampled and held data, which makes it better suited for multiplexed inputs. Like the delta sigma ADC, the incremental ADC also uses a large number of input samples to get high resolution, but the signal processing is done on a sample by sample basis instead of on a continuous data stream. Noise on the input is averaged, and noise which causes a bad comparison can be corrected by later cycles. A fourth order incremental ADC with an internal sample rate of 4.25 MHz, and an external data rate of 50 kHz was chosen for its high accuracy, moderate speed, and rejection of noise.

3 First Order Incremental ADC

The incremental ADC is really a switched capacitor version of the charge balance ADC [13, 14]. The operation of a first order and second order incremental ADC are described in detail in [11, 12]. The operation of the first order incremental ADC, shown in Figure 1 is summarized here. The timing of the switch control signals, shown in Figure 2, is slightly different than that used in [11, 12] to help speed up the conversion rate. The ADC starts in the reset mode, where the integrating capacitor $C2$ is discharged, and the digital counter on the output of the comparator is set to zero. The phase 1 switches are then closed,

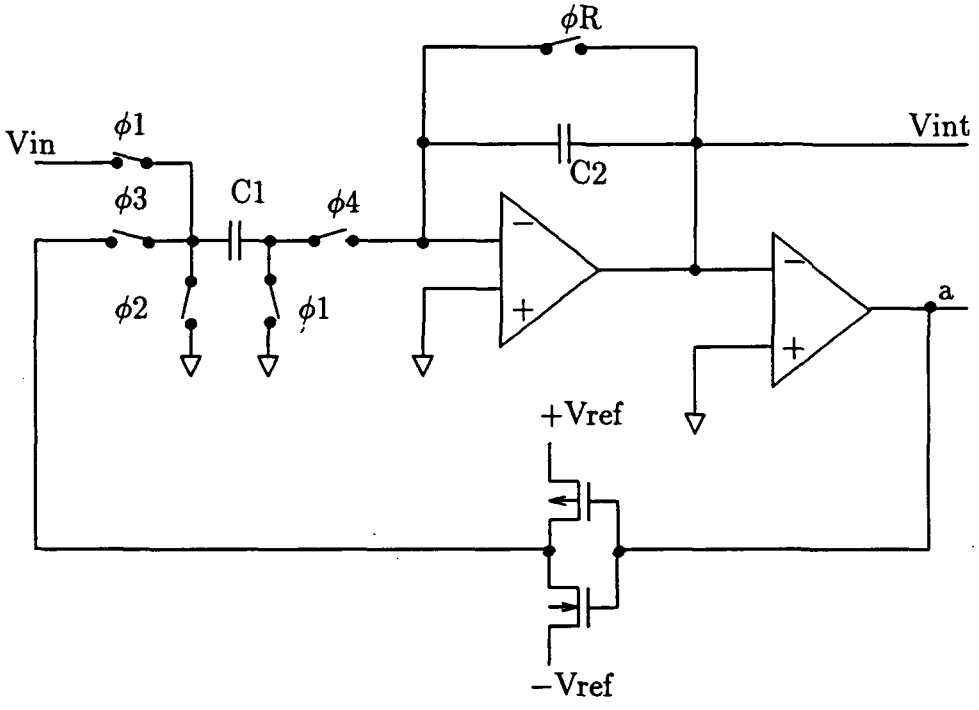


Figure 1: First Order Modulator

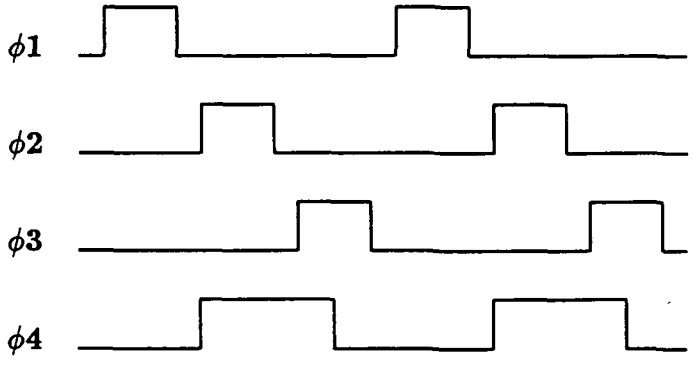


Figure 2: Switch Control Signal Timing

and V_{in} is sampled on $C1$. The phase 1 switches open, then the phase 2 and phase 4 switches close, transferring the charge $-V_{in}C1$ to the $C2$. The output of the integrator is now $V_{int} = V_{in}C1/C2$. Next the phase 2 switch opens, and the comparator is strobed. If $V_{in} > 0$, then the output of the comparator is a logic 0, and $+V_{ref}$ is applied to the input of the phase 3 switch. The phase 3 switch then closes (the phase 4 switch is also still closed) which transfers either $+V_{ref}C1$ or $-V_{ref}C1$ on $C2$, depending on the output of the comparator. If $V_{in} > 0$, the new integrator output is $V_{int} = (V_{in} - V_{ref})C1/C2$, while if $V_{in} < 0$, $V_{int} = (V_{in} + V_{ref})C1/C2$. A gated counter increments if the output of the comparator is a zero. This cycle repeats n times with the same input voltage, until the output of the integrator is given by equation 1 [12].

$$V_{int} = \alpha(nV_{in} - V_{ref}\sum_{i=1}^{n} a_i) \tag{1}$$

where α is the integrator gain ($C1/C2$), and a_i are the individual comparator outputs which have the values $+1$, or -1. Solving for the ratio of the input voltage to the reference voltage yields:

$$\frac{V_{in}}{V_{ref}} = \frac{V_{int}}{n\alpha V_{ref}} + \frac{1}{n}\sum_{i=1}^{n} a_i \tag{2}$$

We can interpret equation 2 to be the average of the comparator outputs, plus a residue. Setting $\alpha = \frac{1}{2}$ allows $V_{in}max = V_{ref} = V_{sw}$ where V_{sw} is the maximum signal swing allowed by the opamp. With $\alpha = \frac{1}{2}$, the maximum possible value of V_{int} is $V_{ref}/2$, so the maximum residue is $1/n$. Therefore, to get n bits of resolution, 2^n integration/comparison cycles are required, which means this technique is as slow as an integrating ADC.

Noise added to the input will be integrated along with the signal, so like the integrating ADC, only the average value of the noise will affect the ADC output. Unlike the integrating ADC, noise at the input of the comparator will not significantly affect the incremental ADC accuracy. If a noise spike causes the comparator in an incremental ADC to output an erroneous value, both the integrator (through the feedback path) and the digital counter will record this bad value. The output of the integrator is now very large, so on the next cycle, the comparator will output a value which compensates for the previous error.

4 Fourth Order Incremental ADC

A higher order integration of the input signal and feedback data can be achieved by cascading the first order modulators as shown in Figure 3 [11].

After n cycles, the voltage on the output of the fourth integrator is:

$$\begin{aligned} V_{int4} = \alpha_4(\alpha_3(\alpha_2(\alpha_1(&\frac{V_{in}(n-3)(n-2)(n-1)n}{6} - V_{ref}\sum_{i=1}^{n-3} a_i\frac{(n-i)(n-i-1)(n-i-2)}{6}) \\ &- V_{ref}\sum_{i=2}^{n-2} b_i\frac{(n-i)(n-i-1)}{2}) - V_{ref}\sum_{i=3}^{n-1} c_i(n-i)) - V_{ref}\sum_{i=4}^{n} d_i) \end{aligned} \tag{3}$$

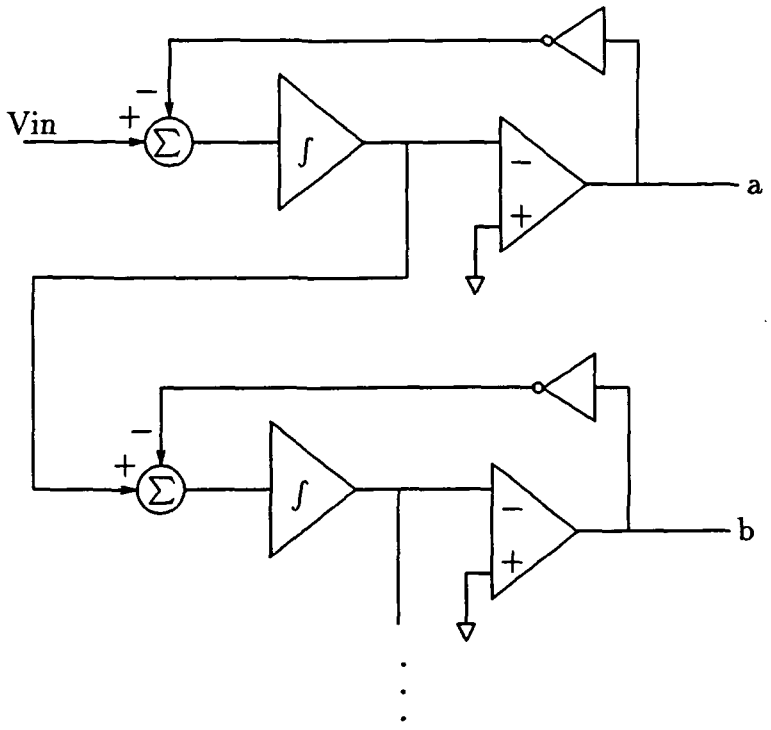


Figure 3: Cascade of First Order Modulators

The indices on the summations in equation 3 are skewed because there is a delay of one clock period in each of the integrators. The gain of the m^{th} integrator is α_m, a_i are the outputs of the first comparator, b_i the outputs of the second comparator, and so on. This equation can be solved for V_{in}/V_{ref}:

$$\begin{aligned}\frac{V_{in}}{V_{ref}} = & \frac{24}{(n-3)(n-2)(n-1)n}\sum_{i=4}^{n}(a_{i-3}\frac{(n-i+3)(n-i+2)(n-i+1)}{6} \\ & +\frac{b_{i-2}}{\alpha_1}\frac{(n-i+2)(n-i+1)}{2}+\frac{c_{i-1}}{\alpha_1\alpha_2}(n-i+1)+\frac{d_i}{\alpha_1\alpha_2\alpha_3}) \\ & +\frac{24V_{int}}{\alpha_1\alpha_2\alpha_3\alpha_4(n-3)(n-2)(n-1)nV_{ref}}\end{aligned} \quad (4)$$

The last term of equation 4 can again be considered to be a residue, which is not accounted for by the digital outputs, the smaller the residue, the higher the resolution. The decrease of the residue as n increases is now on the order of n^{-4}, which explains why the fourth order ADC is so much faster than the first order ADC. The decoder however, is more complex for the fourth order modulator than for the first order circuit. The outputs of the comparators now must be multiplied by a polynomial $f(i)$, where i increments with each clock cycle. We can interpret this to mean that the first digital outputs are to be weighted more heavily in the answer than the later ones. We can also see that the outputs of the first modulator are weighted more heavily than the outputs of the later modulators. If we have noise in the system, errors at the first part of the conversion cycle will affect the accuracy more than errors in the later part of the conversion cycle. Errors which are caused by single event

upsets of the data stored from the outputs of the comparators, and errors in the charge in the integrating capacitors can not be corrected.

5 Digital Processing and Self Calibration

The fourth order incremental ADC offers good trade offs between speed, accuracy, and noise rejection, but unfortunately gain errors in the integrators will cause nonlinearities in the ADC transfer function. The gain errors are caused by capacitor mismatch, and finite gain and bandwidth of the opamps. There are circuit techniques which correct the nonideal behavior of opamps and capacitor arrays, but in this case, the integrator gains do not need to be any particular value for the ADC to function properly, as long as the output processor knows exactly what the gain of each integrator is. It is usually preferable to increase the complexity of digital circuitry instead of analog circuitry, so I have chosen to calibrate this ADC by changing coefficients in the digital decoding logic to match the analog gain, instead of trying to set the gain of the analog integrators to exactly $\frac{1}{2}$. The digital decoding logic is shown in Figure 4.

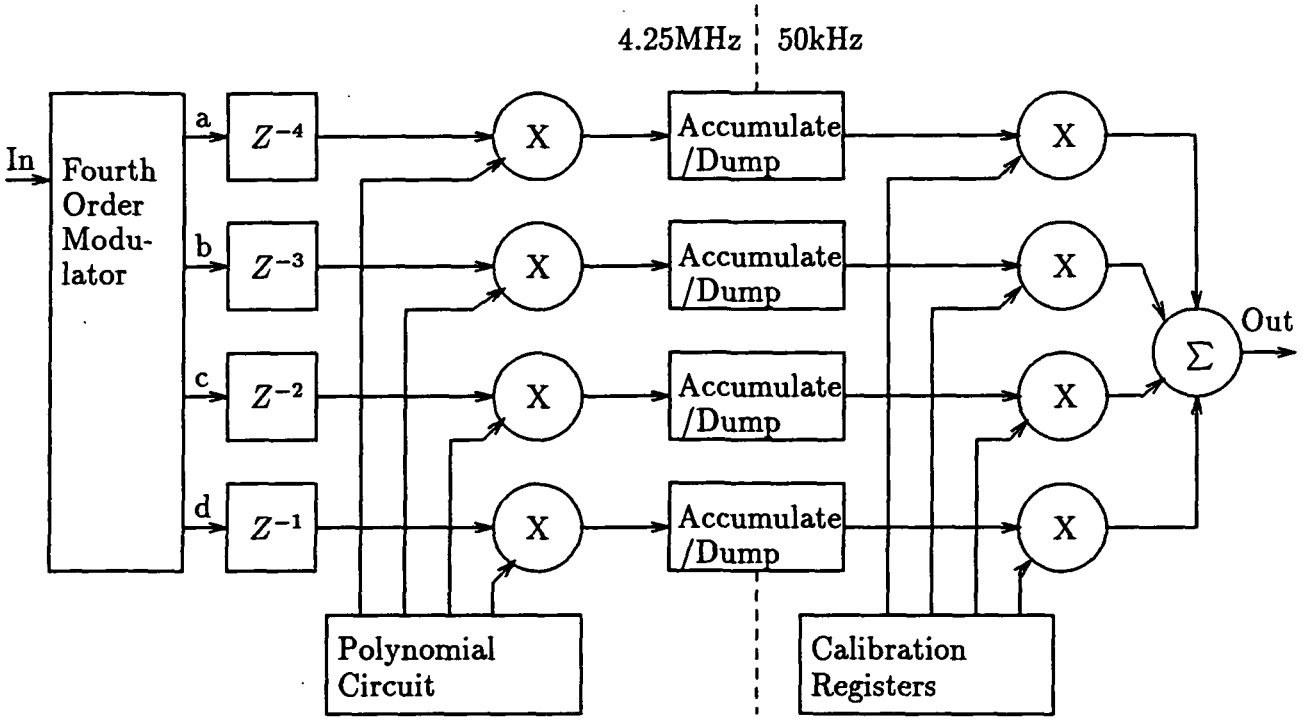


Figure 4: Decode and Calibration Logic

The z^{-x} blocks are delay lines which resynchronize the time shifted comparator outputs in equation 4. The polynomial circuit generates the terms $(n-i+3)(n-i+2)(n-i+1)$, $(n-i+2)(n-i+1)$, and $(n-i+1)$ where i is incremented each clock cycle. The finite differences technique is used to generate these terms, so no multiplications are necessary, which greatly simplifies the hardware. The multipliers of the delayed comparator data are

simply AND gates since the comparator outputs are only one bit wide. The accumulate and dump circuitry is reset when the integrators are reset and a new input sample is acquired. The sample rate of the signals coming into the accumulate and dump circuits is $n \times f_{reset}$ where f_{reset} is the input and output data rate of the overall ADC, and n is the number of comparison/integration cycles per input sample. The calibration registers store the exact values of integrator gains (α_m). The calibration registers are set during a self calibration routine, where inputs are applied which switch from $-V_{ref}$ to $+V_{ref}$ or from $-V_{ref}$ to $+V_{ref}$ part way through the conversion. These inputs precisely emulate input voltages without actually generating calibration voltages which must be accurate to 18 or more bits. An optimization routine then searches for the gains which make the combined errors of all the inputs a minimum, and stores these values in the calibration registers. Self calibration should be performed periodically to compensate for gain temperature coefficients, and component drift. The area of the self calibration circuitry can be made small by multiplexing the multipliers, since the circuitry after the accumulate and dump circuits runs at 50kHz.

6 Modulator Circuit

The modulator circuit in Figure 1 will be fabricated as a fully differential circuit to improve power supply rejection, cancel odd order capacitor nonlinearities, and increase signal to noise ratios. The phase 4 switch is turned off after the phase 3 switch to make the charge injection independent of which reference voltage is selected. The charge injection is then a common mode voltage which is rejected by the differential opamp [15]. Likewise, the phase 1 switch connected to ground is turned off before the phase 1 switch to V_{in}. Class A-B opamps [16], shown in Figure 5, are used to avoid nonlinear settling caused by slew rate limiting, and to reduce power consumption. The common mode feedback circuit for the opamp, Figure 6, provides continuous feedback through capacitors which are periodically zeroed to compensate for leakage.

The comparator, Figure 7, relies on positive feedback [18] to make a quick decision. The differential pair and input current mirrors isolate the integrator outputs from the switching noise in the later stages of the comparator. All circuits were simulated on Hspice [17], using the optimizer to help fine tune the transistor sizes.

7 Results

The number of integration/comparison cycles was chosen to be 85, which makes the sample rate 4.25MHz when the data rate is 50kHz. Simulations show that the fourth order incremental ADC is able to reject noise on it's input of $200\mu V_{rms}$, and noise at the comparator input of $20mV_{rms}$. A test chip will be fabricated to evaluate the feasibility of the self calibration scheme, as well as ADC performance in proton radiation. Test chip layout is in progress.

2.5.8

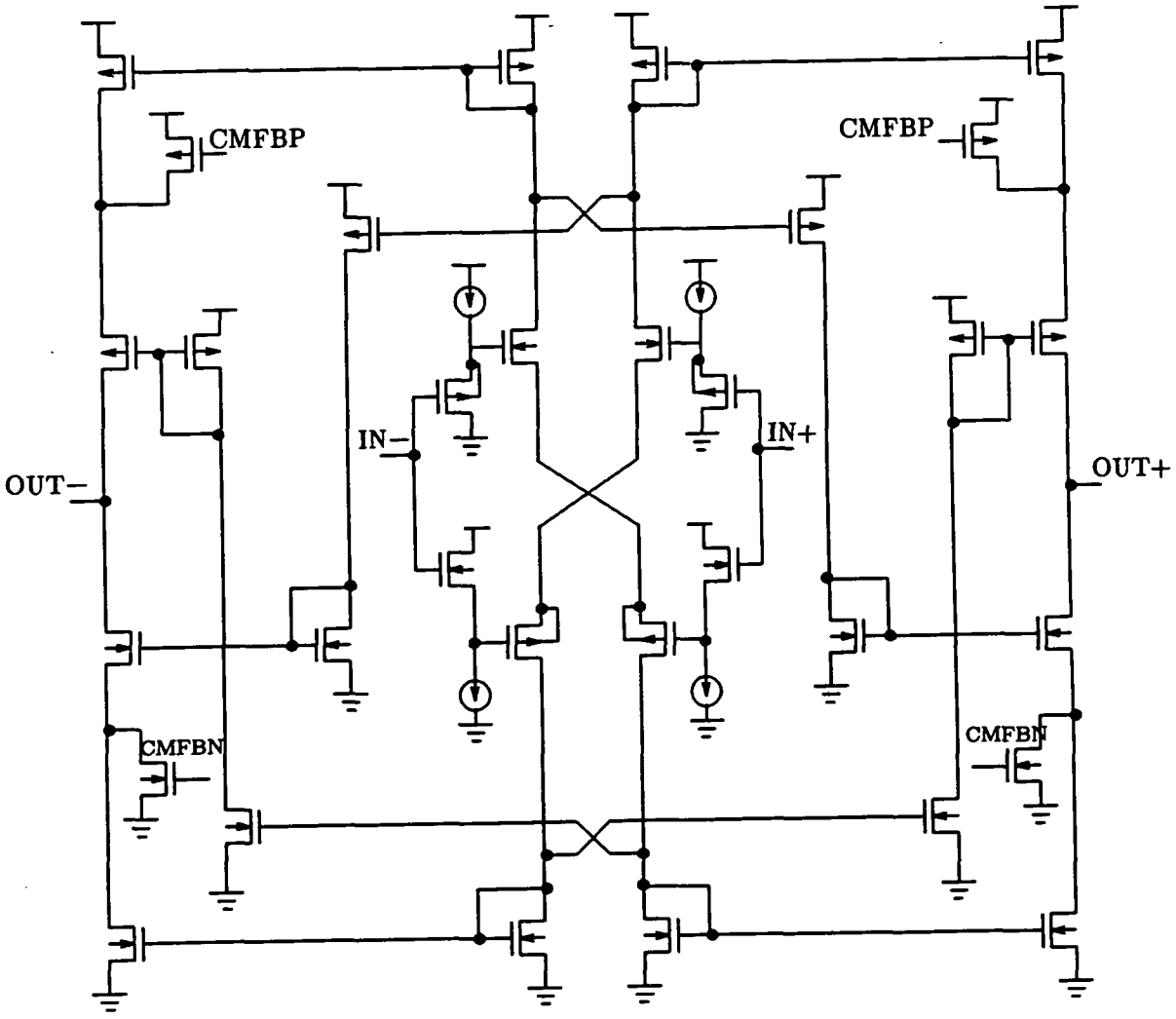


Figure 5: Class A-B Opamp

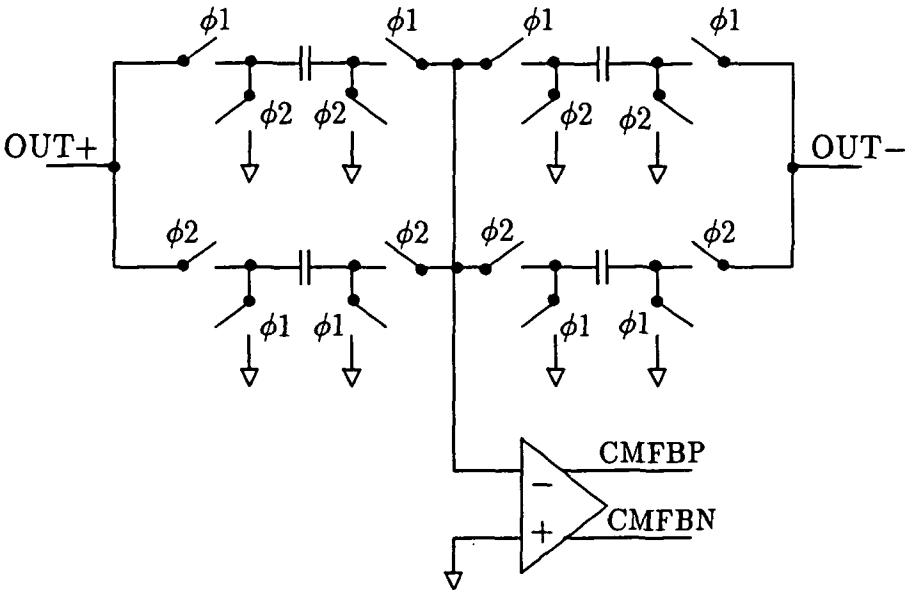


Figure 6: Common Mode Feedback Circuit

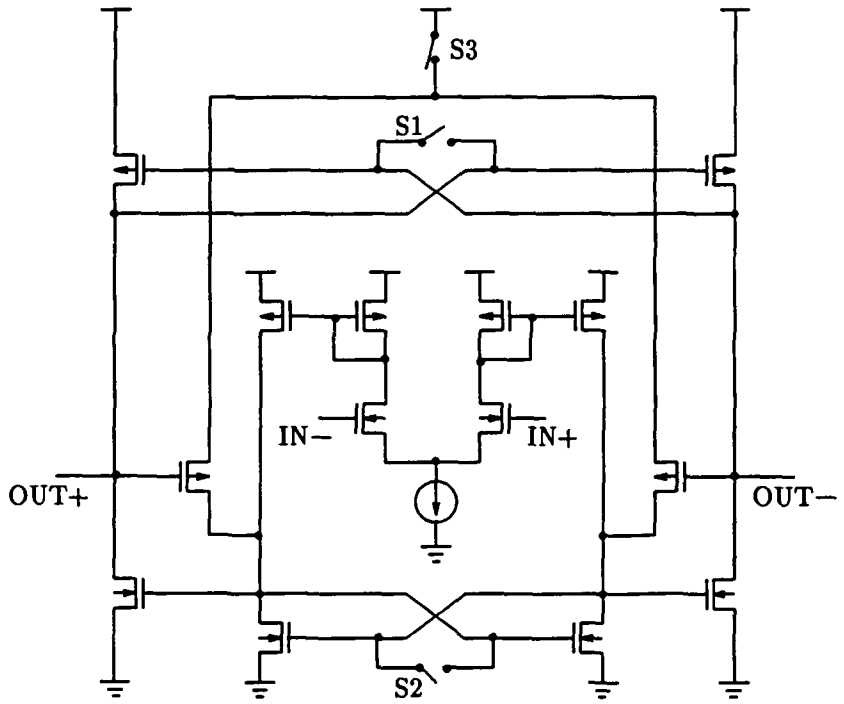


Figure 7: Positive Feedback Comparator

8 Future Work

A three level quantizer [19] can be used instead of the present two level quantizer to raise the resolution, or lower the sample rate. This change should not result in a loss of linearity for a differential circuit realization. MASH type Delta Sigma ADC's [10] are also sensitive to gain matching between integrators. This self calibration scheme could be extended to MASH Delta Sigma ADC's.

9 Acknowledgements

The author would like to thank Dr. Kelly Cameron for his advice on the finite differences technique, and Jody Gambles for the logic and circuit design of the digital postprocessor. This research was made possible by the NASA Space Engineering Research program which is funding the development.

References

[1] T. Turflinger, M. Davey, "Understanding Single Event Phenomena in Complex Analog and Digital Integrated Circuits," IEEE Transactions on Nuclear Science, Vol. 37, pp. 1832-1838, December, 1990.

[2] T. Turflinger, M. Davey, "Transient Radiation Test Techniques for High-Speed Analog-to-Digital Converters," IEEE Transactions on Nuclear Science, Vol. 36, pp. 2356-2361, December, 1989.

[3] G. Miller, M. Timko, H. Lee, E. Nestler, M. Mueck, P. Ferguson, "An 18b 10us Self-Calibrating ADC," ISSCC Digest of Technical Papers, pp. 168-169, February, 1990.

[4] K. Tan, S. Kiriaki, M. De Wit, J. Fattaruso, C. Tsay, W. Matthews, R. Hester, "Error Correction Techniques for High-Performance Differential A/D Converters," IEEE J. of Solid-State Circuits, Vol. SC-25, pp. 1318-1327, December, 1990.

[5] Ping Wai Li, M. Chin, P. Gray, R. Castello, "A Ratio-Independent Algorithmic Analog to Digital Conversion Technique," IEEE J. of Solid-State Circuits, Vol. SC-19, pp. 828-836, December, 1984.

[6] Yuh-Min Lin, B. Kim, P. Gray, "A 13-b 2.5-MHz Self-Calibrated Pipelined A/D Converter in 3-um CMOS," IEEE J. of Solid-State Circuits, Vol. SC-26, pp. 628-636, April, 1991.

[7] D. Draxelmayr, "A Self Calibration Technique for Redundant A/D Converters Providing 16b Accuracy," ISSCC Digest of Technical Papers, pp. 204-205, February, 1988.

[8] Analog Devices, Edited by D. Sheingold, *Analog-Digital Conversion Handbook.* Englewood Cliffs, NJ: Prentice-Hall, 1986.

[9] G. Clayton, *Data Converters.* London: MacMillan Education, 1988.

[10] *Oversampling Delta-Sigma Data Converters, Theory, Design, and Simulation.* Edited by J. Candy, G. Temes, IEEE Press, New York, NY, 1992.

[11] J. Robert, P. Deval, "A Second-Order High-Resolution Incremental A/D Converter with Offset and Charge Injection Compensation," IEEE J. of Solid-State Circuits, Vol. SC-23, pp. 736-741, June, 1988.

[12] J. Robert, G. Temes, V. Valencic, R. Dessoulavy, P. Deval, "A 16-bit Low-Voltage CMOS A/D Converter," IEEE J. of Solid-State Circuits, Vol. SC-22, pp. 157-163, April, 1987.

[13] B. Gordon, "Linear Electronic Analog/Digital Conversion Architectures, Their Origins, Parameters, Limitations, and Applications," IEEE Transactions on Circuits and Systems, Vol. CAS-25, pp. 391-418, July, 1978.

[14] G. Landsburg, "A Charge-Balancing Monolithic A/D Converter," IEEE J. of Solid-State Circuits, Vol. SC-12, pp. 662-673, Dec., 1977.

[15] K. Lee, R. Meyer, "Low-Distortion Switched-Capacitor Filter Design Techniques," IEEE J. of Solid-State Circuits, Vol. SC-20, pp. 1103-1113, Dec., 1985.

[16] R. Castello, P. Gray, "A High-Performance Micropower Switched-Capacitor Filter," IEEE J. of Solid-State Circuits, Vol. SC-20, pp. 1122-1132, Dec., 1985.

[17] Meta-Software Inc., 1300 White Oaks Road, Campbell, CA 95008 *HSPICE User's Manual.*

[18] D. Wiseman, "A High Speed CMOS A/D Converter," Fourth NASA Symposium on VLSI Design, pp. 2.5.1-2.5.13, October, 1992.

[19] J. Paulos, G. Brauns, M. Steer, S. Ardalan, "Improved Signal-to-Noise Ratio using Tri-Level Delta-Sigma Modulation," Proceedings of ISCAS, pp. 436-466, May, 1987.

Session 3
Design Synthesis

Chairman: James Frenzel

On The Decomposition Of Synchronous State Machines Using Sequence Invariant State Machines

K.Hebbalalu, S.Whitaker and K.Cameron
NASA Space Engineering Research Center
University of Idaho
Moscow, Idaho 83843

Abstract - **This paper presents a few techniques for the decomposition of Synchronous State Machines of medium to large sizes into smaller component machines. The methods are based on the nature of the transitions and sequences of states in the machine, and on the number and variety of inputs to the machine. The results of the decomposition, and of using the Sequence Invariant State Machine (SISM) Design Technique for generating the component machines, include great ease and quickness in the design and implementation processes. Furthermore, there is increased flexibility in making modifications to the original design, leading to negligible re-design time.**

1 Introduction

Most digital Very Large Scale Integrated (VLSI) circuits currently designed contain controllers which co-ordinate and control activities occuring inside and outside the chip. Usually, such controllers are comprised, partly or wholly, of synchronous sequential machines. Often, depending on the nature and complexity of the control action required, these sequential machines are of medium to large sizes (requiring five or more state variables). While it is always possible to design these machines as single units using random logic, this implementation presents the designer with a narrow choice of options, since a large number of state variables results in difficult-to-derive and unwieldy design equations. Also, the layout process for these single, large machines becomes complicated and even minor changes in the original flow Table may result in a complete re-design. In addition, large machines tend to be slower and not very amenable to fault diagonostics or to trouble-shooting.

Hence there is a need to decompose such large machines into smaller, more manageable component machines, which offer greater flexibility to the designer. This paper presents a few such decomposition techniques, resulting in faster and better designs. Furthermore, the Sequence Invariant State Machine (SISM) Design Technique [1] is used to generate the component machines, which enhances the ease of the design process, by not requiring the use of complicated design equations and K-Maps. The final and greatest advantage is that the layout is automated.

2 Notations and Definitions

Definition 1: An *active state machine* is one in which some outputs are valid and which is exercising control over the chip in part or whole.

Definition 2: An *inactive state machine* is one in which no output is valid.

Definition 3: A *ready protocol* between two state machines signifies that the machine which originates the *ready* signal is becoming inactive and is handing over control to the machine which receives the signal.

Definition 4: A *start-up state* is a state in which a machine waits until it receives a ready signal from another machine, indicating it is to become active and to go into its next valid state. In the example of Figure 1, states A, X' and X" are start-up states.

Definition 5: An *idle state* or *wait state* is one in which a machine raises a ready signal to another machine to become active, and perhaps waits in this state as long as the machine is inactive: *ie.,* until it receives a ready signal in its turn. In Figure 1, states F, X' and X" are wait states. A single state can function both as a start-up and as a wait state.

Definition 6: State splitting is the replication of a state from the original machine, into corresponding states in component machines. These states serve exactly the same function as the original state with the same outputs. States G' and G" in Figure 1, are split states of the original state G. State splitting is used to limit the interaction and to avoid unnecessary transfer of control between machines.

3 Decomposition and SISMs

Most of the state machines encountered during design, have their own individual peculiarities and constraints, and a formal, all-encompassing procedure for their design or decomposition is difficult. There would be several possible ways of achieving the results, and depending on the criteria, more than one optimal solution would be found. These criteria for finding the 'best' solution usually include speed and area considerations, cost, quickness and ease of implementation and testing, among others. The designer might choose one or more of these, in various orders of priority.

Some decomposition techniques are presented in the following sections, bearing in mind the implementation of the component machines using the SISM Design Technique [1]. There are several advantages to be gained, using this method:

- The structure and design of all the sub-state-machines are the same, provided, they all have equal number of state variables.
- The design of the SISMs is simple and easy, not involving the use of K-Maps or design equations.
- The layout process for these machines is uncomplicated and is automated, leading to very quick results[2]. Also, since the basic structure of the SISM is very regular, the layout is very dense and the area savings are considerable.
- The simulations for the circuits using SISMs are also easy and quick. Any changes to be incorporated are easily done with negligible re-design time.

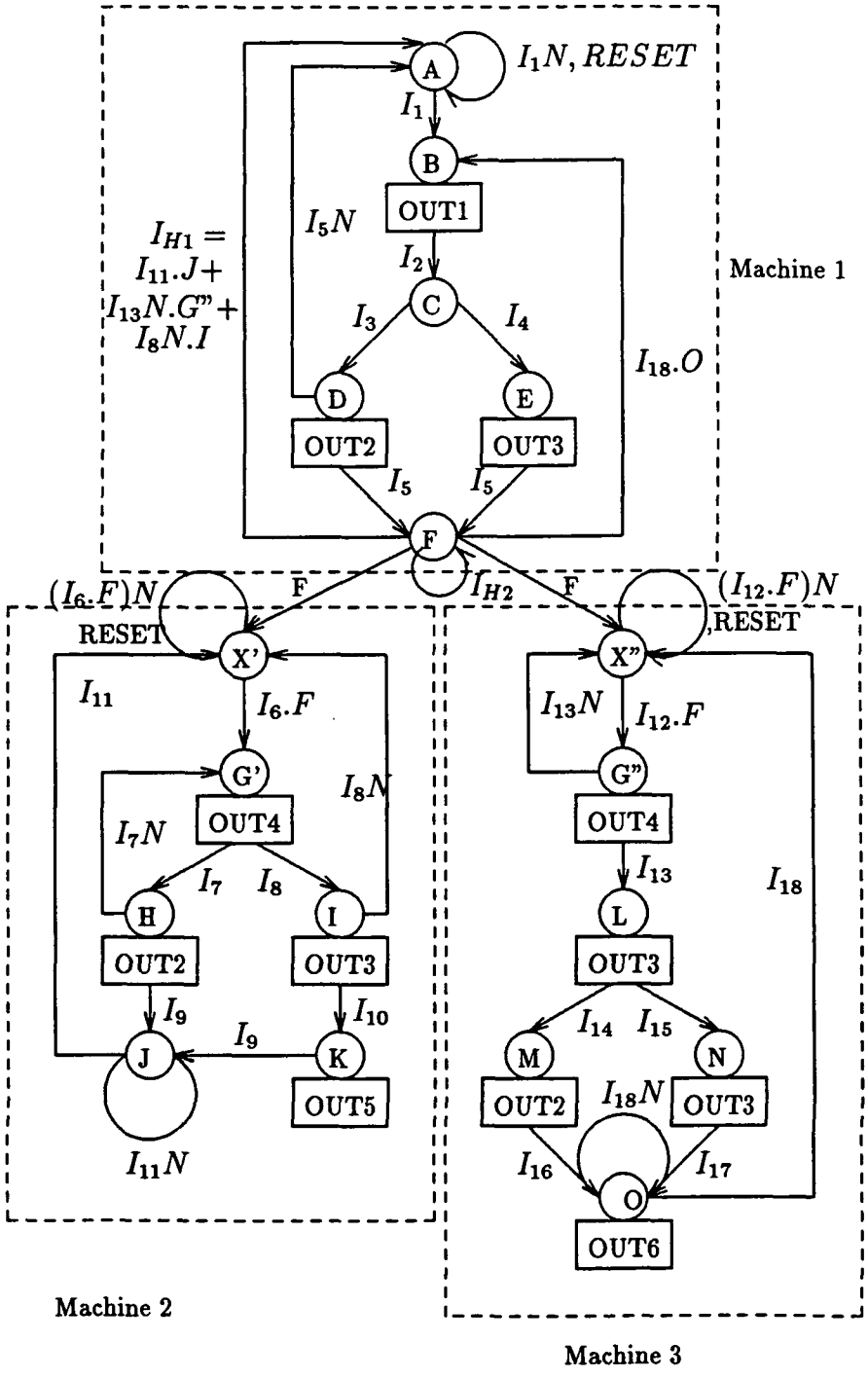


Figure 1: Example of Decomposed State Machine

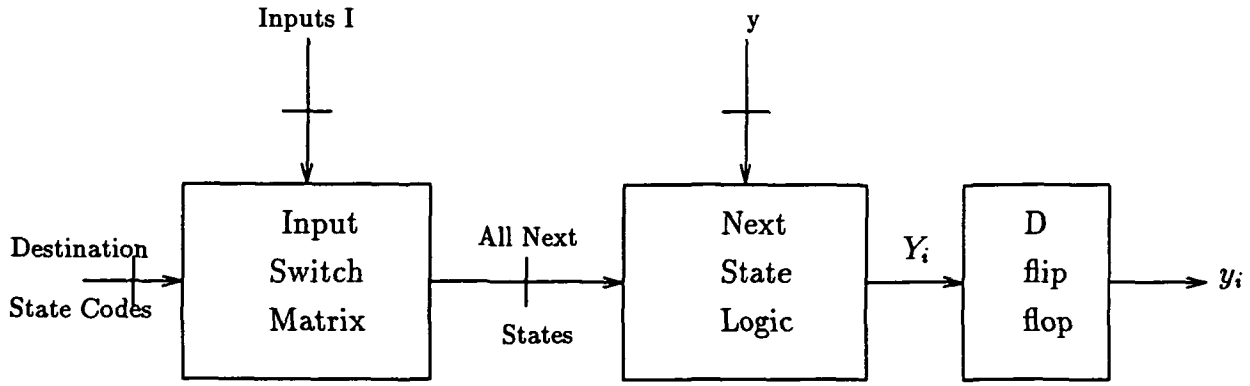


Figure 2: General Block Diagram of SISM

	I_1	I_2	I_3
S_0	N_{01}	N_{02}	N_{03}
S_1	N_{11}	N_{12}	N_{13}
S_2	N_{21}	N_{22}	N_{23}
S_3	N_{31}	N_{32}	N_{33}
S_4	N_{41}	N_{42}	N_{43}
S_5	N_{51}	N_{52}	N_{53}
S_6	N_{61}	N_{62}	N_{63}
S_7	N_{71}	N_{72}	N_{73}

Table 1: General eight-state three-input flow table

3.1 The operation of SISMs

The driving idea behind the SISM architecture is to build a state machine given only the dimensions (number of inputs i and number of states s) of the flow Table. The state machine thus created must be capable of performing the operations of *any* sequence of states that are described in the i by s flow Table, to achieve sequence invariance.

An SISM has the basic block structure as shown in Figure 2. The destination state codes are a representation of the flow Table to be implemented. The input switch matrix selects a single column of the flow Table and passes the next state information for the entire selected column to the next state logic decoder. This next state decoder selects one from the column of states presented to it, as the next state, depending on the present state. The hardware for the SISM depends only on the dimensions of the flow Table to be implemented. The only difference between state machines built for flow tabes with different sequences of states, but with the same dimensions, is in the programming of the destination state codes.

The operation of an SISM can be illustrated with the following example. Let Table 1 represent a general flow Table for a 3 state variable, 3 input state machine. I_1, I_2 and I_3 are the inputs, $S_0 ... S_7$ are the present states and N_{ij} are the next states. The set of N_{ij} also comprises of the destination codes. Let the state assignment be S_0 =000, S_1=001,.....S_7=111.

The next state logic is a general BTS circuit [3,4] with each path decoding a state. The input switch matrix passes the destination codes to the next state logic, as shown in Figure 3. The circuit works as follows: For an active input, I_i, all next state codes in the ith column of the input matrix are passed to the inputs of the next state logic. The present state variables, ys, select only one code among them and pass it on to the flip-flops. For example, if the machine is in state S_1 and input I_2 is asserted, then N_{12} would be passed to the inputs of the flip-flops. The current input selects the set of potential next states that the circuit can assume (selects the input column) and the present state variables select the exact next state (row in the flow Table) that the circuit will assume on the next clock pulse.

4 Decomposition Techniques

This section describes some methods of decomposing large state machines and at the same time, indicates the feasibility of each method for varying situations, with illustrations.

4.1 Functional and Hierarchical Decomposition

As is most often the case, the designer of a digital controller is provided with a word statement of the specifications and the sequence of operations that the controller should conform to, rather than a state diagram or a flow Table. In such cases, a popular method is to decompose the controller into functional blocks, with each block representing an operation in the main sequence of events. In other words, a functional block diagram is generated. Each functional block is progressively decomposed into smaller, more specific operations, culminating in state machines with sets of states, defining macro operations. The requisite flow Tables, design equations and outputs are generated for these machines and are then hierarchically combined to produce the final controller.

4.2 Decomposition using 'bottleneck' states

Many state diagrams reveal on inspection natural 'bottleneck' states which connect groups of states. In the example shown in Figure 1, state F is such a state. The state machine can be decomposed around such states, making the groups of states into sub-machines, with the bottleneck state included in any one of them.

This method of decomposition is convienient and economical in that, it saves the generation of additional dummy states to help decomposition, or the resortion to state-splitting for the same purpose. The only criterion to be observed here is that the number of states in the groups connected by the bottleneck should be roughly the same, so that, when formed into sub-machines, they all need the same number of state variables. This method has been used in fragmenting the state machine in Figure 1, where there are 3 groups of states around state F, with each group having 6 states, needing 3 state variables each.

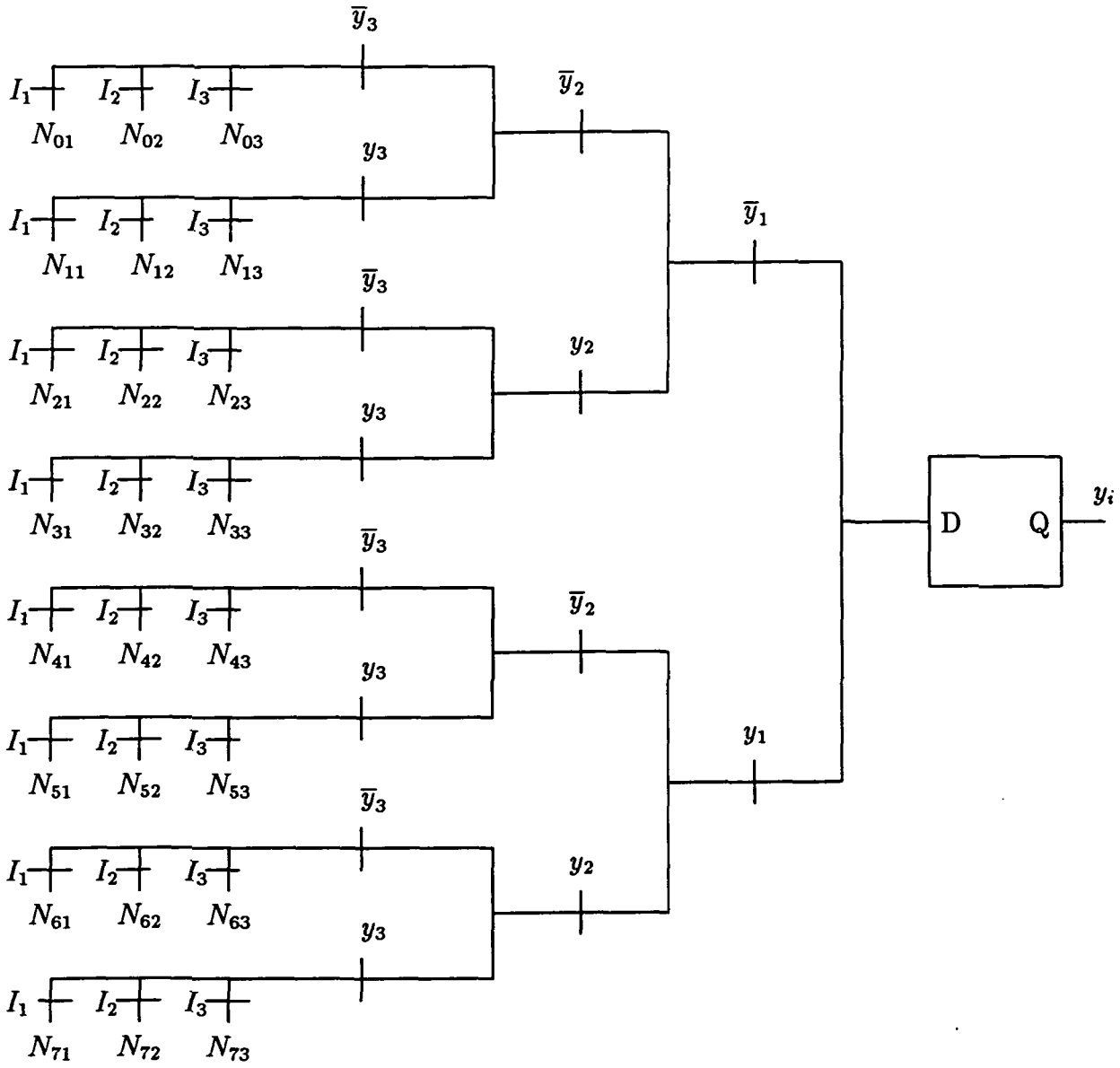


Figure 3: General 8 state 3 input next state equation circuit

	I_1	I_2	I_3	I_4	I_5	I_6
A	D	B	-	-	-	-
B	C	F	D	E	-	-
C	E	A	-	-	-	-
D	F	C	E	B	-	-
E	A	D	B	D	-	-
F	A	B	-	-	A	B

Table 2: Example flow table showing types of inputs

4.3 Decomposition using groups of inputs and states

The inputs to a state machine can be classified under three general categories: 1) those inputs under which *all* the states make transitions to other states in the machine; 2) those under which states of a *group* transition amongst themselves, and 3) inputs under which a *single* state transitions to other states. In Table 2, I_1 and I_2 belong to the first category, I_3 and I_4 to the second and I_5 and I_6 to the third.

Fragmentation of a large machine can be done, depending on the nature of inputs presented to it. An examination of the flow Table reveals which of the three classes of inputs discussed above are predominant, and decomposition can be done accordingly. The central idea here is to seek for *disjoint* groups of inputs, under which groups of states transition amongst themselves, and in addition, there should be little or no overlap between the groups; *ie.,* isolated groups of inputs with corresponding isolated groups of states should be identified. These can be formed into sub-machines.

It is clear that, if a large machine comprises of inputs which are used by *all* the states in the machine to make transitions, or if there is an even mix of the three types of inputs discussed above, as in Table 2, then breaking up of that machine is not viable, or at the most, minimal. This is because decomposing that machine would necessitate a duplication of either the inputs or the states or both in all the component machines leading to larger chip area and reduced speed. On the other hand, if a machine has inputs exclusively of the third class (under which single states transition to other states), then a maximal decomposition of the machine is possible. In other words, the machine can be fragmented into *one bit state machines*, each containing a state of the original machine, with corresponding inputs under which the state transitions to other states (in this case, to other one bit machines). This method is also called as *one hot coding*[5] and is quite popular in applications where the size of the machine is not very large, since it requires the use of one flip-flop for every state of the machine, and thus the area occupied is more. The main advantage here is ease and quickness in design, with little or no design equations, and can be used for standard cell designs. Also, in one bit machines, since there is only one state variable, the series and feedback delays are minimal and the speed of operation is enhanced.

5 Implementation using SISMs

In order to implement the sub-machines using SISMs, a few general guidelines have to be followed, during the decomposition: All the machines should have an equal number of state variables and a common maximum of the number of inputs, to take advantage of the repetitive and automated design and layout processes for SISMs. If any machine has fewer states or inputs than those of other machines, dummy states and inputs can be introduced in that particular machine, to preserve regularity. Every component machine must have a start-up/wait state, to facilitate the ready protocol and the handover of control between machines. Also, such states should have no valid outputs that are used in the normal activities of the controller *ie.,* the machine should be inactive, while it is in the start-up or wait state. If a machine does not have such states naturally, an additional state performing their functions should be introduced. An added constraint is that a machine can have only one wait state, though it can have more than one start-up state. Usually the start-up states of all machines are coded 000 or 111 to help reset the machines with a common reset signal, provided to the flip-flops.

Finally, it is a good practice to keep all states from the original machine, that transition among themselves, in a single sub-machine, rather than distribute them over a few machines. This saves unnecessary interaction and handing over of control between them, resulting in hardware savings and speed enhancement. If this is not feasible under certain circumstances (for example, when there is a single transition from a state in one group, to a state in another, both being separated by many states), then, state splitting can be used to limit the interaction.

An example to illustrate the techniques described so far, is shown in Figure 1. The original machine with 14 states and 6 outputs has been broken up around the natural bottleneck state F, into 3 sub-machines M_1, M_2 and M_3. All the machines have 6 states each and can be coded with 3 state variables. States A and F are the natural start-up and idle states for machine M_1. State X' is the combined start-up and idle state for M_2, while X" serves the same purpose for M_3. States X' and X" have been deliberately added to facilitate the ready protocol. State G of the original machine, common to group HIJK and to group LMNO, has been split into states G' and G", to eliminate any interaction between M_2 and M_3. It can also be observed that there are no common inputs among all the three machines and that there is a maximum of 3 inputs to any state.

In the normal sequence of events, at power-up, a common reset signal is applied, which leaves all the machines in their start-up states, namely, A, X' and X". Machine M_1 is activated first and proceeds through its sequence of states. On reaching state F, it waits there, and depending on the occurance of I_6 or I_{12}, either state G' or G" will be entered, and the corresponding machine activated. The return of control to M_1 is achieved when M_2 sinks back to state X' or when M_3 returns to state X", at which time, M_1 moves from state F to either of states A or B, depending on where the ready signal originated. (The ready signal is just the transition between states; for example, state J going to state X', on receipt of I_{11}, will be a ready signal for machine M_1 to go from state F to state A.)

The generation of the hardware for the machines begins with the construction of the flow Tables for the individual machines. As an example, the Table for machine M_1 is shown in

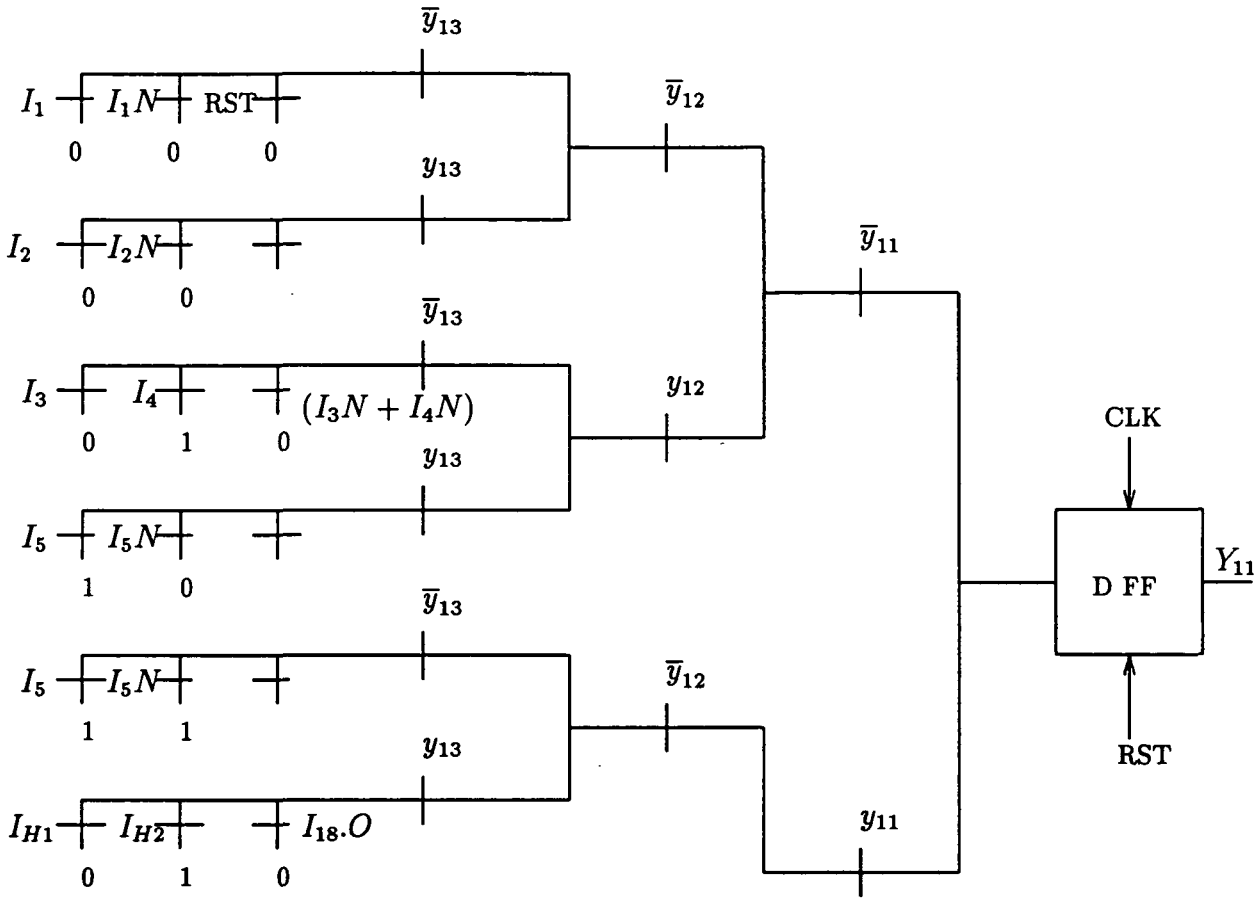


Figure 4: Next state equation circuit for Y_{11} of Table 3

Table 3. Next the circuits for the machines are generated using the SISM procedure. This is illustrated in Figure 4, for one of the next state variables, Y_{11}, of the flow Table shown in Table 3. In Figure 4, the complementary signals of all inputs (I_1N .....I_5N) have been used to prevent the input to the flip-flop from going into a high- impedance state, when the inputs are not active. For example, if the machine is in state B and I_2 has not occured for a clock cycle, and if the complementary signal, I_2N were not present as an additional input, then, the flip-flop would be presented with a high impedance at its input, thereby making its output unpredicTable on the next clock edge. However, if it can be assured externally that the input always occurs within the clock cycle, the complementary signal can be dispensed with. The output forming logic is generated as usual, using the present state codes of the states for their corresponding outputs.

6 Conclusions

The paper discussed the need for the decomposition of large state machines and a few techniques for achieving the same. The use of SISMs for implementing the component machines and the general requirements for doing so, were examined in detail. The methodology pro-

y_{11}	y_{12}	y_{13}		I_1	I_1N	I_2	I_2N	I_3	I_3N	I_4	I_4N	I_5	I_5N	I_{H1}	I_{H2}	RST
0	0	0	A	B	A	-	-	-	-	-	-	-	-	-	-	A
0	0	1	B	-	-	C	B	-	-	-	-	-	-	-	-	A
0	1	0	C	-	-	-	-	D	C	E	C	-	-	-	-	A
0	1	1	D	-	-	-	-	-	-	-	-	F	A	-	-	A
1	0	0	E	-	-	-	-	-	-	-	-	F	-	-	-	A
1	0	1	F	-	-	-	-	-	-	-	-	F	-	A	F	A

$$I_{H1} = I_{11}.J + I_{13}N.G" + I_8N.I$$

$$I_{H2} = \overline{I_{H1} + I_{18}.O}$$

Table 3: Flow Table for the component machine M_1

vided in the paper gives a simple and easy way of implementing the decomposition of large state machines. The process is also flexible, quick and automated, leading to reduced time and cost of design.

References

[1] S. Whitaker, Manjunath Shamanna and G. Maki, "Sequence-Invariant State Machines," *IEEE Journal of Solid State Circuits*, Vol. 26, August 1991, pp. 1145– 1161.

[2] D.Buehler, S. Whitaker, and J.Canaris, "Automated Synthesis of Sequence Invariant State Machines," *2nd NASA SERC Symposium on VLSI Design*, Moscow, Idaho, November 1990, pp. 4.4.1– 4.3.9.

[3] D.Radhakrishnan, S. Whitaker, and G.Maki, "Formal Design Procedures for Pass Transistor Switching Circuits," *IEEE Journal of Solid State Circuits*, April 1985, pp. 531–536.

[4] S. Whitaker, and G. Maki, "Pass Transistor Asynchronous Sequential Circuits," *IEEE Journal of Solid State Circuits*, Vol. SC-24, February 1989, pp. 71–78.

[5] Lee.A.Hollaar, "Direct Implementation of Asynchronous Control Units," *IEEE Transactions on Computers*, vol. C-31, NO.12, December 1982, pp. 1133–1141.

This research was supported in part by NASA under Space Engineering Research Center Grant NAGW-1406.

Heuristic-Based Scheduling Algorithm for High Level Synthesis

Gulam Mohamed, Han-Ngee Tan & Chew-Lye Chng
School of Electrical & Electronic Engineering
Nanyang Technological University
Singapore 2263
Republic of Singapore
e-mail : gmohamed@ntu.ac.sg

Abstract - **A new scheduling algorithm is proposed which uses a combination of resource utilization chart, a heuristic algorithm to estimate the minimum number of hardware units based on operator mobilities and list-scheduling technique to achieve fast and near optimal schedules. The schedule time of this algorithm is almost independent of the length of mobilities of operators as can be seen from the benchmark example (fifth order digital elliptical wave filter) presented when the cycle time was increased from 17 to 18 and then to 21 cycles. It is implemented in C on a SUN3/60 workstation.**

1 Introduction

High level synthesis of digital system involves mapping of an abstract behavioral or algorithmic level specification to a register-transfer-level structure while satisfying a set of constraints. The system will normally output a datapath structure which implements the specification together with a controller unit. The major tasks involved are i) extraction of timing and data precedence relationship from the input ii) scheduling of operators into timesteps iii) allocation of hardware resources like functional units, storage devices and interconnects and iv) creation of the control unit[1]. Among these tasks, operator scheduling in (ii) has been acknowledged to be one of the most crucial step[2]. Decisions made in this step will have direct consequence on the performance and cost of the design in VLSI implementation.

We consider in this paper the problem of scheduling under time constraint, i.e., finding the cheapest schedule without exceeding the given number of time steps. A heuristic based algorithm is proposed which tracks hardware usage, estimates the minimum number of hardware units required based on operator mobilities and uses list-scheduling technique to place operators to timeslots while minimizing the number of hardware functional units, lifetimes of variables and additional multiplexer costs. Our proposed algorithm is able to place operators into timeslots in almost linear time when their mobilities were increased as can be seen in the benchmark example presented. In this paper we consider the scheduling algorithm which minimizes the total number of hardware functional units only. The proposed algorithm in its fullest implementation includes storage device and multiplexer costs considerations. This is not discussed here due to space limitation but can be found in [3].

This paper is organized as follows. Section 2 describes briefly previous related works. The scheduling problem is formulated in Section 3. Section 4 describes our Nanyang Technological University Scheduling System (NTUSS) followed by a brief description of the basic scheduling

algorithm in Section 5. Section 6 gives results for a benchmark example and finally Section 7 concludes this paper.

2 Previous Works

A. C. Parker et al. introduced the notion of freedom-based scheduling in MAHA[4]. Operators with the smallest degree of freedom (bounded by ASAP and ALAP times) are given the highest priority when considered for scheduling. In Force-Directed Scheduling[5], P. G. Paulin and J. P. Knight use 'force' to determine the suitability of an operator to a timestep. The 'force' value favors a balanced distribution of operators over all the control steps. On the other hand, the suitability of placing an operator to a control step is determined by a selection function in the system described in [6] by Park In-Cheol and Kyung Chong-Min. Similarly, their selection function is based on balancing the distribution of operators in each control step. Hwang C. T. et al.[7] use integer linear programming model to find the optimal operator-timeslot combination while minimizing a cost function which includes hardware costs. In most of these systems the time taken to schedule increases tremendously with increasing length of mobilities of operators.

3 Problem Formulation

For a given Acyclic Directed Dataflow graph, Critical Path analysis is used to determine the possible timesteps an operator can occupy based on their given processing times and data precedence relation. The longest path from the input to the output represents the critical path and the latest time the last operator can start will place an upper bound on the maximum timesteps the dataflow graph takes to produce its outputs. Every operator will have to be restricted to within their ASAP and ALAP times if the last operator in the dataflow graph were to start its operation within the upper bound. Operators having their ASAP times equal to their ALAP times are in the critical path and are scheduled to their respective ASAP times. We use a combination of resource utilization chart, heuristic algorithm to estimate the minimum number of hardware units required and list-scheduling technique[8] to place the rest of the operators to one of the timepoints within their ASAP and ALAP times such that :

(i) data precedence relationship as defined in the input dataflow graph is not violated.

(ii) the number of hardware functional units, lifetimes of variables and additional multiplexer costs are minimized.

We assume that only single function modules, for example an add operator is processed by a hardware unit which implements an add function only, are used. Under this circumstance, if we can minimize the number of hardware functional units required to implement each type of operator then we can eventually minimize the total number of hardware units to implement the whole dataflow graph. Consequently, if we can assign all operators to all the available hardware units without logical, timing and resource conflicts[9], then we have found the cheapest feasible schedule.

4 Overview of NTUSS

In this paper we consider time-constrained scheduling with respect to minimizing the hardware costs only. Other considerations such as minimizing lifetimes of variables and minimizing multiplexer costs are not presented here. In our model, basically the problem of assigning off-critical path operators to any one of the timepoints within their ASAP and ALAP has been translated to :
'Squeezing' these operators into their respective resource utilization charts occupying the least height as possible without violating timing and data precedence constraints. We obtain the minimum height through a heuristic based algorithm which estimates the minimum number of hardware units required based on their mobilities (ALAP-ASAP times). It is not known to the author if it is possible to determine the absolute minimum number of hardware units based on their mobilities.

4.1 Resource Utilization Chart

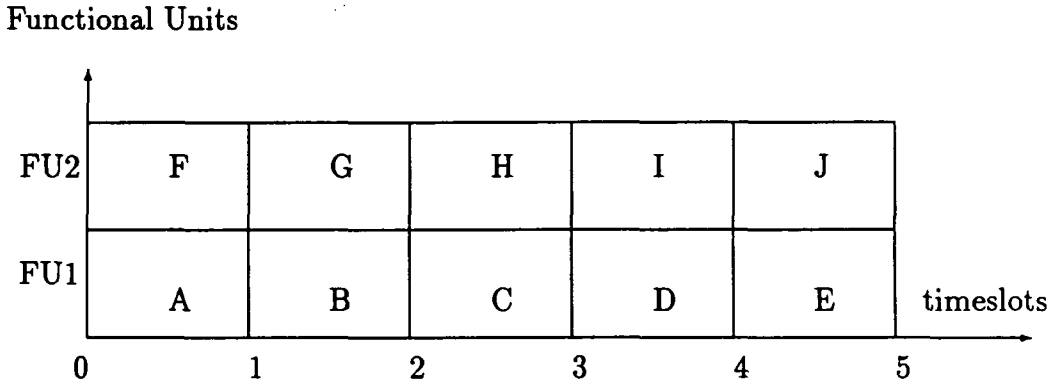


Figure 1: Resource Utilization Chart

A resource utilization chart which is equivalent to a Gantt Chart[10] shows in detail the usage of hardware functional units. Figure 1 shows such a chart. The vertical axis represents the type of hardware functional unit which process the operators that are placed in the chart. All hardware along this axis, for example FU1 and FU2 are identical. The height of the chart represents the total number of hardware units of this type required in the final implementation. The horizontal axis represents physical time which is also the schedule time. A timeslot occupied by an operator in this chart represents the starting time (schedule time) and the duration in which the operator in question is assigned to the stated hardware functional unit on the y-axis. Hence by filling operators in the chart, the algorithm does simultaneous scheduling and allocation. Each type of operator in the dataflow graph has its own resource utilization chart.

4.2 Algorithm to Determine the Minimum Number of Hardware Units

The minimum number of hardware units required to process a particular type of operators within their ASAP and ALAP times (mobility) is strongly dependent on their number and concentration along the timepoints. All critical path operators should also be taken into consideration. To estimate the minimum number, we concentrate our efforts on the timeslot which has the highest overlap of mobilities including that of critical path operators. We assume equal probability distribution of operators similar to the one adopted by Force-Directed Scheduling[5]. For example an operator with ASAP=0 and ALAP=2 will contribute a value of $\frac{1}{3}$(distribution density value) to each timeslot from 0 to 2. Critical path operators contribute a value of '1' to their respective timeslots. The timeslot with the greatest sum value (highest distribution density) is chosen. As shown in Figure 2, the sum may contain an integer portion and a fractional portion.

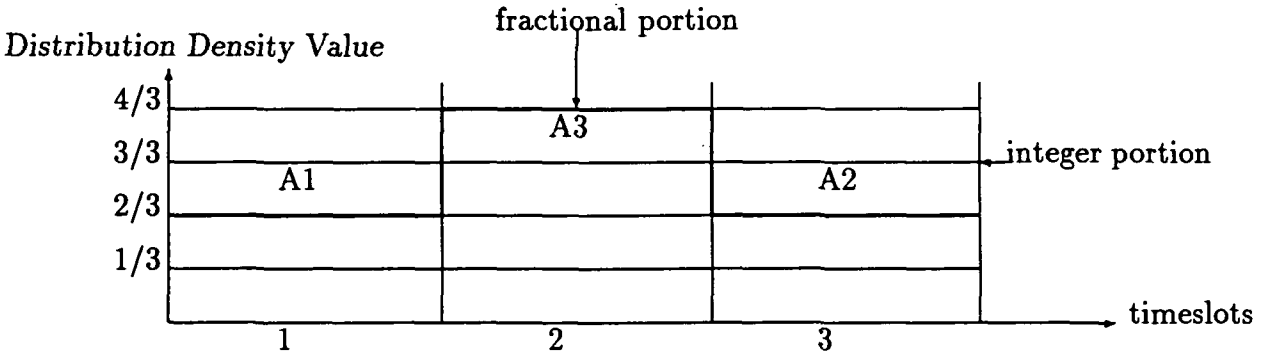


Figure 2: Distribution Density Chart

The highest distribution density value in this figure is $1\frac{1}{3}(1+\frac{1}{3})$. It can be said that $1\frac{1}{3}$,i.e., 2 machines can safely process all the operators whose distribution density values are shown in Figure 2. But we can provide a better estimate by the following method.

Using the integer portion as a reference, conceptually areas A1 and A2 represents the 'idle' time of the machine when it is operated from timeslot 1 to 3. Area A3 on the other hand, represents the amount of 'excess' work that one (reference) machine is incapable of handling at timeslot 2. If area A3 can be 'moved' to fill into areas A1 and A2 without any 'excess', then there is no need to employ an extra machine. This is the criteria used to determine whether the integer portion of the highest distribution density alone is sufficient to be taken as the minimum number or an extra '1' is required. The lower and upper limit of the x-axis of Figure 2 is obtained from step 3 and step 4 respectively in Figure 3. The algorithm to estimate the minimum number of hardware functional units for one type of operator is given in Figure 3.

1. Calculate Distribution Density value for each timeslot in the resource utilization chart.
2. Get timeslot with the highest distribution density value.
3. Get minimum ASAP time among all operators whose mobilities coincide with the timeslot obtained in step 2 (=mintime).
4. Get maximum ALAP time among all operators whose mobilities coincide with the timeslot obtained in step 2 (=maxtime).
5. maxvalue= greatest integer $\leq$ highest distribution density value.
   sum= 0;
   for (i= mintime; i $\leq$ maxtime; i++)
   {
   for (all operators whose mobilities lie within mintime and maxtime)
   /* this also includes operators in the critical path */
   {
   calculate sum(i)= distribution density value for timeslot(i) - maxvalue;
   sum= sum + sum(i);
   }
   }
   if (sum $\geq$ 1)
   minimum number of hardware units required= maxvalue + 1;
   else
   minimum number of hardware units required= maxvalue;

Figure 3: Algorithm to calculate the minimum number of hardware units.

The concepts presented so far can be integrated into a scheduling system whose algorithm is described in Figure 4.

1. Read in the records.
2. Create successor and predecessor list for each operator in the list.
3. Determine the maximum ALAP among operators in each group.
4. Calculate the minimum number of hardware units required for each group.
5. Create resource utilization chart for each group.
6. 'Enter' operators that are in the critical path into their respective resource utilization chart.
7. Sort the rest of the operators in order of decreasing processing times. If adjacent operators in the sorted list have equal processing times, then they are sorted again in order of ascending ALAP times.

8. Select the top (of the sorted list) operator's resource utilization chart.
9. Get the earliest available timeslot from the selected chart.
10. Go back to the sorted list and find the first operator which can legally be placed in the chosen timeslot. Steps 9 and 10 are repeated until an assignment is found.
11. Update mobilities of its predecessors and successors. Also mobilities of its predecessor's predecessors and successor's successors.
12. Update the resource utilization chart of those operators that has fallen into the critical path.
13. Go back to step 7 if there are some more unscheduled operators.

Figure 4 : The basic scheduling algorithm

5 Description of the Basic Algorithm

The basic technique used to 'fill' up the resource utilization chart is similar to the list-scheduling[8]. In list-scheduling used in Deterministic Scheduling Theory, tasks are ordered into a list based on some priority. When the processor becomes available, this list is scanned and the first unexecuted task which can be processed by this processor is then scheduled at that time and allocated to that processor. In our system, the list comprises of all the non-critical path operators. They are sorted in order of decreasing processing times. If adjacent operators in the list have equal processing times, they are then sorted in order of increasing ALAP times. The resource utilization chart of the top operator's type (in the main list) is chosen. The earliest available timeslot is selected (which means the earliest time the machine is free) and the main list of unscheduled operators is scanned for an operator that can be scheduled in that timeslot. If found, the operator is assigned to that timeslot and its predecessor's and successor's mobilities are adjusted to preserve data precedence relation. If not found, then the resource utilization chart is referred to again and the next earliest available timeslot is chosen. The main list is scanned again for a suitable operator. This process is repeated until a selection is made.

The resource utilization chart has to be 'filled' carefully in order to optimize hardware usage. For example in Figure 1, if only two hardware units are required, the sequence A, F, B, G, C, H, D, I, E and J (double-layered selection method) is found to be suitable for single-cycle operators. For two-cycle operators, the most suitable sequence is A, B, C, D, E, F, G, H, I and J (single-layered selection method). Since there could be more than one resource utilization chart to fill and only one chart is updated at any one time, there is a need for co-ordination among them. This is taken care of by ordering all the unassigned operators in the entire dataflow graph into one main list. The next resource utilization chart to be updated is chosen from this list. In this sense it is global in nature. Only when the chart has been chosen then only will the operators within the group of similar type be considered. Scheduling is complete when all the unassigned operators are 'entered' into their respective resource utilization charts.

The approximate worst case time complexity of the algorithm can be shown to be in the order of $4n^3 + 2n^2 - 2n$ where n is the total number of operators in the dataflow graph[3].

6 Experimental Results

Table 1 shows results of the benchmark example of fifth-order digital elliptical wave filter from Kung et al.[11]. In this example, 17 is the least cycle time that can be obtained from Critical Path Analysis at the expense of 3 adders and 3 multipliers. ALPS is able to obtain optimal results from their Linear Programming model. In order to reduce the hardware costs, cycle time is relaxed to 18 and then 21 cycles. NTUSS shows the least increase in the scheduling time taken.

SYSTEM	CYCLES	(+)	(*)	TIME	% CHANGE IN TIME
FDS[5] Xerox 1108 Lisp Machine	17	3	3	1 min	100 %
	18	3	2	3 min	300 %
	19	2	2	7 min	700 %
	21	2	1	13 min	1300 %
ALPS[7] Vax-11/8800	17	3	3	0.26 secs	100 %
	18	2	2	3.1 secs	1192 %
	21	2	1	34.5 secs	13269 %
[6] SUN4/280	17	3	3	0.067 secs	100 %
	18	2	2	0.101 sec	150 %
	21	2	1	0.783 sec	1169 %
NTUSS SUN3/60	17	3	3	1.25 secs	100 %
	18	2	2	1.30 secs	104 %
	21	2	1	1.42 secs	114 %

Table 1: Fifth Order Digital Elliptical Wave Filter[11]

7 Conclusions

A heuristic based scheduling algorithm is presented. Our algorithm differs from others. For example in MAHA[4], an operator is chosen based on its degree of freedom and then only a timeslot is chosen for this operator based on some other criteria. Our algorithm on the other hand, chooses a timeslot (which minimizes the idle time of the machine) first and then picks the most suitable operator from a sorted list. Some other heuristics like Force -Directed Scheduling[5] and the one described in [6], the most suitable (operator,c-step) pair is chosen based on some figure of merit. The decision making process is based on selecting the best figure among a group of computed values. Our algorithm minimizes numerical computation by making decisions based on direct usage of hardware resources in the resource utilization chart and storage device/multiplexer cost considerations (although storage device/ multiplexer costs considerations are described here). Using the scheme just described, NTUSS is capable of scheduling operators in almost linear time when their mobilities are increased.

There are also some limitations of the algorithm. For example it can handle static scheduling problems only. Currently, we are also not able to handle loops and allocation to pipelined datapaths. For the example presented we were able to obtain optimal results but like all heuristics, the same cannot be guaranteed for all cases.

We are able to incorporate storage device and multiplexer costs considerations into the model presented. Other real-world extensions like operator-chaining, multi-cycle and mutually-exclusive operations can also be handled.

Reference

1. M. C. McFarland, A. C. Parker and R. Camposano 'High Level Synthesis Of Digital Systems', Proc. of IEEE, vol 78, no.2 , February 1990.

2. B. M. Pangrle and D. D. Gajski 'Design Tools For Intelligent Silicon Compilation', IEEE Trans. Computer Aided Design pages 1098-1112, November 1987.

3. Gulam Mohamed 'A Heuristic-Based Scheduling Algorithm For High Level Synthesis Of Digital Systems', Master's Thesis, Nanyang Technological University (in preparation).

4. A. C. Parker, Jorge "T" Pizarro and Mitch Mlinar 'MAHA : A Program For Datapath Synthesis', Proc. 23rd Design Automation Conference, pages 461-466, July 1986.

5. P. G. Paulin and J. P. Knight 'Force-Directed Scheduling For Behavioral Synthesis Of Asics', IEEE Trans. Computer Aided Design, vol 8 pages 661-679, June 1989.

6. P. In-Cheol and K. Chong-Min 'Fast And Near Optimal Scheduling in Automatic Data Path Synthesis', Proc. 28th Design Automation Conference pages 680-685, June 1991.

7. Hwang C. T., Lee J. H. and Hsu Y. C. 'A Formal Approach To Scheduling Problem In High Level Synthesis', IEEE Trans. Computer Aided Design, pages 464-475, April 1991.

8. E. G. Coffman Jr. et al., **Computer And Job-Shop Scheduling Theory**, John Wiley & Sons, New York, 1976.

9. D. E. Thomas et al. , **Algorithmic And Register-Transfer Level Synthesis: The System Architect's Workbench**, Kluwer Academic Publishers, 1990.

10. K. G. Lockyer, **An Introduction To Critical Path Analysis**, The Pitman Press, Bath, 1969.

11. S. Y. Kung, H. J. Whitehouse and T. Kailath, **VLSI And Modern Signal Processing**, Englewood Cliffs, NJ, Prentice-Hall, pages 258-264, 1985.

Evaluation of Floating-Point Sum or Difference of Products in Carry-Save Domain

A. Wahab, S. Erdogan and A. B. Premkumar
School of Applied Sciences
Division of Computer Technology
Nanyang Technological University
Nanyang Avenue
Singapore 2263
ASABDUL@NTUVAX.BITNET, (65) 7994948

Abstract - **An architecture to evaluate a 24-bit floating-point sum or difference of products using modified sequential carry-save multipliers with extensive pipelining is described. The basic building block of the architecture is a carry-save multiplier with built-in mantissa alignment for the summation during the multiplication cycles. A carry-save adder, capable of mantissa alignment, correctly positions products with the current carry-save sum. Carry propagation in individual multipliers is avoided and is only required once to produce the final result.**

1 Introduction

In evaluating sum of products, the mantissas of the current sum-of-products and the newly examined product are compared. The mantissa of the one with the smaller exponent is aligned first and then added to the mantissa of the other one. One alternative is to use fast shifting units for mantissa alignment. However the cost of such units is prohibitive and may well slow down the overall speed of the system if pipelining has to be used. The technique used in the proposed architecture is to incorporate shifting capability into the multiply/add unit, thus eliminating use of conventional shifters. Sequential carry-save multipliers are used for evaluating the product of the mantissas involved in the floating-point operation [1]. For an n by n multiplication, the result is available in carry-save form after n cycles. Conventional result can be obtained through n additional cycles or a fast carry propagation unit. Since the sum of products evaluation is the main concern in the proposed architecture, carry propagation is deferred until the final stage. Thus, carry propagation is avoided in individual product evaluations. A high throughput is achieved by incorporating the mantissa alignment into the product evaluation cycles.

2 Floating-Point Sum-of-Products Evaluation in Carry-Save Form

The summation of products involves the multiplication of multiplier and multiplicand pairs and subsequent addition of these results. In the floating-point domain, the multiplication itself involves addition of the exponents of operands and multiplication of the mantissas.

Carry-save techniques have been proposed to reduce the timing penalty associated with the propagation of carry during the mantissa multiplication operation.

The multiplication of two floating-point numbers in carry-save domain is achieved by performing a carry- save multiplication of their mantissas and adding their exponents. The exponent part of the product is obtained by the addition of the exponents. The mantissa of the product is the most significant half of the result in carry and sum registers. For a conventional 16-bit mantissa carry-save multiplier, the result is obtained in 16 clock cycles. Additional 16 clock cycles are necessary to produce a conventional result. A fast carry-propagation circuit could also be used for this purpose. However, if the multiplier is to be used in evaluating sum-of-products, the propagation of carry can be deferred till the final stage. If the products in carry-save form are to be used for the evaluation of sum-of-products, carry-save results are satisfactory and timing penalty due to propagation of carry only occurs once as opposed to m times for a sum-of-products with m terms.

The results obtained from two multipliers can be added to produce a carry-save result provided their results are of the same exponent (i.e., their mantissas are correctly aligned first). Further, addition of other products to the current sum is possible provided their mantissas are properly aligned. However, special care must be taken to ensure that the summation result does not overflow.

Let us consider the evaluation of F=A+BW, where A, B, F and W are 24-bit floating-point numbers (16-bit mantissa, excess 64 exponent). First the sum of the exponents of B and W must be obtained. If the exponent of BW is smaller than the one of A, the mantissa of the product should be adjusted prior to addition with A, otherwise the mantissa of A should be adjusted. The mantissa associated with the lowest exponent should be shifted right by the difference in the exponents and added to the other mantissa. In the worst case, a shift of the mantissa by 15 bit positions may be required. The resulting exponent is the higher of the two exponents. In carry-save representation, a correction must be applied, if a carry-out is produced from the most significant digit [2].

One alternative to achieving mantissa alignment is to provide a fast shifting unit. The cost of such a unit is prohibitive and may slow down the overall speed of the system if pipelining is to be used. A 16-bit shifter would require 16×16 connections and sixteen 16 to 1 multiplexers. Since shifting is performed at the end of the multiplication prior to summation a timing penalty occurs. The other alternative would be to incorporate bit shifting capabilities into the summation and multiplier units such that no dedicated shifter would be necessary to sum the products. Furthermore, if this could be achieved with little or no timing penalty during the standard carry-save cycles, a number of multiplier units could be run in parallel and very high summation rates could be obtained. To achieve massive pipelining, multipliers operate on different sets of data and their results are added through a dedicated carry-save adder.

Although the architectural details of an implementation are given in Section 4, the control requirements associated with individual multipliers and the summation unit to allow for massive pipelining are examined below:

1 When a new multiplier and a multiplicand are selected for multiplication, the product's exponent is determined by adding their exponents and is compared with the exponent

of the current sum-of-products.

2.a One in the current sum-of-products, then the multiplication is performed and resulting mantissa is added to the current sum-of-products after necessary mantissa alignment.

2.b If the exponent of the product is higher than the one in the current sum-of-products, then the mantissa of the current sum-of-products is adjusted to its correct position. Since the exponent of the product can be precomputed, the summation unit has 16 clock cycles to correctly align its mantissa with respect to the product while the multiplication is being performed.

The above requirements can be achieved by using a dedicated shifter unit and many multipliers. Pipelining maximizes the usage of the fast shifter. Another alternative is to incorporate mantissa alignment into the multiplication cycles. To achieve this objective, a concept called "global exponent" that is basically the exponent of the current sum-of-products, is proposed in this paper. Instead of recording the exponent of individual products currently being evaluated, their relative difference to the global exponent is recorded. Every multiplier unit and the summation unit have a counter. Since we are dealing with 16-bit mantissas, the maximum relative difference is 15. A unit is required to align its carry-save value one bit at a time as long as its counter is positive. The hardware realization of this alignment for the multiplier and summation is presented in Section 3.

Initially, the global exponent is the first product's exponent and is subsequently updated. The succeeding products are examined one by one, and the relative difference between their exponent and the global exponent are evaluated. The shifting requirement is tagged to the assigned multiplier unit to ensure correct mantissa alignment at the end of 16 clock cycles. For an exponent difference of 16 or above, the multiplier can be set to zero and no further intervention would be necessary. Some sophisticated buffering techniques can also be developed not to schedule this product evaluation at all. The multiplier could therefore operate on a set of operands leading to a significant result. We have considered such an optimization, however, it has not been discussed in the current paper.

Another point of concern is the overflow out of the summation unit. When an overflow is detected, all counters in the system and the global exponent are incremented by one to ensure consistency of mantissa alignment and global exponent. The overflow can be detected before it occurs by examining the most significant carry and sum bits. A correction action is initiated by incrementing all counters in the system and the global exponent.

3 Built-In Mantissa Alignment for Floating-Point Additions

In order to achieve the alignment requirement associated with the sum-of-products, two units are of special interest. The first, is a floating-point multiplier that incorporates alignment into the multiplication cycles and the second is a summation unit that allows for alignment. We first concentrate on the design of the former. The design of the latter is fairly trivial and is described later in this Section.

The conventional carry-save floating-point multiplier as described earlier generates a product in 16 clock cycles for a 16-bit mantissa. To incorporate mantissa alignment into the multiplication operation, this architecture is modified in order to produce a correctly aligned carry-save result. This result is ready for subsequent addition to the current sum-of-products without further manipulation. In the proposed sum-of-products architecture, the alignment requirement of a multiplier unit is determined by a counter. In a 24-bit floating-point environment (16-bit mantissa, excess 64 exponent), a 4-bit counter is necessary to record the shifting requirement. The value of the counter is set prior to the start of a multiplication and is subsequently modified to reflect the requirements of the system.

To achieve alignment required for subsequent summation, the multiplier could be shifted right as many positions as required prior to the actual multiplication cycles and then the multiplication could proceed with the remaining bits of the multiplier. Since fewer bits of the multiplier are examined (starting with the most significant ones) reduced precision may result. Furthermore, the sum-of-products evaluation approach as described in Section 2, requires a multiplier to shift its current product in the middle of a multiplication. This can not be handled simply by preshifting the multiplier prior to the start of the multiplication.

Figure 1 shows a carry-save floating-point multiplier with built-in mantissa alignment capability. The multiplication follows the conventional carry-save algorithm. However additional paths are created to achieve mantissa alignment while multiplication is being performed. Unlike the approach proposed above, the multiplier is examined exactly as in the conventional carry-save algorithm and the multiplication is performed accordingly. However, when shifting is required, the partial product is added to the current product and is fed one bit further right compared to the original implementation. This new configuration ensures correct alignment of the mantissa as specified at the end of the multiplication cycles. Further, to conserve the integrity of the multiplication process, the multiplicand is also shifted by one bit right when a shift is required.

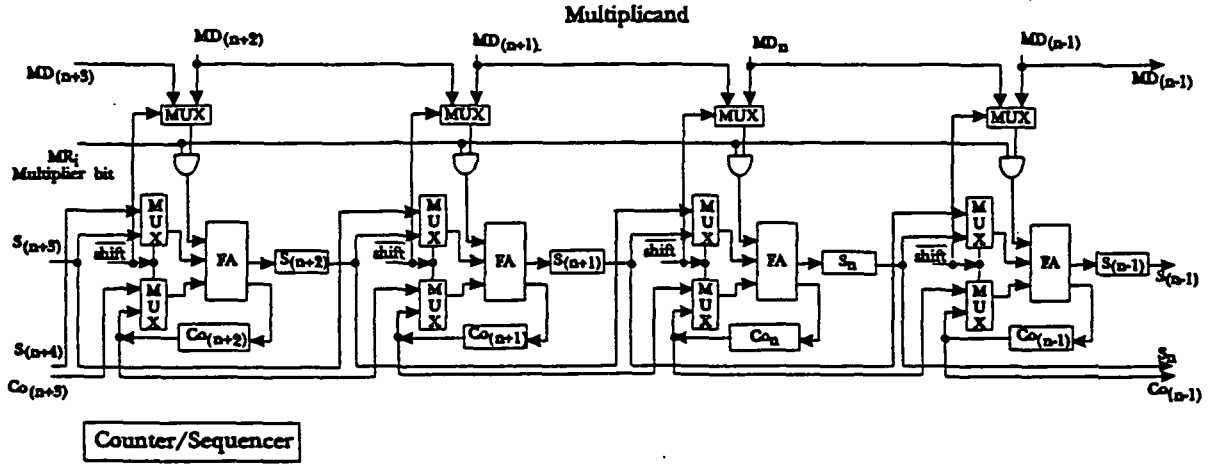


Figure 1: Block diagram of a carry-save multiplier with built-in mantissa alignment capability

Figure 2 shows a carry-save adder for adding the mantissa of products with built-in mantissa alignment capability. N is the number to be added to the current carry-save quantity.

When alignment is not needed, the unit adds an ordinary number to an existing carry-save number. When alignment is required, the carry-save result is fed one unit right compared to the original configuration. This is similar to the approach used in the implementation of the multiplier with built-in mantissa alignment. Two clock cycles are necessary to add the carry and sum components of a recently evaluated product.

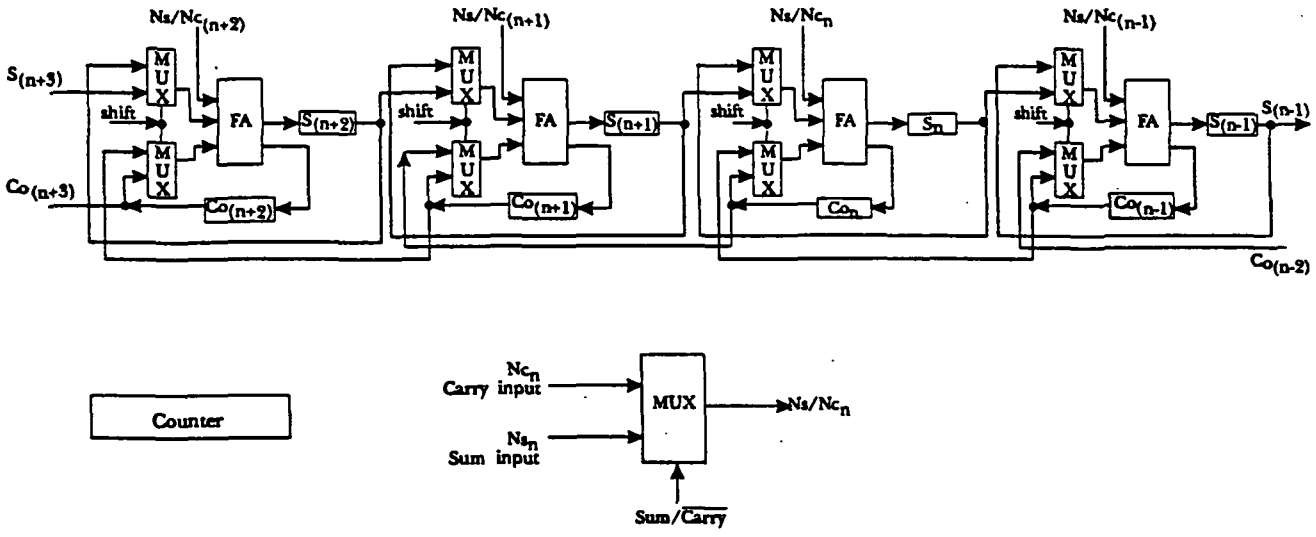


Figure 2: Block diagram of a carry-save adder for summing of products with built-in mantissa alignment capability

4 Architecture for Sum-of-Products Evaluation

The architecture presented in this paper is a novel approach to explore carry-save representation for sum- of-products evaluation as described in Section 2. The two components that are of special concern in this architecture are: a floating-point carry-save multiplier that can align the mantissa of a product during regular multiplication cycles (ready for subsequent summation) and a special floating-point adder with a mantissa alignment capability to sustain continuous addition of products in carry-save domain.

Figure 3 shows the architecture of the sum-of-products module. The module is composed of 8 floating-point multipliers with built-in mantissa alignment capability. The counters associated with individual multiplier units are counter/adders to store the shifting requirements as described in Section 3. Delta exponent is obtained by subtracting the exponents of the product currently examined from the global exponent. When delta exponent is positive (the exponent of a product to be evaluated is less than the global exponent), then the counter of the multiplier is set to reflect this difference. Otherwise, all counters in the system except the one of multiplier unit assigned to the current product evaluation are incremented by that difference.

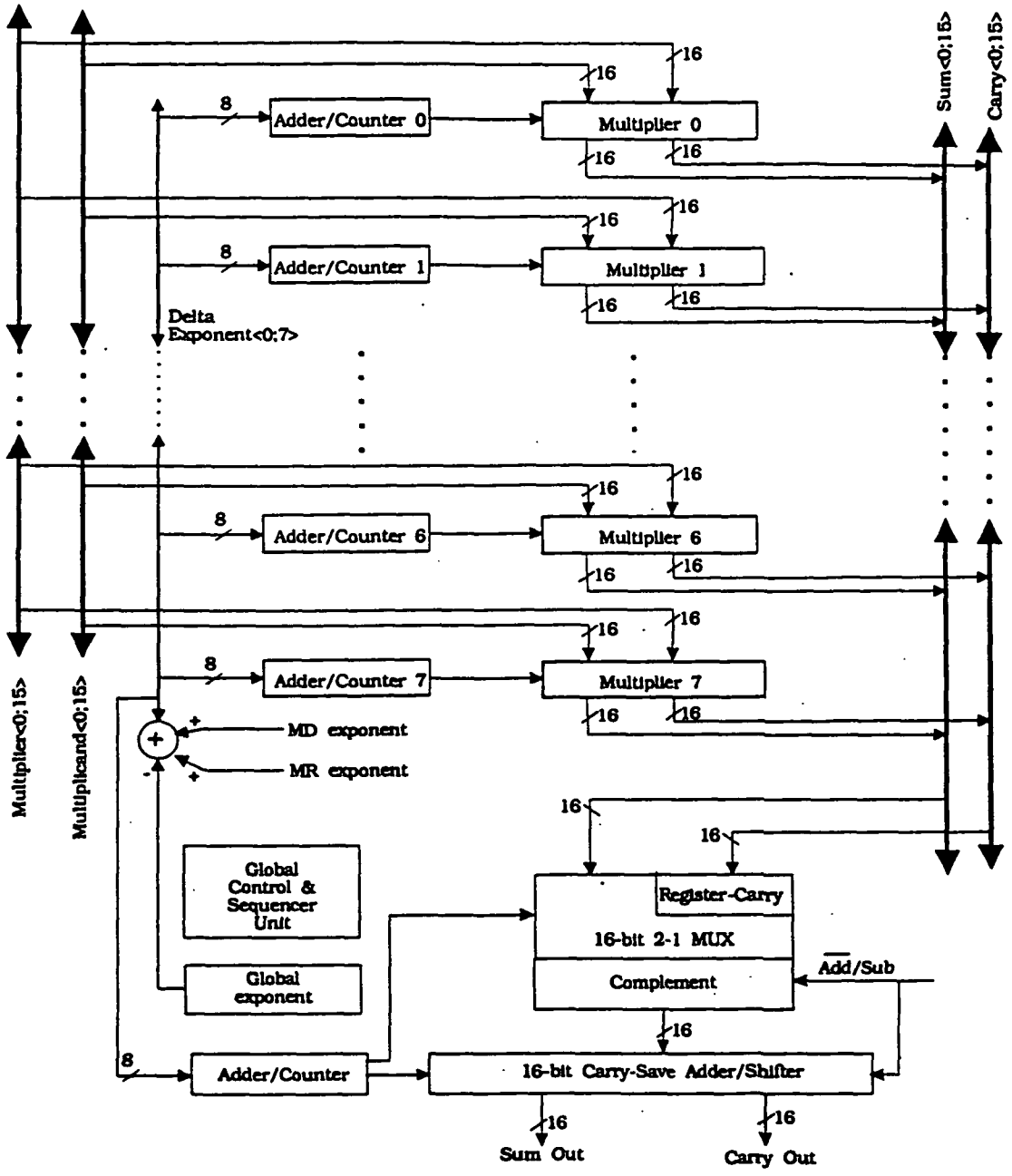


Figure 3: Architecture of a 24-bit floating point (16-bit mantissa, 8-bit exponent) Sum of Products system.

Figure 4 shows the timing characteristics of the architecture. The multiplicand and the multiplier pairs are fed to each multiplier with a phase difference of two clock cycles to ensure that their results can be summed without unnecessary buffering. Depending on the implementation platform, the exponent of the product may need to be pre computed since only two cycles are available for implementing the counter settings in the system. A multiplication result is output to the bus every two clock cycle. Each multiplier requires 16 clock cycles to produce a result.

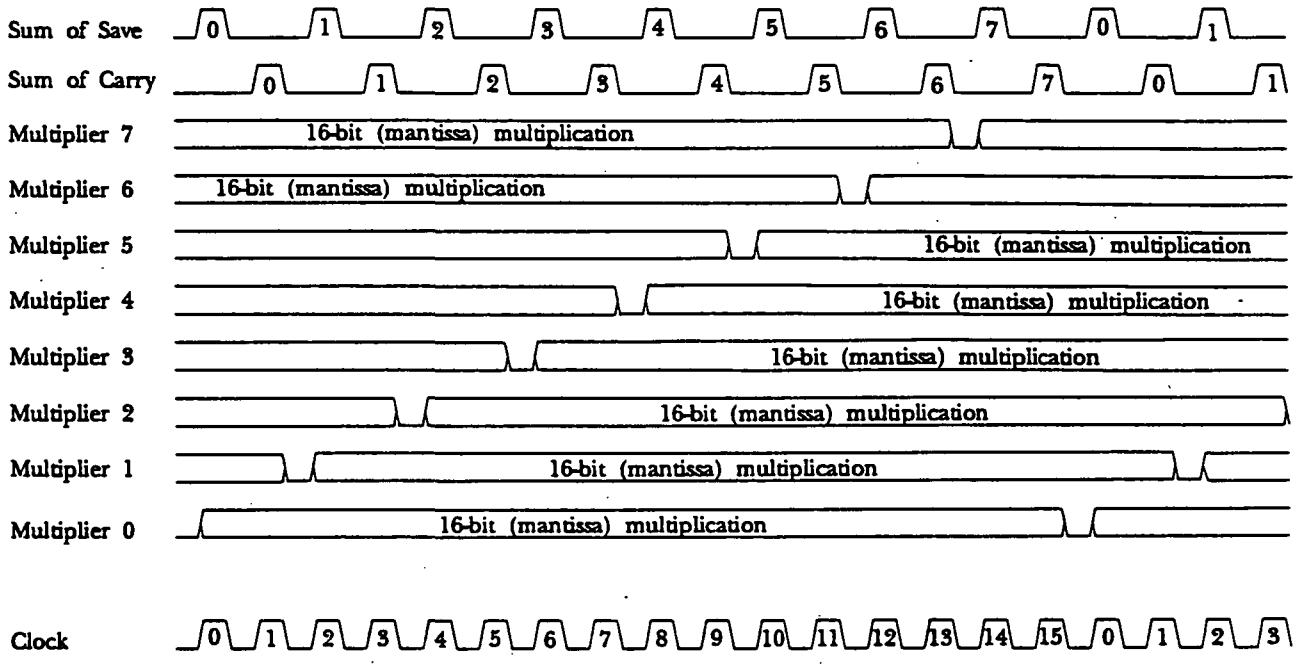


Figure 4: Timing diagram of the pipelined Carry-save Sum of Products architecture

The results obtained through the multipliers are fed through a wide bus to a summation circuit. The 32-bit bus allows for the transfer of a result in one clock cycle to achieve the desired 16-cycle pipelining. The sum component is fed directly to the summation circuit and shifted while being added as required by the counter associated with it. The carry component of the mantissa is moved to a shift register that ensures that its alignment is correct by following the control applied on the summation circuit. The numbers associated with the clock cycles in the summation unit indicate which multiplier is responsible for a particular result.

The carry-save adder shown in Figure 2 is used to accumulate the sum-of-products in carry-save form as required. A radix-2 carry-save configuration is sufficient since a new product is generated every two cycle. When an overflow out of the summation unit is detected, all counters and the global exponent are incremented by one to conserve floating-point representation. Summation of negative products is also supported. Since products are evaluated using positive mantissas, negative products are summed by using one's complement of their mantissas to avoid carry propagation associated with two's complement. Two's complement is later achieved in the summation unit by forcing a "one" to the least significant

carry-in bit of the carry-save adder.

The final result in conventional form is obtained when there are no more products to be summed by propagating the carry. This is achieved by feeding the carry and sum components of the summation through an additional fast carry propagate circuit. The same result can be obtained by allowing the carry to propagate through the summation unit. The 16-bit sum component becomes the conventional result when carry is allowed to propagate for an additional 16 clock cycles. The global exponent is the exponent of the final result and is directly available from the circuit.

The architecture presented in Figure 3 uses 8 multipliers to achieve a throughput of one product summation every two clock cycle. A doubling of this performance is possible by using a radix-4 carry-save adder to accumulate summation results. Sixteen carry- save multipliers or 8 radix-4 multipliers could be used to sum a product every clock cycle in conjunction with this wide adder [3]. Dual prefetching of operands may be necessary to allow for the examination of exponent of a product to meet the timing requirements. Time/hardware tradeoff should be considered for a given implementation environment for maximum efficiency [4].

5 Conclusions

The proposed architecture achieves effective evaluation of floating-point sum-of-products in carry-save domain. A product summation every two clock cycle is performed at steady state at moderate cost by pipelining 8 floating-point carry-save multipliers. Better performance can be achieved by performing 2 bit decode of the multiplier but at the cost of increased complexity of the control. The performance of the proposed architecture may be further increased by detecting and aborting multipliers that will produce insignificant results with respect to the current summation value. Further performance increase is also possible by detecting and not scheduling those products with very low exponents with respect to the global exponent at a very early phase. The merits of such a system will depend on data and should be considered in applications such as artificial neural network simulation hardware [5].

The multiplier with built-in mantissa alignment capability and the special carry-save adder described in this paper can also be used in other environment that requires high performance computing [6]. The carry-save result of the multiplier is useful for applications such as Fast Fourier Transform and complex function evaluations where further computations are necessary prior to the output [7,8]. This concept is similar to the residue-number-system concept where an intermediate form of a number is generated to perform arithmetic operations more effectively [9]. However, we believe that carry-save representation offers a better alternative in sum-of-products evaluation, since the carry-save form is obtained as an intermediate form in the carry-save multiplication and does not require a costly initial conversion [10]. The final conventional result can also be obtained without additional hardware.

References

[1] J. J. F. Cavanagh, Digital Computer Arithmetic, Design and Implementation. McGraw-Hill, 1984.

[2] M. R. Santoro, G. Bewick, M. A. Horowitz, "Rounding Algorithms for IEEE Multipliers", in Proc. IEEE 9th Symp. on Computer Arith., pp. 176-183, 1989.

[3] S. S. Erdogan and Abdul Wahab, "Design and Implementation of a Radix-4 Carry-save Multiplier", submitted to IEEE Trans. on Comp. (December 1991).

[4] A. G. Ferreira, "A Parallel Time/Hardware Tradeoff for the Knapsack Problem", IEEE Trans. Comput., Vol. 40, No. 2, pp. 221-225, February 1991.

[5] "Design of RM-nc: A Reconfigurable Neurocomputer for Massively Parallel-Pipelined Computations", Proc. of the IEEE International Joint Conference on Neural Networks '92 Baltimore, U.S.A., Vol. 2, pp. 33-38, 7 - 11 June, 1992.

[6] "Floating-point Fast Fourier Transform Evaluation in a Parallel-Pipelined Carry-Save Architecture", accepted for publication in the proceedings of the ICARCV '92, Singapore, 15-18 September, 1992.

[7] P. Kornerup, D. W. Matula, "Exploiting Redundancy in Bit-Pipelined Rational Arithmetic", in Proc. IEEE 9th Symp. on Computer Arith., pp. 119-126, 1989.

[8] R. H. Brackert, M. D. Ercegovac, and A N Wilson, "Design of an On-line Multiply-Add Module for Recursive Digital Filters", in Proc. IEEE 9th Symp. on Computer Arith., pp. 34-41, 1989.

[9] F. J. Taylor. "Residue Arithmetic: A tutorial with Examples", IEEE Comp. Magazine, pp. 50-62, May 1984.

[10] K. M. Ibrahim and S. N. Saloum, "An efficient Residue to Binary Converter Design", IEEE Trans. Circuits Syst., Vol. 35, pp. 1441-1444, November 1988.

A Statistical-based Scheduling Algorithm in Automated Data Path Synthesis

Byung Wook Jeon and Chidchanok Lursinsap
The Center for Advanced Computer Studies of
The University of Southwestern Louisiana
Lafayette, LA 70504
Email: bwj@cacs.usl.edu and lur@cacs.usl.ed

Abstract - **In this paper, we propose a new heuristic scheduling algorithm based on the *statistical analysis* of the *cumulative frequency distribution* of operations among control steps. It has a tendency of escaping from local minima and therefore reaching a globally optimal solution. The presented algorithm considers the real world constraints such as chained operations, multicycle operations, and pipelined data paths. The result of the experiment shows that it gives optimal solutions, even though it is greedy in nature.**

1 Introduction

The high level synthesis task is to transform an abstract behavioral specification of a digital system into a register transfer level (RTL) structure that realizes the given behavior. This task can be decomposed into a number of distinct but not independent subtasks [?]. The first one is to specify the behavior of a digital system using a programming language or a hardware description language (HDL), and then to translate the description into a graph-based representation which is said to be control and data flow graph (CDFG). This subtask is followed by operation scheduling process that assigns each operation to a control step. The third phase is the resource allocation process that assigns the operations and values to hardware. Among these subtasks, operation scheduling and resource allocation stages are the main processes which are closely interrelated. The operation scheduling usually determines the major design decisions such as the number and types of functional units, clock cycle time, lifetimes of variables, and implementation styles (pipeline, chained and multicycle operations, etc.). Therefore, a good scheduler is critical to an automated data path synthesis system [?, ?, ?, ?].

The simplest scheduling scheme is to schedule operations "as soon as possible" (ASAP) or "as late as possible" (ALAP) which is done in Emerald system [?]. This scheme may produce an unrealizable scheduling if there is resource limitation. To manage this problem, ASAP scheduling with postponement of operations is proposed in MIMOLA msystem [?] and Flamel system [?]. Another approach to scheduling is the list scheduling technique in which operations are sorted in topological order using the precedences specified in CDFG, and these operations are then scheduled into control steps using some heuristic priority function such as operation mobility used in SLICER [?]. Freedom-based scheduling used in MAHA [?] determines the critical path. The operations on the critical path are scheduled first and assigned to functional units. Then the other operations are scheduled and assigned one at a time. In force-directed scheduling [?], the operations are iteratively scheduled into control

steps based on the evaluation using distribution graph, which illustrates the distribution of fixed operations and unscheduled operations in each control steps.

All these schemes which are the constructive algorithms build up a schedule by adding operations one at a time until all operations have been scheduled. None of these constructive algorithms is guaranteed to find the best possible schedule. In ALPS [1], the scheduling problem is formulated as an integer linear programming (ILP) which gives a globally optimal solution. Since ILP is an exponential algorithm in nature, it is not acceptable for some large example even if an optimal solution can be generated. Practically, it is not always desirable to produce an optimal solution in the scheduling problem space because the problem itself is NP-complete. ¿From this observation, Park and Kyung [8] proposed an efficient algorithm based on the multiple exchange pair selection algorithm with cumulative gain which was proposed by Kernighan and Lin in their min-cut graph partitioning problem [2]. They devised the selection function used in their algorithm [2] by taking into account the fact that the density of functional units was the important factor for determining an operation to be moved, but no theoretical justification for it was provided. They also intentionally gave the preference to a certain operation in a particular control step.

The number of times that an operation occurs in each control step shows how the operations in the control and data flow graph *(CDFG)* are distributed among control steps. It also determines the *frequency distribution* of the operations and thus characterizes the feature of CDFG. From the basis of this statistical property, some measures can be derived which may be used for selecting a good candidate operation to be moved. The theoretical justification for these is quite simple but strong enough to devise a new selection function. Based on this observation, we propose a new scheduling algorithm using *statistical analysis* of the cumulative *frequency distribution* of operations among control steps. The presented algorithm considers the real world constraints such as chained and multicycle operations, and pipelined data paths. The essential method being used in our algorithm is also based on the multiple exchange pair selection algorithm [2].

We will start by describing the scheduling process in the next section. The scheduling process consists of two phase: one is the prephase for the scheduling phase and the other is the scheduling phase. The prephase transforms a given CDFG into the intermediate form containing information necessary for the scheduling phase. The scheduling phase selects a set of trials, and accepts even locally undesirable movements if they belong to a set of movements which maximizes the object function as a whole. This will be followed by the extention of algorithm to the real world constraints such as chained operations, multicycle operations, and pipelined data paths. Then, the experimental results for the examples used in several other systems will be given. Finally, concluding remarks will be made in the last section.

2 Scheduling Process

The presented scheduling algorithm takes into account the *frequency distribution* of operations occurred in each control step. The essential method being used is a modified version of the algorithm in [2]. This algorithm selects a set of trials, and accepts even locally undesirable movements if they belong to a set of movements which maximize the object function

as a whole. Even though this algorithm may prone to get stuck in a local minimum rather than finding the global optimum, it has a *hill climbing* property that can escape from local minima and therefore reach a globally optimal solution. In the practical point of view, it is not always necessary to produce an optimal solution as long as a near optimal one can be obtained.

The scheduling process consists of two phases; one is a prephase for the scheduling phase and the other a scheduling phase. We will now describe the prephase that transforms a given *CDFG* into the intermediate form containing information necessary for the scheduling phase. Then, the selection function which is the kernel of our scheduling phase is derived. This will be followed by description of the scheduling algorithm which improves the solution iteratively. To illustrate our scheme, we will use the example given in [11]. Figure 1 illustrates the *CDFG* of this example.

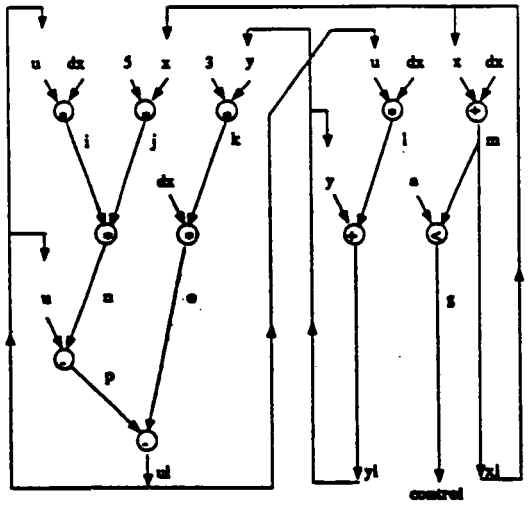


Figure 1: The data flow graph for the example

2.1 Prephase for Scheduling

This phase transforms a given *CDFG* into a constraint graph *(CG)* to identify the precedence relationships that are due to both data flow and control flow dependencies, and the timing behavior of each operation. Each node corresponds to an operation and has weight which specifies the minimal number of cycles required to execute the corresponding operation. Two nodes in *CG* are connected by a directed edge if and only if there is a precedence relation between two operations. The weight of an edge indicates the number of control steps between two operations. This transformed graph will be the input to the scheduling phase. Similar descriptions of *CG* used in our system can be found in the literature [3, 8].

2.2 Selection Function

The number of times that an i-type operation occurs in each control step illustrates how the operations are distributed among control steps in a given *CDFG* and therefore characterizes the feature of that *CDFG* . This implies that if one operation in control step j is moved

to another step k, the *CDFG* may have different properties after this movement. That is, the distribution of operations in a given *CDFG* can be changed after the movement, which implies that the concurrency of operations may be affected. We may also compute measures such as the average number of operations for each control step and the spread of operations among control steps. For the purpose of our scheduling process, we primarily consider the two most important of such quantities, which are the *mean* and the *variance* of the number of operations, to describe the balanced distribution of operations. Based on these measures, we will explore how to balance the concurrency of the operations among control steps in a given *CDFG* .

Roughly speaking, the average number is a measure of the minimum number of operations needed to uniformly distribute them among control steps and the variance measures the spread or dispersion of operations in the corresponding control step. We first want to introduce a measure for the average number of operations. The most common such measure is the arithmetic mean. The *mean* of the number of operations of i-type operation is denoted by M_i and defined by the formula

$$M_i = \lceil \frac{N_i}{N} \rceil$$

where N_i denotes the total number of i-type operations and N the minimum number of control steps required to perform a given *CDFG*.

Now we want to introduce a measure for the spread or variation of the number of operations to distinguish between two *CDFGs*. We choose a quantity that measures the deviation from the mean M_i in each control step and then take square of such quantity to derive another measurement. This quantity of i-type operation in control step j can be defined by

$$V_i(j) = (n_i(j) - M_i)^2$$

where $n_i(j)$ is the number of i-type operations in control step j. By using this quantity, another measurement which is said to be the *variance* of the distribution of operations between two control steps can be introduced by

$$\begin{aligned} V_i(j,k) &= V_i(j) + V_i(k) \\ &= (n_i(j) - M_i)^2 + (n_i(k) - M_i)^2 \end{aligned}$$

where $V_i(j,k)$ denotes the *variance* of i-type operation at control steps j and k.

It can be seen that as the value of $V_i(j,k)$ approaches to zero, the number of operations at both control steps i and j tends to be balanced uniformly. That is, it measures the *degree* of the *balance* of operations between two control steps. Then, the *variance* of the distribution after the movement of an operation O_{i_p} from step j to k can be also defined by

$$\begin{aligned} V_i'(j,k) &= V_i'(j) + V_i'(k) \\ &= (n_i(j) - 1 - M_i)^2 + (n_i(k) + 1 - M_i)^2 \end{aligned}$$

where $V_i'(j,k)$ denotes the *variance* of i-type operation at control steps j and k after the movement.

From the above results, we can derive another measurement, the *Change of Variance (CV)*, which is the difference between variances before and after the movement. The *CV* of

an operation is defined by the formula

$$C_{i_p}(j,k) = V_i(j,k) - V_i'(j,k)$$

where $C_{i_p}(j,k)$ denotes the value of the CV when p-operation of i-type operations is moved from control step j to k.

We can observe that if the value of this function is greater than zero, it is preferable to move O_{i_p} from step j to k. The negative value indicates that the movement is not desirable. Clearly, from the values of this function, we can make a decision whether or not the movement of an operation is preferable. That is, the balance of the concurrency of operations can be achieved by decreasing the number of operations in the control step where the value of CV is maximal. This provides the *selection function* used in our scheduling algorithm to choose a good candidate operation to be moved.

2.3 Scheduling Algorithm

The goal of our scheduling algorithm is to reduce the number of functional units required but not to lengthen the total execution time. This can be achieved by balancing the concurrency of the operations assigned to the functional units, which implies the high utilization of the units and in turn minimizes the number of units required. The balance of concurrency can be specified by the degree of the distribution of operations appeared in each control step. Therefore, we can accomplish this by distributing those as uniformly as possible without violating any constraints.

We now describe our scheduling algorithm on the basis of this observation. For the simplicity, we temporarily assume that each operation executes in one control step and all operations are either arithmetic or logic operations only. The objective function is the cost function whose arguments are each type of functional units and the number of those in each control step. The *ASAP* and *ALAP* schedules are shown in Figure 2 and Figure 3, respectively. Figure 4 depicts the final scheduled graph for the example given in Figure 1.

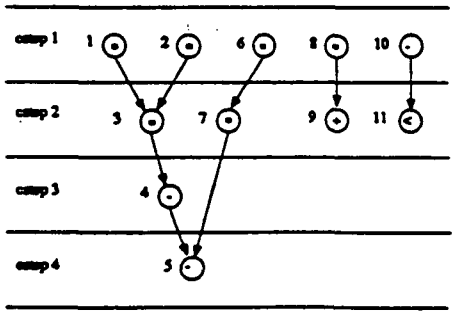


Figure 2: The ASAP schedule for the example

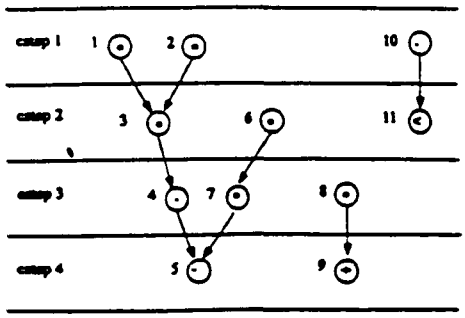


Figure 3: The ALAP schedule for the example

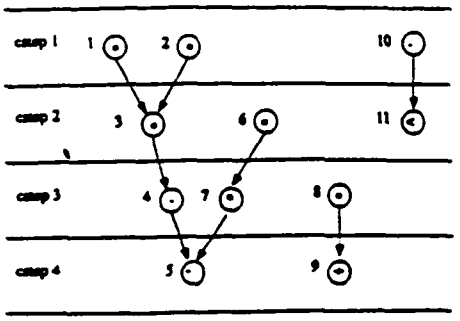


Figure 4: The scheduled data flow graph for the example

STEP 1 Determine the critical path and compute the mobility of each operation using both *ASAP* and *ALAP* schedules. In the following steps, the operations in the critical path are not considered because each of them can not be moved to the other control step. We consider either an *ASAP* or an *ALAP* schedule as an initial feasible solution.

STEP 2 Set counter to zero. Find the movable operations from the feasible solution determined in the previous iteration. If there is no movable operation, then go to *STEP5*. Otherwise, the following steps are performed.

STEP 3 Select the control step k for an operation O_{i_p} such that the selection function $C_{i_p}(j,k)$ has the maximum value and then move O_{i_p} to control step k. O_{i_p} denotes an i-type operation which is numbered as p in *CDFG*. Then, that operation is locked at the control step k temporarily.

STEP 4 Compute the gain which is the change of the value of the objective cost function when the operation O_{i_p} is moved from control step j to k. Each of these gains computed in this step is stored somewhere for the later use. Increment counter and go to *STEP2*.

STEP 5 Find a sequence of operations such that its cumulative gain is maximum among those sequences generated up to now. If there is no improvement for the objective cost function, the scheduling process is terminated. Otherwise, the current feasible solution determined during the previous iteration is modified and go to *STEP2*.

We will now illustrate how the algorithm works using the example shown in Figure 1. For the simplicity, it will be temporarily assumed that the available functional units are multipliers and ALUs, and that multiplier cost is four and ALU cost is one. We also assume that ALU is capable of performing addition, subtraction, and comparison. Let us consider the ASAP schedule depicted in Figure 2 which is chosen to be an initial feasible solution. It shows that the movable operations are O_{mul_7}, O_{alu_9}, and $O_{alu_{11}}$. We then compute $C_{i_p}(j,k)$ for each of these operations as described before. For example, if the operation O_{mul_7} is moved from control step 2 to 3 in Figure 2, then we can compute the selection function as follows:

$$\begin{aligned} M_{mul} &= \lceil \tfrac{6}{4} \rceil \\ &= 2 \\ C_{mul_7}(2,3) &= \{(2-2)^2 + (0-2)^2\} - \{(1-2)^2 + (1-2)^2\} \\ &= 2 \end{aligned}$$

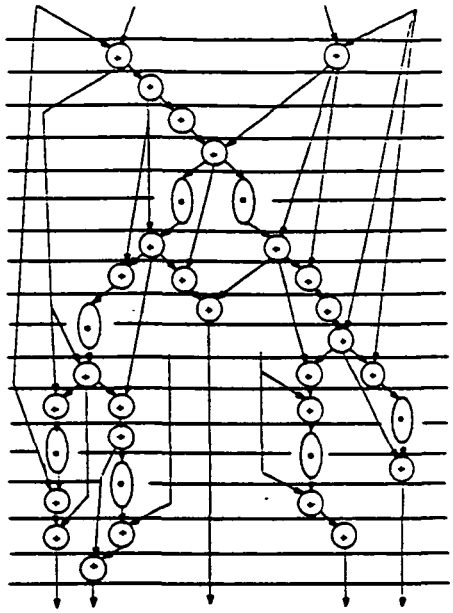


Figure 5: Fifth-order wave digital eliptical filter example

5 Conclusion

We have presented a new heuristic scheduling algorithm based on multiple exchange pair selection algorithm using the statistical analysis on the cumulative frequency distribution of the number of operation among control steps. The presented algorithm considers the real world constraints such as chained and multicycle operations, and pipelined data paths. The theoretical justification of this statistical method used in our selection function is simple but is strong enough to choose a good candidate operation to be moved. The proposed algorithm has a *hill climbing* property that can escape from local minima and reach a globally optimal solution, even though it is greedy in nature and therefore may prone to get stuck in a local minimum rather than finding the global optimum.

The experimental result shows that this algorithm can generate the optimal solutions for the examples used in the literatures [4, 10, 11].

References

[1] Cheng-Tsung Hwang, Jiah-Hurng Lee, and Yu-Chin Hsu. A Formal Approach to the Scheduling Problem in High Level Synthesis. *IEEE Tran. Computer-Aided Design*, 10(4):464–475, April 1991.

[2] B. W. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *Bell Syst.Tech. J.*, 49(2):291–308, 1970.

[3] David C. Ku and Giovanni De Micheli. Constrained Resouce Sharing and Conflict Resolution in Hebe. In *Integration*, volume 12, pages 131–165. Elsevier, December 1991.

[4] S. Y. Kung, H. J. Whitehouse, and T. Kailath. *VLSI Modern Signal Processing.* Prentice Hall, 1985.

[5] P. Marwedel. A New Synthesis Algorithm for MIMOLA Software System. In *Proceedings of the 23th IEEE/ACM Design Automaton Conference*, pages 271–277, July 1986.

[6] M. C. McFarland, S. J. Parker, and Raul Camposano. The High-Lvel Synthesis of Digital System. *Proceeding of IEEE*, 78(2):301–318, February 1990.

[7] B. M. Pangrle and D. D. Gajski. Slicer: A state synthesis for intelligent silicon compilation. In *Proceedings of the IEEE International Conference on Computer Design*, pages 42–5, October 1987.

[8] In-Cheol Park and Chong-Min Kyung. Fast and Near Optimal Scheduling in Automatic Data Path Synthesis. In *Proceedings of the 28th IEEE/ACM Design Automaton Conference*, pages 680–685, July 1991.

[9] N. Park and A. C. Parker. Sehwa: A Software Package for Synthesis of Pipelines from Behavioral Specifications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, CAD-7(3), March 1988.

[10] A. C. Parker, G. T. Pizarro, and M. Mlinar. MAHA: A Program for Datapath Synthesis. In *Proceedings of the 23th IEEE/ACM Design Automaton Conference*, pages 461–66, July 1986.

[11] P. G. Paulin and J. P. Knight. Force-directed Scheduling in Automatic Data Path Synthesis. In *Proceedings of the 24th IEEE/ACM Design Automaton Conference*, pages 195–202, July 1987.

[12] P. G. Paulin and J. P. Knight. Force-Directed Scheduling for the Behavioral Synthesis of ASIC's. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, CAD-8(6):661–79, June 1989.

[13] H. Trickey. Flamel: A high-level hardware compiler. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, CAD-6(2):259–69, March 1987.

[14] C. J. Tseng and D. P. Siewiorek. Automated Synthesis of Data Paths in Digital Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, CAD-5(3):379–95, July 1986.

Micro-rollback and Self-recovery Synthesis

Byung Wook Jeon and Chidchanok Lursinsap
The Center for Advanced Computer Studies of
The University of Southwestern Louisiana
Lafayette, LA 70504
Email: bwj@cacs.usl.edu and lur@cacs.usl.ed

Abstract - **A new approach to high level synthesis with *micro-rollback* and *self-recovery* method is presented whose objective is to generate a *reliable* system from the behavioral descriptions. The approach is formulated as a *rollback point insertion* problem based on the *probability* function of being inserted rollback points after the given *CDFG* is scheduled. This function allows us not only to search the problem space efficiently but also to avoid the exhaustive search.**

1 Error Detection Scheme

When one of the components of system fails, it causes an *error,* i.e., it results in an erroneous internal system state that can lead to system failure unless some special action is taken by the system to recover or to reconstruct a valid internal state. In order to prevent system failure, such errors must be detected soon after they occur. Thus, error detection is a vital step in all recovery schemes and the system must include a mechanism for detecting the failure of its components. The key to developing an effective error detection scheme is to minimize the distance between error occurrence and detection in both space and time. Ideally, these distance can be reduced to zero so that as soon as an error occurs, i.e., a component produces incorrect results, the error is detected by all the other system components that are receiving this erroneous information. This *ideal* can be achieved if all the components in the system are *self-checking* so that in addition to their normal outputs they also indicate to the rest of the system whether these outputs are correct.

No component can be self-checking with respect to all possible combinations of hardware faults. Instead, for all likely faults, the component must either produce the correct output or produce the output that contains an error indication. A component that satisfies this requirement is said to be *fault secure* [1]. No component is fault secure with respect to all possible combinations. Since the component is not guaranteed to produce a noncode output[1] immediately following the occurrence of the first fault, several additional different faults may exist in a component without any indication to the rest of the system and may cause the destruction of the fault secure property of the component. In order to prevent this situation, a component must be *self-testing* [1] such that it is guaranteed to produce a noncode output, due to the occurrence of one or more faults, before additional faults can occur and lead to the failure of the self-checking mechanism. Thus, *self-checking* must be both *fault secure* and *self-testing* so that the reliable error detection can be provided. It is therefore assumed that the error detection scheme must consist of such self-checking components. We further assume that the data flowing in the data path are coded. The error detection scheme in our

[1]Output that contain an error indication

system can be achieved by applying such self-checking design and coding techniques in the literature [17].

2 Micro-rollback Scheme

In this section we present the rollback technique for multiple retries in the high level synthesis which is at the heart of our system. Some constraints such as the number of available registers, the maximum allowable recovery time, and the number of retries are given by the user. The retries of a sequence of microinstructions are performed on the basis of control steps. Although a large number of retries may not be necessary from the point of view of recovery time, the need for multiple retries arises due to the fact that a single retry may not guarantee the disappearance of the error. If an error is detected while a sequence of instructions is being processed, our rollback algorithm brings the processing back a few control steps to the state that had existed in the past before the error first occurred, thus returns the system to an *error free* state where the offending microinstructions can be retried. We mean the state of the system is the contents of all storage information between two consecutive control steps, such as program status word, register contents, program counter, and the instruction register.

In order to be able to perform such an operation, the system must store all such information necessary to undo the state changes that have occurred in the last few control steps. Thus, setting up the *rollback points* is necessitated at the control step boundaries. This implies that the scheduled *CDFG* is partitioned into several segments and each rollback point is introduced at the beginning of every segment. A rollback point consists of a set of registers which store the data flowing into that particular segment. This set of registers stores the data that is only external to that segment, and the contents of these registers must be stored until the next rollback point is located. The registers having the contents of the previous rollback point will be overwritten with those of the current one's after a new rollback point is located. While the execution of the microinstructions is continued, the remaining registers are used to contain information for them. When backing up the processing to the previous *error free state* is necessitated in response to an error signal, all the required information which is the contents of registers at the most recently located rollback point is readily available. Note that no memory access is required and in turn the time required to save and load the status across the control step boundaries can be saved. To illustrate our rollback scheme, we define the following terminologies:

- N : The minimum number of control steps required to perform a given *CDFG*.
- N_i : The total number of $i-$ type operations in a given *CDFG*.
- N_r : The number of retries which is specified by the user.
- N_{rp} : The number of rollback points inserted at the control step boundaries.
- r_{min} : The minimum number of registers required to execute a given *CDFG*.
- r_{avail} : The maximum number of available registers specified by the user.

- T_{max} : The maximum allowable recovery time for N_r retries given by the user.
- D_{max} : The maximum allowable recovery distance which is defined by $\lfloor T_{max}/N_r \rfloor$.
- T_i :The time needed to run the set of microinstructions from the last rollback point to C_i.
- T_{r_i} : The recovery time for N_r retries at any control step C_i (i.e., $N_r \cdot T_i$).

The value of T_{max} is also assumed to be greater than the maximum execution time among those of different type functional units. Furthermore, we assume the values of T_{max} and T_{r_i} are specified in terms of the number of control steps.

2.1 Insertion Function

In selecting a good candidate location where the rollback point can be set up, the most important factor to be considered is the probability of being inserted a rollback point at each control step boundary. Let n denotes the number of control steps required to perform the remaining part of a scheduled *CDFG*. Since T_{max} and N_r are given by the user, the maximum allowable recovery distance can be computed by $\lfloor T_{max}/N_r \rfloor$. Let the insertion function $P_1^{D_{max}}(n,i)$ be the probability that a rollback point is inserted between two successive control steps C_i and C_{i+1} , $1 \leq i < n$, in the *CDFG*. The subscript 1 indicates the fact that the rollback point can be inserted. In order to be able to compute this probability, we must find all possible patterns of rollback points being set up at every control step boundary. These patterns can be illustrated by the binary number sequences of length n. For example, a sequence 101... indicates the following facts:

- The rollback point was set up at the point before C_1 , i.e. , the initial state of the system is saved, and
- The rollback point was not inserted between C_1 and C_2 , and so on.

Note that the probability of being inserted a rollback point depends on the maximum allowable recovery time T_{max} , number of retries N_r , recovery time T_{r_i},, and the location of the previous rollback point. The initial state of the system must be stored before the processing begins to start. Thus, inserting an element into i -th position in the sequence must be satisfied the following conditions:

- 1 must be inserted if either $i = 0$ or there is 1 in $(i - D_{max})$-th position and there is no 1 in the range $[i - D_{max}, i - 1]$, where $D_{max} < i \leq n$.
- 0 can not be inserted if either $i = 0$ or there is 1 in $(i - D_{max})$-th position and there is no 1 in the range $[i - D_{max}, i - 1]$, where $D_{max} < i \leq n$.
- Otherwise, either an 1 or a 0 can be inserted

All possible patterns can be generated on the basis of these features. For example, we may generate all possible sequences for $D_{max} = 2$ and $n = 4$ as follows:

$sequence_i$	rp_0	rp_1	rp_2	rp_3
$sequence_1$	1	0	1	0
$sequence_2$	1	0	1	1
$sequence_3$	1	1	0	1
$sequence_4$	1	1	1	0
$sequence_5$	1	1	1	1

During the generation of such sequences, it can be easily seen that these sequences have the same characteristics as the *Fibonacci* sequence. Figure 5 illustrates how such sequences can be generated. In the following context, the first position (i.e., 0-th position) in the sequence is not considered due to the fact that it is always set to one. Thus, we can derive the following formula to compute $P_1^{D_{max}}(n,i)$. Given D_{max} and n, let $f^{D_{max}}(n)$ denotes the number of all possible sequences. Then, $f^{D_{max}}(n)$ are the D_{max}*-th order* Fibonacci numbers and defined by the rules

$$f^{D_{max}}(n) = \sum_{k=1}^{D_{max}} f^{D_{max}}(n-k), \text{ if } n > D_{max};$$

$$f^{D_{max}}(n) = 2^n - 1, \text{ if } n = D_{max};$$

$$f^{D_{max}}(n) = 2^n, \text{ if } 1 \le n \le D_{max} - 1.$$

That is, they are started with $2^1, 2^2, \cdots, 2^{D_{max}-1}$'s, then $2^{D_{max}} - 1$, and then each number is the sum of the previous D_{max} values. When $D_{max} = 2$, this is the usual Fibonacci sequence; for larger values of D_{max} the equation can be solved by using the generating function[7].

Let the functions $f_0^{D_{max}}(n,k)$ and $f_1^{D_{max}}(n,k)$ denotes the number of 0's and 1's on the k-th position respectively for some given integers n and D_{max}, where $1 \le k \le n$. Then, the function $f^{D_{max}}(n)$ can be redefined in terms of two functions $f_0^{D_{max}}(n,k)$ and $f_1^{D_{max}}(n,k)$ by

$$f^{D_{max}}(n) = f_0^{D_{max}}(n,k) + f_1^{D_{max}}(n,k).$$

These numbers also have the characteristics of the Fibonacci number. Thus, these functions can be defined by the similar way described previously. Due to the limited space, we will not describe the detailed derivations for these functions.

Note that the initial conditions of these recurrence functions are depending on the function $f^{D_{max}}(n)$. This provides an important fact that the number of 1's occurrences on i-th position in all possible patterns can be represented in term of the number of all possible sequences. Consequently, the probability $P_1^{D_{max}}(n,i)$ can be computed by the following rule:

$$P_1^{D_{max}}(n,i) = \frac{f_1^{D_{max}}(n-1,i-1)}{f^{D_{max}}(n-1)}$$

This is the definition of the insertion function used in our rollback scheme. The function $P_0^{D_{max}}(n,i)$ denotes the probability that a rollback point is not inserted at the location between two successive control steps C_i and C_{i+1}, $1 \le i < n$, in the $CDFG$. It can be defined in a similar fashion. However, since we only have interests in finding the location that the rollback point can be inserted among control steps, it is not further considered.

2.2 Rollback Point Insertion Algorithm

It is necessary to find the appropriate points in between every two consecutive control steps and set up the rollback points on those locations in order to support the rollback scheme. The following procedure processes a such task and is iterated until all necessary rollback points are inserted. A rollback point consists of a set of registers, i.e., *rollback registers,* that store the values received from previous segment and these registers can not be shared with others. The remaining registers can be used for storing the intermediate results of computations.

STEP 1 Construct the *live- variable* graph from the scheduled *CDFG.* The initial state of the system is stored before the first control step. That is, the external input values of *CDFG* are stored into *rollback registers.* Compute the maximum allowable recovery distance D_{max} for T_{max} and N_r. Set *current_check_point* to zero. This variable specifies the current possible rollback point between two consecutive control steps.

STEP 2 If the length of critical path minus *current_check_point* is less than or equal to one, the process of inserting the rollback points is terminated. Otherwise, the following *STEP 3 - 6* are repeated. That is, these steps are iterated until all the necessary rollback points are set up in a given scheduled *CDFG.*

STEP 3 Compute the *insertion function* for the remaining control steps in order to decide the possible rollback point. Select the earlier one of either the control step number whose insertion function has the largest value among those or the value of *current_check_point* plus maximum allowable recovery distance D_{max}. If there is a tie among the computed values of insertion functions for the corresponding control steps, the earliest point will be chosen.

STEP 4 Increase the value of *current_check_point* by the value selected in *STEP 3.* This value of *current_check_point* specifies the *most probable* point between two consecutive control steps where the rollback point can be inserted after the most recently located rollback point. Compute the number of registers required for each control step from the previous rollback point to *current_check_point* .

STEP 5 If the largest number of registers computed in *STEP 3* is greater than the given number of available registers r_{avail}, the value of *current_check_point* is subtracted by the control step number which has the largest number of registers and then repeat this step. Otherwise, go to *STEP 6.*

STEP 6 If the recovery time T_{r_i} is greater than T_{max} (i.e., $T_i > D_{max}$), decrement *current_check_point* by one and repeat this process. Otherwise, the rollback point is set up at *current_check_point.* Then, the *live-variable* graph is modified such that the life regions of variables received values from the previous segment are extended to the rollback point set up in this step and go to *STEP 2.*

The algorithm is illustrated for the scheduled *CDFG* shown in Figure 4 where $T_{max} = 2$, $N_r = 1$, $r_{min} = 4$, and $r_{avail} = 6$. Let us consider the initial live-variable graph for this

CDFG which is given in Figure 6. Let rp_i denotes the *current_check_point* variable in the algorithm. Since $N_r = 1$, $D_{max} = T_{max} = 2$ and $T_{r_i} = T_i$. The contents of variables y, u, x, which are the initial status of *CDFG* are saved on the point rp_0. We then compute the insertion function for each control step in the range [rp_1 ,$rp_0 + T_{max}$] as follows:

$$\text{For control step 1, } P_1^2(4,1) = \frac{f_1^2(3,1)}{f^2(3)} = \frac{3}{5}$$

$$\text{For control step 2, } P_1^2(4,2) = \frac{f_1^2(3,2)}{f^2(3)} = \frac{4}{5}$$

Thus, rp_2 is chosen to be the most probable point. From the live-variable graph shown in Figure 5, the number of registers required to execute the operations in the range [C_1, C_2] is 6 and $T_{r_2} = T_2 = 2 \leq D_{max} = 2$. Therefore, a rollback point is inserted in rp_2 without examining another possible location rp_1. The live-variable graph is then modified to reflect the fact that the saved variables can not be shared with the others in the range [C_1, C_2]. Similarly, next rollback point is inserted on rp_3 and the final live-variable graph is given in Figure 7.

3 Further Considerations

In this section, the basic rollback algorithm is extended to handle real world constraints such as mutually exclusive operations, chained operations, and multicycle operations.

3.1 Mutually Exclusive Operations

When two operations can never be both executed at the same time, they are said to be *mutually exclusive* and therefore can share hardware. The existence of the conditional statements such as *if-then-else* and *case* statements causes some operations to be mutually exclusive, since only one branch in a conditional block occurs at a time. Such operations can be scheduled into the same control step without increasing the number of functional units if they are performed on the same unit. In other words, they can be simply treated by taking them as a single operation. Similarly, two variables are *mutually exclusive* if they can never be both alive at the same time. They can also share a register for storage, even though their life times are overlapping. In order to handle such variables for the mutually exclusive operations, we must color the edges in the initial live-variable graph. Thus, the basic algorithm described in the last section can be used to handle mutually exclusive operations with coloring algorithm. The algorithm used for the coloring is a modified version of the node coloring algorithm in [10].

3.2 Chained Operations

As the duration of one control step is increased, more operations can be performed during one control step if there are enough components available to perform the operations. When more than one operation is performed sequentially within one state, the operations are said

to be *chained* together. Chaining allows higher utilization of functional units, but not the hardware sharing, by assigning them within the same control step. It can be also eliminated the need for registers to save data between two successive chained operations, because the data produced by the first one is consumed in the same state by the second. Thus, for the case of the chained operations, it is not necessary to modify the proposed rollback schemes.

3.3 Multicycle Operations

The possibility of scheduling operations that require more than one control step to execute has been also incorporated into the system. This feature can be implemented by taking into account the fact that multicycle operation can not be preempted to the completion of its execution. That is, once the multicycle operation is assigned, the functional unit cannot be shared by other operations until the operation is completed. In order to reflect this fact, we simply use the following strategy in the course of examining the possible location that a rollback point can be inserted;

- After the most probable location is decided, check the location if there is a multicycle operation.
- If so, *current_check_point* is decreased to the control step at which that multicycle operation begin to start.
- Otherwise, proceed the basic algorithm as usual. In other words, there is no effect on this scheme due to the above fact.

4 Experiment

We have implemented the proposed algorithm in *C* on a SUN SPARK station 2 system. The proposed system has been tested on several examples which were used in some systems described in the literature [9, 11, 12]. However, direct comparisons are not possible to make due to the following facts:

- There has been no published report for rollback and recovery system from the behavioral synthesis point of view which considers the real environment such as chained and multicycle operations.
- Since the requirement of our design is reliability of the system, it may involve either temporal or physical redundancy.

Thus, instead of giving the experimental results for several examples, we will classify the results for a popular example into several cases, and then intensively analyze the effect on the system. This will be followed by a brief discussion of another example borrowed from [9].

The first example is the differential equation which were described in [12]. Two types of overhead should be introduced due to our design requirement. One is *register* overhead and the other is *recovery time* overhead [13, 16]. They are defined by

- $register\_overhead = (r_{avail} - r_{min}/r_{avail}) \times 100$
- $recovery\_time\_overhead = (N_{rp}/N) \times 100$

We consider two cases in each table. In the first case we hold the maximum allowable recovery time D_{max} constant and compute the number of rollback points inserted for the various number of available registers r_{avail}. since $r_{min} \leq r_{avail}$, r_{avail} begins with r_{min}. The columns show the result of this case. In second case we fix r_{avail} and compute the number of rollback points required for the various D_{max}. Each row illustrates the results for this. The entries of each table indicate the number of rollback points required in this example.

Table 1 specifies the result for the case of which neither chained nor multicycle operation is allowed. After the scheduling process, $r_{min} = 5$. The assumptions for this case are as follows.

- one control step is equivalent to $100ns$
- delay time: ALU = $60ns$, multiplier =$80ns$

Table 1 shows that the optimum values for r_{avail} and D_{max} lie in the range of 20% – 60% of register overhead and 20% – 40% of recovery time overhead, respectively. The result of the experiment that has been taken into account the multicycle operations are summarized in the Table 2. In this case, $r_{min} = 6$. We also made the following assumptions:

- one control step is equivalent to $60ns$
- delay time: ALU = $40ns$, multiplier =$80ns$

r_{avail}	D_{max}						
	1	2	3	4	5	6	7
5	3	3	3	3	3	3	3
6	3	2	1	1	1	1	1
7	3	2	1	0	0	0	0
8	3	1	1	0	0	0	0
9	3	1	1	0	0	0	0
10	3	1	1	0	0	0	0
11	3	1	1	0	0	0	0

Table 1: The case for one cycle operations only.

r_{avail}	D_{max}					
	2	3	4	5	6	7
6	4	3	3	3	1	1
7	3	3	2	2	0	0
8	2	3	1	1	0	0
9	2	3	1	1	0	0
10	2	2	1	1	0	0
11	2	2	1	1	0	0

Table 2: The case for the multicycling operations.

For the case of multicycle operations, the suitable values of r_{avail} and D_{max} lie in 13% – 50% of register overhead and 30% – 50% of recovery time overhead, respectively.

Table 3 shows the result that the chained operations are considered. r_{min} is 6 as the result of scheduling process.

For this case, the following assumptions are made:

r_{avail}	D_{max}						
	1	2	3	4	5	6	7
6	2	1	1	1	1	1	1
7	2	1	0	0	0	0	0
8	2	1	0	0	0	0	0
9	2	1	0	0	0	0	0
10	2	1	0	0	0	0	0
11	2	1	0	0	0	0	0

Table 3: The case for the chained operations.

- one control step is equivalent to $100ns$
- delay time: ALU = $40ns$, multiplier =$80ns$

In general, it can be easily seen that the register overhead and recovery time overhead have conflicting requirements. In other words, as the recovery time overhead becomes greater we should insert more rollback point, and vice versa. Similarly, we can estimate those suitable values for each case by analyzing the corresponding tables. ¿From this analysis of the results, it can be observed that as the duration of one control step is increased, more register overhead can be expected. Conversely, we can expect more recovery time overhead as there are more multicycle operations. Another example shown in Figure 8 is the fifth-order wave digital elliptical filter borrowed from [9]. This example contains multicycle operations and the assumption we made in the previous example is also applied. This figure illustrates the final result of our algorithm when $r_{min} = 11$ for given $r_{avail} = 15 \leq 37\%$ and $T_{max} = 6$.

5 Conclusion

This paper presented a new approach to high level synthesis for the micro-rollback and self-recovery system whose objective is to generate a self-recoverable system from a given behavioral description. Ragahvendra and Lursinsap [13] proposed this new approach to high level synthesis. However, they did not consider the real world constraints such as chained and multicycle operations. As we described in the previous sections, such constraints have been incorporated into our system.

The probability function for the rollback point insertion allows us to avoid searching all the locations between two consecutive control step in *CDFG* in order to find the position that the rollback point can be inserted. From the result of the experiment, we have formulated a way of arriving at the reasonable values of the number of additional registers and the allowable recovery time in spite of conflicting requirements.

References

[1] D. A. Anderson and G. Metz. Design of Totally Self-checking Check Circuits from m-Out-of-n. *IEEE Trans. Computers*, C-22(3):263–269, March 1973.

[2] M. O. Ball and F. Hardie. Effects and Detection of Intermittent Failures in Digital Systems. Technical Report IBM 67-825-2137, IBM, 1967.

[3] M. Brodsky. Hardening RAMs Against Soft Errors. In *Electronics*, volume 53. McGraw-Hill, April 1980.

[4] K. M. Chandy and C. V. Ramamoorthy. Rollback and Recovery Strategies for Computer Programs. *IEEE Trans. Computers*, c-21(6):546–556, June 1972.

[5] J. J. Horning et al. A Program Structure for Error Detection and Recovery. In G. Goos and J. Hartmanis, editors, *Lecture Notes in Computer Science*, volume 16, pages 177–193. Berlin: Springer-Verlag, 1974.

[6] J. P. Roth et al. Phase II of an Architectural Study for a Self-Repairing Computer. Technical Report SAMSO-TR-67-106, U. S. AirForce Space and Missile Division, El Segundo, CA, 1967.

[7] D. E. Knuth. *The Art of Computer Programming. Volume 3/Sorting and Searching.* Addison-Wesley, 1973.

[8] Israel Koren, Zahava Koren, and Stephen Y. H. Su. Analysis of a Class of Recovery Procedures. *IEEE Trans. Computers*, c-35(8):703–712, August 1986.

[9] S. Y. Kung, H. J. Whitehouse, and T. Kailath. *VLSI Modern Signal Processing.* Prentice Hall, 1985.

[10] N. Park and A. C. Parker. Sehwa: A Software Package for Synthesis of Pipelines from Behavioral Specifications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, CAD-7(3), March 1988.

[11] A. C. Parker, G. T. Pizarro, and M. Mlinar. MAHA: A Program for Datapath Synthesis. In *Proceedings of the 23th IEEE/ACM Design Automaton Conference*, pages 461–66, July 1986.

[12] P. G. Paulin and J. P. Knight. Force-directed Scheduling in Automatic Data Path Synthesis. In *Proceedings of the 24th IEEE/ACM Design Automaton Conference*, pages 195–202, July 1987.

[13] Vijay Raghavendra and Chidchanok Lursinsap. Automated Micro-Roll-Back Self-Recovery Synthesis. In *Proceedings of the 28th IEEE/ACM Design Automaton Conference*, pages 385–390, July 1991.

[14] D. D. Siewiorek, V. Kim, H. Mashburn, S. R. McConnel, and M. M. Tsao. A Case Study of C.mmp and Cm* : Part I-Experiences with Fault Tolance in Multiprocessor Systems. *Proceeding of the IEEE*, 66(10):1178–1199, October 1978.

[15] Yuval Tamir and Marc Tremblay. High-Performance Fault-Tolerant VLSI Systems Using Micro Rollback. *IEEE Trans. Computers*, 39(4):548–554, April 1990.

[16] J. Shambhu Upadhyaya and Kewal K. Saluja. A Watchdog Processor Based General Rollback Technique with Multiple Retries. *IEEE Tran. Software Engineering*, SE-12(1):87–97, January 1986.

[17] John Wakerly. *Error Detecting Codes, Self-checking Circuits and Applications.* Elsevier Science, 1978.

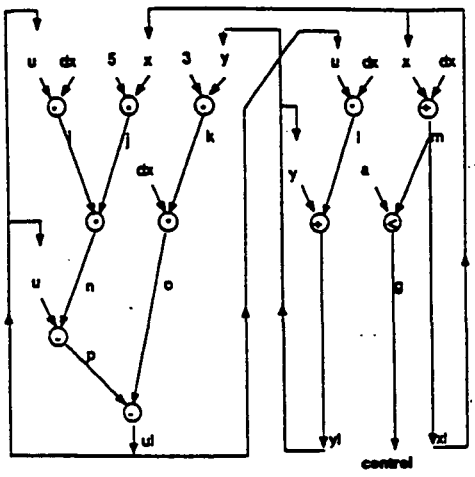


Figure 1: The data flow graph for the example

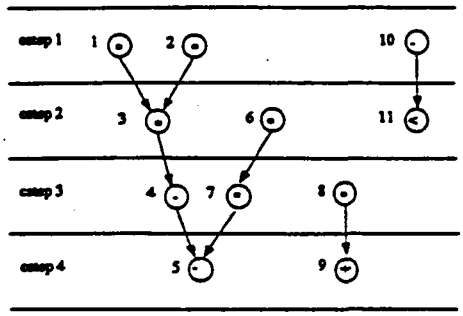


Figure 2: The scheduled data flow graph for the example

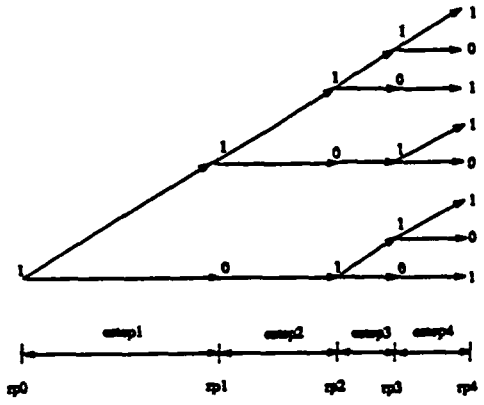


Figure 3: The generation of all possible sequences

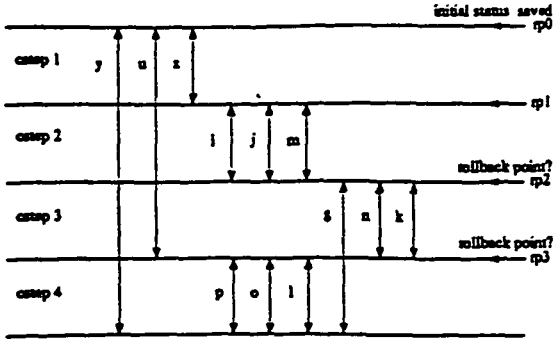


Figure 4: initial live variable graph for the scheduled CDFG

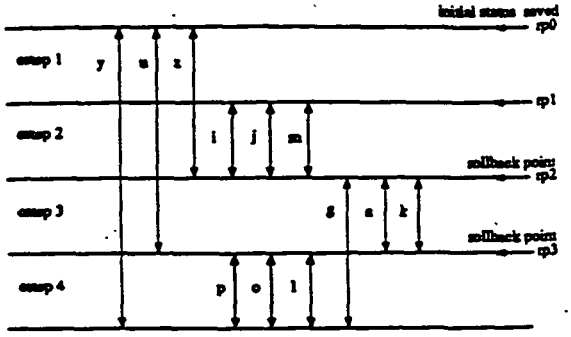


Figure 5: The final live variable graph

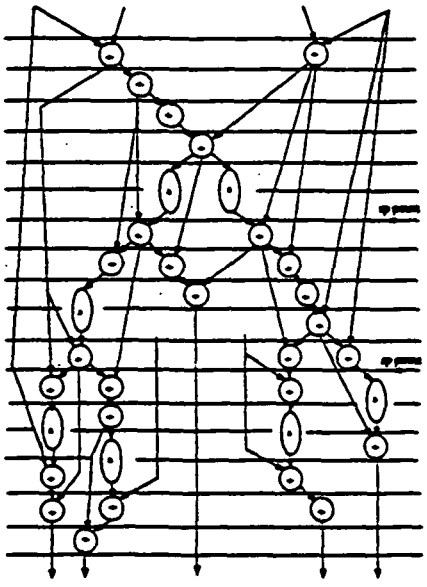


Figure 6: Fifth-order wave digital elliptical filter example

Session 4
Featured Presentations II

Chairman: Sterling Whitaker

System Development Using VLSI for Space Applications

T.R. McKnight, D.E. Rodriguez, J.C. Barnett and C.R. Valverde
Johns Hopkins University Applied Physics Laboratory
Johns Hopkins Road
Laurel, MD 20723
(301) 953-5000

Abstract - **This paper describes the authors' experience in designing a high performance digital processor for a space application using commercially available VLSI parts. Schedule, cost, reliability, and performance issues were addressed in order to arrive at a successful design. In some cases, the use of "off-the-shelf" parts led to unanticipated problems. This experience provides a backdrop for a discussion of the relative merits of custom VLSI versus off-the-shelf alternatives.**

NASA SERC Digital Correlator Projects

John Canaris
NASA Space Engineering Research Center for VLSI System Design
University of Idaho
Moscow, Idaho 83843

Abstract - **Interest in custom digital correlator processors is growing, in both the radio astronomy and earth sensing communities, as scientists realize that VLSI technology is available to them. This paper presents three digital correlator projects currently underway at the NASA SERC for VLSI Systems Design. The projects are a 60MHz chip for the ESTAR satellite, a 100MHz, 1024 channel autocorrelator and a 64 MHz, VLSI system consisting of 8 32 channel complex crosscorrelators, including phase rotators.**

1 Introduction

This paper presents three digital correlator projects currently underway at the NASA SERC for VLSI Systems Design. The projects are a 60MHz chip for the ESTAR satellite, a 100MHz, 1024 channel autocorrelator and a 64 MHz, VLSI system consisting of 8 32 channel complex crosscorrelators, including phase rotators.

2 ESTAR Correlator

This correlator chip is intended for use in synthetic aperture radiometer systems which require the non-time lag cross-correlation of multiple data streams. The large number of correlators (1600) makes the chip especially useful for applications requiring many antennas. The study phase for this chip has been completed. The design and fabrication of this IC is awaiting funding.

The correlator chip accepts 80 input streams, clocked at a maximum rate of 60 MHz. The 80 inputs are divided into two 40 input sections, one horizontal (**HBUS**), the other vertical (**VBUS**). Each horizontal input is multiplied by all vertical inputs with the products being accumulated in separate registers. This process continues for one integration period, as defined by the user. At the end of the integration period, data is made available for reading from an asynchronous interface. This interface is capable of operating at a maximum rate of 20 MHz. Correlated data is read out in 32-bit words. A new integration period can begin when all data has been read out. Input data can be either 2-bit, 3 level or 1-bit, 2 level, under user control. An 8 pin external test port is provided to simplify system testing. The key features of this correlator IC are listed below:

- 1600 correlators
- 60 Megasamples/second maximum input rate
- 20 MHz maximum data output rate
- Low Power ($\approx 500\mu W$/correlator)
- 2-bit/3-Level or 1-bit/2-level correlation supported

- Up to 1.11 second integration time at 60 MHz
- External Test Port
- Maskable Interrupt/Polling Supported
- Data output is independent of system clock
- 32-bit Data Bus
- Control circuitry uses SEU immune memory cells
- TTL compatible inputs, can be driven with CMOS

3 High Performance Correlator

Currently in design, this chip is targeted for radio astronomy applications and digital spectrometers, where large numbers of lags are needed. The chip is designed to be easily used in configurable systems.

This correlator chip accepts two data streams (Stream A and Stream B) at a maximum sample rate of 100 MHz. Samples from Stream A are successively delayed (in a 1024-stage internal shift register) and multiplied by the (undelayed) data from Stream B. Arithmetic is performed in 1024 individual multiplier/accumulators, operating in parallel. At the end of an integration period the data in the accumulators is parallel-loaded into 1024 output registers. New integration can begin immediately. The contents of the output registers can be shifted out at rates up to 20 MHz, via a 32-bit tri-stating output port. The chip supports data multiplexing and all control signals are passed through the chip, to simplify cascading. The key features of this correlator IC are listed below:

- Autocorrelation or Crosscorrelation
- 1024 lags
- 100 Megasamples/second
- Double Nyquist sampling supported
- 32-bit accumulator stages (no prescaler)
- 3-level or 2-level arithmetic supported
- Low Power
- Data and Control signals completely cascadable
- Selectable auxiliary ports on the data inputs
- Data input blanking supported
- Integration can continue while data is output
- TTL compatible inputs, can be driven with CMOS

4 Cross Correlator

This digital correlator is intended for use in interferometry applications in radio astronomy. This chip will integrate, in addition to the correlator itself, many of the functions needed by a multiple antenna interferometry system. The die will contain the equivalent of 512 real correlator channels, organized into 8 sections of 64 channels each. Each section can be configured into a 32 channel complex correlator. There will also be a phase generator circuit associated with each of the 8 sections. Internal muxing circuits will allow flexibility in

interconnecting the correlator sections. The correlator will be contain a control register and a 16 bit asynchronous I/O port. This crosscorrelator chip is currently in the study phase and is awaiting development funding. The key features of this correlator IC are listed below:

- Real or Complex Crosscorrelation
- 512 equivalent lags
- Flexible internal data path configurations
- 64 Megasamples/second
- 16-bit accumulator stages (plus 7 bit prescaler)
- 4-level arithmetic
- Low Power
- Asynchronous I/O port
- Selectable auxiliary ports on the data inputs
- Data input blanking supported
- Integration can continue while data is output
- TTL compatible inputs, can be driven with CMOS

5 Conclusion

The several correlator chip development projects at the NASA SERC are intended to provide the scientific community with levels of integration and correlator system complexity which have not previously been available.

6 Acknowledgements

This research was supported in part by NASA, under the NASA Space Engineering Research Center grant NAGW-1406 and contract G5-NAG-5-1635. The author would also like to thank Jon Hagen, at the NAIC Lab, and Will Aldrich, at the MIT-Haystack Observatory.

Fully-Depleted Silicon-on-Sapphire and Its Application to Advanced VLSI Design

Bruce W. Offord
NCCOSC Research and Development
San Diego, Ca. 92152-5000

Abstract - **In addition to the widely recognized advantages of full dielectric isolation, e.g., reduced parasitic capacitance, transient radiation hardness and processing simplicity, fully-depleted silicon-on-sapphire offers reduced floating body effects and improved thermal characteristics when compared to other silicon-on-insulator technologies. The properties of this technology and its potential impact on advanced VLSI circuitry will be discussed.**

1 Introduction

Compared to the junction isolation of VLSI technologies based on bulk silicon, the full dielectric isolation afforded by silicon-on-insulator (SOI) offers many well known advantages. Of primary importance is the significant reduction in parasitic junction capacitance resulting from the replacement of the silicon under the source/drain regions with an insulator. This reduction in capacitance yields higher device switching speed and decreased dynamic power consumption. Additional benefits of SOI include reduced interconnect capacitance, unconditional latch-up immunity, insensitivity to transient radiation and single-event phenomena and the process simplicity inherent to the natural device isolation.

A further refinement of the SOI concept has led recently to the use of ever thinner silicon layers. MOS devices fabricated on very thin Si films operate in a fully depleted (FD) mode in which the gate-controlled depletion region extends to the back interface as the top interface inverts. Under these conditions SOI devices display some improved characteristics when compared to conventional, partially depleted SOI devices. Some of the advantages of full depletion include; reduction of the kink effect due to the absence of the neutral layer in the channel region [1]; better control of the channel region by the gate leading to reduced punch-through current [2]; higher carrier mobility due to reduced transverse electric field [3]; and reduced short-channel effects [4].

Of the leading SOI candidate materials silicon-on-sapphire (SOS) has the longest history of reliable service in harsh environments. We describe here the properties of a fully depleted VLSI technology based on ultra-thin (< 100 nm) SOS films and show some of the advantages of this technology to the designer of digital and analog CMOS circuitry. As with other fully depleted SOI approaches this technology can be scaled laterally with relative ease because the vertical scaling has already been accommodated by the thin active region. Because of the full dielectric isolation, source, drain, p-well and n-well design rules become limited only by the alignment capability of the lithographic tool and the ability to transfer patterns by dry etching. These advantages lead to processing simplicity and very high density circuitry.

2 Material

Until recently, SOS technologies were based on silicon layers of at least 300 nm in thickness because of the high density of grown-in defects occupying the region near the Si/sapphire interface. This high density of twinning defects had two effects which precluded CMOS operation in the fully depleted regime. First, these defects significantly reduced carrier mobilities in the near-interfacial region to levels which significantly limited transistor performance. Second, the presence of intra-gap states associated with these defects pinned the potential of the Si/sapphire interface thus inhibiting the control of back interface potential by the gate.

To deal with these two related problems we refined a defect reduction technique based on solid-phase epitaxial regrowth [5] to improve the SOS films in the region near the Si/sapphire interface. This method uses Si ions implanted at 185 keV at a dose of $6 \times 10^{14}/cm^2$ to amorphize the film near the interface. Regrowth through the amorphous region at 900 deg C results in dramatically improved films with significant reduction in twinning defect concentrations as shown by RBS [6] and TEM [7]. CMOS devices fabricated from 100 nm thick improved SOS show low field channel mobilities of $500cm^2$/V-sec and $200cm^2$/V-sec for NMOS and PMOS devices, respectively and back interface state densities of $1 - 2 \times 1011/cm^2$ [8].

3 Design Advantages

Figures 1 and 2 are cross sectional views of a CMOS/SOS device and a CMOS bulk device, respectively. The full dielectric isolation offered by SOS eliminates the need for field oxides, channel stop implants, and isolation techniques which can limit density and which make CMOS latchup a design issue. The natural isolation together with the fully depleted nature of the thin films lend a freedom to design novel structures and analog circuitry with different power supply voltages. For example, an analog multiplier that is an important part of a neural network design has been made in SOS at NRaD[9]. Design rules in SOS are made simple and planarization schemes can be relaxed because of the absence of a thick field oxide. Latchup is not an issue and radiation sensitivity is minimized. Decreased subthreshold slope factor due to the fully depleted nature of the films allows for lowered threshold voltages and hence faster switching times. The use of FD films also eliminates the need for body ties which results in more compact layouts. Parasitic capacitances are reduced to their absolute minimum in a FD SOS process, and if a silicide process is used, resistance of silicon and polysilicon areas can also be very low. These advantages can result in fast, dense, and low power designs.

4 Device Characteristics

Figures 2 and 3 show some typical drain current characteristics for a fully depleted n-channel device and a non-fully depleted device with an L_{eff} of $1.0 - \mu m$. Both devices have a nominal silicon film thickness of 100-nm but the channel of the non-fully depleted device has been doped such that the depletion width is less than 100-nm. Elimination of the kink effect in the FD device is clearly shown. It is also interesting to note that no negative slope is

observed in the output characteristics for the fully-depleted device. This is in contrast to other SOI technologies where a negative slope in the output characteristics of an n-channel device for a gate length of $1.0 - \mu m$ has been observed [10]. The poor thermal conductivity of the buried oxide in other SOI technologies as opposed to the higher thermal conductivity of sapphire is the reason attributed to this effect. The drain breakdown voltage (V_{bds}) of the FD $1.0 - \mu m$ device has been measured at 6.8 volts which is sufficient for use with a 5.0 volt power supply. This drain breakdown voltage is in sharp contrast to other $1.0 - \mu m$ n-channel SOI devices which have shown a breakdown less than 5.0 volts [11]. This reduction in Vbds for n-channel SOI devices without body ties has been shown to be due to parasitic bipolar conduction [12]. This effect is particularly troublesome for devices with gate lengths below about 1 micrometer fabricated in long lifetime materials such as SIMOX, ZMR, and bonded wafer SOI. Bipolar effects are enhanced in these materials because of relatively long minority carrier lifetimes which may exceed 100 ns in some of the more recently developed material. In comparison, drain breakdown voltages measured on fully-depleted SOS nMOS devices are shown in Fig. 5 as a function of effective channel length. From these data we conclude that a similar reduction in Vbds exists in down-scaled, fully-depleted nMOS/SOS devices but with a significant mitigation of this effect when compared to devices fabricated in other SOI materials.

5 Circuit Applications

SOS has been used for years for radiation hard, medium scale integration circuits. With the film improvement techniques available, SOS can be used for VLSI designs with all of the benefits and advantages of an ultra-thin, fully depleted technology. Analog-to-Digital converters, neural networks, and any space based application where low power, high speed, high functionality, and radiation insensitivity are needed, can all benefit from the advantages offered by SOS.

The microelectronics research facility at NRaD has fabricated a few technology demonstration chips using the process described in Table II. A 16×16 bit parallel multiplier with $0.75 - \mu m$ gate lengths and a 1K SRAM with effective gate lengths of $1.25 - \mu m$ have been fabricated in thin film fully depleted SOS. The 16×16 bit multiplier has shown CMOS loaded gate delays of 243 ps at 5.0 volts VDD. Although the design of the 1K SRAM was conservative, access times of 20ns were obtained. These designs at least demonstrate the viability of FD SOS for VLSI applications.

6 Challenges

Floating body effects, while mitigated by fully depleted films, are not eliminated entirely. This effect leads to early drain breakdown although for deep submicron devices it is predicted that SOI will have a higher breakdown voltage than a bulk device[13]. Body ties, although successful at eliminating the floating body (albeit at the expense of less dense circuitry) for thicker, non-fully depleted films are not effective for a FD film due to the lack of free carriers in the FD film. Lightly doped source and drains will increase drain breakdown voltage while

creating a trade-off of saturation current.

7 Summary

High quality thin films of SOS have made the use of fully depleted, sub-micron transistors possible. Due to the low minority carrier lifetimes in SOS, drain breakdowns over 5.0 volts for a gate length of 1.0-m have been observed. The potential for very high speed, low power and dense layout design has been demonstrated. The natural isolation offered by SOS and the very thin films described here offer advantages to the designer and make SOS a viable VLSI technology.

References

[1] J.-P. Colinge, "Reduction of kink effect in thin-film SOI MOSFETs", IEEE Electron. Dev. Lett., vol. EDL-9, p.97, 1988.

[2] M. Yoshimi, H. Hazama, M. Takahashi, S. Kambayashi, T. Wada, K. Kato, H. Tango, "Two-dimensional simulation and measurement of high-performance MOSFET's made on a very thin SOI film,"IEEE Trans. Electron Devices, vol. 36, p.493, 1989.

[3] M. Yoshimi, H. Hazama, M. Takahashi, S. Kambayashi, and H. Tango, "Observation of mobility enhancement in ultrathin SOI MOSFETs," Electron. Lett., vol. 24, p. 1078, 1988.

[4] K. Throngnumchai, K. Asada, and T. Sugano, "Modeling of 0.1 micron MOSFET on SOI structure using Monte Carlo simulation technique", IEEE Trans Electron Devices, vol. ED-33, p. 1005, 1986.

[5] S.S. Lau, S. Matteson, J.W. Mayer, P. Revesz, J. Gyulai, J. Roth, T.W. Sigmon, and T. Cass, "Improvement of crystalline quality of epitaxial Si layers by ion-implantation techniques", Appl. Phys. Lett. vol. 34, p.76, 1979.

[6] R. Reedy, T.W. Sigmon, and L.A. Christel, "Suppressing Al outdiffusion in implantation amorphized and recrystallized silicon-on-sapphire films", Appl. Phys. Lett., vol.42, p. 707, 1983.

[7] M.A. Parker, R. Sinclair and T.W. Sigmon, "Lattice images of defect-free silicon-on-sapphire prepared by ion implantation", Appl. Phys. Lett., vol. 47, p.626, 1985.

[8] G. A. Garcia, R.E. Reedy and M. L. Burgener, "High-quality CMOS in thin (100 nm) silicon on sapphire", IEEE Electron. Dev. Lett., vol. EDL-9, p.32, 1988.

[9] R.L. Shimabukuro, M.E. Wood, and P.A. Shoemaker, "A Neural Network Synapse Cell in 90nm SOS," Proceedings 1991 International SOI Conference, pp. 162-163.

[10] L.J. McDaid, S. Hall, P.H. Mellor, W. Eccleston," Physical Origin of Negative Differential Resistance in SOI Transistors," Electron. Lett., vol 25, no. 13, pp. 827-828, June 1989.

[11] G.A. Armstrong, J.R. Davis and A. Doyle, "Characterization of Bipolar Snapback and Breakdown Voltage in Thin-Film SOI Transistors by Two-Dimensional Simulation," IEEE Trans. Electron Devices, vol. 38, no. 2, February 1991.

[12] M. Haond, and J. Colinge, "Analysis of Drain Breakdown Voltage in SOI n-Channel MOSFETs," Electron. Lett., vol. 25, pp 1640-1641, Nov. 1989.

[13] J.P. Colinge "Problems and Issues in SOI CMOS Technology," Proceedings 1991 IEEE International SOI Conference, pp. 126-127.

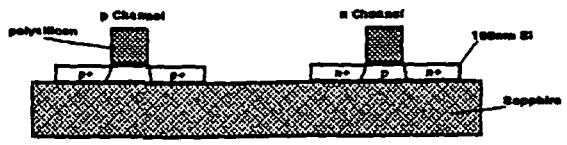


Figure 1: Cross section of CMOS/SOS device.

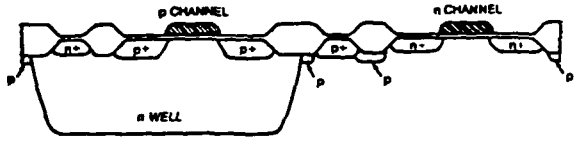


Figure 2: Cross section of CMOS/bulk device.

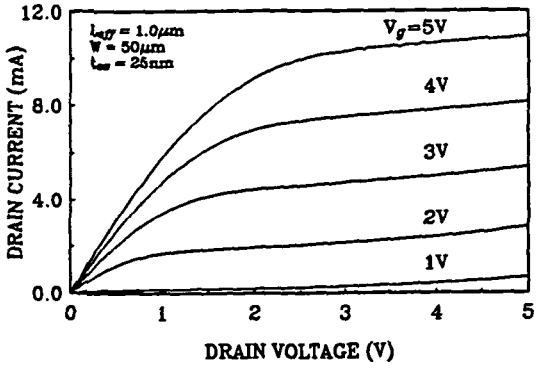


Figure 3: Drain characteristics of a fully-depleted NMOSFET.

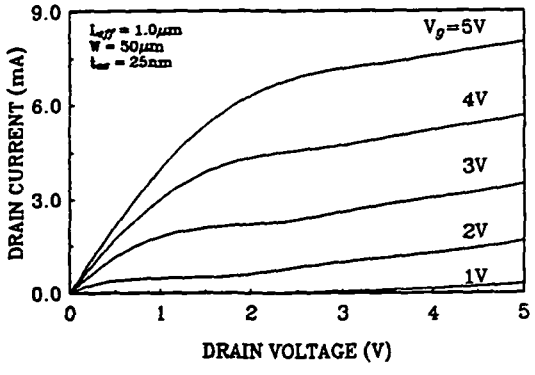


Figure 4: Drain characteristics of a non-fully depleted NMOSFET.

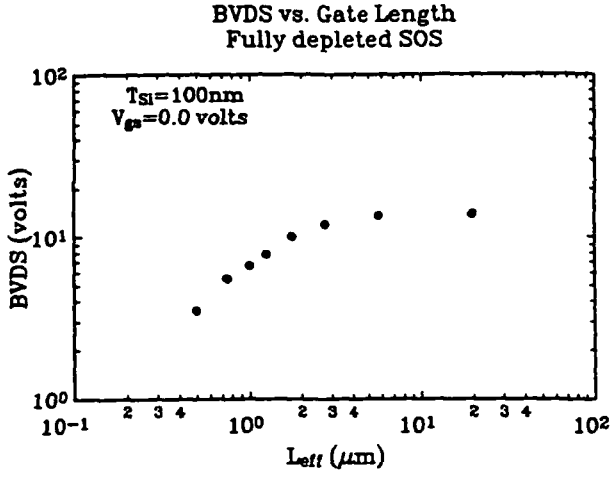


Figure 5: Drain breakdown vs. effective gate length at VGS=0.0 volts for fully-depleted SOS.

TABLE I

Advantages of a SOS/Fully Depleted Technology

Advantages	Primary Reason	Application
Reduced capacitance	Reduced junction capacitances due to very thin film and smaller area junctions	High-speed electronics
Immunity to transient upset	Very small active volume for electron-hole generation	Soft-error free memories; defense electronics
Elimination of "kink" effect	No neutral floating substrate (due to substrate fully depleted)	Analog VLSI
Increased drain breakdown voltage	Higher recombination rate due to short lifetime cause a reduction in gain of parasitic npn bipolar	Deep submicron VLSI

Table II
Silicide CMOS/SOS Process Steps

Description	Details
• Silicon thinned to 1000-Å	oxide grown in wet O_2 then stripped
• Island isolation	plasma etch
• NMOS threshold implant	boron: 2.5 x 10^{12} cm^{-2} at 35 keV
• Gate oxidation	11 min at 875°C in wet O_2
• Polysilicon deposition,doping, etch	T_{dep} = 580°C
• NMOS source/drain implant	arsenic: 2.0 x 10^{15} cm^{-2} at 55 keV
• PMOS source/drain implant	BF_2: 2.0 x 10^{15} cm^{-2} at 50 keV
• Sidewall oxide deposition	3000-Å undoped LTO
• Source/drain anneal	550°C 30min, 850°C,30min in N_2
• Sidewall oxide etch	plasma etch
• Titanium deposition	T_{Ti} = 500-Å
• Silicon implant	silicon: 1.0 x 10^{15} cm^{-2} at 100 keV
• RTA	675°C, 60sec, N_2
• Ti etch	1:1:5NH_4OH:H_2O_2:H_2O,60°C,3 min
• RTA	850°C, 60sec, N_2
• Contact oxide, etch contacts	6000-Å undoped LTO
• Metal deposition, etch	10,000-Å Al/1%Si
• Sinter	450°C for 30 min in N_2+H_2

Session 5
VLSI Design

Chairman: Kel Winters

Design and Test of Field Programmable Gate Arrays in Space Applications

Priscilla L. McKerracher, Russel P. Cain,
Jon C. Barnett, William S. Green and James D. Kinnison
Johns Hopkins University Applied Physics Laboratory
Johns Hopkins Road
Laurel, MD 20723
(301) 953-5000
Priscilla_McKerracher@spacemail.jhuapl.edu

Abstract - **Field Programmable Gate Arrays (FPGAUs) offer substantial benefits in terms of flexibility and design integration. In addition to qualifying this device for space applications by establishing its reliability and evaluating its sensitivity to radiation, screening the programmed devices with Automatic Test Equipment (ATE) and functional burn-in presents an interesting challenge. This paper presents a review of the design, qualification and screening cycle employed for FPGA designs in a space program, and demonstrates the need for close interaction between design and test engineers.**

1 FPGA Technology

Utilizing Actel Field Programmable Gate Arrays for flight designs has several advantages. The first is that the Actel FPGAUs are capable of being programmed with moderately complex designs. This eliminates the need for a great deal of 'glue logic' and can also take the place of complex logic that is not available in space qualified parts. The second is that the design cycle is much shorter than that of conventional gate arrays. This allows design changes in prototype hardware in as little as one hour instead of waiting eight weeks for a typical masked gate array. The last advantage is the ready availability of reliable, space qualified parts that only need be programmed and tested before use in a flight application.

2 Radiation Testing

However, a part is only acceptable for space applications if it can withstand the harshness of the applicable radiation environment. Of the FPGAUs available in 1990, only the 2000 gate Actel A1020, using antifuse technology, had demonstrated adequate radiation tolerance to make it a viable choice for space applications.

The Actel FPGAUs from the Matsushita foundry showed good radiation tolerance in all areas of concern. Total dose testing with Co-60 indicated that the Matsushita devices stay within data sheet parameters for doses above 100 Krads (Si), while those from the alternative Texas Instruments line perform poorly after only 6 Krads (Si). Examination of two versions of the Actel 1020 FPGA, the original 2.05 ACT1020-A from the Matsushita foundry and the 1.25 TI-1210D by Texas Instruments showed an acceptable amount of sensitivity to Single

Event Upsets for both parts. We will present a review of radiation testing by APL and others.

3 Device Qualification and Screening

Actel reports 37.4M device hours of dynamic burn-in on programmed devices at 125 deg C producing a failure rate of 83 FITS. We performed 1000 hour burn-in life test at 125 deg C on 8 sample programmed devices. No failures were observed. We will discuss the life test design.

4 Testing of Unprogrammed Devices

ActelUs functional screening of unprogrammed devices is quite comprehensive. Obtaining 883C qualified parts with Particle Impact Noise Detection (PIND) testing and additional 240 hour burn-in greatly simplified screening for APL. The major work then involved adequately screening the devices Rpost-programming.

5 Testing of Programmed Devices

Although the programmed devices represent Application Specific Integrated Circuits (ASICUs) of relatively low gate count, ensuring testability of the completed device is still important and complex. By following good Design for Testability (DFT) techniques the designer can enhance the testability of the device; thereby, significantly decreasing the cycle time for functional test development. Only then can a designer take true advantage of the short design turn-around time of FPGAUs.

Through cooperation between the design and test engineers, it is possible to develop simulation vectors appropriate for final electrical test. An in-house RCS program translated the simulation vectors from Mentor listings to the Sentry S-15 ATE tester environment. Ensuring the completeness of these vectors via fault-grading these vectors is still an area of concern. We will discuss these issues.

6 Screening of Programmed Devices

In order to successfully screen a completed FPGA for possible internal defects, it is necessary to combine a thorough electrical test program with a complete burn-in regimen. Each design required an individual set of burn-in circuitry, designed in-house. A discussion of this work is included.

7 Final Results

We will present a summary of the successes, as well as the failures and problems encountered for each of eight design types.

Defect-Sensitivity Analysis of an SEU Immune CMOS Logic Family [1]

Erik H. Ingermann and James F. Frenzel
Department of Electrical Engineering
University of Idaho, Moscow, ID 83843
208-885-7888 jfrenzel@groucho.mrc.uidaho.edu

Abstract - **Fault testing of resistive manufacturing defects is done on a recently developed single event upset immune logic family. Resistive ranges and delay times are compared with those of traditional CMOS logic. Reaction of the logic to these defects is observed for a NOR gate and an evaluation of its ability to cope with them is determined.**

1 Introduction

This paper reviews prior research on the effects of manufacturing defects in CMOS logic. CMOS logic was chosen because it is becoming the technology of choice for digital microelectronics as it is well suited for VLSI designs. It requires low power while still providing high performance. It has been shown that stuck-at fault models do not accurately characterize all manufacturing defects, particularly for CMOS technology [5]; therefore, there exists an array of single fault physical models which cover critical manufacturing defects. This paper focuses upon resistive fault models as described in [3]. The goal is to apply these physical models to a newly developed SEU (single event upset) immune logic family [1]. Standard CMOS is sensitive to radiation effects in space, thus a robust CMOS logic family was developed which could tolerate this environment. The purpose of applying these physical models to this new SEU immune logic family is to observe how it deals with manufacturing defects as compared to traditional CMOS logic. Such information will be useful in evaluating the manufacturability of the technology.

2 Manufacturing Defects

Manufacturing defects are upsets in the fabrication process or environmental contaminations which affect the layout or the material of an IC causing it to function incorrectly. These manufacturing defects are physically represented by fault models. Fault models are used to generate test sets which in turn are used to determine faults in IC's. There are two types of manufacturing defects: global and local [6], [2]. Global defects involve major fabrication errors such as cracks or scratches on the wafer, mask misalignments, etc. These are easily detectable since many sections of the IC are damaged. Local defects, on the other hand, affect a localized area on an integrated circuit layer. These are seen as changes of the electrical properties at a point. Local (spot) defects are explored in this paper.

[1]This research was supported by NASA under Space Engineering Research Center Grant NAGW-1406

V_{ILMAX}	1.84V
V_{IHMIN}	2.76V

Table 1: Noise Margin Parameters for Traditional CMOS

Manufacturing defects can generally be attributed to the absence or excess of material in one of the conductive, semiconductive, or insulating layers [4]. Airborne particles and droplets are the most predominant causes of spot defects as they contaminate the lithographic mask. Thus, most spot defects are generated during the lithographic process [6].

3 The SEU Immune Logic Family

While CMOS logic has many benefits, it is sensitive to radiation. In space applications, a high radiation level environment, particles pass through the electronic circuitry and momentarily alter the electrical properties in the area of penetration, thereby creating potential failures. The failure occurs when a particle passes through a diffusion region of one logic level to the substrate at the opposite logic level. This results in a short which gives rise to a glitch. An SEU immune (SEU-I) logic family was developed to provide radiation immunity, while maintaining many of the benefits of traditional CMOS logic [1]. In the SEU-I logic family, additional weak feedback transistors momentarily maintain the logic level in the impacted area and thus prevent the glitch. Figure 2 shows an SEU-I NOR gate and three SEU-I inverters. For more information on this family please see [1].

4 Fault Simulation Methods

Schematics were entered using PIGLET, an artwork and schematic editor, and translated to SPICE input using SCIP, a schematic translator and SPICE interface. Simulation was performed using HPSPICE with CMOS26 process models on a Hewlett Packard 9000/385 workstation. CMOS26 is a 1μm CMOS N-Well process. All simulations were done in the HPSPICE "slow case" in order to provide worst case conditions.

Resistive shorts were simulated on a two input NOR gate, driven by two inverters and driving an inverter. The shorts simulated for each transistor were drain-to-source, gate-to-source and gate-to-drain. Transistors were sized to provide a two to one β ratio between the NMOS and PMOS transistors and between the strong and weak transistors of the n and p networks. The process parameters were left as the HPSPICE CMOS26 default values.

The simulated input sequence was AB=(01 00 10 11). Each input was applied for 100ηsec with a transition time of 3ηsec. Input logic levels ranged from 0 to 4.5 volts, while the Vdd supply was set to 5 volts. The delay times for the faulted gates were determined for the 01 to 00 and 00 to 10 patterns since these are the only valid delay times for the sequence.

Resistive shorts have two primary effects upon a logic gate, depending upon the input pattern: one, the output voltage level may degrade, and two, the transition (delay) time may change. Minimum and maximum input levels for logic high and low were determined by applying a ramp input signal to an inverter and observing the input voltage levels corre-

	POUT	NOUT
V_{ILMAX}	2.93V	.57V
V_{IHMIN}	4.41V	2.65V

Table 2: Noise Margin Parameters for SEU-I CMOS

Input Pattern	Traditional OUT	SEU-I POUT	SEU-I NOUT
00→01	1	7.5	1.5
00→10	1	15	3
00→11	.5	6.5	1
01→00	1	1.5	3
11→00	1	1.5	3
10→00	1	1.5	3.5

Table 3: Fault Free Delay Time (ηsec). Resolution $\mp.5\eta$sec

sponding to a negative one slope on the output voltage [7]. This was done for both traditional CMOS and SEU-I CMOS, as shown in Tables 1 and 2. These levels separate the output voltage range into determinate and indeterminate regions. Careful considerations were made to maintain the same transistor parameters between the traditional CMOS gates, the SEU-I gates, and their respective noise margin inverters.

Simulations were performed to identify the critical short resistance at which a gate produced a valid output, meaning the output voltage fell within the determinate region. Below this resistance, the output was in the indeterminate region, while above this resistance the output voltage was always valid. For simulated shorts in which the output was affected for several input states, this resistance was recorded for each of the affected states, resulting in a resistive range. The smallest number in a resistive range indicates the short resistance at which the output was valid for one of the affected input states and the largest number indicates the short resistance at which the output was valid for all input states.

For shorts resulting in a valid output, delay times were recorded which indicate the time required for the output to transition. As the simulated short resistance was increased, the delay time eventually returned to that observed for a fault-free gate.

Capacitive loading was only provided through the driving of an inverter. If additional loading was simulated, the delay of both the traditional and SEU-I CMOS gates would increase.

5 Results

The output steady state level, resistance range, and delays were evaluated for the given shorts of the traditional and SEU-I logic. Fault free delay times are given in Table 3. The circuit used for the traditional CMOS simulations can be seen in Figure 1, whereas that used for SEU-I CMOS simulations is shown in Figure 2.

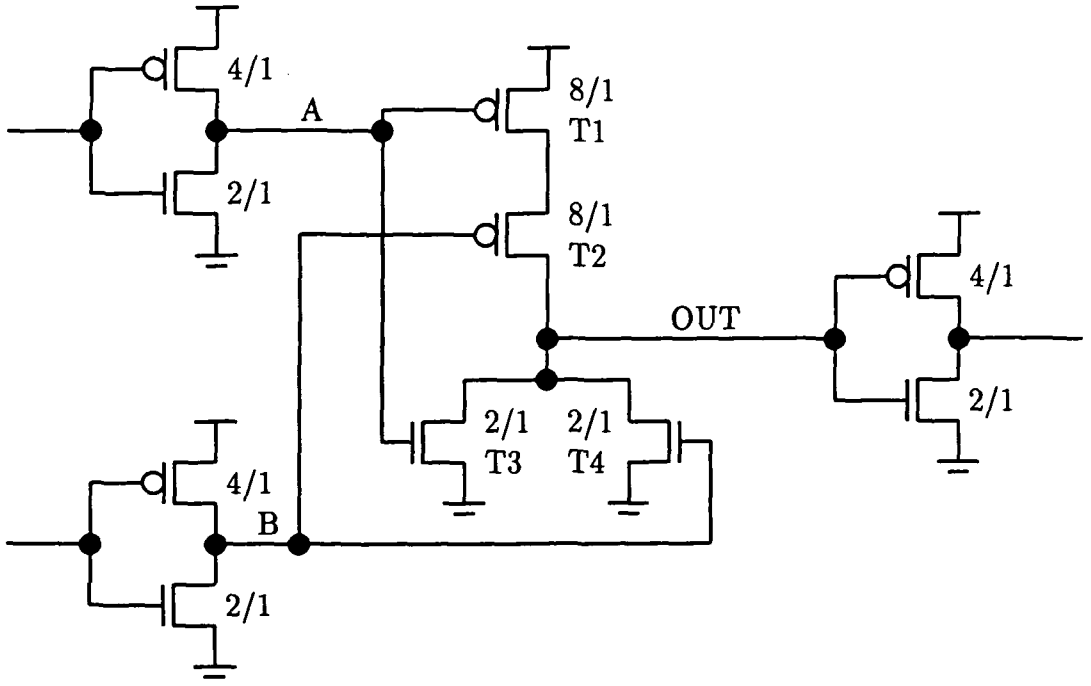


Figure 1: Traditional CMOS Circuit

5.1 Traditional CMOS Shorts

Drain-to-source shorts (DS) have the least effect since they only involve one transistor, meaning that they do not alter the electrical characteristics of other transistors. The output for each short was faulty under only a single input pattern, namely, 01(T2), 00(T3 and T4) and 10(T1). The input 00 activated the fault for both NMOS transistors because they are in parallel.

Gate-to-source shorts (GS), on the other hand, affect two transistors in the NOR gate, the first being the shorted transistor itself and the second being the transistor that connects to the shorted gate node (each input connects to two transistors, one n and one p). Again the output was only affected under a single input pattern, specifically 00(T1 and T2), 10(T3) and 01(T4). As the simulated resistance is decreased, the shorted transistor can not "turn on" completely. Consequently, the transistor's channel resistance is higher. Furthermore, the other transistor connected to the shorted node does not "turn off" completely.

Gate-to-drain (GD) shorts have the greatest effect since most of the shorts connect between the gate and the output, as opposed to the gate-to-source shorts which connect to supply nodes. Therefore, the input has a direct influence on the output while also altering the logic levels at other gates. For this reason, gate-to-drain shorts affect the output for multiple input states, those of 01(T2 and T4), 00(T1, T2, T3 and T4), 10(T1 and T3), and a mild effect (not enough to reach noise margin limits) from 11(T2, T3 and T4). As was the case for DS shorts, the GS short at T2 is identical to the GS short at T4. The delays are limited by the physical characteristics of the affected transistor. As the short's resistance is decreased, the gate and drain approach equipotential, causing the channel resistance to change. This resistance will continue to change until an equilibrium is reached between the

	T1	T2	T3	T4
DS	3.30	5.60	7.10	7.10
GS	6.25	5.76	7.84	7.08
GD	2.30→3.80	6.0→8.0	4.60→8.20	6.0→8.0

Table 4: Resistances for traditional CMOS (in K ohms)

	T1	T2	T3	T4
DS	(10)1.5...	(01)*N.S.*	(00)1.0...	(00)1.0...
GS	(00)4.0...	(00)3.0...	(10)6.5...	(01)*N.S.*
GD	(00)1.5...	(00)1.5...	(00)1.0...	(00)1.5...
	(10)1.5...	(01)*N.S.*	(10)1.5...	(01)*N.S.*

Table 5: Delay times for traditional CMOS (in ηsec). Resolution $\mp.5\eta$sec

"on" (channel) resistance and the short resistance. For example, PMOS transistors (i.e. T1) pass ones, thus, a GD short would cause the transistor to begin turning off, causing the channel resistance to increase. This is the same idea as the gate to source short except that the drain potential is dependent on how fully "on" the transistor is. It is for this reason that the channel resistance will reach an equilibrium with the fixed short resistance. This prevents the delay from becoming large. The NMOS transistors behave similarly.

Tables 4 and 5 show resistance values and delay times respectively. The affected input states are shown as (AB). Take for example entry T4/GD where the input (00) has a delay of 1.5ηsec. As the resistance increases the delay time drops to its fault free value of 1.0ηsec. The N.S. (Not Simulated) for input (01) means that although the DS short did cause faulty behavior for the input A=0 and B=1 it was not possible to calculate a delay time because input (01) was the starting input of the sequence. All delay times return to the fault free times as the resistance goes towards infinity.

5.2 SEU-I CMOS Shorts

As can be expected, the common transistors between the SEU-I and the traditional CMOS behave in a similar manner to the same input patterns for the given faults. For this reason, only the differences will be presented.

Drain-to-source shorts in the feedback transistor affect several input patterns, those where the output is high (00) for T5 and those where the output is low (01 10 and 11) for T6. T5 pulls POUT low while T6 pulls NOUT high. This short type, as can be seen, has a higher resistive range since there is a direct influence on the output. Delays occur for the T6 feedback gate where the output is low and the T5 feedback gate where the output is high. As T6 is shorted, NOUT is pulled up causing T5 to turn off and thus increasing the resistance to ground. Since T5 is the means by which POUT is pulled low, any effect upon this gate would greatly affect the fall time. For the case of T5, POUT is driven by the pull up transistors, thus, shorting T5 has little effect outside of causing POUT to be pulled down. When POUT is pulled down the resistance from NOUT to Vdd is increased, thereby

5.2.6

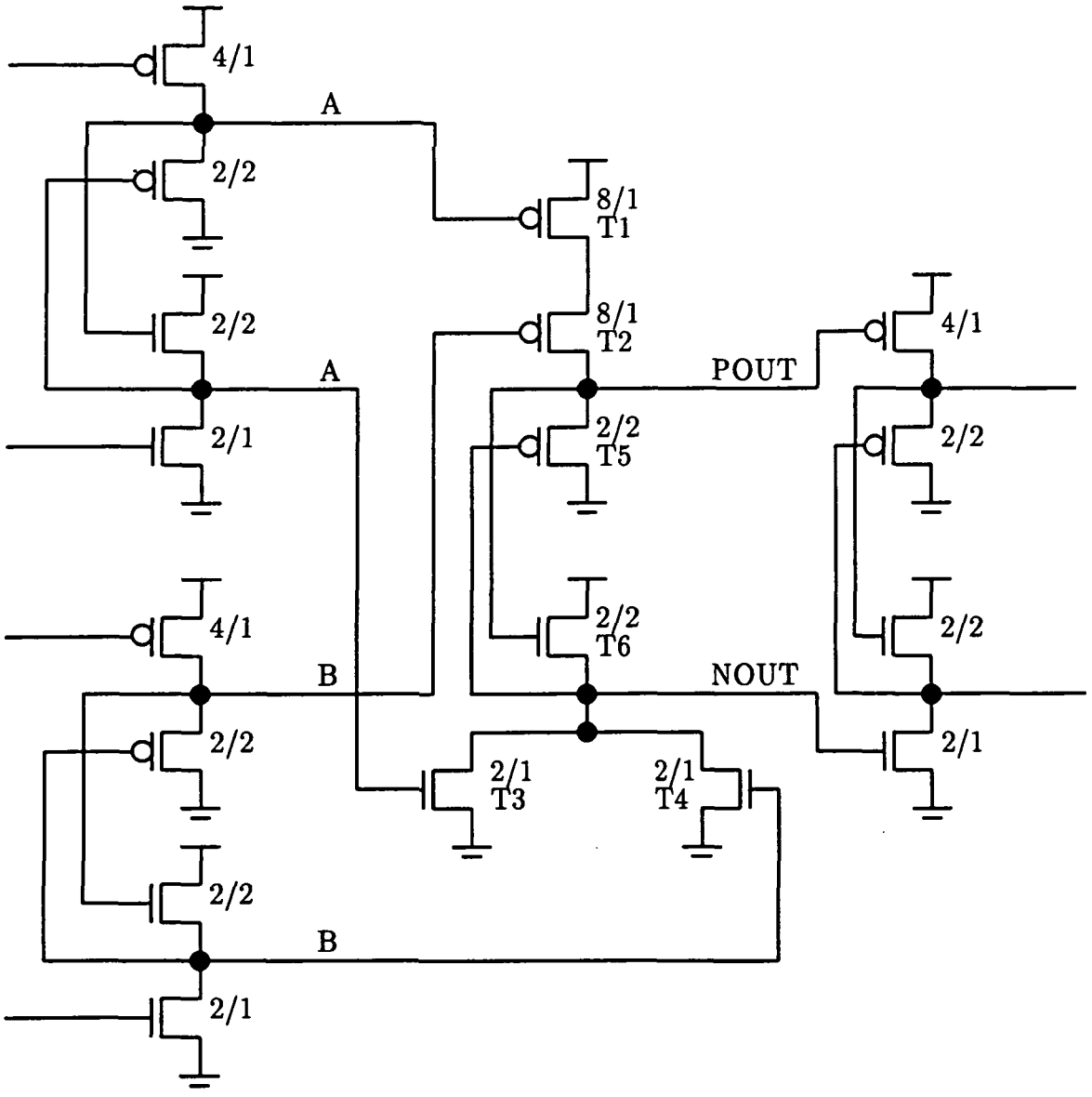


Figure 2: SEU-I Circuit

		T1	T2	T3	T4
DS	POUT	140.0	150.0	No Effect	No Effect
	NOUT	30.0	30.0	100.0	100.0
GS	POUT	26.0	30.0	6.0	5.0
	NOUT	22.0	24.0	10.0	9.0
GD	POUT	33.0→136.0	39.0→140.0	10.0	5.0
	NOUT	15.0→27.0	16.0→35.0	6.0→95.0	5.0→95.0

Table 6: Resistances for SEU-I CMOS common transistors (in K ohms)

		T5	T6
DS	POUT	50.0	10.0→15.0
	NOUT	30.0	20.0→40.0
GS	POUT	No Effect	No Effect
	NOUT	No Effect	No Effect
GD	POUT	No Effect	120.0→140.0
	NOUT	100.0	40.0

Table 7: Resistances for SEU-I CMOS feedback transistors (in K ohms)

increasing the rise time for NOUT.

Gate-to-source shorts for the feedback transistors have no effect on the outputs. This type of short connects POUT to NOUT; because these nodes are always at the same logic levels, the result is to simply to pull both nodes to the appropriate rail. The resistive range, for the common transistors, is the lowest of all SEU-I transistor shorts. These shorts affect the transistor's gate as opposed to affecting the output directly, as is the case for the DS and GD shorts (i.e. the transistors "on" state is manipulated instead of the transistor being bypassed).

Gate-to-drain shorts of the feedback transistors affect several input states, which are the same as those of the drain-to-source short. In this case the T5 transistor pulls NOUT down while T6 pulls POUT up. This happens when POUT is low. This fault type has a high resistive range much like the DS short, where the input to the gate affects the output directly. As a result of the shorted feedback transistors, there is a minimum effect upon the delays. A GD short on T5 has no effect since it occurs at the input pattern (00) where the transistor is not driving the output. However, it is also the DS short for T3, but this short too has virtually no influence on the rise time. The feedback transistor T6 has an influence for the input patterns (01,10 and 11), although, the effect to the output is again minimal. For these input cases the output NOUT is driven by either transistor T3, T4 or both; thus, as T6's resistance increases the effect is not noticed in output's fall time.

Tables 6 and 7 show resistance values for SEU-I CMOS common and feedback transistors. Tables 8 and 9 present delay times for the common and feedback transistors. Shorts that did not have an effect on the circuit are indicated by "No Effect".

		T1	T2	T3	T4
DS	POUT	(10)5.0...	(01)*N.S.*	(00)1.5...	(00)1.5...
	NOUT	(10)1.5...	(01)*N.S.*	(00)2.0...	(00)2.0...
GS	POUT	(00)21.0...	(00)12.5...	(10)45.0...	(01)*N.S.*
	NOUT	(00)25.0...	(00)24.0...	(10)32.5...	(01)*N.S.*
GD	POUT	(10)4.5... (00)2.5...	(01)*N.S.* (00)2.5... (11)*N.S.*	(00)1.5... (10)20.5... (11)*N.S.*	(01)*N.S.* (00)1.5... (11)*N.S.*
	NOUT	(10)1.5... (00)6.5...	(01)*N.S.* (00)2.0... (11)*N.S.*	(00)2.0... (10)8.5... (11)*N.S.*	(01)*N.S.* (00)2.0... (11)*N.S.*

Table 8: Delay times for SEU-I CMOS common transistors (in ηsec). Resolution $\mp.5\eta$sec

		T5	T6
DS	POUT	(00)2.5...	(01)*N.S.* (10)21.0... (01)*N.S.*
	NOUT	(00)4.5...	(01)*N.S.* (10)3.0... (01)*N.S.*
GS	POUT	No Effect	No Effect
	NOUT		
GD	POUT	(00)1.5...	(01)*N.S.* (10)5.0... (11)*N.S.*
	NOUT	(00)2.0...	(01)*N.S.* (10)1.5... (11)*N.S.*

Table 9: Delay times for SEU-I CMOS feedback transistors (in ηsec). Resolution $\mp.5\eta$sec

5.3 Discussion of Results

The results have shown that there are some considerable differences between the behavior of the SEU-I and traditional CMOS logics. The differences are due to the weak feedback transistors and, most importantly, the SEU-I gates which drive them.

Feedback transistors by their design have a higher resistance (this is required for SEU-I operation [1]) which results in higher delays when driving the output, as in, the output POUT during a 01, 10 or 11 input. This transistor feedback also provides a means by which the faulted output can return and intensify the faulting effect. An example would be a drain-to-source short on T1 with an input of 10. As the short's resistance decreases, POUT gets pulled up from its ground level causing the feedback transistor T6 to start turning "on". With this event NOUT gets pulled up from its ground level causing the feedback transistor T5 to start turning "off". The result of T5 not being fully "on" leads to an increase of POUT, thus intensifying the fault.

Since SEU-I logic is driven by SEU-I logic, an increase in delay and resistive range occurs. This can be seen in the gate-to-source and gate-to-drain shorts. For example, consider the gate-to-source short on T1 for the input 00. As the simulated resistance is decreased, T1 begins to shut "off", causing the rise time to increase. For traditional CMOS, T1's gate is being driven low by a strong (lower channel resistance) NMOS transistor in the preceeding inverter. In the case of the SEU-I CMOS, T1's gate is being driven low by a weak (higher channel resistance) PMOS transistor. The result of being driven by a weak transistor is an increased rise time since it takes longer for the driving transistor to pull the driven transistor (T1) low. The high SEU-I delay is not confined to only faulted conditions, as can be seen from the fault free values. SEU-I logic transistors are driven "on" by weak gates which pass poor signals. An example would be the PMOS transistor T1, the gate of which is driven low by a weak PMOS transistor in the preceeding logic gate. PMOS transistors do not pass good zeros. Thus T1 is never fully "on". The increase in resistive range, as compared to the traditional CMOS, is due to the resistive ratio between the short of the driven transistor and the channel of the driving transistor. A higher channel resistance must be matched by a corresponding increase in the short resistance in order to maintain the same voltage level at the gate. This means that traditional CMOS, because it drives the output with "strong" transistors, can overcome, or tolerate, lower resistance shorts.

It is important to note that resistive shorts will also effect the gate being driven. For example, if T6's channel resistance increases due to a short then a gate being driven by it will experience a much greater delay time on its output. This can propagate through the circuit.

6 Summary

As the results show, SEU-I CMOS logic is more sensitive to resistive shorts than its traditional CMOS counterpart. It is, however, important to note that realistic manufacturing faults may not span from zero to infinity. That is to say that resistive faults may only occur for a small range of values. Consequently, the fact that the SEU immune logic is more sensitive may not be a concern. The higher rise time is, on the other hand, a concern in the

area of performance since a circuit is dependent on its worst case time. A reduction of the RC time constant could be achieved by strengthening the feedback transistors. However, strengthening the feedback transistors affects the transient suppression abilities [1].

This research has only covered the narrow field of resistive faults. The author wishes to continue exploring other manufacturing faults such as bridging, stuck-at's and stuck-on's to make a more complete judgment on the logic's ability to tolerate such defects.

7 Acknowledgments

The authors would like to thank the members of the MRC for the help provided in understanding and the support.

References

[1] John Canaris. An SEU immune logic family. In *3rd NASA Symposium on VLSI Design*, pages 2.3.1–2.3.11. University of Idaho, October 1991.

[2] F. Joel Ferguson and Tracy Larrabee. Feasibility study on the costs of IDDQ testing in CMOS circuits. *submitted to the International Journal on VLSI Design special issue*, September 1993.

[3] Hong Hao and Edward J. McCluskey. "Resistive shorts" within CMOS gates. In *International Test Conference*, pages 292–301, 1991.

[4] Wojciech Maly. Realistic fault modeling for VLSI testing. In *24th ACM/IEEE Design Automation Conference*, pages 173–180, 1987.

[5] C. Timoc et al. Logical models of physical failures. In *International Test Conference*, pages 546–553, 1983.

[6] Hank Walker and Stephen W. Director. VLASIC: A catastrophic fault yield simulator for integrated circuits. *IEEE Transactions on Computer-Aided Design*, CAD-5(4):541–556, October 1986.

[7] Neil Weste and Kamran Eshraghian. *Principles of CMOS VLSI Design*, chapter 2, pages 51–53. Addison-Wesley, 1988.

Dimension Scaling Effects on the Yield Sensitivity of HEMT Digital Circuits

Jogendra C. Sarker and John E. Purviance
NASA Space Engineering Research Center,
Department of Electrical Engineering
University of Idaho, Moscow, ID 83843
jsarker@denali.ee.uidaho.edu

Abstract - **In our previous works, using a graphical tool, yield factor histograms, we studied the yield sensitivity of High Electron Mobility Transistors (HEMT) and HEMT circuit performance with the variation of process parameters. This work studies the scaling effects of process parameters on yield sensitivity of HEMT digital circuits. The results applied to two HEMT circuits are presented.**

1 Introduction

The GaAs High Electron Mobility Transistor (HEMT) is being considered for VLSI circuit design for space applications because of its high speed and its tolerance to many forms of radiation. The digital HEMT circuit performances are strongly dependent on the process parameter variations and on the device dimensions. So, the study of circuit performances and their sensitivities to process parameter variations and to device dimension scalings is very important to avoid failure of costly and time-consuming circuit designs.

In our previous work [1], we studied the statistical sensitivities of the HEMTs to process parameter variations. Then in our second phase of the work [2], we designed two digital circuits with these HEMTs and studied the yield sensitivity to process parameter variations. For the HEMT yield sensitivity study, we statistically varied three parameters, gate length, L, gate width, Z and the carrier mobility, μ. It is important to study the yield and the yield sensitivity with more parameter variations like parasitic resistances and the threshold voltage and also observe the effect of HEMT sizing on yield and the yield sensitivity. So, the purpose of this work is to study the circuit yield sensitivity as a function of more parameter variations and also to study the HEMT sizing effect on the circuit yield sensitivity.

2 The HEMT Model and the Dimension Scaling

The basic structure of a uniformly doped AlGaAs/GaAs HEMT is shown in Figure 1. The dc model used in this work is taken from reference [3]. This model uses the Trofimenkoff type carrier velocity-field relationship to model the dc I-V characteristics for both normal and compressed transconductance regions. In the model, three parameters which are directly related to the physical parameters are needed to describe the I-V characteristics in the normal transconductance region. These are shown below:

$$A = \frac{Zq\beta\mu}{L} \tag{1}$$

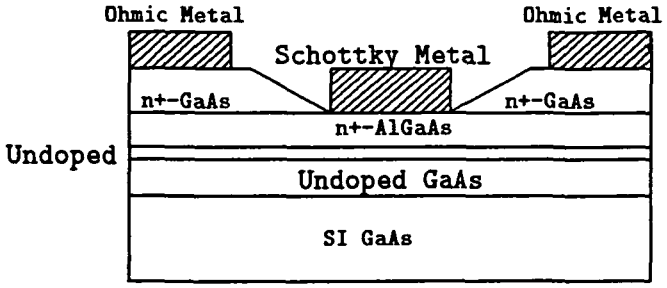


Figure 1: Structure of a Uniformly Doped AlGaAs/GaAs HEMT.

$$B = LE_c \tag{2}$$

$$C = \frac{L^2}{2\epsilon Z v_s \delta} \tag{3}$$

where β, E_c, ϵ, v_s and δ are the charge control coefficient, saturation electric field, permittivity of AlGaAs, saturation velocity and effective width of the conduction channel respectively. This dc model was validated in reference [4].

¿From the dc model, we have analytically derived the expressions for the small-signal parameters such as transconductance, g_m, gate to source capacitance, C_{gs} and the current gain cutoff frequency, f_T in the normal linear region. The expressions are given below:

$$g_m = \frac{\partial I_D}{\partial V_G} = \frac{AV_D}{1 + V_D/B} \tag{4}$$

$$C_{gs} = \frac{\partial Q_T}{\partial V_G} = \frac{AL^2(2 + V_D/B)}{\mu} \tag{5}$$

$$f_T = \frac{g_m}{2\pi C_{gs}} = \frac{\mu V_D}{2\pi L^2(1 + V_D/B)(2 + V_D/B)} \tag{6}$$

In the normal saturation region, we have applied numerical techniques to compute these circuit parameters. See reference [4] for details.

Another parameter "BETA" which is used in the JFET SPICE circuit model is calculated from the relation

$$BETA = \frac{g_m}{2(1 + \lambda V_D)(V_G - V_{th})} \tag{7}$$

where λ is the channel length modulation parameter.

In the dc or the small-signal model, the parasitic effects are not included explicitly. This effect was taken into account in the model by solving the nonlinear equations which are given below:

$$V_{GS} = V_G + I_D(V_G, V_D)R_S \tag{8}$$

$$V_{DS} = V_D + I_D(V_G, V_D)(R_S + R_D) \tag{9}$$

where V_{GS} and V_{DS} are the externally applied bias voltages and R_S and R_D are the parasitic source and the drain resistances.

Parameter	Normal Size Value	Scaled Size Value
L(μm)	0.35	0.35
Z(μm)	65	10
$\mu(m^2/Vs)$	0.44	0.44
$V_{tho}(V)$	-0.017	-0.017
$A(mA/V^2)$	49.517	7.618
B(V)	5.285	5.285
$C(K\Omega)$	8.341	54.216
D	0.015	0.015
$R_S(\Omega)$	5.9	38.35
$R_D(\Omega)$	6.0	39.0

Table 1: Physical and Model Parameters of the TRW #2078 HEMT.

In the subthreshold region of operation of the HEMT, a model parameter, D is used to model the threshold voltage shift of the two dimensional electron gas caused by the drain voltage. This threshold voltage correction is shown by the equation

$$V_{th} = V_{tho} - D \times V_D \tag{10}$$

The physical and the model parameters used for the TRW #2078 HEMT are given in Table 1 [3].

2.1 Scaling Rules for the HEMT

As we mentioned earlier, we did not consider the dimension sizing of the HEMT in the statistical sensitivity study of our previous works. Scaling rules for the HEMT can be arrived at from the analytical expressions for the parameters or from the measurements of different size HEMTs from the same foundry. Most of the HEMT small-signal parameter values are scaled up or down based on the gate width [5].

The drain source current, I_D and the depletion charge are proportional to the gate width, Z; so the partial derivative of these quantities, such as $g_m = \frac{\partial I_D}{\partial V_G}$, $g_{ds} = \frac{\partial I_D}{\partial V_D}$, $C_{gs} = \frac{\partial Q}{\partial V_G}$ and $C_{gd} = \frac{\partial Q}{\partial V_D}$ are also expected to be proportional to gate width. In addition, because C_{ds} is a geometric capacitance from source to drain, it is also proportional to gate width.

The circuit parameters of a HEMT with gate width Z can be related to a scaled HEMT with gate width Z_1 by using a scale factor $S_1 = \frac{Z}{Z_1}$. Then the circuit parameters are scaled to some arbitrary gate width, Z_1 as follows:

$$I_D' = \frac{I_D}{S_1} \tag{11}$$

$$g_m' = \frac{g_m}{S_1} \tag{12}$$

$$g'_{ds} = \frac{g_{ds}}{S_1} \tag{13}$$

$$C'_{gs} = \frac{C_{gs}}{S_1} \tag{14}$$

$$C'_{gd} = \frac{C_{gd}}{S_1} \tag{15}$$

$$C'_{ds} = \frac{C_{ds}}{S_1} \tag{16}$$

Scaling rules for the parasitic resistances R_S and R_D are different from the above parameters. These rules can be derived from simple Ohm's law: $R = \frac{V}{I}$. For either resistor

$$R_{S,D} = \frac{V}{I_D} \tag{17}$$

where V is the voltage drop across the resistor and I_D is the drain current that is proportional to gate width. Using the scale factor defined above, R_S and R_D can be scaled as

$$R'_S = R_S S_1 \tag{18}$$

$$R'_D = R_D S_1 \tag{19}$$

As we see from the above equations, the parasitic resistances are inversely proportional to the gate width. It can be shown from the definition of f_T, it is independent of gate width.

In this work, we have used the $65\mu m$ normal gate width HEMTs in the example circuits. We have used a scale factor of 6.5 to reduce the gate width to $10\mu m$ so that the HEMTs could be used for the VLSI design. The last column of Table 1 shows the scaled parameters.

A computer simulation program was developed to calculate circuit parameters such as drain current, transconductance, "BETA", gate-to-source capacitance and the current gain cutoff frequency for both the normal and scaled HEMT. In Table 2, we present the simulation results for the linear and the saturation regions. The bias conditions were chosen arbitrarily. As we see from the table, the results agree excellently. For all cases, the scaling factor is close to 6.5 as we expected.

3 Statistical Analysis

In our statistical analysis, we used the Monte Carlo technique and a computer program called SPICENTER developed at the University of Idaho. Mathematical development of the yield factor histograms and the yield sensitivity to process parameter variations can be found elsewhere [1,2,6]. Here we briefly describe the method we used in our work.

3.1 Monte Carlo Simulator

In this work, the scaled TRW #2078 HEMT was used to implement the example circuits. This HEMT shows only the normal transconductance effect. To describe this effect, six

Circuit Parameter	Bias Condition	Normal Size	Scaled Size
$I_D(\mu A)$	$V_{GS} = 0.5V, V_{DS} = 0.4V$	5091	783
$g_m(mS)$	$V_{GS} = 0.5V, V_{DS} = 0.4V$	15.79	2.43
$BETA(mA/V^2)$	$V_{GS} = 0.5V, V_{DS} = 0.4V$	16.05	2.47
$C_{gs}(fF)$	$V_{GS} = 0.5V, V_{DS} = 0.4V$	28.46	4.38
$f_T(GHz)$	$V_{GS} = 0.5V, V_{DS} = 0.4V$	88.32	88.34
$I_D(\mu A)$	$V_{GS} = 0.2V, V_{DS} = 1.0V$	1376	212
$g_m(mS)$	$V_{GS} = 0.2V, V_{DS} = 1.0V$	10.87	1.67
$BETA(mA/V^2)$	$V_{GS} = 0.2V, V_{DS} = 1.0V$	24.20	3.72
$C_{gs}(fF)$	$V_{GS} = 0.2V, V_{DS} = 1.0V$	6.51	1.00
$f_T(GHz)$	$V_{GS} = 0.2V, V_{DS} = 1.0V$	265.58	265.58

Table 2: Circuit Parameters of the TRW #2078 HEMT Calculated in this Work.

process related parameters such as L, Z, μ, V_{tho}, R_S and R_D are needed. In our Monte Carlo simulation, we have chosen a $\pm 5\%$ uniform and uncorrelated variation of these parameters about their nominal values listed in Table 1. In the simulation, we varied these parameters randomly at the same time. The random parameter values are used in the model program to calculate the circuit parameters g_m, C_{gs} and "BETA" for a particular bias condition. To create 1000 statistical HEMTs for normal size or for scaled size HEMT, 1000 simulations are performed. These 1000 HEMT circuit parameter values form a "TRUTH MODEL" [6] input to the JFET SPICE circuit model. Gate-to-drain capacitance C_{gd} is very small compared to the C_{gs}, so in the SPICE model we keep this capacitance constant.

For the Monte Carlo analysis, the program SPICENTER takes each set of parameter values created from the model program and computes the circuit performances like delay time, rise time, fall time etc. The circuit performance calculated from each simulation are compared with the nominal performance specification. If the circuit meets the specification then the circuit is accepted otherwise it is rejected. From the accepted circuits, the program creates the yield factor histograms for each of the independent parameters, L, Z, μ, R_D, R_S and V_{tho}. Our analysis technique is summarized in Figure 2. Details are to be found elsewhere [1,7].

The yield sensitivity was determined by linear fitting the yield factor histograms. The slope of each fit determines the yield sensitivity with respect to a particular parameter. These yield sensitivities are presented as yield%/parameter% and also as yield% per each unit dimension.

4 Results

In this work, we have chosen two example HEMT circuits to study the dimension scaling effects on the circuit performances and their yield sensitivities.

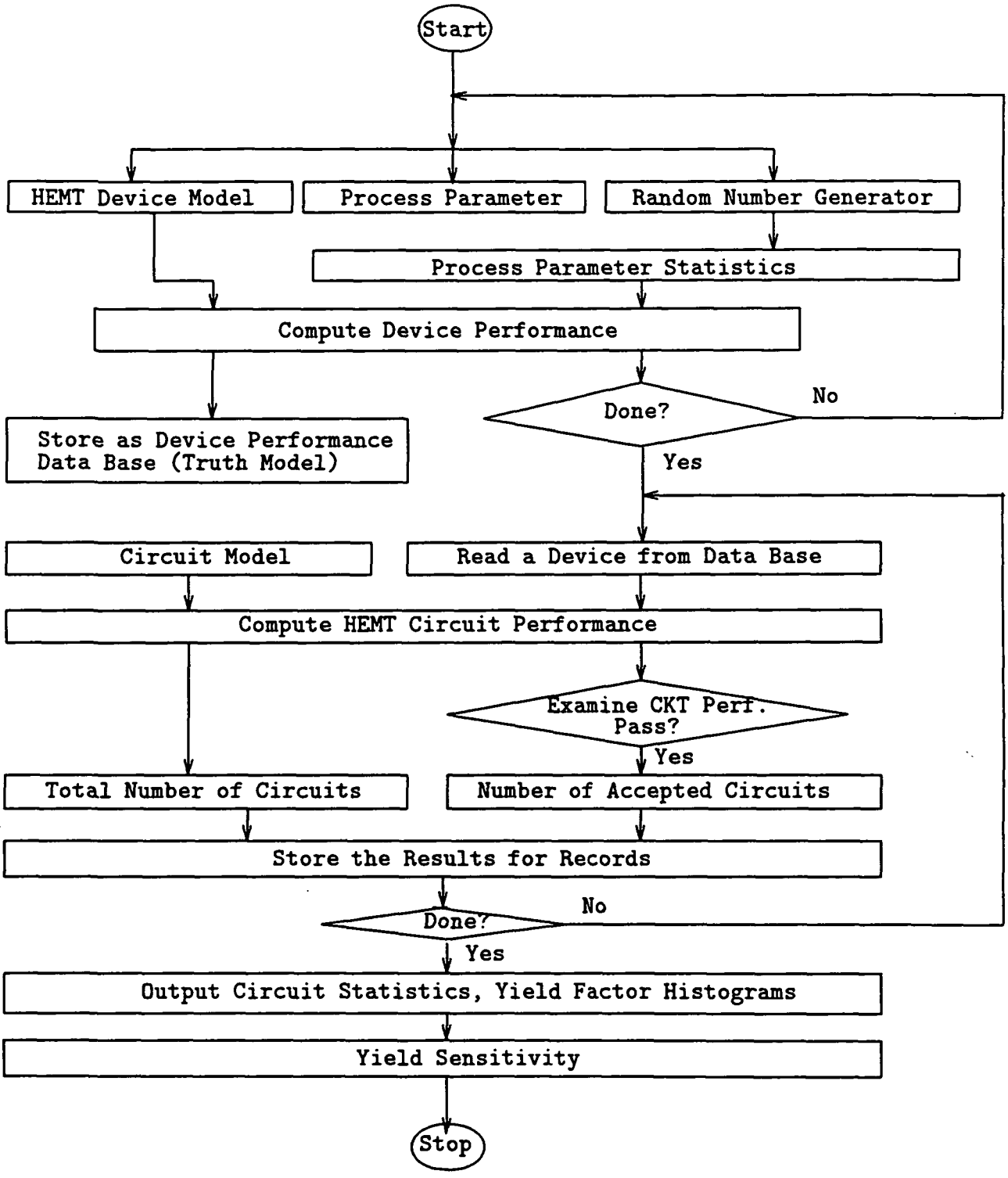


Figure 2: Algorithm of the Monte Carlo Simulator.

Parameter	Normal Size Yield Sensitivity (yield%/parameter%)	Normal Size Yield Sensitivity	Reduced Size Yield Sensitivity (yield%/parameter%)	Reduced Size Yield Sensitivity
Param. 1: L	-9.14 ± 0.49	$-2602.56 \pm 137.14\%/\mu m$	-9.81 ± 0.42	$-2794.87 \pm 120.92\%/\mu m$
Param. 2: Z	$+6.33 \pm 0.44$	$+9.74 \pm 0.67\%/\mu m$	$+5.29 \pm 0.49$	$52.95 \pm 4.87\%/\mu m$
Param. 3: μ	$+7.77 \pm 0.45$	$+1761.91 \pm 102.60\%/m^2/Vs$	$+7.79 \pm 0.59$	$+1765.31 \pm 132.94\%/m^2/Vs$
Param. 4: R_D	-0.90 ± 0.83	$-15.00 \pm 13.74\%/\Omega$	-1.05 ± 0.84	$-2.69 \pm 2.16\%/\Omega$
Param. 5: R_S	-2.19 ± 0.50	$-37.13 \pm 8.56\%/\Omega$	-1.47 ± 0.44	$-3.83 \pm 1.15\%/\Omega$
Param. 6: V_{tho}	-0.07 ± 0.50	$+0.44 \pm 2.90\%/mV$	$+0.36 \pm 0.45$	$+2.11 \pm 2.64\%/mV$

Table 3: Yield Sensitivity of the FFL NOR Gate.

4.1 Example I: 2-Input NOR Gate with Feedback FET Logic

Our first example circuit is a 2-input NOR gate with feedback FET Logic (FFL). This high-speed and low power feedback FET logic is based upon a push-pull output stage with a feedback inverter that shuts off the output current after the logic high state is reached. It is two to four times faster than the comparable GaAs direct-coupled FET logic. Figure 3 shows the circuit diagram of this FFL NOR gate [8]. All of the transistors in the circuit are n-channel type transistors. Using one type of transistors raises yield and reduces process complexity.

We have chosen delay time as the performance criterion for this NOR gate. In our analysis, the delay time is defined as the time taken at the output to reach 90% of its final steady state value from the instant the input is triggered with a step voltage. The nominal delay times for both the normal and the scaled size HEMT circuits were calculated by simulating the circuit with the nominal parameter values. The delay time was found to be 35.88 ps for normal size circuit and 34.06 ps for scaled size circuit. The output load capacitance was scaled down by the same factor to calculate the delay time of the scaled size circuit.

The Monte Carlo simulations were performed first with the normal size HEMT circuit parameters. The yield was only 12.2% which is very low. To increase the yield, we relax the delay time by 5% and resimulate. With this 5% increased delay time specification we achieve a 48.1% yield. The yield factor histograms for 1000 simulations of this circuit are shown in Figure 4. As we see from the figure, the yield is very sensitive to the gate length, width and the mobility. But it is almost insensitive to the parasitic resistances and the threshold voltage. However, we found a small decrement of yield with source resistance increment.

1000 simulations were also performed for the scaled size circuit and the yield was 13.2%. So, we increase the delay time by 5% in the specification to increase the yield to 56.2%. We observe similar results as the normal size HEMT circuit.

The yield sensitivity to each parameter variation was calculated for both the circuits and are shown in Table 3. These are presented as yield%/parameter% and also as yield% per unit dimension of the parameter.

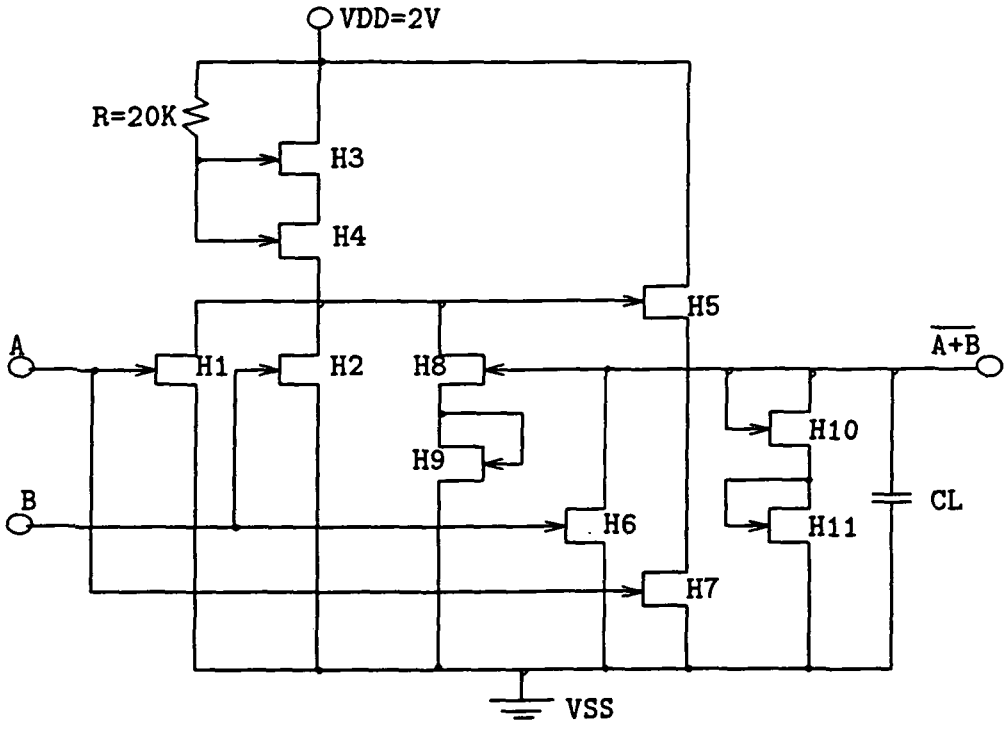


Figure 3: Circuit Diagram of an FFL HEMT NOR Gate.

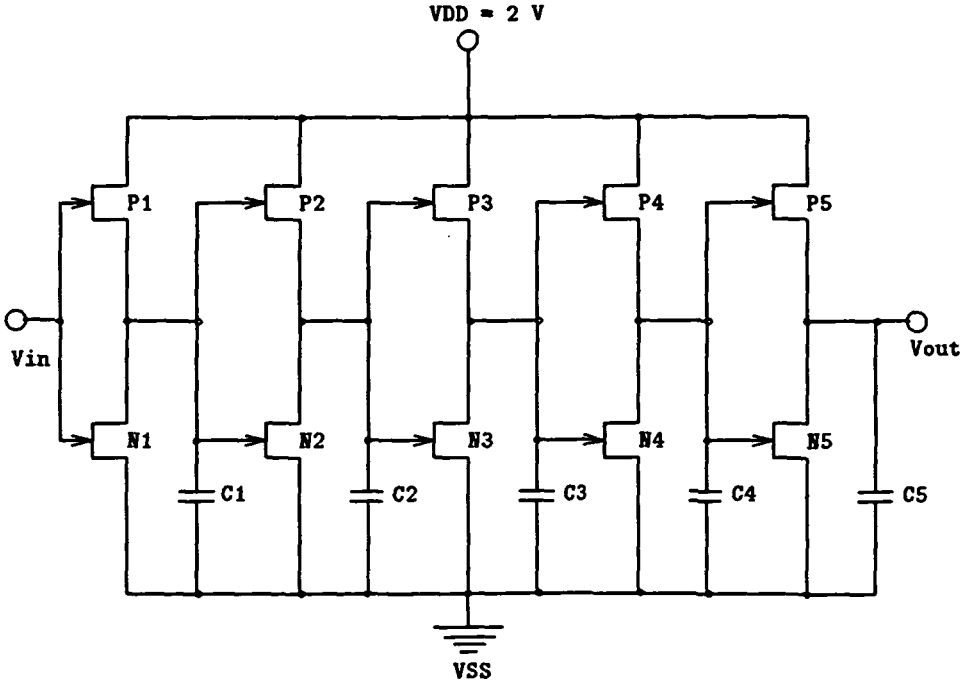


Figure 4: Yield Factor Histograms, Yield % vs. Parameter %, of an FFL NOR Gate.

Parameter	Normal Size Yield Sensitivity (yield%/parameter%)	Normal Size Yield Sensitivity	Reduced Size Yield Sensitivity (yield%/parameter%)	Reduced Size Yield Sensitivity
Param. 1: L	-7.80 ± 0.65	$-2222.22 \pm 183.60\%/\mu m$	-5.69 ± 0.59	$-1619.66 \pm 168.78\%/\mu m$
Param. 2: Z	$+2.21 \pm 0.49$	$+3.39 \pm 0.74\%/\mu m$	$+1.96 \pm 0.49$	$+19.65 \pm 4.82\%/\mu m$
Param. 3: μ	$+3.92 \pm 0.41$	$+887.76 \pm 91.74\%/m^2/Vs$	$+3.84 \pm 0.67$	$+870.75 \pm 149.91\%/m^2/Vs$
Param. 4: R_D	-1.64 ± 1.03	$-27.23 \pm 17.10\%/\Omega$	-5.29 ± 0.94	$-13.58 \pm 2.39\%/\Omega$
Param. 5: R_S	$+0.15 \pm 0.60$	$+2.53 \pm 10.26\%/\Omega$	$+3.67 \pm 0.70$	$+9.58 \pm 1.82\%/\Omega$
Param. 6: V_{tho}	$+0.52 \pm 0.58$	$+3.07 \pm 3.34\%/mV$	$+0.99 \pm 0.71$	$+5.79 \pm 4.14\%/mV$

Table 4: Yield Sensitivity of the HEMT Inverter Chain.

4.2 Example II: HEMT Inverter Chain with Complementary Logic

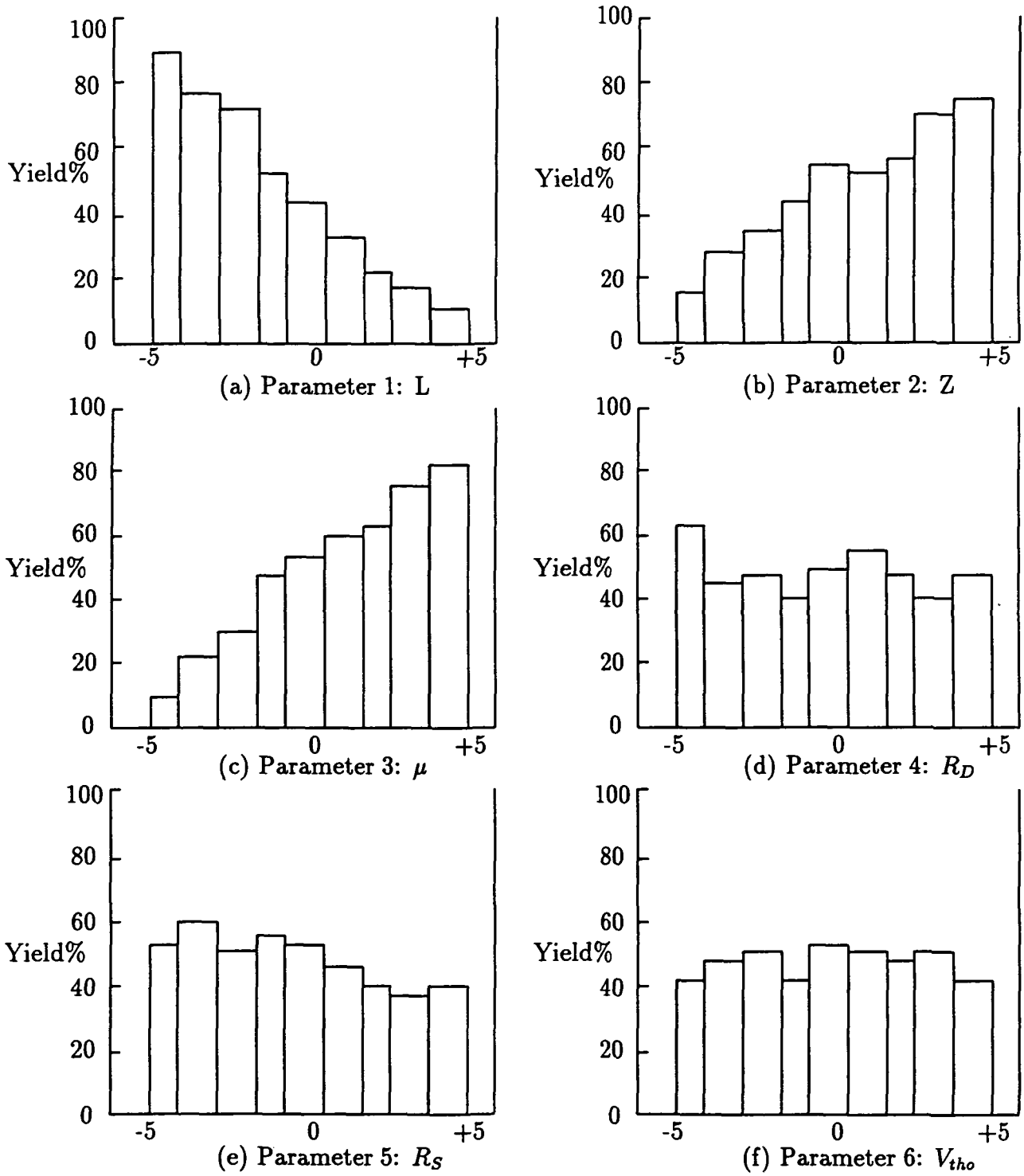
A chain of inverters mostly used in memory circuits is chosen as our second example circuit. We propose a chain of five inverters implemented with complementary logic [9] and the circuit is shown in Figure 5. In the circuit, P1-P5 are the p-channel and N1-N5 are the n-channel depletion mode HEMTs. The capacitances C1 through C5 are the output capacitances for each stage which are arbitrarily chosen to 0.1pF for the normal size circuit. A step input is applied and output is observed at the capacitor C5.

For this circuit, we have chosen rise time of the output at C5 as the performance criterion. The rise time calculated from the nominal values of the parameters were 20.44 ps for normal size and 23.77 ps for the scaled size circuit. Here also, we scale all capacitors down by the same factor.

1000 Monte Carlo simulations were performed first for the normal size HEMT and then for the scaled size HEMT inverters. The yields were not high enough, so we relax the specification by increasing the rise time of 5% and simulate again. With the new specification the yield went up to 52.1% and 56.8% for normal and scaled size circuit respectively. In Figure 6, we present the yield factor histograms of the Scaled size HEMT circuit with 5% relaxed rise time specification. For this circuit both in normal and scaled size, the yield and the yield sensitivity as a function of L, Z, μ and V_{tho} variation is similar as we observed for circuit I. The yield is not very sensitive to the parasitic resistances R_S and R_D for the normal size circuit but it is sensitive for the scaled size circuit. Yield increases with R_S and decreases as R_D increases. As a verification of our result, we choose fall time as performance specification in the Monte Carlo simulation and perform 1000 runs and we observed a consistent result as we expected. That is, the yield is higher for higher R_D value and lower for lower R_S value. The yield sensitivity of both normal and scaled HEMT inverters are shown in Table 4.

5 Conclusion

In this work, we first studied the dimension scaling rules of the HEMT circuit parameters. Results of our study are given in Table 2. Then we used the yield factor histograms graphical tools to study the dimension scaling of the HEMTs on the yield and the yield sensitivity of two circuits. From our study, we found, for the NOR gate, that the scaling of the gate width



(a) Parameter 1: L

(b) Parameter 2: Z

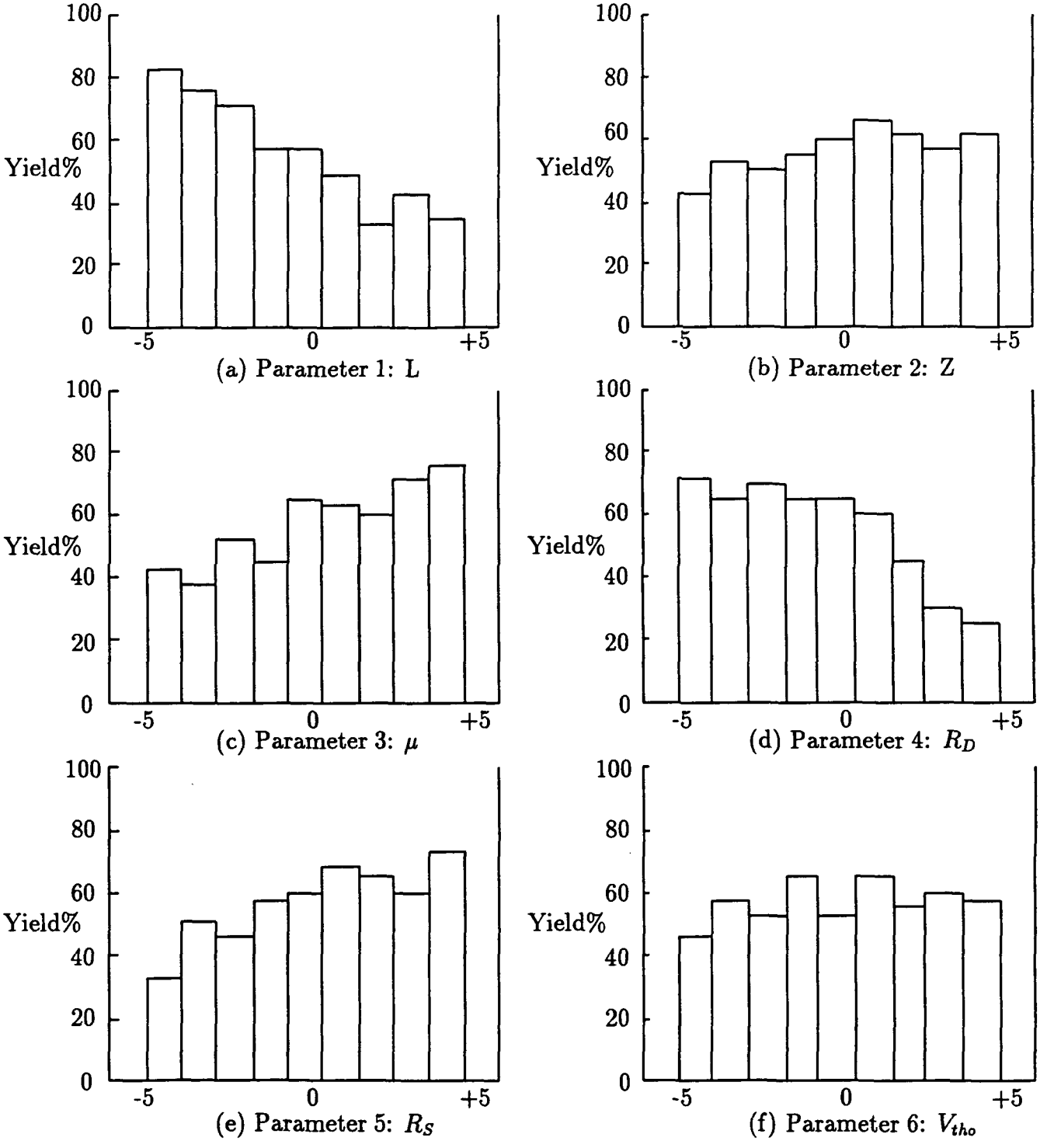
(c) Parameter 3: μ

(d) Parameter 4: R_D

(e) Parameter 5: R_S

(f) Parameter 6: V_{tho}

Figure 5: HEMT Inverter Chain with complementary Logic.



(a) Parameter 1: L

(b) Parameter 2: Z

(c) Parameter 3: μ

(d) Parameter 4: R_D

(e) Parameter 5: R_S

(f) Parameter 6: V_{tho}

Figure 6: Yield Factor Histograms, Yield % vs. Parameter %, of HEMT Inverter Chain.

and the parasitic resistances do not change the yield and the yield sensitivity significantly. The yield and the yield sensitivity of the inverter chain do not change much with the scaling of the HEMT width. But yield is more sensitive to R_S and R_D for the scaled HEMT inverter.

6 Acknowledgement

The authors thank the NASA Space Engineering Research Center and the Chairman of the Electrical Engineering Department for partial funding of this project. One of us (J.E.P) acknowledges Jesus of Nazareth for His sacrifice and example.

References

[1] J.C. Sarker and J.E. Purviance, "Yield Sensitivity Study of AlGaAs/GaAs High Electron Mobility Transistor," *International Journal of Microwave and Millimeter-Wave Computer-Aided Engineering*, Vol. 2, Jan. 1992, pp. 12-27.

[2] J.C. Sarker and J.E. Purviance, "Yield Sensitivity of HEMT Circuits to Process Parameter Variations," *IEEE Transactions on Microwave Theory and Techniques*, Vol. 40, July 1992, pp. 1572-1576.

[3] G.W. Wang and W.H. Ku, "An Analytical and Computer- Aided Model of the AlGaAs/GaAs High Electron Mobility Transistor," *IEEE Transactions on Electron Devices*, Vol. ED-33, May 1986, pp. 657-663.

[4] J.C. Sarker and J.E. Purviance, "DC and Small-Signal Physical Models for the AlGaAs/GaAs High Electron Mobility Transistor," *3rd NASA Symposium on VLSI Design*, UI Moscow, Oct. 1991.

[5] J. Michael Golio, *Microwave MESFETs and HEMTs*, Artech House, Boston, 1991.

[6] M. Meehan and T. Wandinger, "Accurate Design Centering and Yield Prediction Using the "Truth Model"," *IEEE Microwave Theory and Techniques Symposium Digest*, June 1991, pp. 1201-1204.

[7] A. MacFarland, J. Purviance, D. Loescher, K. Diegert, and T. Furguson, "Centering and Tolerancing the Components of Microwave Amplifiers," *IEEE Microwave Theory and Techniques Symposium Digest*, June 1987, pp. 633-636.

[8] D.E. Fulkerson, "Feedback FET Logic: A Robust, High-Speed, Low-Power GaAs Logic Family," *IEEE Journal of Solid-State Circuits*, Vol. 26, Jan. 1991, pp. 70-74.

[9] M. Shur, *GaAs Devices and Circuits*, Plenum Press, New York, 1987.

Links between N-Modular Redundancy and the Theory of Error-Correcting Codes

V. Bobin*, S. Whitaker** and G. Maki**

* California Design Center
Hewlett-Packard Co.
5301 Stevens Creek Blvd.
Santa Clara, California 95051

** NASA Space Engineering Research Center
University of New Mexico
2650 Yale SE, Suite # 101
Albuquerque, New Mexico 87106

Abstract – **N-Modular Redundancy (NMR) is one of the best known fault tolerance techniques. Replication of a module to achieve fault tolerance is in some ways analogous to the use of a repetition code where an information symbol is replicated as parity symbols in a codeword. Linear Error-Correcting Codes (ECC) use linear combinations of information symbols as parity symbols which are used to generate syndromes for error patterns. These observations indicate links between the theory of ECC and the use of hardware redundancy for fault tolerance. In this paper, we explore some of these links and show examples of NMR systems where identification of good and failed elements is accomplished in a manner similar to error correction using linear ECC's.**

1 Introduction

In a repetition code, the information symbol is replicated as parity symbols for the purpose of error detection and correction. An NMR system with voting also has the same purpose; ie., tolerate failures in some modules. In an NMR system, voting can be replaced by Galois Field (GF) arithmetic operations to identify the good and failed modules from the results of these operations, just as GF arithmetic operations are used to identify and correct the erroneous symbols in a received codeword, provided the number of errors or failed modules is within the correction capability of the code. Some necessary results from the theory of linear ECCs are reviewed in the next section [1, 2].

2 Useful Principles from Coding Theory

This paper attempts to adapt some results from coding theory for use in fault tolerance using modular redundant systems. Some definitions and results from coding theory [1, 2] are reproduced below.

Definition 1 *A* block code *of size* M *over an alphabet with* q *symbols is a set of* M q*-ary sequences of length* n *called* codewords.

If $q = 2$, the symbols are called *bits* and the code is a binary code. Usually $M = q^k$ for some integer k, and the code is an (n, k) code. Each sequence of k q-ary information symbols is associated with a sequence of n q-ary symbols making a codeword.

C-3

Definition 2 *The* Hamming distance d(x, y) *between two q-ary sequences x and y of length n is the number of places in which they differ.*

Definition 3 *Let $C = c_i$, $i = 0, \ldots, M-1$ be a code. Then the* minimum distance *of C is the Hamming distance of the pair of codewords with smallest Hamming distance.*

Suppose that a codeword is transmitted, and some symbols are corrupted by the channel. If t errors occur, and if the distance from the received word to every other codeword is larger than t, then the decoder will properly correct the errors, if it presumes that the closest codeword to the received word was actually transmitted. This always occurs if $d \geq (2t+1)$.

Within the space of all q-ary n-tuples, a set of n-tuples is selected and the elements are designated as codewords. If d is the minimum distance of this code and t is the largest integer satisfying $d \geq 2t+1$, then nonintersecting spheres of radius t can be drawn about each of the codewords. A received word that falls in a sphere is decoded as the codeword at the center of that sphere. If t or fewer errors occur, then the received word is always in the proper sphere, and the decoding is correct. Some received words with more than t errors will be within the decoding spheres of other codewords and will be decoded incorrectly. Other received words with more than t errors will lie in the interstitial space between decoding spheres. (Notice that if the minimum distance is $(2t+1)$, a received word with $2t$ or fewer errors can be recognized as being in error, since it will be a non codeword.)

There is a class of codes called *linear codes* which are defined by imposing strong structural property on the codes. This class includes most of the known good codes. Good codes are those which have good error-correction and detection capabilities. The structure helps in the search for good codes as well as in the design of encoders and decoders. There are good codes that are not linear, but non linear codes are hard to deal with because of their lack of structure. The theory of linear codes involves the concept of *vector spaces*. From group theory, it is known that under componentwise vector addition and componentwise scalar multiplication, the set of n-tuples of elements from *GF(q)* (the Galois Field with q elements), is a vector space called $GF(q^n)$. A special case of major importance is $GF(2^n)$, the vector space of all binary n-tuples with two such vectors added by modulo-2 addition in each component.

Definition 4 *A* linear code *is a non-empty set of n-tuples over $GF(q)$ called codewords such that the sum of two codewords is a codeword and the product of any codeword by a field element is a codeword.*

There are q^k codewords, each n symbols long out of which k are information symbols and $(n-k)$ are parity symbols added for error-correction purposes. Another way of defining a linear code is that it is a subspace of $GF(q^n)$.

It follows from the theory of vector spaces that any (n, k) linear code can be represented by its *generator matrix* which is a k by n matrix with the property that any codeword is a linear combination of the rows of this matrix. The generator matrix is a concise way to describe a linear code. The generator matrix is formed by using any set of basis vectors for the subspace as its rows. Any one-to-one pairing of k-tuples and codewords can be used as the encoding procedure, but the most natural approach is to use the equation

$$\mathbf{c} = \mathbf{iG} \tag{1}$$

where, $\mathbf{i}$, the information word, is a k-tuple of information symbols to be encoded and $\mathbf{c}$ is the codeword n-tuple. $\mathbf{G}$ is the generator matrix.

Obviously the parity symbols can be generated as a linear combination of some of the information symbols. Every generator matrix $\mathbf{G}$ is equivalent to one with a k by k identity in the first k columns. That is,

$$\mathbf{G} = [\mathbf{I}\vdots\mathbf{P}] \tag{2}$$

and $\mathbf{P}$ is a k by $(n-k)$ matrix. We call this the *systematic form* of the generator matrix. This generator matrix leads to systematic codes, where the information symbols appear as a block in the first k positions of the codeword, and the parity symbols appear as a block in the next $(n-k)$ positions.

Associated with every generator matrix is a *parity check* matrix $\mathbf{H}$, defined such that

$$\mathbf{G}\mathbf{H}^{\mathbf{T}} = \mathbf{0}. \tag{3}$$

For the systematic form of $\mathbf{G}$, an appropriate definition of a systematic parity check matrix is

$$\mathbf{H} = [-\mathbf{P}^{\mathbf{T}}\vdots\mathbf{I}] \tag{4}$$

where $\mathbf{I}$ is an identity matrix of dimension $(n-k)$, because then $\mathbf{G}\mathbf{H}^{\mathbf{T}} = \mathbf{0}$.

When a codeword is transmitted through a communication channel, it can become corrupted. The effect is the same as adding an n-tuple called the *error pattern* to the codeword. The $GF(q)$ sum of the codeword and error pattern gives the received word. If it is assumed that the number of errors is within the error-correction capability of the code, each correctable error pattern gives rise to a unique *syndrome*. The syndrome is generated from the received word by $GF(q)$ operations. Thus the syndrome helps to recover the original codeword from its corrupted received version. One way of generating the syndrome from the received word is to take its product with the transpose of the parity check matrix, ie.,

$$\mathbf{S} = \mathbf{r}\mathbf{H}^{\mathbf{T}} \tag{5}$$

where $\mathbf{S}$ is the syndrome corresponding to the received word $\mathbf{r}$. From the structure of $\mathbf{H}$, the structure of $\mathbf{H}^{\mathbf{T}}$ can be seen to be

$$\begin{bmatrix} -\mathbf{P} \\ \cdots \\ \mathbf{I} \end{bmatrix}$$

where, $\mathbf{I}$ is an identity matrix of dimension $(n-k)$. The structure of $\mathbf{H}^{\mathbf{T}}$, shows that the syndrome can be generated as the vector sum of the received parity symbols and the *additive inverse* of the parity symbols recomputed from the received information symbols. For any extension field of GF(2), the additive inverse of an element is the element itself.

3 NMR as a Repetition Code

Consider a repetition code, where there is one information symbol and $(N-1)$ parity symbols, which are copies of the information symbol. This is an $(N,1)$ repetition code. The codewords

are [0 0 ... 0] and [x x ... x], where x is any member from the code alphabet with q symbols. This is a special case of linear codes. This code has a minimum distance of N and hence can correct t errors where $N \geq (2t+1)$. Consider a received word $[r_1\, r_2 \ldots r_N]$. From the theory of linear ECC, we can find the syndrome corresponding to this received word as follows. First, we recompute the parity symbols from the received word. Notice that for a repetition code, the parity symbols are copies of the information symbol itself. Thus the reconstructed parity symbol vector corresponds to the $(N-1)$-tuple $[r_1 r_1 \ldots r_1]$. To generate the syndrome, we find the GF(q) vector sum of the additive inverse of the recomputed parity symbol vector and the received parity symbol vector. That is, the syndrome

$$S = [(-r_1 \oplus r_2)(-r_1 \oplus r_3) \ldots (-r_1 \oplus r_N)] \tag{6}$$

where $\oplus$ represents GF(q) addition. Thus the syndrome of the error pattern is obtained by the GF(q) addition of the additive inverse of the received information symbol with each of the received parity symbols. For all correctable errors, the syndrome uniquely determines the error pattern, ie., it shows which symbols are in error and by how much.

An NMR system resembles a repetition code in that the module is replicated just as the information symbol is replicated in the repetition code. The modules in an NMR system can be either good or faulty. A faulty module can, in general, take an undefined state but the module output is always equivalent to an element from GF(q), where $q = 2^m$, m being the number of module output lines. In other words, we are representing the module outputs as GF(q) symbols. Imagine that the first module in an NMR system corresponds to the information symbol of a repetition code, and the other modules correspond to the parity symbols. Consider that the first module is GF(q) added with every other module using $(N-1)$ two-input GF(q) adder units. Assume that the adder units are fault-free. This system is shown in Figure 1, where the c_i's represent GF(q) addition.

The following theorem can be proved. Some terms are defined first.

Definition 5 *A Module Fault Pattern is an N-bit vector over GF(2), with weight less than or equal to* $\lfloor \frac{N-1}{2} \rfloor$, *where 1's represent faulty modules and 0's represent good modules by position in an NMR system. The most significant position represents the first module and the least significant position represents the last module.*

Definition 6 *A Fault Signature is a vector over GF(q) whose symbols are the outputs of the GF(q) adders for a particular fault pattern. The symbols of the fault signature are assigned by position; they depend on the physical position of the corresponding adder in the system.*

Theorem 1 *Assuming fault-free adders in an NMR system where a particular module output is GF(q) added with the outputs of all other modules as in Figure 1, there is a unique correspondence between all module fault patterns of* $\lfloor \frac{N-1}{2} \rfloor$ *faulty modules or less, and fault signatures, so that the faulty and fault-free modules can be identified.*

Proof:

Consider the equivalent (N,1) repetition code. This code has a minimum distance of N. Therefore it can generate unique syndromes for all error patterns of $\lfloor \frac{(N-1)}{2} \rfloor$ errors or less. The NMR system is configured such that the first module output is GF(q) added with

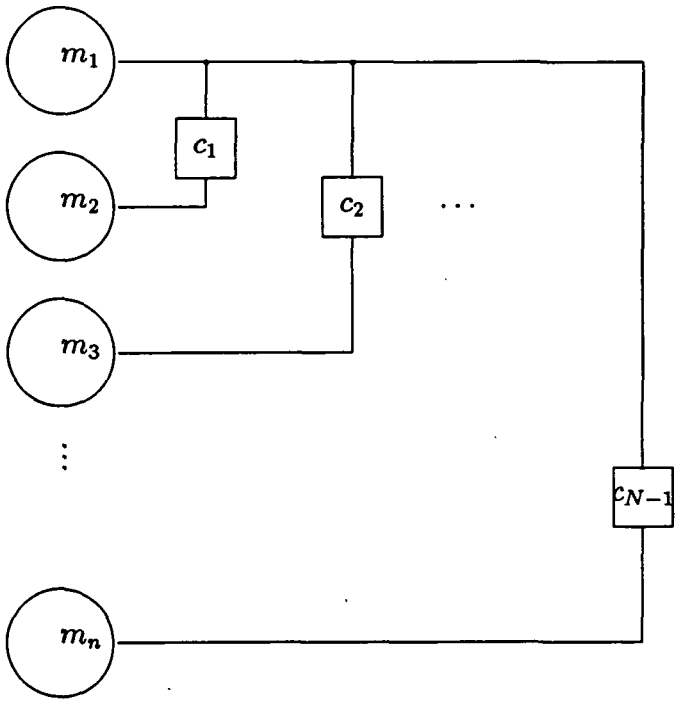


Figure 1: Star-Connected NMR System

every other module output to generate the fault signature. The syndrome of the received word is generated by the GF(q) addition of the additive inverse of the received information symbol with the received parity symbols. Since the module output lines have boolean values (elements of GF(2)), and q is equal to 2^m, GF(q) is always an extension field of GF(2). Therefore, the additive inverse of any element in GF(q) is the element itself. Thus the given configuration of the NMR system exactly determines the syndrome of the fault pattern which in turn gives the position of the failed module and what its output should have been. Therefore, the fault signature is identical to the syndrome of the corresponding error pattern or the module fault pattern. □

Thus diagnosis of an NMR system is exactly similar to error correction using a repetition code if the GF(q) arithmetic modules are fault-free. If multiple module failures are assumed to be common-mode in nature, then the code reduces to a binary code (the only elements of the alphabet are good and faulty, corresponding to 0 and 1 arbitrarily), and the GF(q) adder blocks reduce to comparators. This leads to very simple systems [3]. The same situation results if single element failures alone are considered.

4 Extending the Code to Cover Checker Units

In the previous section, the similarity between an NMR system and an (N,1) repetition code was elaborated. One of the features of the implementation was that the determination of the syndrome had to be error-free, ie., the GF arithmetic units had to be assumed fault-free. This is because the repetition code only covers the modules; it does not cover the checker units. This observation opens up a new possibility. It seems as if an encoding scheme that

somehow includes the checker units also might lead to a system which can also tolerate checker units failures. This would be a very desirable property indeed. In this section, this idea is pursued.

Consider an NMR system capable of tolerating t faulty modules. Let each module have an output bus with m lines. Thus the module outputs can be represented by the elements of GF(q), where $q = 2^m$. In other words GF(q) is an extension field of GF(2) and hence the additive inverse of an element in GF(q) is the element itself. Consider a t error-correcting linear code over GF(q) with N information symbols and p parity symbols. There will be q^N valid code words, and the minimum distance of the code is $(2t + 1)$. Each parity symbol will be a linear combination of some of the information symbols. We will constrain the generation of parity symbols thus; each parity symbol must be a GF(q) sum of an *even* number of information symbols. Assume that in an NMR system, the modules correspond to the information symbols and the checker units roughly correspond to the parity symbols. In the no-fault case for an NMR system, we are dealing with codewords whose information symbols are all the same. Notice that in all the cases, the parity symbols are all zeroes because of the way the code is constructed. Thus we know beforehand what the parity symbols should be when there are no faults. GF(q) adders are connected at the module outputs exactly the way the parity symbols are generated. The outputs of the adders are actually reconstructions of the parity symbols from the received information symbols. We know that the received information symbols (or equivalently, the output set of all the N modules) must be of the N-tuple form $[x\,x \ldots x]$ where x is any element from GF(q). All these N-tuples have the same parity symbols represented by the p-tuple $[0\,0 \ldots 0]$ because of the construction of the code. The GF(q) sum of the outputs of the p GF(q) adders and the p-tuple $[0\,0 \ldots 0]$ gives the "syndrome" of the fault pattern. If the fault pattern is correctable, ie., if t or less modules are faulty, the modules can be identified because the syndrome identifies the fault pattern. Notice that there can be faults in the adder units also (which correspond to faults in received parity symbols) as long as the total number of faulty elements does not exceed the correction capability of the code, t. A total of t faulty elements can be identified unambiguously using this scheme. This will be formalized by the following theorem.

Theorem 2 *Consider an NMR system whose module outputs are considered as elements of GF(q), an extension field of GF(2). Consider also a minimum distance N linear ECC with N information symbols and p parity symbols whose parity symbols are all GF(q) sums of an even number of information symbols. A network of p GF(q) adders is connected at the output of the NMR system, such that each adder takes as input the modules corresponding to the information symbols that make up a single parity symbol. This system gives fault signatures that uniquely identify all possible fault patterns of* $\lfloor \frac{N-1}{2} \rfloor$ *faulty elements or less.*

Proof:
The system described here corresponds to a linear code, where the module outputs correspond to the information symbols and the GF(q) adders roughly correspond to the generation of parity symbols. Also, since GF(q) is an extension field of GF(2), additive inverses of field elements are the elements themselves. Due to the structure of the code, the fault-free case corresponds to code words of the structure $[x\,x \ldots x(information N-tuple)0 \ldots 0(parity p-$

tuple)], where x is an element of GF(q). In this proof, we will assume that the number of faulty elements does not exceed the capability of the code.

Case A: Faults confined to modules

Consider a received word whose information part is identical to the N-tuple formed by the module outputs. We know apriori, that the correct parity p-tuple is the all zero p-tuple. The received (N+p)-tuple obviously is a correctable received word, as far as the code is concerned, as long as the number of corrupted symbols (module outputs) is less than or equal to the error-correction capability of the code. Notice that the parity symbols (adder outputs) recomputed from the received information symbols (module outputs) are identical to the corresponding symbols of the fault signature generated by the GF(q) adders. Since there is no error in the received parity symbols, the fault signature is identical to the syndrome of the error patter. Thus the faulty modules can be uniquely identified.

Case B: Faults confined to GF(q) adders

The parity symbols corresponding to the faulty adders are corrupted and all the other parity symbols are 0's. Consider a received word whose information part is identical to the module outputs (all correct and identical), and parity symbols are all zeroes except the ones that are in error which correspond to the locations of the failed adders. Obviously, the syndrome of the error pattern is the same as the parity part of the received word, which is identical to the fault signature. Thus the faulty GF(q) adders are identified from the syndrome.

Case C: Faults mixed among the modules and GF(q) adders

Consider a received word whose information part is identical to the module outputs and whose parity symbols are zeroes in all positions except the ones corresponding to the faulty adders. The syndrome is generated by recomputing the parity symbols and then GF(q) addition of the recomputed parity symbols with the received parity symbols. On close inspection, it can be seen that the resultant p-tuple is identical to the fault signature. The adders recompute the parity with errors corresponding to the positions of the faulty adders. This is the same as if the parity symbols were recomputed without error but the received parity symbols corresponding to the positions of the faulty adders were in error. Hence the fault signature is identical to the syndrome of the error pattern and the failed elements can be identified from the fault signature.

To conclude the proof, if there are no faulty elements, the fault signature will be the all zero vector. □

It is obvious that we are not using all the redundancy inherent in the coding structure, since we are really dealing with only a few codewords. But this is a necessary evil since we want the effects of error-correction to extend over the computation of the syndrome. This is the main difference from the theory presented in the previous section.

5 A Systematic Way of Generating Required Codes

It has been shown that we can create an NMR system that tolerates $\lfloor \frac{N-1}{2} \rfloor$ faulty elements by creating a special type of linear error-correcting code of minimum distance N. This section introduces one systematic way of doing this. This method is not unique; there could be other ways of doing this.

Theorem 3 *Consider a linear block code over GF(q), where $q = 2^m$, with q^N code words, ie., N information symbols. If ${}_NC_2$ parity symbols are appended to the information symbols such that each parity symbol is the GF(q) sum of a unique pair of information symbols, the resulting code has minimum distance equal to N.*

Proof:
We will first show that the minimum distance is at least N. Let the information symbols be represented by the N symbol vector $\hat{\mathrm{I}}$, and the parity symbols be represented by the ${}_NC_2$ symbol vector $\hat{\mathrm{P}}$. For any two code words i and j, $\hat{\mathrm{I}}_i$ and $\hat{\mathrm{I}}_j$ differ in a positions, where $0 < a \leq N$. Consider one of those differing symbol positions (For example, consider that the left most position in $\hat{\mathrm{I}}_i$ and $\hat{\mathrm{I}}_j$ are different). The parity vectors $\hat{\mathrm{P}}_i$ and $\hat{\mathrm{P}}_j$ have parity symbols corresponding to the GF(q) sum of the each information symbol with each other information symbol. $\hat{\mathrm{I}}_i$ and $\hat{\mathrm{I}}_j$ agree on $(N - a)$ positions. Consider the parity symbols generated by a symbol at a differing symbol position (for example the left most symbol in $\hat{\mathrm{I}}_i$ and $\hat{\mathrm{I}}_j$ in the example being considered) with these $(N - a)$ agreeing positions. It is clear that $\hat{\mathrm{P}}_i$ and $\hat{\mathrm{P}}_j$ differ in at least $(N - a)$ positions. But $\hat{\mathrm{I}}_i$ and $\hat{\mathrm{I}}_j$ differ in a positions. Therefore any two code words differ in at least N positions.

Now we will show that the minimum distance is not more than N. Consider the codewords whose information vectors are the all zero vector and the all x vector, where x is any element of GF(q). These two code words have the same parity vectors equal to the all zero vector (notice that GF(q) is an extension field of GF(2)). Obviously, the distance between these two codewords is N. Therefore, the minimum distance of the code is not greater than N. The two results together prove the theorem. □

An example 5MR system is shown in Figure 2. The X_i's represent GF(q) adders. In this system a maximum of two faulty elements (the code being used is a double error-correcting code) can be identified, irrespective of the positions of the faulty elements, including the adders.

6 An Example; Following a (7,4) Hamming Code

As was mentioned in Section 3, if only a single faulty element needs to be tolerated, the code reduces to a binary code, and results in easy implementation of the NMR system. Additionally, since only one failed element is anticipated, GF(2) addition on module outputs can be simulated by a comparison operation if we arbitrarily assume that the element '1' represents a failed module, and the element '0' represents a good module. In Section 4, we introduced the need to compose a parity symbol from an even number of information symbols. This was to ensure that in a fault-free NMR system following the code, a GF(q) addition on the module outputs would always yield the additive identity (**0**) since an even number of module outputs are being added in GF(q) which is an extension field of GF(2). This results in the advantageous situation where we know apriori, what the parity symbols in an error-free case would be. In the case of a single failed element represented by the element 1 of GF(2), a straight comparison operation always simulates a GF(2) addition since there can be at the most one differing input to the comparator. Also, since the fault-free element is represented by element 0 (the additive identity of GF(2)), no matter how many fault-free

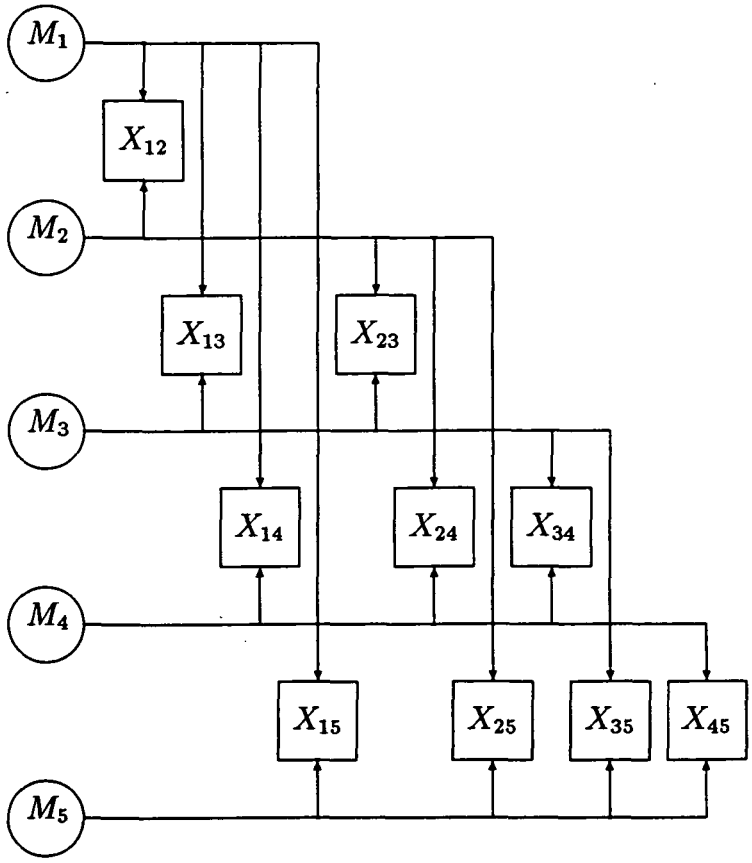


Figure 2: Linear ECC Implementation of 5MR System

Failed Element	Comparator Outputs	Corresponding Error Pattern	
	$C_1\ C_2\ C_3$	Information	Parity
	$p_1\ p_2\ p_3$	$i_1\ i_2\ i_3\ i_4$	$p_1\ p_2\ p_3$
M_1	1 1 0	1 0 0 0	0 0 0
M_2	1 0 1	0 1 0 0	0 0 0
M_3	1 1 1	0 0 1 0	0 0 0
M_4	0 1 1	0 0 0 1	0 0 0
C_1	1 0 0	0 0 0 0	1 0 0
C_2	0 1 0	0 0 0 0	0 1 0
C_3	0 0 1	0 0 0 0	0 0 1

Table 1: Relation to (7,4) Hamming Code

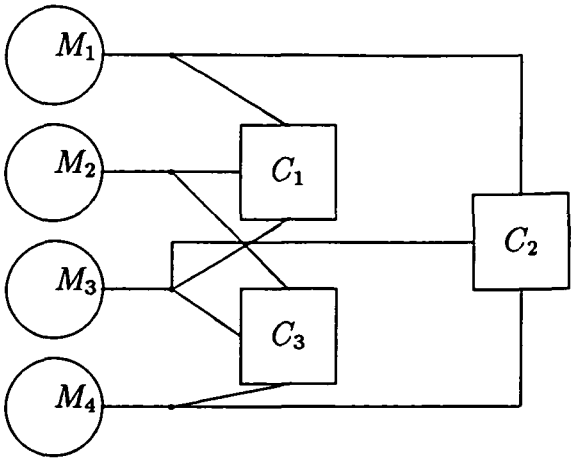


Figure 3: (7,4) Hamming Code Example

elements are compared, the result would be 0. Hence there is no need to compose the parity bits using an even number of information bits, in the corresponding binary linear code. These principles are illustrated in the example.

Consider a (7,4) binary Hamming code. The i's represent information bits and the p's represent parity bits. Note that a Hamming code has single error correction capability. We are trying to isolate a single faulty element. The parity bits are generated as follows.

$$\begin{aligned} p_1 &= i_1 + i_2 + i_3 \\ p_2 &= i_1 + i_3 + i_4 \\ p_3 &= i_2 + i_3 + i_4 \end{aligned}$$

Notice that the parity bits are composed of an odd number of information bits. This is allowed since we are considering the fault-free modules to be represented by the additive identity of GF(2). Let the modules being checked be represented by the information bits. Let bus-bus comparators be connected at the outputs of the modules according to the way the parity bits are generated; ie., one three-way comparator compares the outputs of modules $M_1, M_2, \& M_3$, a second comparator compares the outputs of $M_1, M_3, \& M_4$, and a third comparator compares $M_2, M_3, \& M_4$. The relations are summarized in Table 1 given. The system is shown in Figure 3.

The special feature of this implementation is that the comparators roughly correspond to the parity bits. Thus the comparators are also included as part of the code. The outputs of the comparators indicates which element is faulty. Notice that when a module is faulty, the outputs of the comparators indicate the syndrome of the fault pattern (or the error pattern in the equivalent code). For example, when the module M_1 is failed, the comparators output the vector [1 1 0]. This vector corresponds to the syndrome of the error pattern [1 0 0 0 0 0 0], where the first four positions represent the information bits which are the modules in the equivalent NMR system, and the last three represent the parity bits which are the comparators. Since 1 represents a failed element by our representation, we know that module M_1, corresponding to the first information bit is the failed element. As another example, consider the case where comparator C_1 is failed. Now the comparators output the vector [1 0 0], which is the same as the syndrome of the error pattern [0 0 0 0 1 0 0]. This error

pattern indicates that the first parity bit is in error, or in the equivalent NMR system, the first comparator is failed. Similarly, all possible single element failures are uniquely identified by this system.

7 Conclusions

In this paper, we explored the links between the use of N-Modular Redundancy for hardware fault tolerance, and the use of linear Error-Correcting Codes. We showed how NMR is an application of a (N,1) repetition code, and also showed how special encoding structures can be designed to cover failures in the checker units themselves. We adapted some well-known results from coding theory for use in NMR for fault tolerance. The rich mathematical theory that finds application in error-protection techniques in digital data communication has faces that touch other subjects. The use of modular redundancy for hardware fault tolerance is one of them.

Acknowledgements– This research was supported in part by NASA under Space Engineering Research Center Grant NAGW-1406. The authors would like to express sincere gratitude to Barbara Martin for help in the preparation of the manuscript. The first author acknowledges the help of colleagues, especially Suresh Gopalakrishnan, at the California Design Center of Hewlett-Packard Company.

References

[1] R. Blahut, *Theory and Practice of Error Control Codes*, Addison-Wesley, May 1984.

[2] S. Lin and D. Costello, Jr. *Error Control Coding: Fundamentals and Applications*, Prentice-Hall, 1983.

[3] V. Bobin, "Modular Redundancy and the Theory of Linear Codes", Ph D Dissertation, Dept. of Elec. Engr., University of Idaho, 1992.

Reduction of Blocking Effects for the JPEG Baseline Image Compression Standard

Gregary C. Zweigle
Microelectronics Research Center
Elec. and Comp. Eng. Dept.
University of New Mexico
Albuquerque, NM 87131-1356

Roberto H. Bamberger[1]
Imaging Research Laboratory
School of Elec. Eng. and Comp. Science
Washington State University

Abstract- **Transform coding has been chosen for still image compression in the JPEG [1] standard. Although transform coding performs superior to many other image compression methods [2] and has fast algorithms for implementation [3], it is limited by a blocking effect at low bit rates. The blocking effect is inherent in all nonoverlapping transforms. This paper presents a technique for reducing blocking while remaining compatible with the JPEG standard. Simulations show that the system results in subjective performance improvements, sacrificing only a marginal increase in bit rate.**

1 Introduction

Digital images demand large amounts of data to faithfully duplicate the analog scene. As a result, image coding for compression has been a major area of research since the earliest days of digital image processing. While memory capabilities for digital storage and channel bandwidth for digital transmission have increased in recent years, so have the applications for digital images and the need for compression remains. However, compressed data is not useful if it is unreadable by those who need it. An image compression standard allows images which have been compressed for storage or transmission to be easily decompressed and used.

A proven compression method is transform coding. This technique first uses a unitary transform to map image data into a space which allows more efficient representation. In the ideal case, the mapping results in data which is independent or uncorrelated. However, such transforms are difficult to implement. The discrete cosine transform (DCT) is a transform with a known fast algorithm that also performs close to the ideal for many images [3]. Subsequent to transformation the data is typically quantized and then entropy coded, taking advantage of the uncorrelated transformed data. Transform coding using the DCT has been chosen for the Joint Photographic Experts Group (JPEG) still image compression standard [1].

JPEG has been shown to perform well for greyscale compression ratios of five to fifteen. A major limitation to further compression is a blocking effect. This visually annoying effect has its roots in the method used for transform computation. In order to exploit local stationarity and reduce computational load, an image is first divided into nonoverlapping areas and then each area is acted upon individually. A JPEG standard codec divides the imaged into 8x8 blocks. If subsequent quantization is coarse, a noticeable discontinuity between neighboring regions is visible. This is especially noticeable to the viewer because the human visual

[1]This work was supported in part by the National Science Foundation under grant number MIPS-9116683.

system is very sensitive to edges [4] and even more so to edges in the vertical and horizontal directions [5], which is the direction of the blocking effect edges in JPEG. Although in a mean-square error sense, the blocking effect may not contribute much to the overall error, research has shown that it can be up to ten times more objectionable to the human viewer than random noise distortion [6].

Several approaches to reducing the blocking effect have been researched. These include postprocessing with a smoothing function [7], doing quantization with a constraint on the amount of distortion that is allowed between neighboring blocks [8], using human visual system properties in coding [9], and adaptively changing block sizes [10]. Additionally, the discovery of useful overlapping transforms has provided another framework for reducing blocking effects. Lapped Orthogonal Transforms (LOT) [11], are a family of such transforms. Of these methods, only the approach using postprocessing can be utilized with a JPEG standard codec.

In this paper, a technique for reducing the blocking effect is introduced which uses an overlapping mean estimation operator. Compatibility with JPEG is maintained by using a pre processor before compression to make the estimate and a postprocessor after decompression to restore the estimate. The extra mean information is subtracted from the original image data before JPEG compression to allow JPEG to perform more efficiently and is then transmitted as a small amount of side information. It will be seen that the overall data rate remains approximately the same with this system, but the blocking effect is dramatically reduced.

Section 2 introduces and explains the Baseline sequential JPEG codec, which employs only the most used features of the full JPEG standard. The differences between these standards do not effect the theories of this paper and subsequent references to JPEG will imply the Baseline version. Section 3 presents the pre and postprocessors used to reduce the blocking effect. The performance of the new implementation will be compared to the performance of an unadulterated JPEG codec in Section 4. Section 5 concludes the paper.

2 The Baseline JPEG Standard

A Baseline JPEG standard image compression system begins by converting the input image data into a form suitable for processing, Fig. 1. This is necessary because JPEG is a compression standard and not a file format standard. Each pixel in the input image must, however, be eight bit unsigned for the Baseline lossy codec.

After conversion to a functional format, the image data is shifted to be centered around zero. A mapping is performed from $[0, 2^P - 1]$ to $[-2^{P-1}, 2^{P-1} - 1]$ by subtracting 2^{P-1}. For eight bit data, P = 8, and 128 is subtracted as shown in Fig. 1. This step can be considered as a simple mean subtraction.

The next step in the compression scheme is the DCT. Although an integer 8x8 DCT is specified in the JPEG standard, the implementation details are left to the user. This leaves room for improved algorithms to be utilized as they become available. The 8x8 forward DCT

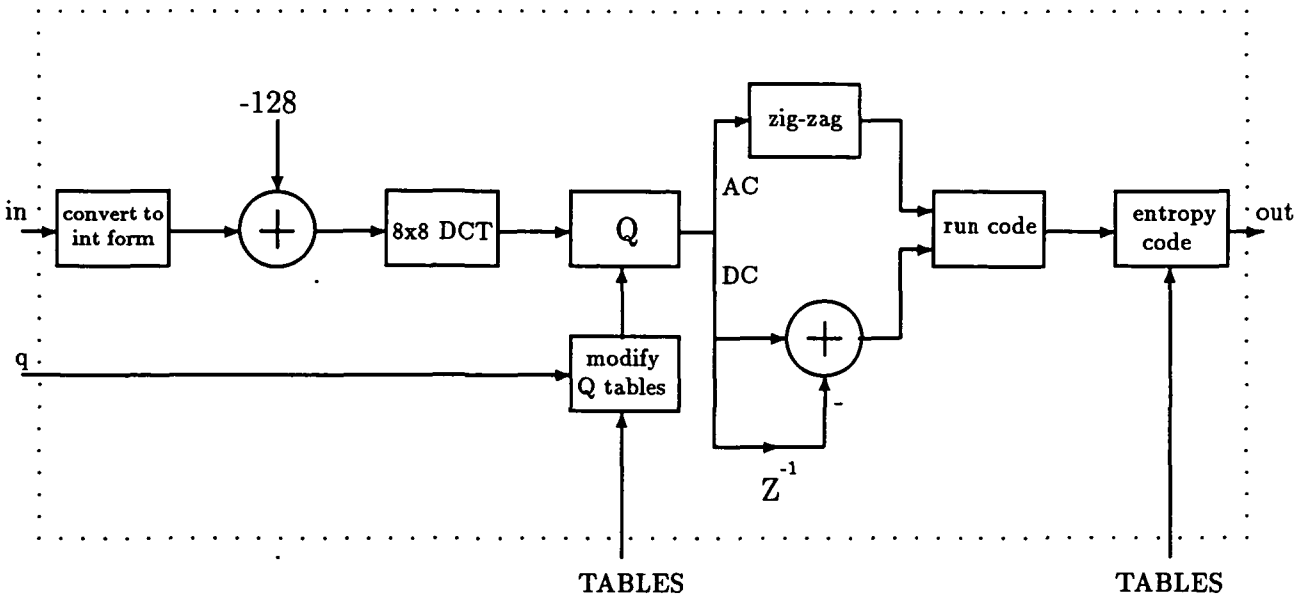


Figure 1: Block diagram of JPEG compression scheme

is defined according to

$$F(u,v) = \frac{1}{4}C(u)C(v)\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y)cos\left[\frac{(2x+1)u\pi}{16}\right]\left[\frac{(2y+1)v\pi}{16}\right]. \quad (1)$$

For decoding, the inverse DCT is defined as

$$f(x,y) = \frac{1}{4}\sum_{u=0}^{7}\sum_{v=0}^{7} C(u)C(v)F(u,v)cos\left[\frac{(2x+1)u\pi}{16}\right]\left[\frac{(2y+1)v\pi}{16}\right]. \quad (2)$$

The values C(u) and C(v) are both defined according to

$$C(k) = \left\{ \begin{array}{cc} \frac{1}{\sqrt{2}}, & k=0 \\ 1, & otherwise \end{array} \right\}. \quad (3)$$

Extensive computer simulations have shown that the outputs from the DCT are integers in the range $[-2^{R-1}, 2^{R-1}-1]$ where R = P + 3 [1].

Each 8x8 block exiting from the DCT contains 64 frequency domain coefficients. These coefficients are then quantized using uniform threshold quantization in conjunction with a 64-element quantization table, Q(u,v), specified by the user. Quantization is the principle source of loss in the JPEG standard codec. The integer DCT also introduces a small amount of loss.

The quantization is done according to

$$F_Q(u,v) = Integer\ Round\ \{\ \frac{F(u,v)}{Q(u,v)}\ \}. \quad (4)$$

It can be seen that as the value of Q(u,v) approaches unity, the quantization step goes away. Note also that there is no saturation point in the quantizer. Saturation is not necessary

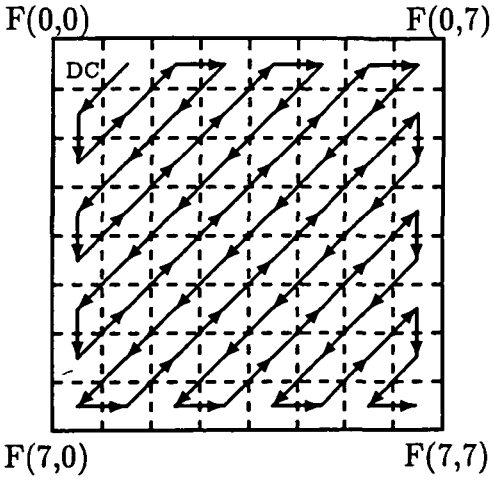


Figure 2: Zig-zag sequence

because the input dynamic range is known apriori. Variable length entropy coding is sufficient to deal with low probability, high valued inputs.

For this JPEG implementation, the user has control over a free parameter, q, at the time the image is compressed. This "q-factor" is used to modify the quantization tables before quantization. The initial values in the quantization table are in the range [1,255]. A q-factor of 100 sets all the quantization table values to one. As the q-factor is decreased, the quantization table values are multiplied by an at first linearly increasing and subsequently exponentially increasing factor. When q is zero, the large quantization values basically set all DCT coefficients to zero.

After quantization the F(0,0), or dc, coefficient of each DCT block is separated from the other coefficients. The dc coefficient is treated differently because it represents a local mean of image intensities on a block by block basis. Most images can be modeled as a source whose power spectral density is concentrated in the low frequencies [12] and so the dc coefficient is known to change slowly as the image is traversed. This correlation between blocks is exploited by encoding only the difference between dc components.

Meanwhile, the other DCT coefficients (called the ac coefficients) are arranged into a vector according to a zig-zag pattern, Fig. 2. This arrangements facilitates run-length coding by placing the higher frequency coefficients, which are likely to be zero, after the lower frequency coefficients which typically have more energy.

Entropy coding for the JPEG Baseline standard is tackled in two steps. The first is an intermediate symbol coding step which does a type of run-length coding and outputs a pair of symbols associated with each run. The second step does entropy coding on these symbols. In Fig. 1 the first step is denoted "run code" and the second "entropy code".

For run-length coding, only runs of zeros are counted. Two symbols are used for representation. The first symbol contains the number of zero valued coefficients that preceded the nonzero valued coefficient which terminated the run of zeros. Symbol one also contains the size in bits of the variable length integer (VLI) code that will be used to represent the amplitude of the nonzero value in the entropy coder. Symbol two is the actual amplitude of the nonzero coefficient.

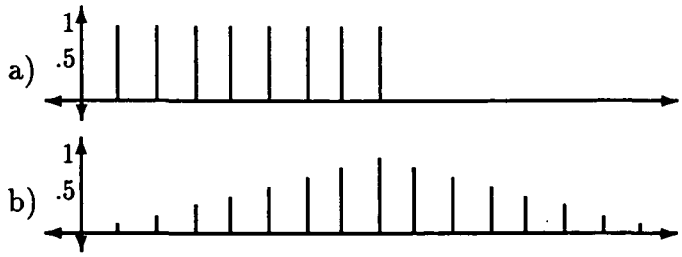


Figure 3: First and second order interpolators

The dc coefficients are also placed in a symbol one, symbol two pair by the "run coder", but no run-length coding is done. Symbol one is only the size in bits of the VLI code needed to represent the amplitude of the dc coefficient which is stored in symbol two.

The entropy coding treats symbols one and two from the "run coder" separately. Symbol one is encoded using a Huffman code. The Huffman coder requires the use of table sets which are supplied by the user. Each set consists of a table for the ac coefficients and a table for the dc coefficients. Symbol two is encoded using a variable length integer coder. Although marginally less efficient than the Huffman code, the VLI code has the advantage of being hardwired into the codec resulting in faster computation speeds and simpler implementation.

A decoder based on the JPEG standard is basically just the inverse of the encoder. For the Baseline codec, only two sets of Huffman tables can be used by the decoder at a time. This limits what can be attempted in the coder. The inverse quantization is accomplished according to

$$F'(u,v) = F_Q(u,v)Q(u,v). \tag{5}$$

At the close of decoding, the offset is added back to the data.

3 Blocking Effect Reductions

Because images tend to be low frequency in nature and the human visual system has a lowpass spatial frequency response [13], the DCT dc coefficient conveys much of the image information to the human viewer. Quantization of this coefficient without regard to neighboring regions is a major contributor to the blocking effect. If the dc information can be represented with an operator that has block overlap then the effect of the quantization will be smoothed between regions and the blocking effect reduced.

Figure 3a shows the basis function that the DCT uses to calculate the dc coefficient. This boxcar shaped function is of length eight, the same size as the transform. Because of the size, discontinuities between blocks cannot be smoothly interpolated. Figure 3b shows an alternate basis function, the second order interpolator, which has similar frequency content as the DCT dc basis function. This function is able to represent the dc level of the image more smoothly because the function of Fig. 3b overlaps the block size. But the triangular shaped basis function cannot be directly substituted for the boxcar shaped DCT basis function because of the size difference and the desire for an orthogonal transform. In order to use the new function, a preprocessor is utilized. This is shown in Fig. 4.

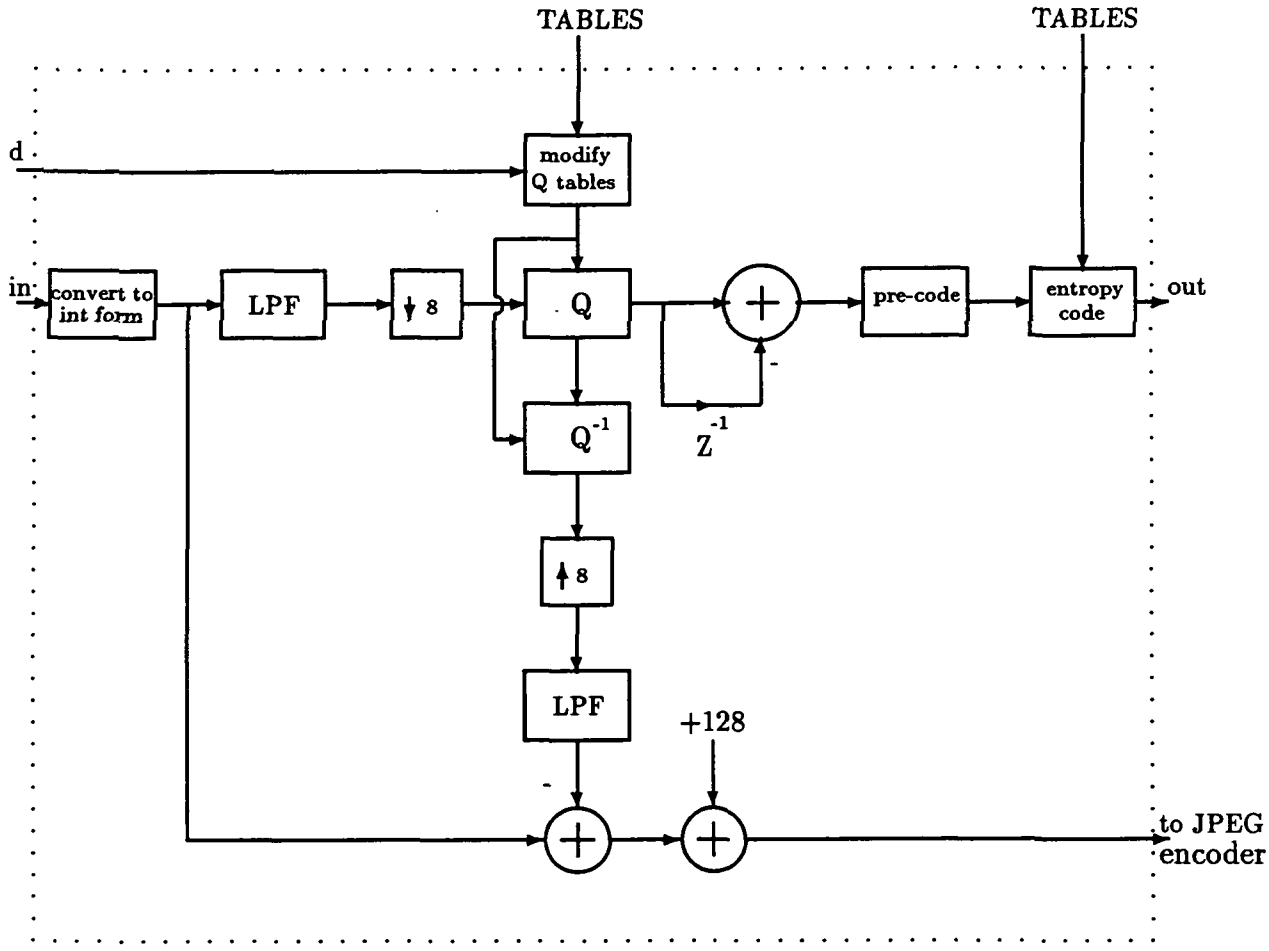


Figure 4: Block diagram of preprocessor

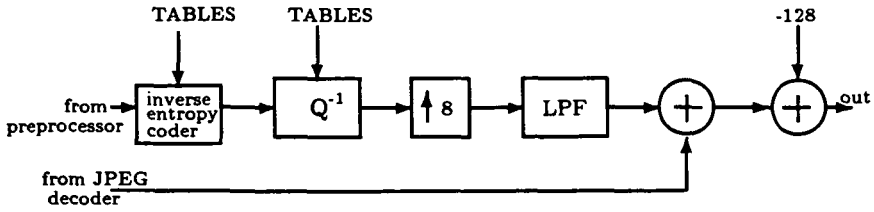


Figure 5: Block diagram of postprocessor

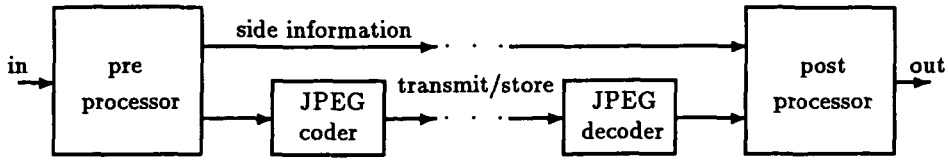


Figure 6: Block diagram of JPEG with pre and postprocessor

The preprocessor operates similar to a laplacian pyramid [14]. After conversion to a useable internal format, image data is lowpass filtered ("LPF" block) by circular convolution with the second order interpolator of Fig. 3b. Circular convolution is performed instead of linear convolution to avoid data expansion, although some picture edge effects result. The lowpass filtering results in an estimate of the mean of the image with the new basis function. The image is then decimated by eight to take advantage of the now oversampled image spectrum. The quantization and entropy coding steps are performed with the same tables as the ones the JPEG codec will use for its dc coefficient. However, a new parameter, 'd', is used to adjust the quantization table of the preprocessor. When 'd' is large, the effect of quantization is reduced and the mean estimate is smooth between neighboring regions. As 'd' approaches zero, the quantization becomes more coarse and the error more blocky. When d = 0, the preprocessor goes away.

The inverse quantization, upsampling, filtering, and subtraction of the new mean estimate from the original signal allows the JPEG coder to use less information for its dc estimate which reduces the JPEG bit rate. In the ideal case, the JPEG bit rate decreases by the same amount as the extra bit rate needed for side information. The addition of 128 before JPEG processing is necessary to cancel the effect of JPEG's initial subtraction of 128.

The postprocessor used after JPEG decompression to reconstruct the mean estimate is shown in Fig. 5. All LPF blocks in Figs. 4 and 5 represent circular convolution with the function of Fig. 3b. The JPEG input to the postprocessor is the output of a JPEG decoder. The postprocessor simply puts back the information that was subtracted from the original input. The complete system is shown in Fig. 6.

4 Performance Comparisons

Test results show that the performance of the system is dependent on the choice of 'd'. For large 'd', the total bit rate is marginally increased due to the side information that must be transmitted. However, subjective analysis shows the blocking effect to be significantly reduced.

5.5.8

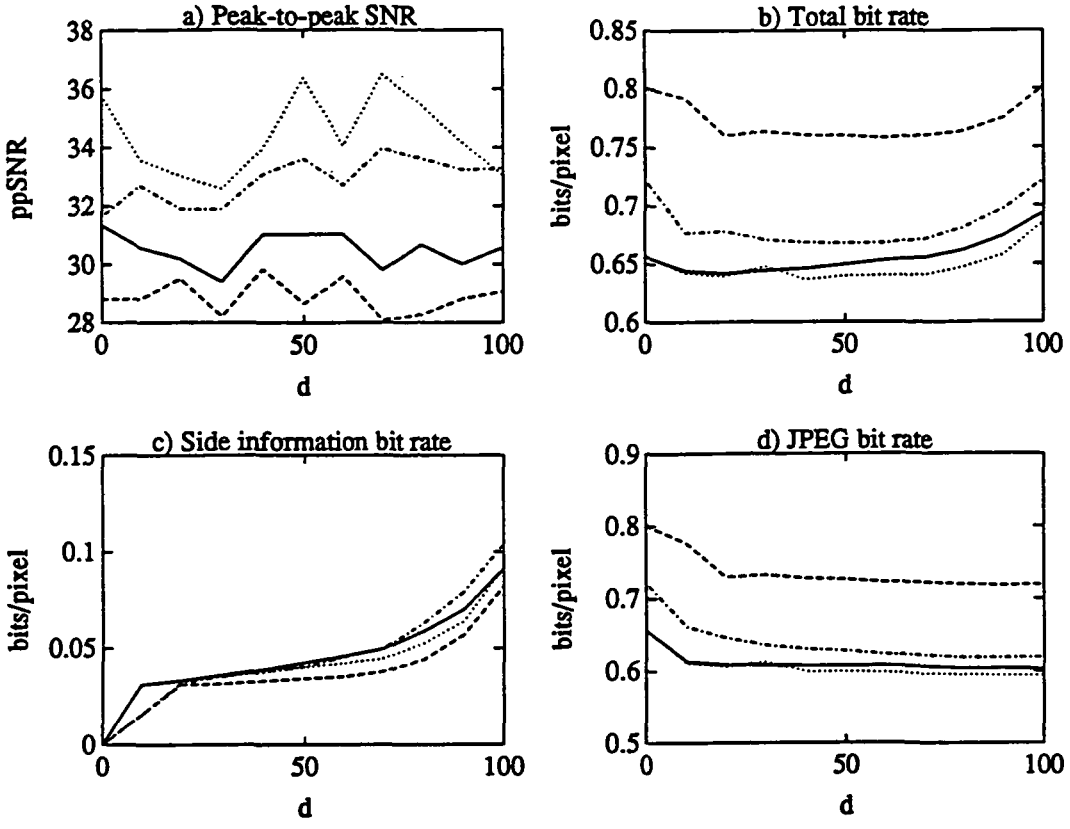


Figure 7: Bit and distortion rates, q = 15, d increases from 0 to 100

Figure 7 shows test results for four different images all compressed and decompressed with the system of Fig. 6 and a JPEG q-factor of 15. This results in greyscale compression ratios of 10 to 12, depending on the images. The data for d = 0 represents compression by JPEG without the extra processors. The solid line represents data for the `building` image, long dashed line for `kboat`, short dashed line for `cameraman`, and short/long dashed line for `lenna`. These are standard comparison images and will be shown at the symposium. Along the horizontal axis of all four plots is the preprocessor parameter 'd'. Figure 7a shows the bit rate associated with the complete system. For d = 0, the plain JPEG codec has the lowest bit rate. The images however, are blocky. As d increases, the images become less blocky but the bit rate is slightly increased. Figure 7b shows the bit rate associated with just the JPEG portion of the system of Fig. 6. It is apparent that the subtraction of the mean information from the original image before JPEG compression helps reduce JPEG's data rate.

In order to make better comparisons as to the amount image blockiness is reduced with the use of the extra processors, the JPEG dc coefficient quantization table value is altered for further tests to reduce blocking effects associated with JPEG alone. The quantization table value for the dc coefficient is set to 1 for the JPEG codec used for d = 0 data. For the JPEG codec used in conjunction with the pre and postprocessor the quantization table value is not changed. These test results are shown in Fig. 8. From Fig. 8a, the objective distortion measure, ppSNR, is seen to show no significant change as 'd' changes. The subjective distortion is reduced due to a reduction in the blocking effect. This can be seen by looking at the images. Figure 8b shows the total bit rate of the system. It can be

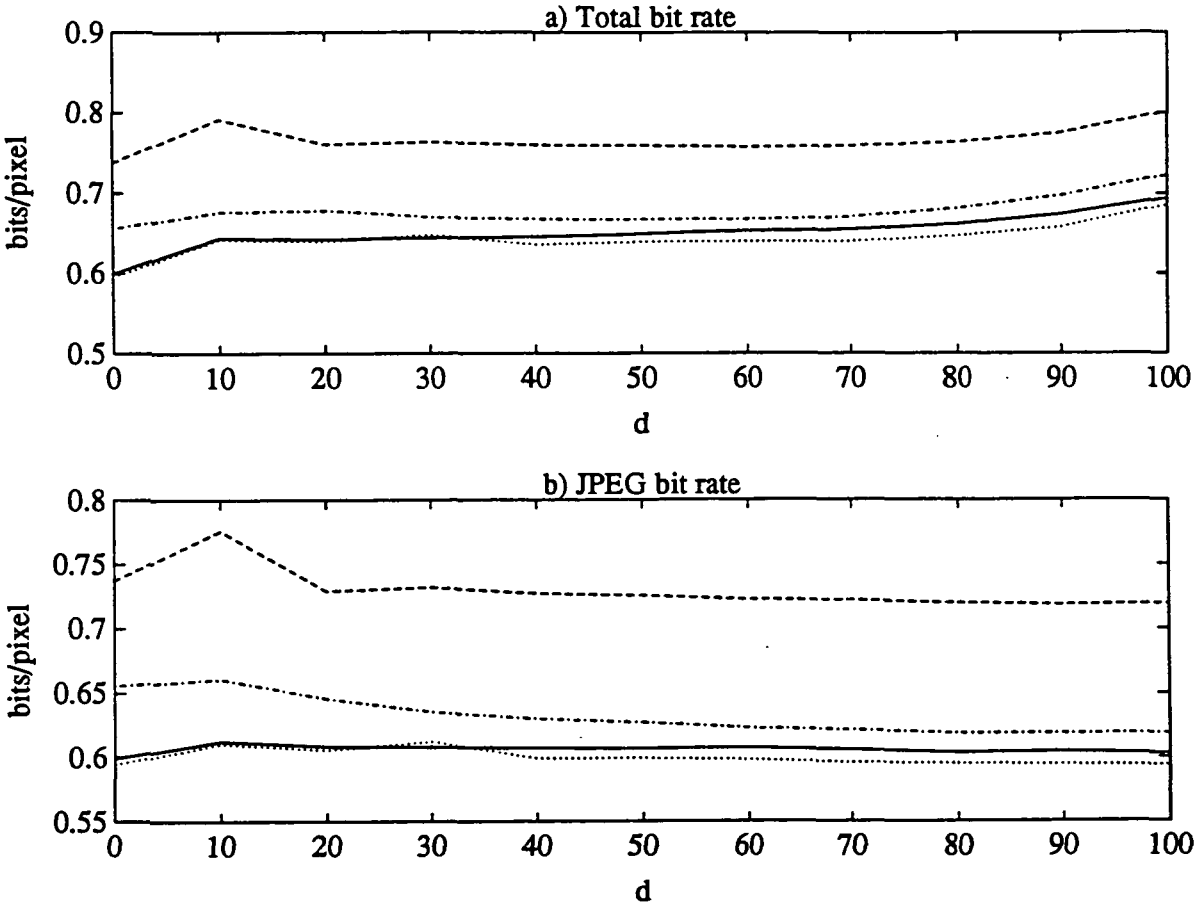


Figure 8: Bit and distortion rates, different Q table for d = 0

seen that the total bit rate is lowest for intermediate values of 'd'. For d = 0, the JPEG codec has a higher bit rate because of reduced dc coefficient quantization. As 'd' increases, the JPEG rate (shown in Fig. 8d) decreases more than the side information rate increases, (side rate shown in Fig. 8c) resulting in the dip in Fig. 8b. Notice from Fig. 8c, that the side bit rate associated with the pre and postprocessor is less than a tenth of a bit for all four of the images and all values of 'd'. This side rate is independent of the JPEG q-factor. So Fig. 8c gives an indication of the side rate regardless of what q-factor is chosen. Because of image degradation that can occur in printing the symposium proceedings, the images will be presented at the symposium for subjective comparisons.

5 Conclusion and Future Work

A JPEG compatible codec has been described which reduces the blocking effect for low bit rates. The codec uses an overlapping basis function for average pixel intensity estimation. The mean estimate is transmitted as a small amount of side information and subtracted from the original image before JPEG compression. Experimental results show that the blocking effect is reduced.

There are some areas of work that could lead to better results for this system. The

second order interpolator was used for computational simplicity, but it is not orthogonal to the other DCT basis functions. Use of a LOT basis function could possibly give better results. Another area that should be pursued is the reduction of picture edge effects that result from circular convolution. This area of research has been well studied for subband coding systems [15] and these results should be extended to this system. Also, there is still a slight blocking effect in the neighborhood of edges for very low bit rates. This is a result of edge information being contained in ac coefficients which are not dealt with in the extra processors. The possible reduction of this effect needs to be addressed.

References

[1] Gregory K. Wallace, "The JPEG Still Picture Compression Standard," *Comm. of the ACM*, vol. 34, pp. 31-44, April 1991.

[2] Gregory K. Wallace, R. Vivian, and H. Poulsen, "Subjective Testing Results for Still Picture Compression Algorithms for International Standardization," *Proceedings of the IEEE Global Telecommunications Conference*, pp. 1022-1027, Nov. 1988.

[3] N. Ahmed, T. Natarajan, and K. Rao, "Discrete Cosine Transform," *IEEE Trans. Comput.*, vol. 23, pp. 90-93, Jan. 1974.

[4] Murat Kunt, A. Ikonomopoulos, and Michel Kocher, "Second-Generation Image-Coding Techniques," *Proc. IEEE*, vol. 73, pp. 549-574, April 1985.

[5] F.W. Campell and J.J. Kulikowski, "Orientational Selectivity of the Visual System," *J. Physiol.*, vol. 187, pp. 437-445, 1966.

[6] Makoto Miyahara and Kazunori Kotani, "Block Distortion in Orthogonal Transform Coding - Analysis, Minimization, and Distortion Measure," *IEEE Trans. on Comm.*, vol. 33, pp. 90-96, Jan. 1985.

[7] Bhaskar Ramamurthi and Allen Gersho, "Nonlinear Space-Variant Postprocessing of Block Coded Images," *IEEE Trans. Acoust., Speech, Signal Processing*, vol.34, pp. 1258-1267, Oct. 1986.

[8] William A. Pearlman, "Adaptive Cosine Transform Image Coding with Constant Block Distortion," *IEEE Trans. on Comm.*, vol. 38, pp. 698-703, May 1990.

[9] King N. Ngan, Kin S. Leong, and H. Singh, "Adaptive Cosine Transform Coding of Images in Perceptual Domain," *IEEE Trans. Circuits and Systems*, vol. 37, pp. 1743-1749, Nov. 1989.

[10] Its'Hak Dinstein, Kenneth Rose, Arie Heiman, "Variable Block-Size Transform Image Coder," *IEEE Trans. on Comm.*, vol. 38, pp. 2073-2078, Nov. 1990.

[11] Henrique S. Malvar and David H. Staelin, "The LOT: Transform Coding Without Blocking Effects," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, pp.553-559, April 1989.

[12] N.S. Jayant and P. Noll, *Digital Coding of Waveforms.* Englewood Cliffs, NJ: Prentice-Hall, 1984.

[13] J.G. Robson, "Spatial and Temporal Contrast- Sensitivity Functions of the Visual System," *J. Opt. Soc. Amer.*, vol. 56, pp.1141-1142, 1966.

[14] Peter J. Burt and Edward H. Adelson, "The Laplacian Pyramid as a Compact Image Code," *IEEE Trans. on Comm.*, vol. 31, pp. 532-540, April 1983.

[15] Roberto H. Bamberger, Steven L. Eddins, and Veyis Nuri, "Generalizing Symmetric Extension: Multiple Nonuniform Channels and Multidimensional Nonseparable IIR Filter Banks", *IEEE Intl. Symp. on Circuits and Systems*, San Diego, CA, May, 1992.

Session 6
Verification

Chairman: Phil Windley

A Novel Visual Hardware Behavioral Language

Xueqin Li
Dept of Electrical Engineering
Utah State University
Logan UT 84322
Email:xueqin@slow.cs.usu.edu

H. D. Cheng
Dept of Computer Science
Utah State University
Logan UT 84322

Abstract - **Most hardware behavioral languages just use texts to describe the behavior of the desired hardware design, but it is inconvenient for VLSI designers who enjoy using schematic approach. The proposed visual hardware behavioral language has the ability to graphically express design information using visual parallel models (blocks), visual sequential models (processes) and visual data flow graph (which consists of primitive operational icons, control icons and Data and Synchro links).**

Thus the proposed visual hardware behavioral language not only can specify hardware concurrent and sequential functionality but also can visually expose parallelism, sequentiality and disjointness (mutually exclusive operations) for the hardware designers. That would make the hardware designers capture the design ideas easily and explicitly using this visual hardware behavioral language.

1 Introduction

The success of hardware design is heavily dependent on how effectively the input language captures the ideas of the designer in a simple and understandable way. A hardware description is behavioral when it is expressed in a manner which conveys only what the hardware module is supposed to do without committing to an implementation. The hardware behavior languages are basically the best way we know today to describe what a system looks like at a more abstract level. Many hardware behavioral languages have been proposed and used in both academical and industrial environment. For example, VHDL, Verilog, Esim, etc. Some HDLs provide a mechanism for attaching code written in Fortran, Lisp, Pascal or C in order to express complex behavior such as HardwareC[10]. But VLSI designers prefer drawing diagrams rather than writing codes in design hardware. If the hardware behavioral language can provide direct, manipulatable graphic models, which are consistent and complete from the hardware designer's point of view, then simulation and synthesis based on these visualized models will greatly improve user interface, and the users only need to understand the visual models. This requires that the hardware behavioral language has the ability to graphically express abstract design information. The proposed hardware behavioral language is such a visual hardware behavioral language that can not only specify hardware concurrent and sequential functionality but can also visually expose parallelism, sequentiality and disjointness (mutually exclusive operations).

There are two kinds of visual models in the proposed language: parallel model (block) and sequential model (process). Using blocks, we are describing the hardware modules as collections of interconnected objects which operate in parallel. Using processes, we are

Figure 1: The visual representations for the port

encapsulating certain aspects of the behavior which can be described in the form of a sequence of steps and expressing these steps in a visual pictorial dataflow style. Our visual behavioral HDL has its own hardware semantics and visual representations (such as process, block, signal link, port, wait operation icon, drive operation icon, etc) with adoption of some data flow syntax and visual representations from other visual language[4, 5, 6] to support the specification of the hardware design.

2 Definitions and Visual Representations

1. port: A connected point associated with objects which may appear in blocks and processes, and which represents information flow into or out of the objects. The attributes of a port are name, type, width, mode and sensitive token. The port name represents its own ID which may be a character or a letter and digit string. The type represents the type of information to be carried through the port which may be a type of bit, integer, float, character or string. The width indicates an array of units of information, and the mode represents the direction of data flow into or out of the object which may be IN, signifying information flow from the "outside" to the "inside", OUT, signifying flow from the "inside" to the "outside", or INOUT, signifying bidirectional information flow. The sensitive token has only two values 0 or 1 which indicates whether the port is a sensitive port to the attached process. The visual representation of the port is shown in Figure 1. During editing, a double-click on these port icons would enter the port editor as shown in Figure 2.

2. Signal link: A signal link connects two ports together. A signal link is represented as a line in the block editor. Signal link has the attributes of type and width which must be compatible with the ports to which they are connected.

3. Block (parallel model): A form of description managed by the block editor, which represents description as a collection of interconnected objects which are considered to operate in parallel. A block has an external port list and contains objects and signal links. An object may be: 1) an instance of another block; 2) a process. The ports of the block and ports on the contained objects may be interconnected with signal links. The visual representation for the block is an empty rectangle with some ports sticking with it which is shown in Figure 3.

4. Process (sequential model): A form of description expressed in a dataflow style. A

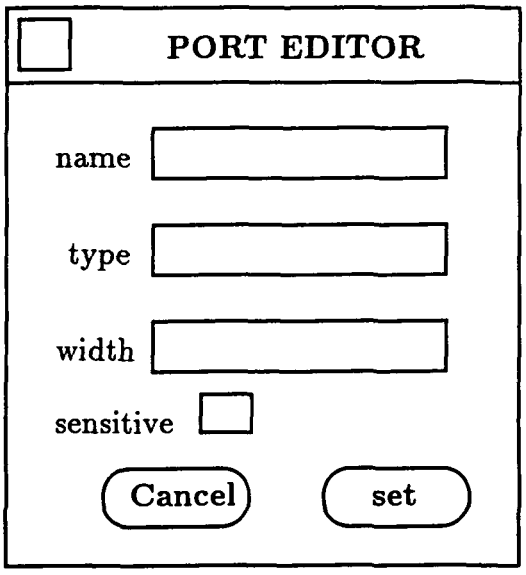


Figure 2: The port editor

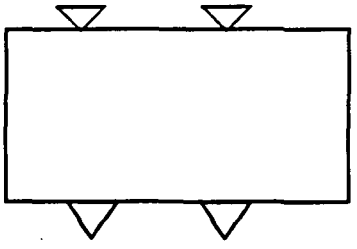


Figure 3: The visual representations for the block

Figure 4: The visual representation for the process

process has an external port list and contains data flow primitive operation icons which are linked by the data/synchro links to express certain behavior which is supposed to do within the process. A process has some sensitive ports which are indicated by tokens. A process is said to be sensitive to a particular set of its IN or INOUT ports at any time and such that execution will be resumed by the simulator whenever there is a state change at any currently sensitive ports of the process. A process which suspends when completes its execution (reaching the end of its dataflow graph) will be deemed at that time to be sensitive to all of its IN and INOUT ports and will be restarted at the beginning of its dataflow graph upon a state change on any of these ports. We assume that a process does not contain subprocesses. The visual representation for the process is a hexagon with some ports sticking with its edges as shown in the Figure 4.

5. Dataflow graph: Some ordered operational icons connected by data and synchro links represent certain algorithms. The visual data flow graph represents the algorithm as a set of operational icons connected by Data links and Synchro links. Consider a process which has the function: $F = X_1X_2 + X_3$ after 20ns. A process showing the dataflow model of this function is shown in Figure 5. The process will be activated whenever any one input variable of the input list: (X_1, X_2, X_3) changes. Once the process is activated, the "and" operation will be executed before the "or" operation because of its data-dependence. The "drive" operation is to model the time of function. That is, "F" will be updated after 20ns from the time the process is activated. The operational icons connected by Datalinks are guaranteed to execute in a serial order. The Synchro link allows the designer to specify control dependencies among data-independent icons. In Figure 6, the computation $A_1 + A_2$ must be carried out before the computation $A_3 + A_4$. The multiplication must be carried out after two additions because of its data-dependence.

   Following elements are needed to specify hardware functional behavior within the data flow graph.

   - Primitive icons: primitive icons visually represent arithmetic, relational or logical operations. These icons have at most two inputs (terminals) and one output (root) as shown in the Figure 7. This form corresponds closely to the operations that can be carried out directly in the hardware. Table 5 gives the definition of these primitive operations.

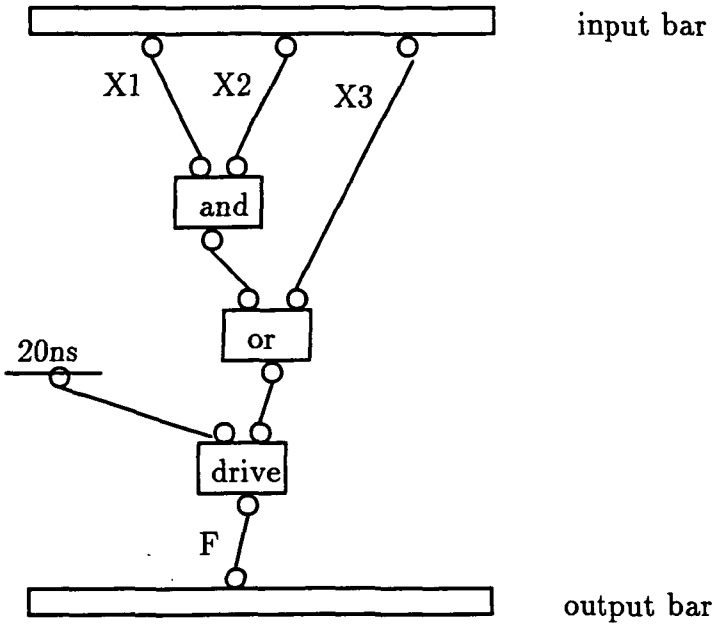


Figure 5: An example of dataflow graph to model a combinational logic

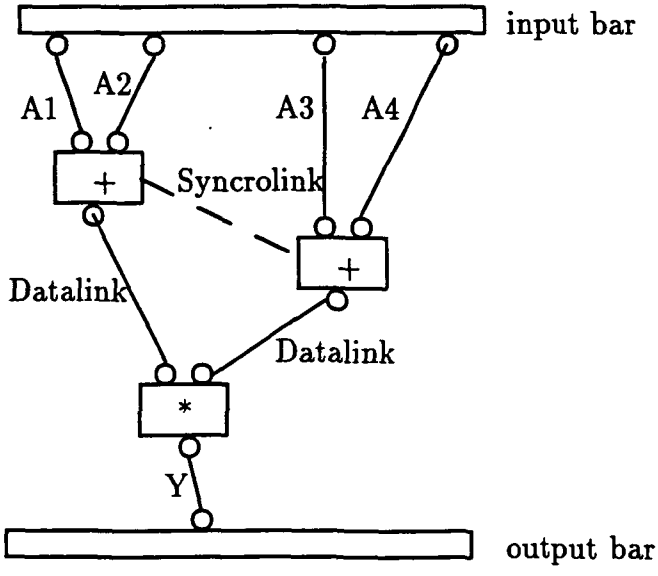


Figure 6: The visual representation for the Datalink and Synchro link

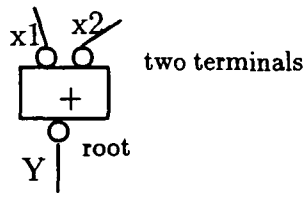


Figure 7: The example of primitive icon

Table 1: The definition of the primitive operations

Group	Symbol	Function
arithmetic	+	addition
(binary)	-	subtraction
	*	multiplication
	/	division
	**	exponentiation
	mod	modulus
	rr	rotate right
	rl	rotate left
	>>	shift right
	<<	shift left
	drive	signal assignment
arithmetic	+1	unary plus
(unary)	-1	unary minus
	abs	absolute value
relational	==	equal
	!=	nor equal
	<	less than
	>	greater than
	≤	less than or equal
	≥	greater than or equal
logical	and	logical and
	or	logical or
	nand	complement of and
	nor	complement of or
	xor	logical exclusive-or
	not	complement
input	ask	read data
output	show	display data

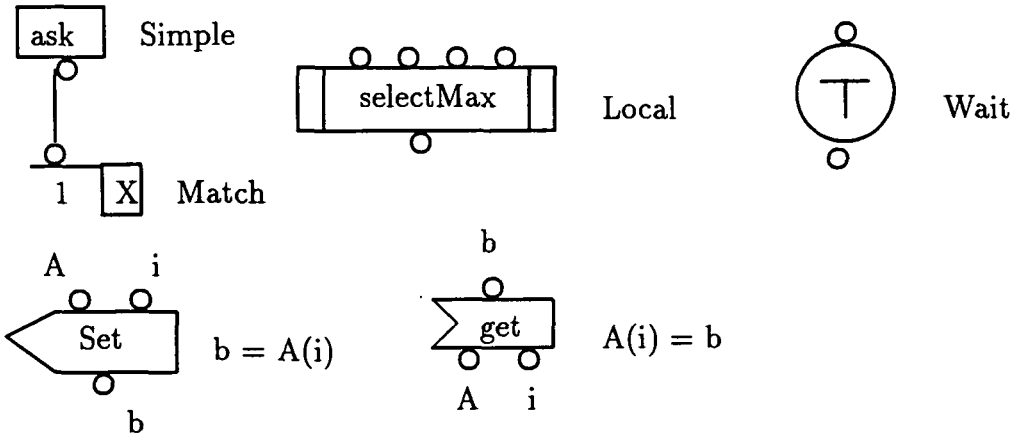


Figure 8: The Operation icons

- Operation icons: There are several operation icons: **Simple, Match, Wait, Constant, Set, Get, Local.** Figure 8 shows these operational icons. An operation when first created is Simple. After its name has been typed and the Return Key pressed, the interpreter parses the name and determines whether it refers to a primitive or a user-defined operation, altering the appearance of the operational icon accordingly. A constant icon has a value and a single root. When the operation executes, this value is made available on its root. When a match operation executes, its value is compared to the value flowing into its terminal. The match operation may be 1 or 0 with the success or failure of the comparison. Wait operation causes the process to suspend itself until the event being waited occurs or the time it is waiting for has elapsed. Wait operation provides strong synchronization across processes. Get and Set operation are used for the variable reference. Local operation is an encapsulation of a body of codes into a single icon. We can treat each local icon as a closed box that communicates with the rest of the program within the process only through its inputs and outputs. We can alter or rearrange the statements inside the box at will, provided the changes do not affect the local icon's input or output. In this way, we can define a hierarchy of procedure call within a process. Consider a simple example: $F = (a+b+c+d+x+y+z)*s = [(a+b+c+d)+x+y+z]*s$. In Figure 9, we make the sum of 4 operants into one local icon **sum4**.
- Control icons: Control icons are attached to the operation icons. Control icon is visually a small icon attached to the right side of an operation icon and is used for implementing the condition and loop functions. A control has two aspects: the action to be taken, and whether this action is taken on success or failure of the operation. Control icons depict these two aspects as follows:
  - A check mark (V) within the control icon indicates that it is activated on success of the operation.
  - A mark (X) within the control icon indicates that it is activated on failure of

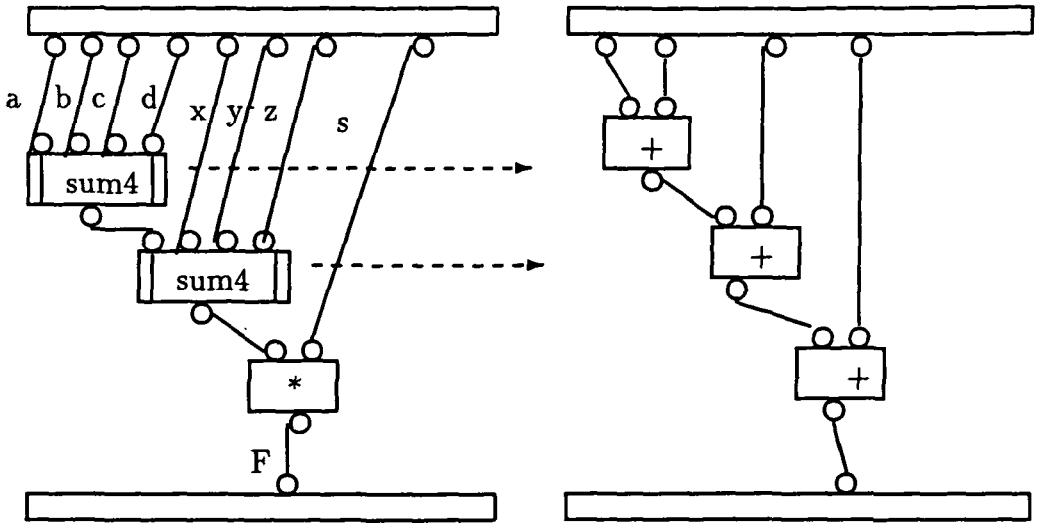


Figure 9: The example of the local icon

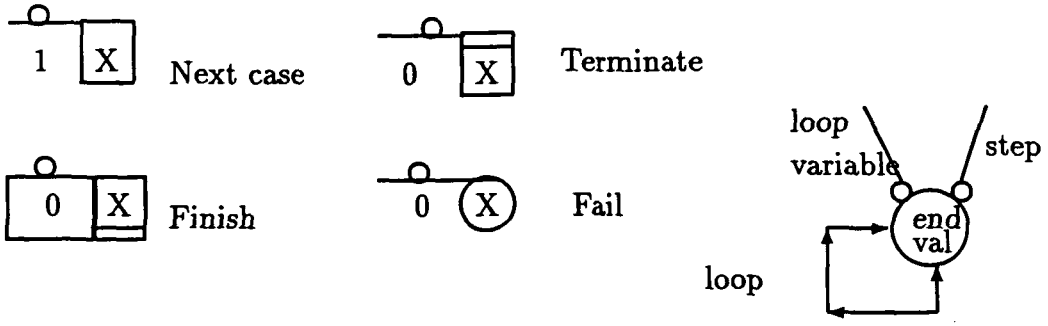


Figure 10: The control icons

the operation.

- The other graphics within the control icons indicate the action to be taken.

We use following types of controls: Next case, Terminate, Finish, Fail and loop as shown Figure 10.

Consider an example of a Mod-4 up counter. The counter counts up if the control input **enable** equals 1 at the rising edge of the clock, **clk**, signal. Figure 11 shows the behavioral representation of this counter. A method with two cases is required to express the counter. The first case is to be activated whenever **clk** and **enable** equal to 1. A primitive operation of the form **(1 + count) mod 4** is to be executed revealing the new value of count which will be scheduled after **clk-width**. The second case is activated if either **clk** or **enable** equal to 0. In this case, the counter will retain its current value.

6. Template library: An external collection of the blocks, which consist of some generic components such as: ALU, counters, comparators, registers and interconnect units

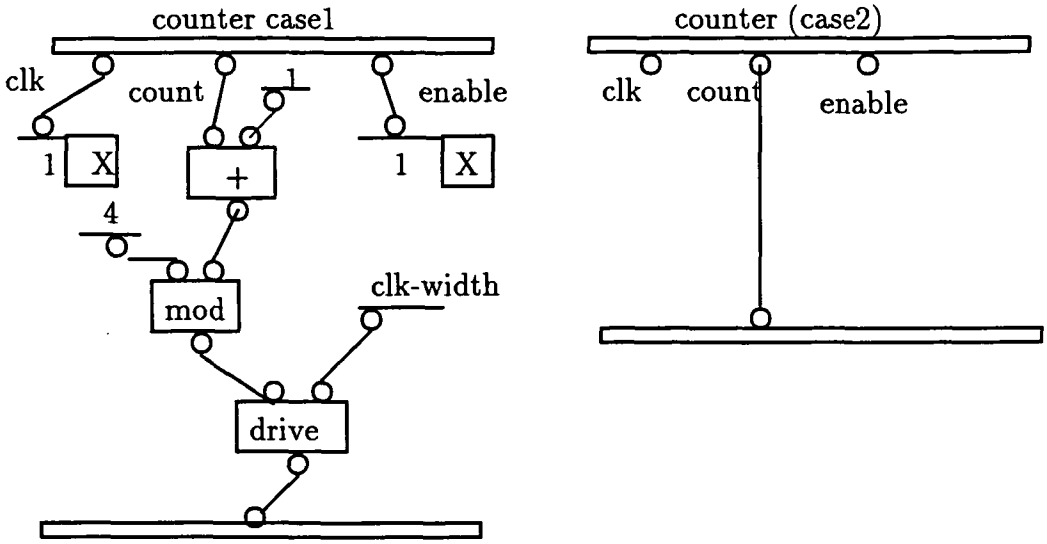


Figure 11: An example 4 mod up counter

(bus, multiplexors), etc. Each process in the template library has a certain function associated with it.

3 Visual Editors for the Visual Hardware Behavioral Language

In our proposed visual hardware behavioral language, the block is the most top model which may contain some processes and the instances of other blocks. When one is working in the block editor, a new model is required. The block editor is shown in Figure 12. In this editor, a mouse is used to direct a cursor about the screen, to control selections from drawing modes, editing modes and a set of pop-up menus, and to select and position design objects on the screen. Pop-up menus and overlapping windows are created on the fly as needed to select items for creation and to provide access to the file system. Drawing a rectangle or hexagon within the editing area creates an anonymous ("unbound") model instance, the model instance can be given a name, and ports can be added and annotated (given names, types, widths and modes). If we type the name inside the model instance area, a search can take place for a block or a process instance created before with the same name and the model instance is bounded if a match is found. Otherwise the model instance remains unbound. If the new model instance is a block and unbound, a double click on this instance can create a new block of the same name and enter the block editor. If the new model instance is a process and unbound, a double click on this unbounded process would enter the dataflow editor which is shown in Figure 13. A double click on a bounded model instance would enter the editor corresponding to the model that the instance is bounded to.

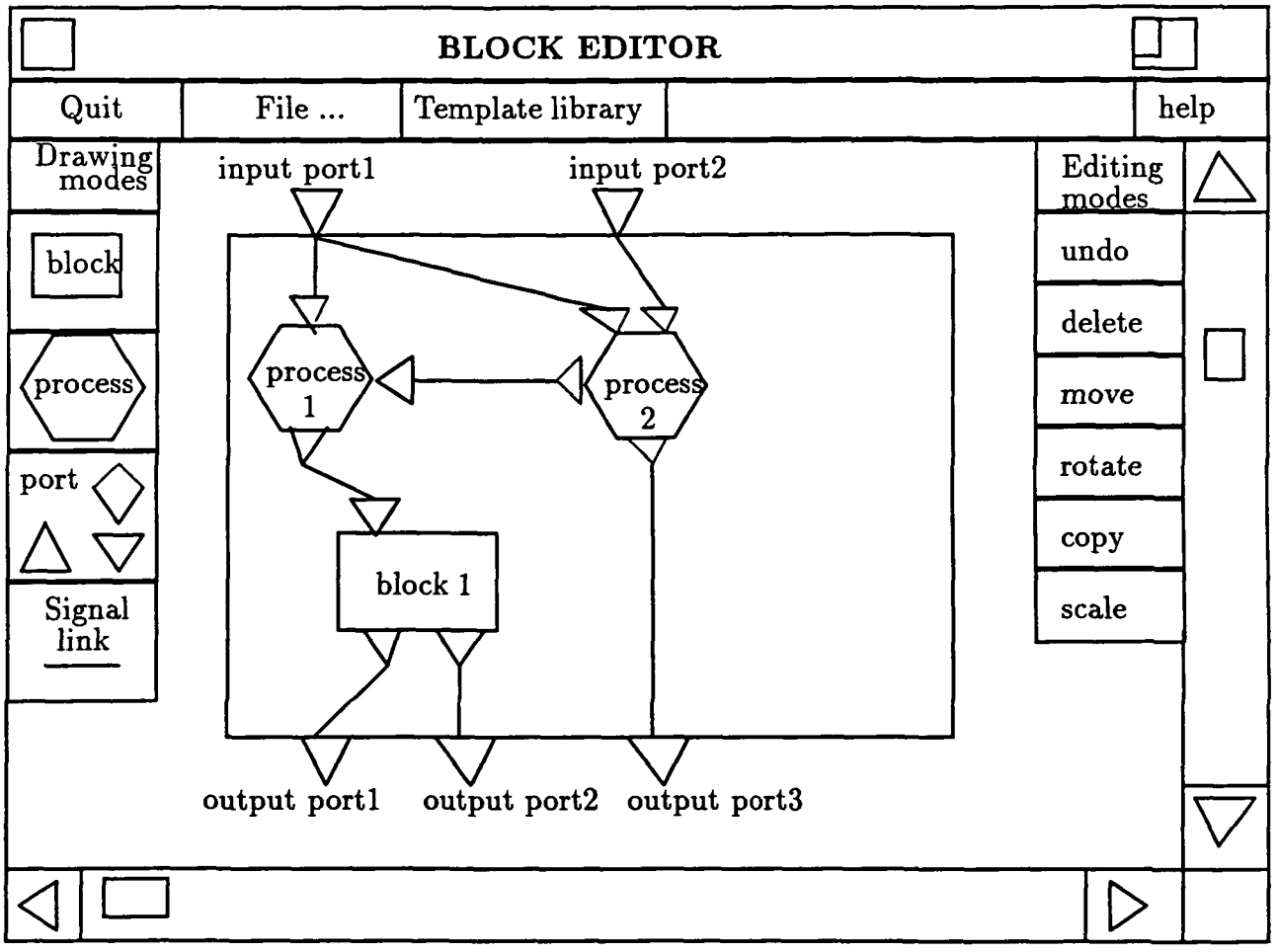


Figure 12: The Block Editor

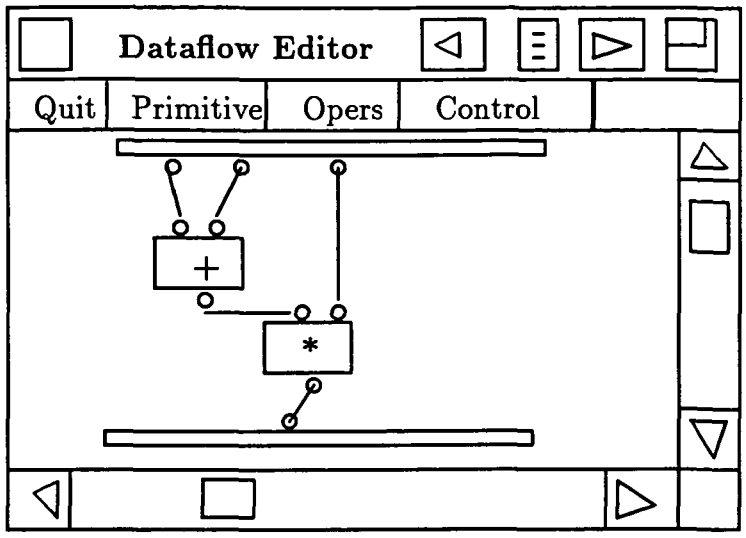


Figure 13: The data flow editor

4 Conclusions

The major features of the proposed visual hardware behavioral language are as follows:

- The ability to design abstract information graphically.

  It can graphically display parallelism, sequentiality and disjointness using parallel model (visual blocks), sequential model (visual processes) and data flow graph on the screen.

- Two level behavioral descriptions.

  At the architectural level, we use blocks and cooperating processes to describe the communication and connected behavior of the processes within the design. At the functional level, we use data flow graph to describe the algorithm accomplishing certain functions.

- Data flow style.

- Easy to learn and use.

  Because the proposed visual hardware behavioral language provides graphical information such as diagrams (rectangles for blocks and circles for processes) and operational icons in the actual process of hardware design, it is easy to learn and use for VLSI designers.

References

[1] A Rountable, "Behavioral description languages - PART I: Are designers Benefiting?". IEEE Design Test of Computers, February 1990, pp56 - 62.

[2] R. Camposano, L. F. Saunders and R. M. Tabet, "VHDL as Input for High - Level Synthesis" IEEE design Test of Computers, March 1991, pp43 - 49.

[3] D. Ku and G. De Micheli, "Hardware C: A Language for Hardware Design" tech, rpt. CSL-TR 90- 419, Computer System Lab., Stanford University, August 1990(version 2.0).

[4] "Prograph Tutorial" The Gunakara Sun Systems Limited, 1990.

[5] "Prograph Reference" The Gunakara Sun Systems Limited, 1990.

[6] Shi-kuo Chang, "Principles of Visual Programming Systems" Prentice-Hall. Inc, 1990.

[7] Alex Orailoglu and Daniel Gajski, "Flow Graph representation" IEEE 23rd Design Automation Conference, 1986, pp503 -509.

[8] Akira Sugimoto, "VEGA: A Visual Modeling Language for Digital Systems" IEEE Design Test of Computers, June,1991, pp38 - 45.

[9] Paul J. Dronggowski, Juahar R. Bammi and Tsu-Hua, Wang,"A graphical hardware Design Language", IEEE 25th Design Automotion Conference, 1988, pp108-144.

A Mechanized Process Algebra for Verification of Device Synchronization Protocols [1]

E. Thomas Schubert [2]
Department of Computer Science
University of California, Davis

Abstract - **We describe the formalization of a process algebra based on CCS, within the HOL theorem-proving system. The representation of four types of device interactions and a correctness proof of the communication between a microprocessor and MMU is presented.**

1 Introduction

This paper describes a methodology to formally verify the correctness of the synchronization and communication within a hardware system. The methodology is based on the formalization of a process algebra within the HOL theorem-proving system. Using this formalization, we show how four types of device interactions can be represented and proven to behave as specified. This paper also includes a correctness proof for the communication between the microprocessor and an memory management unit.

Previous system integration efforts have focused on vertical integration of one layer on top of another [1, 2]. This work examines the "horizontal" integration of communicating devices. Device communication may require only a single message or a series of messages to be passed between two devices. The types of interactions can be loosely described as belonging to one of the following categories: remote procedure call, process creation (fork), message passing, or rendezvous. To demonstrate how system integration can be achieved, we will show how several devices can be composed to form a system that uses these communication protocols.

The remainder of this section discusses background material, Section 2 presents the mechanization of the process algebra within the HOL theorem prover, and Section 3 presents specification and verification examples using the formalized process algebra, including a CPU-MMU composition proof. The final section summarizes this work and describes future extensions.

1.1 Related Work

Hardware verification requires that the design of a system is formally shown to satisfy its specification through a mathematical proof. Using theorem-proving techniques, an expres-

[1]This work was performed under Boeing Contract NAS1-18586, Task Assignment No. 3, with NASA-Langley Research Center.

[2]Author's current address is: Department of Computer Science, Portland State Univeristy

1.1 Related Work

Hardware verification requires that the design of a system is formally shown to satisfy its specification through a mathematical proof. Using theorem-proving techniques, an expression describing the behavior of a device is proven to be equivalent in some sense to an expression describing the implementation structure of the device. These expressions concisely describe the behavior of devices in an unambiguous way. An additional benefit of hardware verification is that the behavioral semantics of the hardware are clearly defined. This provides an accurate basis for building correct software systems [1].

Hardware verification efforts thus far have focused primarily on a microprocessor as the base for computer systems [2, 3, 5]. However, these processors are quite limited. The processors verified have modeled small instruction sets and, generally, have not included modern CPU features such as pipelines, multipled functional units, and hardware interrupt support. Tamarack-3 [2] and AVM-1 [6] do provide sufficient interrupt support to connect with an interrupt controller. However, no system currently verified provides the memory management functions necessary to support a secure operating system.

Process algebras and concurrency theories seek to provide formal models that aid in our understanding of the behavior of such systems. The archetype for the process algebra developed, is the Calculus of Communicating Systems (CCS) [7]. Despite the seemingly simple syntactic definition of CCS, the semantics of its primitive operators are carefully defined and allow complicated communication schemes to be accurately represented.

2 Mechanization of the Process Algebra

This section will describe our work to represent CCS in HOL. This representation has allowed us to easily prove several of the CCS semantic laws. To develop the process algebra in the HOL logic, we take advantage of several type definition mechanisms provided by the logic. Using these facilities we define an initial algebra for *agents*, which are constructed from sequences of *actions*. The purpose of types in higher-order logic is to prevent the inconsistency that higher-order variables can induce. The recursive type definition facility [8] automates the process of defining new data types in terms of already existing types. Both constants of the new type and type operators can be defined. Type operators are constructor functions, used to build compound members of new type from the type constants. The properties of these new types are then derived by formal proof. This guarantees that the new type does not introduce inconsistency into the logic. Additional recursive operators can be defined to operate on the concrete data representation of the type.

2.1 Actions

Using the recursive type definition facility, *actions* are defined to be either internal or external transitions. The internal action represents the τ action of CCS. External transitions require an *label*, which consists of a name (string) and boolean value, that denotes whether the action is a send or receive synchronization operation.

```
new_type_abbrev('name', ":string");;
new_type_abbrev('label', ":bool#name");;

let ACTION = define_type 'action'
   'action = INTERNAL | EXTERNAL label';;
```

2.2 Agents

Type representations the syntax of a concrete data type and a term algebra is also formed. Good practice suggest that the representation definition of any type, should minimize the number of type operators. This is certainly true if the semantics of some operators can be defined in terms of others. For determining the equivalence of two agents, we would like to define all agents in a normal form with agent terms consisting of only the prefix and summation operators. To describe processes that execute in parallel, we adapt a method described in [9]. This technique allows concurrently executing agent expressions to be represented with only the prefix and summation operators. Three mutually recursive functions replace the composition type constructor:

1. A communication operator, COMM, that declares that two agents will communicate if they have complementary, enabled actions.
2. A left-merge operator, LMERGE, that creates a new agent from two agents, such that the new agent must first behave as though only the left agent (first argument) were present and followed by an agent constructed by the composition of the resulting left agent and original right agent.
3. A compose operator ("merge" in the literature), COMPOSE, that creates an arbitrary interleaving of the two agents (through use of the summation type constructor).

The functions relate through the following laws, where the symbol "||", denotes the COMPOSE operator, the symbol "|" denotes the COMM operator, and "L" denotes the left-merge operation.

1. Given $P = a.P' and Q = b.Q'$

   $P \mid Q = (a = COMPLEMENT\ b) \rightarrow \tau.(P' \mid\mid Q')\ else\ \mathbf{0}$
2. $x \mid\mid y = (xLy) + (yLx) + (x \mid y)$

3. $aLx = a.x$
4. $a.xLy = a.(x \,||\, y)$
5. $(x + y)Lz = (xLz) + (yLz)$
6. $ax \mid bx = (a \mid b).(x \,||\, y)$
7. $(x + y) \mid z = (x \mid z) + (y \mid z)$
8. $x \mid (y + z) = (x \mid y) + (x \mid z)$

HOL does not provide a mechanism for defining mutually recursive functions, such as COMPOSE and LMERGE. Instead, two auxiliary compound type operators are present in the type definition with axiomd defining their semantic meaning.

```
let AGENT = define_type 'agent'
   'agent = INACTIVE
            | PREFIX action agent
            | SUMM agent agent
            | COMPOSE agent agent
            | LMERGE agent agent
            | RESTRICT label agent';;

COMM_DEF_AX =  ⊢ ∀ A B a b.  COMM (PREFIX a A) (PREFIX b B) =
         IS_COMPL_ACT a b ⟶ PREFIX INTERNAL (COMPOSE A B) |  INACTIVE

COMM_DIST_AX = ⊢ ∀ A B C. (COMM (SUMM A B) C = SUMM (COMM A C) (COMM B C))  ∧
               (COMM A (SUMM B C) = SUMM (COMM A B) (COMM A C))
COMPOSE_AX = ⊢ ∀ A B. COMPOSE A B = SUMM(SUMM(LMERGE A B) (LMERGE B A)) (COMM A B)

LMERGE_AX = ⊢ ∀ A B a. LMERGE (PREFIX a A) B = PREFIX a (COMPOSE A B)

LMERGE_DIST_AX = ⊢ ∀ A B C. LMERGE (SUMM A B) C = SUMM(LMERGE A C)(LMERGE B C)
```

2.3 Transition Semantics

Having defined the form of agents in the previous section, the transition semantics of an evolving agent (e.g., A $\xrightarrow{a}$ B) can be captured using the inductive relation definition package. The inductive relation definition package provides a set of theorem-proving tools based on a newly derived principle of definition in HOL for defining relations inductively by a set of rules [10]. Rules consist of a list of premisses and side conditions and a conclusion. Each premiss must make a positive assertion about membership in the relation. Side conditions may be arbitrary propositions not involving the relation being defined (as an example, see the TRANS_RES law definition below). The rules are essentially implications: if the premisses and side conditions hold, then the conclusions hold. The relation is inductively defined by a collection of such rules to be the least relation closed under all the rules.

The TRANS function defined below states that for an agent to evolve to another agent, there must be an immediate transition by a prefixed action or a set of premisses must be

satisfied. To embody the CCS laws, symmetric forms are needed for summation laws and the composition laws.

```
let (TRANS_rules, TRANS_ind) =
    let  TRANS = "TRANS:agent->action->agent->bool" in
    new_inductive_definition false 'TRANS'  ("^TRANS x a y",[])
    [
 % ACT LAW %
[],
 %----------------------------------------%
        "^TRANS (PREFIX a y) a y"
;% SUM1 LAW %
[           "^TRANS x a y"    ],
 %----------------------------------------%
              "^TRANS (SUMM x z) a y"
;% SUM2 LAW %
[           "^TRANS x a y" ],
 %----------------------------------------%
              "^TRANS (SUMM z x) a y"
;% COM1 LAW %
[           "^TRANS x a y" ],
 %----------------------------------------%
  "^TRANS (COMPOSE x z) a (COMPOSE y z)"
;% COM2 LAW %
[           "^TRANS x a y" ],
 %----------------------------------------%
          "^TRANS (COMPOSE z x) a (COMPOSE z y)"
;% COM3 LAW %
[   "^TRANS x (EXTERNAL (T,s)) x'";
    "^TRANS y (EXTERNAL (F,s)) y'" ],
 %----------------------------------------%
 "^TRANS (COMPOSE x y) INTERNAL (COMPOSE x' y')"
;% RES LAW %
[   "^TRANS x a y";
    "~( (LBL a) = (SND (r:label)))" ],
 %----------------------------------------%
          "^TRANS (RESTRICT r x) a (RESTRICT r y) "    ];;
```

2.4 Agent Equivalence

Several notions of equivalence between agents can be defined. The kinds of models that the initial algebra satisfies will be determined by which notion of equivalence is used. Trace semantics can be shown by defining the meaning of an agent as a set of traces where a trace is a list of actions. Trace semantics permit fairly broad equivalence classes to be constructed. We are frequently interested in a narrower definition of equivalence where two agents are equivalent if an external agent cannot distinguish between the visible behavior (traces) of the two agents. When using the notion of strong equivalence, traces consist of both external and internal actions. For our application, a weaker notion of observation equivalence where internal actions cannot be detected by the external agent, is sufficient.

We are actually interested in a weaker form: one-way observation equivalence. Under this definition, P implements Q's behavior if for every action α of Q, every α-derivative of Q

is one-way observation equivalent to some α-descendant of P. For the remainder of the paper we will use the term observation equivalence to mean one-way observation equivalence.

Observation equivalence can be defined in terms of an inductive relation definition. Observation equivalence laws are defined for compound terms, which are constructed using only the PREFIX or SUMM operators.

```
let (OE_rules, OE_ind) =
    let OE = "OE:agent->agent->bool" in
    new_inductive_definition false 'OE'
    ("^OE A B",□)
    [[
      %1----------------------------------------% ],
                "^OE A A"
;
     [          "^OE A B"
      %2----------------------------------------% ],
             "^OE (PREFIX a A) (PREFIX a B)"
;
     [          "~(TRANS_ACT_EXISTS B C) /\
                ^OE A C"
      %3----------------------------------------% ],
             "^OE (SUMM A B) C"
;
     [          "~(TRANS_ACT_EXISTS A C) /\
                ^OE B C"
      %4----------------------------------------% ],
             "^OE (SUMM A B) C"
;
     [
                "^OE A C";
                "^OE B C"
      %5----------------------------------------% ],
             "^OE (SUMM A B) C"
;
     [
                "^OE A B";
                "^OE A C"
      %6----------------------------------------% ],
             "^OE A (SUMM B C)"
    ];;
```

Rules 1 and 2 are straightforward; observation equivalence is reflexive, and two agents prefixed with the same action are observation equivalent if the agents without the prefixed action are observation equivalent.

If the left-hand-side agent is a summation of two agents (i.e. OE (SUMM A B) C) rules 3–5 may apply, and one-way observation equivalence is achieved if either:

1. Both of the summation agents (A and B) satisfy observation equivalence with the right-hand-side agent (C). This is rule number 5, OE_LSUM.

2. The left summation agent (A) satisfies observation equivalence with the right-hand-side agent (C) **and** there is no action for which a transition for both the right summation agent (B) and the right-hand-side agent (C) exists. This is rule number 3, OE_LSUML.

3. The right summation agent (B) satisfies observation equivalence with the right-hand-side agent (C) **and** there is no action for which a transition for both the left summation agent (A) and the right-hand-side agent (C) exists. This is rule number 4, OE_LSUMR.

The last rule states that if the right-hand-side agent is a summation agent, then the left-hand-side agent must satisfy observation equivalence for both of the right-hand-side summation agents. This is the symmetric case of rule number 5 for a left-hand summation.

Note that symmetric rules for rules 3 and 4 do not exist. If they were present, the OE relation would specify trace semantics. For example,

$$OE\ (a.0 + b.0)\,(b.0 + a.0)$$

requires both:

$$OE\ (a.0 + b.0)\,(a.0) \quad \text{and} \quad OE\ (a.0 + b.0)\,(b.0)$$

Adding the symmetric rules would allow relations such as $OE\ (b.0)\,(a.0 + b.0)$ to be true. If the semantic meaning of the OE relation is read as "implements", then $(a.0+b.0)$ implements $(a.0)$, but,$(a.0)$ does not satisfy/implement $(a.0 + b.0)$.

3 Mechanization of Device Interactions

3.1 Generic Interactions

Device communication may require only a single message or a series of messages to be passed between two devices. At a lower level, the information is passed over a bus using a hardware protocol (e.g., 4-phase handshaking). The types of interactions can be loosely described as belonging to one of the following categories:

1. Remote Procedure Call,
2. Message Passing,
3. Process Creation (fork), and
4. Rendezvous.

These forms of interaction are described in concurrent programming literature. While the Ada programming language provides primitive support for only remote-procedure-call and rendezvous features, the SR programming language provides primitive statements for each of the above interaction forms.

6.2.8

3.1.1 Remote Procedure Call

A CPU and a memory subsystem interact in a remote-procedure-call manner. The CPU sends a memory request to the subsystem and waits for a response. If the memory subsystem includes a memory management unit (MMU), the MMU and the actual memory may also interact in a remote procedure form. In the example below, we show a proof of a 4-phase handshaking protocol between a CPU and a bus.

```
let cpuRaiseRead = '    -'cpu_addr'.-'cpu_rReq'  ';;
let cpuReadMem   = '    . 'dataAvail'.'data'       ';;
let cpuDropRead = '     . -'cpu_addr'.-'cpu_rReq'  ';;
let cpuReadComplete = '  . SUCCESS ';;

let busGetRead     = '    'cpu_addr'.'cpu_rReq'  ';;
let busReturnMem   = '    . -'dataAvail'.-'data'       ';;
let busDropRead = '     . 'cpu_addr'.'cpu_rReq'  ';;
let busReadComplete = '  . SUCCESS ';;

let cpuRead = cpuRaiseRead^cpuReadMem^cpuDropRead^cpuReadComplete;;
let busRead = busGetRead^busReturnMem^busDropRead^busReadComplete;;

let system_def = '('^cpuRead^')|('^busRead^')';;
let system = Agent system_def;;

let system_done = ' SUCCESS | SUCCESS ';;
let success = Agent system_done;;
```

3.1.2 Message Passing

Devices also interact through a message-passing format. For example, consider the interaction between a CPU and an interrupt controller. These devices operate simultaneously, performing independent tasks, and may never interact (this is, of course, unlikely). If an enabled I/O device generates an interrupt, the system's interrupt controller will send a message to the CPU indicating that some device requires attention. At this point the CPU may or may not respond. Depending on the signaling discipline (e.g., *signal-and-continue* or *signal-and-wait*), the interrupt controller may also continue to operate. The example below shows how the interaction between a CPU and a PIC can be modeled.

```
let cpuInt         = ' 'intPending'.CPU_PROCESS_INT';;
let cpuIntEnabled  =  cpuInt^ '+ tau.FETCH_INSTRUCTION';;
let cpuIntDisabled = ' FETCH_INSTRUCTION ';;

let picInt         = '  -'intPending'.PIC_PROCESS_INT ';;
let picReq         =  picInt^ ' + tau.PIC_CHECK  ';;
let picNoReq       =  ' PIC_CHECK ';;

let system_def = '('^pic^')|('^cpu^')';;
```

3.1.3 Process Creation and Rendezvous

The interaction between a CPU and a Direct Memory Access device (DMA), can be characterized as process creation followed by a rendezvous. The process of initializing the DMA to supervise the transfer of data from an I/O device to memory, is a form of process creation. When the DMA has completed its task, the CPU (through an interrupt controller) will be signaled. In many circumstances, the CPU activity must rendezvous with the completion of the DMA activity. For example, when a new code page must be brought into memory for an executing program to continue, the CPU may switch to other executing programs, but once all other programs have completed, the CPU (really the OS kernel) must wait for the DMA activity to complete.

```
let dmaProcCreate  =   ‘  cpuWriteReg.cpuInitExec.DMA_EXEC‘;;
let dmaSleep       =   dmaProcCreate;;
let dmaDone        =   ‘ -’dmaInt’.dmaSleep ‘;;
let dma            =    dmaSleep;;

let cpuDMACreate   = ‘ -’cpuWriteReg’.-’cpuInitExec‘;;
let cpuDMAWait     = ‘ ’dmaInt’.cpuContinue ‘;;
```

3.2 CPU/MMU Connection Specification

To determine how a CPU and MMU interact, we examine the external interface of AVM-1 with memory and the interface that the MMU provides to the CPU. Below we show the AVM-1 memory interface specification. If a write request is made at time t the memory will reflect this request at time $t + 1$, otherwise, the memory will remain unmodified. If a read request is made at time t then the data value returned is a function of the memory contents at time t.

```
let MEM = new_definition
   (‘MEM‘,
    "! (rep:^rep_ty) wr_s rd_s addr data mem.
     MEM rep wr_s rd_s addr data mem =
     !t:time .
        (mem (t+1) =
           (wr_s t => store rep (mem t, address rep (addr t), (data t))
                    | mem t)) /\
        (rd_s t ==> (data t = (fetch rep (mem t, address rep (addr t)))))"
   );;
```

This specification is incomplete for the purposes of connecting the CPU with the MMU verified in [11]. There is no external line to indicate the security status of an executing process (e.g., the supervisor line); nor is there a return code indicating whether the memory request was validated. Additionally, the specification assumes that memory operations occur in a single cycle. This last consideration could be dealt with by an appropriate temporal abstraction.

Below is a modified specification that includes these features. A security line is added to the specification and an acknowledgment variable (ack) is added to inform the CPU of invalid memory requests. This variable is set based on a new abstract function memMgt. The abstract CPU functions, store and fetch, now expect the supervisor state as an additional argument (superV). These functions should not perform as requested when security/safety requirements are not satisfied.

```
let MEM = new_definition
('MEM',
 "! (rep:^rep_ty) wr_s rd_s addr data mem superV ack.
  MEM rep wr_s rd_s addr data mem superV ack =
  !t:time . (mem (t+1) =
      (wr_s t => store rep (mem t, address rep (addr t), (data t), superV t)
               | mem t)) /\
  (rd_s t ==> (data t=(fetch rep (mem t, address rep (addr t),superV t)))) /\
  (ack t = memMgt rep (mem t, address rep (addr t),data t,superV t,wr_s t))";;
```

With these additions, the process algebra term for this interface can be defined as below.

$$\mathsf{cpu\_write\_request} = (\overline{userMode} + \tau.\overline{superMode}).\overline{write}.\overline{address}.\overline{data}.(ack + nack)$$

$$\mathsf{cpu\_read\_request} = (\overline{userMode} + \tau.\overline{superMode}).\overline{read}.\overline{address}.(ack + nack).data)$$

$$\mathsf{CPUtoMEM} = \mathsf{cpu\_write\_request} + \tau.\mathsf{cpu\_read\_request}$$

$$\mathsf{CPU} = \mathsf{CPUtoMEM.CPU}$$

These terms are abbreviations for their actual representations in HOL. The PREFIX operator is defined to construct an agent from an action and an agent rather than from two agents as presented here.

Note the use of the internal operator to indicate that at given points, the CPU will behave in one of two ways: communicate through the $\overline{userMode}$ action or through the $\overline{superMode}$ action. This choice is made internally by the CPU. Without this use of τ, the terms would mean that communication could occur by either action, depending on what an external agent might choose. This use of the τ action can also be seen in the process algebra terms for the MMU below. To express the notion that the MMU performs some work before responding, the τ action is inserted before the response is returned. Part of this action may be to update the segment-table pointer value. While the system is able to accept either a *userMode* or a *superMode* communication, the MMU chooses to respond with only $\overline{ack}$ or $\overline{nack}$ communication. The MMU specification also states when the MMU segment-table pointer is updated. This action is not relevant to the CPU -MMU interface, so it is expressed as an internal (τ) action. This specification yields the process algebra terms:

$$\mathsf{mmu\_process\_write} = (userMode + superMode).write.address.data.\tau.(\overline{ack} + \tau.\overline{nack})$$

$$\mathsf{mmu\_process\_read} = (userMode + superMode).read.address.\tau.(\overline{ack} + \tau.\overline{nack}).\overline{data}$$

$$\mathsf{MMUtoCPU} = \mathsf{mmu\_process\_write} + \mathsf{mmu\_process\_read}$$

MMU = MMUtoCPU.MMU

System = CPU | MMU

3.3 Proof of Correct Composition

The agents CPU and MMU are recursively defined and exhibit an infinite behavior. To show that the composed CPU and MMU communicate correctly, we must show that the system is always able to return to its initial state. It is also necessary to show that progress is made when the CPU initiates a dialogue with the MMU. The proof goal then can be stated as:

1. If either of the CPU actions, $\overline{userMode}$ or $\overline{superMode}$, are enabled, the memory subsystem will engage in communication.
2. The communication protocol will complete and the system returns to its initial state.

To formalize and prove this goal in HOL, agent terms are defined based on process algebra terms. To reason about the finite protocol communication sequences, the recursive behavior of the agents is removed. The recursive agent reference is replaced with an (undefined) agent constant SUCCESS.

In the box below, we show the construction of the MMU process algebra term. The CPU term is defined in a similar manner. The term is constructed in parts by building up an ML string. The ML function Agent parses the ML string and returns a HOL agent term. ML strings are delimited by backquotes (‘) and the string concatenation operator is the caret symbol (^).

```
let muw = '('write'.'addr'.'data'.((-'ack'.SUCCESS)+(T.-'nack'.SUCCESS)))';;
let mur = '('read'.'addr'.((-'ack'.'data'.SUCCESS)+(T.-'nack'.'data'.SUCCESS)))';;

let msw ='('write'.'addr'.'data'.((-'ack'.SUCCESS)+(T.-'nack'.SUCCESS)))';;
let msr ='('read'.'addr'.((-'ack'.'data'.SUCCESS)+(T.-'nack'.'data'.SUCCESS)))';;

let user_mmu  = '( 'user'  .('^ muw ^'+'^ mur ^'))';;
let super_mmu = '( 'super' .('^ msw ^'+'^ msr ^'))';;
let mmu = '('^ user_mmu ^'+'^ super_mmu ^')';;
let MMU = Agent mmu;;
```

To express the goal in a general form, several auxiliary definitions are defined. A recursive definition (ENABLED) is defined to construct a list of all output actions that an agent can perform. A goal predicate definition BECOMES is defined. The predicate states that for all possible enabled output actions, a complement action exists such that a success agent is reached immediately or reached in a descendent.

```
let ENABLED = new_recursive_definition false AGENT 'ENABLED'
    "(ENABLED(INACTIVE)      = []) /\
     (ENABLED(PREFIX a A)    = ( (INTERNAL=a) => ENABLED A |
                                 (( (TYP a) = F) => [a] | [] ))) /\
     (ENABLED(SUMM A B)      = APPEND (ENABLED A) (ENABLED B)) /\
     (ENABLED(COMPOSE A B)   = APPEND (ENABLED A) (ENABLED B) )";;

let BECOMES = new_definition('BECOMES',
  "!(system success :agent).  BECOMES system success =
      (EVERY (\x. (TRANS system x success)) (ENABLED sys) )");;
```

The success agent for the composed MMU-CPU is $SUCCESS \mid SUCCESS$. By unwinding the agent definitions and using the TRANS laws, all possible paths are found to reach the success agent through internal transitions. The MMU-CPU composed communication proof shows that:

```
⊢ BECOMES  (CPU | MMU)  (SUCCESS | SUCCESS)
```

4 Conclusions

We have presented a framework to formally verify the correctness of communication between composed devices. Previous system verification research has developed *vertically verified systems.* However, the hardware bases for these systems have been simplistic. Our research is developing a framework to verify a more realistic *horizontally verified system.* This work demonstrates that CCS is a good choice for describing interdevice implementation-level connections within a computer system. Additional research will expand the calculus and address automating the derivation of process algebra expressions from interpreter specifications. Several improvements are being investigated, including an additional type constructor for recursive agents, greater proof support, and automation of the tedious aspects of the proofs.

References

[1] W. R. Bevier, W. A. Hunt, and W. D. Young, "Toward verified execution environments," *IEEE Symposium on Security and Privacy*, 1987.

[2] J. J. Joyce, *Multi-Level Verification of Microprocessor-Based Systems.* PhD thesis, Cambridge University, December 1989.

[3] A. Cohn, "A proof of correctness of the VIPER microprocessor: the first level," in *VLSI Specification, Verification, and Synthesis* (G. Birtwhistle and P. Subrahmanyam, eds.), pp. 27–71, Kluwer Academic Press, 1988.

[4] W. A. Hunt, "A verified microprocessor," Tech. Rep. 47, The University of Texas at Austin, Dec. 1985.

[5] W. A. Hunt, "Microprocessor design verification," *Journal of Automated Reasoning*, vol. 5, pp. 429–460, 1989.

[6] P. J. Windley, *The Formal Verification of Generic Interpreters.* PhD thesis, University of California, Davis, 1990.

[7] R. Milner, *Communication and Concurrency.* Prentice Hall, 1989.

[8] T. Melham, "Automating recursive type definitions in higher order logic," in *Current Trends in Hardware Verification and Automated Theorem Proving* (G. Birtwhistle and P. Subrahmanyam, eds.), pp. 341–386, Springer-Verlag, 1989.

[9] J. C. M. Baeten and W. P. Weijland, *Process Algebra.* Cambridge University Press, 1990.

[10] T. Melham, "A package for inductive relation definitions in HOL," 1991.

[11] E. T. Schubert, "Verification of memory management units using HOL," Tech. Rep. CSE-90-27, University of California, Davis, August 1990.

HDL to Verification Logic Translator

J. W. Gambles and P. J. Windley
NASA Space Engineering Research Center for VLSI Systems Design
University of Idaho, Moscow, Idaho 83843
jgambles@groucho.mrc.uidaho.edu, 208-885-9041
windley@panther.cs.uidaho.edu, 208-885-6501

Abstract -**The increasingly higher number of transistors possible in VLSI circuits compounds the difficulty in insuring correct designs. As the number of possible test cases required to exhaustively simulate a circuit design explodes, a better method is required to confirm the absence of design faults. Formal verification methods provide a way to prove, using logic, that a circuit structure correctly implements its specification. Before verification is accepted by VLSI design engineers, the stand alone verification tools that are in use in the research community must be integrated with the CAD tools used by the designers.**

One problem facing the acceptance of formal verification into circuit design methodology is that the structural circuit descriptions used by the designers are not appropriate for verification work and those required for verification lack some of the features needed for design. We offer a solution to this dilemma: an automatic translation from the designers' HDL models into definitions for the higher-ordered logic (HOL) verification system. The translated definitions become the low level basis of circuit verification which in turn increases designers confidence in the correctness of higher level behavioral models.

1 Introduction

As higher transistor counts increase the complexity of VLSI circuits and the number of potential test cases explode, traditional simulation methods can expose only a fraction of design faults – not guarantee their absence. Formal verification methods, which *prove* circuit correctness, will play an important role in design fault exclusion. It is common in modern design methodologies to utilize abstract circuit models in a hierarchical design:

- An architectural model (i.e. highly abstract) can be used to simulate an entire system, at an early date, to help confirm that the system specification truly meets the customers needs.
- In a top-down design, a model of the system's architecture is refined to a less abstract model, and this decomposition process proceeds iteratively from algorithmic description, to large functional blocks, to detailed logic, and right down to the circuit level.
- After the circuit structure is modeled and designed, the logic simulation of complex systems can become very slow. Simulations run faster using behavioral models.

A problem with these design approaches is that there is no *formal* way to relate a circuit's structural model to its abstract behavioral model. Formal verification allows these models

to be related through mathematical analysis so that designers can enjoy increased confidence that behavioral models are *correct* abstractions of their structure. Before formal verification is accepted by design engineers, stand alone verification tools that are used in academic research must be integrated with the CAD tools used by VLSI designers.

The hardware description languages (HDL) used by VLSI CAD tools can provide the link between these tools and the verification environment. Engineers can design using their HDL and the models can be automatically translated for use in the verification tool. The translation process consists of two steps.

- Recognizing the syntax of the HDL.
- Constructing the translation from the syntax to the HDL's semantic domain.

The parser, for recognizing the syntax, and the translation semantic construction functions can be built directly into the verification system.

The NOVA simulation engine, one of the CAD tools being developed and used at the NASA Space Engineering Research Center (SERC) for VLSI Systems Design, located on the University of Idaho campus, uses the BOLT (Block Oriented Logic Translator) HDL. BOLT was chosen for this research because it provides ready access to many real-world VLSI designs at the SERC. This paper presents a translator from BOLT to the HOL theorem proving system.

Much has been published about theories for modeling MOS circuits in a verification environment [4, 5, 7, 11, 15]. Our work linking verification with VLSI design tools is related, but has a different motivation. While we are concerned that the model accurately reflect the true behavior of the devices being specified, we must also be concerned that the HOL circuit primitive definitions are consistent with the BOLT primitives used in NOVA. Correct modeling of MOS circuits requires a complex multi-valued, multi-strength data type for signal values[3]. Reasoning about such a signal value system can be done in HOL, where the signal value type (`STATE`) definition and a collection of theorems about manipulating `STATE` values is known collectively as the `STATE` theory[6]. Other work has been published dealing with translating HDLs to verification logics [2, 14]. Our interest in tying HOL to the NOVA simulator has motived our work to include lower level structures (i.e. multi-strength signal values and their resolution functions) where these problems have been largely ignored by others.

2 HOL

HOL, an acronym for higher-order logic, is a general theorem proving system developed at the University of Cambridge [4, 8] based on Church's theory of simple types. Higher-order logic is suitable for specifying all aspects of hardware, including both structure and behavior [8, 10]. In using higher-order logic, predicates are defined to represent both circuit primitives and behavioral definitions [4]. First-order logic is well suited to represent simple combinational circuits, but not sequential circuits. In higher-order logic, variables are allowed to range over functions and predicates, which makes it possible to represent sequential circuit behavior

[10]. HOL is not an automated theorem prover but is more than simply a proof checker. It could, more appropriately, be called a proof assistant.

The HOL system is implemented on top of Cambridge LCF, which is a direct descendant of the work of Robin Milner [8]. Milner originally developed an approach to mechanizing logic for a system called Logic of Computable Functions (LCF) designed for reasoning about higher-order recursively defined functions. The LCF meta-language is called ML, a functional programming language.

3 The BOLT to HOL Translator

The BOLT to HOL translator is comprised of a syntax parser, built using a parser-generator tool included with version 2.0 of the HOL system[13], and a set of ML functions that construct the semantics of the parsed BOLT syntax into HOL definition terms. The semantic construction functions were written *ad hoc*.

3.1 The Syntax Parser

The parser-generator takes as input a grammar representing the formal syntax of BOLT given in a modified Backus Naur Form (BNF) notation similar to Prolog's definite clause grammar (DCG) [13]. The output of the generator is a ML program that recognizes the HDL syntax and makes appropriate calls to the ML semantic construction functions, whose names are included as action symbols in the input grammar. The BOLT syntax is defined in [1]. The HOL parser-generator library developed at the University of Cambridge was found to be very useful in building the syntax recognizer portion of the translator. For example the syntax of a statement-body, as given in the BOLT manual, is:

```
BEGIN

[ { Module-Invocation | Case-Statement } ... ]

END ;
```

Where upper-case words are keywords, the expression [ ] contains an optional item that may be omitted, { } indicates choose one of the enclosed items, and ... indicates that an item may be repeated any number of times is entered into the parser-generator input grammar as the following recursive production:

```
statement_body  --> [BEGIN] invocation_list [END] [;]

invocation_list --> mod_invocation invocation_list |
                    case_statement invocation_list |
                    □.
```

3.2 The Semantic Construction Functions

Because no formal semantic definitions for BOLT exist, the semantic construction functions have been initially written in an *ad hoc* fashion. In order to construct a HOL definitional translation the following data from the BOLT module is required:

1. The module name from the BOLT module declaration. The same name is used for the HOL definition.
2. The set of ports declared to be external to the top module in the BOLT declaration. The signals on these ports must be universally quantified in the HOL definition.
3. The names of the component modules that are invoked inside the BOLT module. Each of these module invocations will cause an identically named HOL predicate to be conjoined in the HOL definition predicate.
4. The set of ports declared to be an output of any invoked module. This set will be used to determine the signals that are driven by more than one device. A `JOIN` predicate must be added to the HOL definition to resolve all interconnected outputs. This set will also be used in the identification of the internal ports that are to be hidden. In BOLT, the ports are not explicitly declared to be module input or outputs. By *convention* the outputs are listed before the module name and the inputs are listed following the module name. **Our translator relies on conformance to this convention.**
5. The set of ports declared to be an input to any invoked module. The union of this set and the set of invoked module output ports defines the set of all ports in the module. The set difference of the set of all ports minus the set of externally declared ports is used to define the set of internal signals that must be existentially quantified in the HOL definition.
6. The only module parameter with any meaning to our translation is the `STR` parameter, which in BOLT is used to define the output strength of an invoked module. If no `STR` parameter is included in the module invocation then the default strength is active.

As the BOLT syntax is parsed, the ML functions construct a data structure from the parsed tokens. It is this structure that is used for the creation of the HOL definition terms once the BOLT module `END;` statement is found. As the BOLT syntax is parsed, the ML functions construct a data structure from the parsed tokens. It is this structure that is used for the creation of the HOL definition terms once the BOLT module `END;` statement is parsed. The data structure is implemented as a list of lists of lists of strings. The form of the structure is:

$$
\begin{aligned}
Structure &\triangleq head : Header; body : Body \\
Header &\triangleq name : Identifier; ext\_out, ext\_in, int\_out, int\_in : Nodes \\
Identifier &\triangleq id : String \\
Nodes &\triangleq Identifiers^{*} \\
Body &\triangleq Invocation^{+} \\
Invocation &\triangleq name : Identifier; out, in : Nodes; param : STR \\
STR &\triangleq Identifier^{*}
\end{aligned}
$$

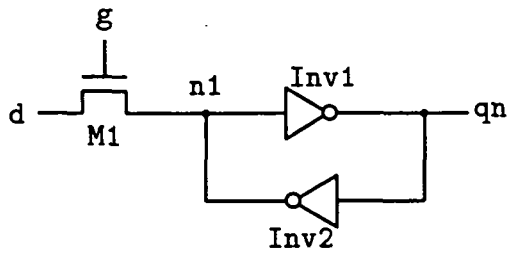


Figure 1: Latch Schematic

In this grammar, the asterisk ("Kleene star") has the standard language theory meaning — a list with zero, one, or more elements [12]. The plus means a list with one or more elements. The first sublist is the header part. It contains the module name and node lists corresponding to the external outputs, external inputs, internal outputs, and internal inputs. The body part is a list of component module invocations. Each invocation contains a module name, a list of output ports, a list of input ports, and a possibly empty paramenter list containing device output strength information. An example structure is shown in Section 4.5.

As each new module invocation is parsed, the module name, output node names, input node names, and optional output strength parameters are added to the data structure. Additionally, a check is made to see if the invoked module output node(s) are already a member of the set of internal output nodes for the current module. If it is, then two outputs are connected to drive the signal value on that node and a `join` resolution function from the `STATE` theory is required[6]. The `join` function is added by renaming the first instance of that output node name to the decorated (primed) variation of the name and the current invocation output is given the double-decorated node name variation. An invocation of `JOIN` is then added to the end of the data structure where the output of the `JOIN` is the original node name and the inputs are the new decorated and double-decorated nodes. If either the decorated or double-decorated names are already used then the first two unused decoration variations are added. A new blank sub-list is also appended to the end of the structure in anticipation of the next module invocation.

When the `END;` statement is encountered, the set of external output nodes unioned with the external input nodes are universally quantified in the resulting HOL definition. The set of internal nodes subtract the external nodes are existentially quantified (hidden). HOL terms are then generated by matching each invoked module with a previously defined HOL constant whose name and type both match the structure built by the parser and construction functions. The terms from all of the invoked modules are conjoined to complete the HOL definition for the current module. A stack is maintained for current module data structures so that embedded BOLT modules can be properly defined and translated.

4 Translator Demonstration

A data latch, implemented with gate level and pass transistor primitives, is used to demonstrate the translator (Figure 1). This circuit is interesting because without a signal value

representation and resolution function that realizes output dominance this circuit cannot be correctly modeled. Fundamental to the operation of this circuit is that the output strength of pass-transistor M1 dominates the output of inverter Inv2 to force node n1 to the state of the input d while the gate g is 1 (high voltage). The feedback inverter Inv2 acts to store the state, by dominating the pass-transistor after the gate goes to 0, turning the transistor off.

4.1 The BOLT Structural Description

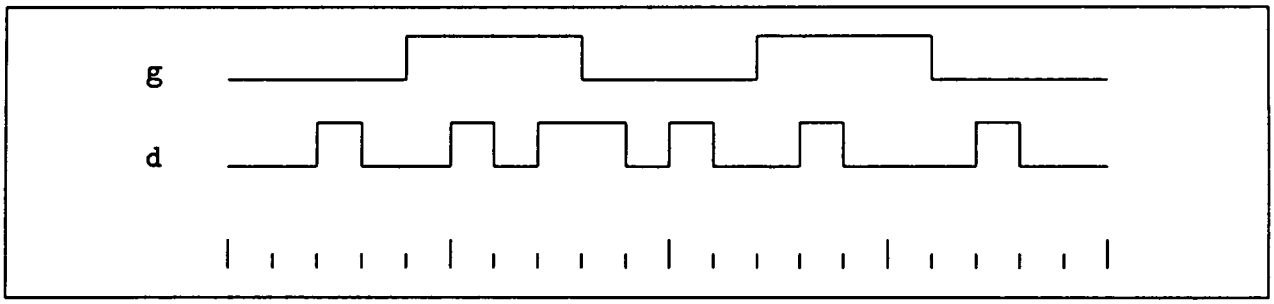
A BOLT description of the latch is:

```
MODULE qn .LATCH g d;
  BEGIN
    n1  .NTRAN g d;
    qn  .INVR  n1;
    n1  .INVR  qn  (STR='RR');
  END;
```

The STR='RR' parameter in the second .INVR invocation defines the output strength of that inverter as resistive. The default value used for the first invocation is active.

4.2 Simulating the Latch

The operation of the latch can be tested by exercising it with the NOVA simulator. The LATCH module was run in NOVA with the following waveforms on the g and d inputs:



The resulting simulator output is shown in Table 1, where the symbol 1 represents 1aa, 0 represents 0aa, and X represents Xaa.

4.3 The HOL Circuit Primitives

The latch structure includes three predicate definitions; a NMOS-transistor element, inverter element, and the JOIN operation. These primitive element definitions must be made in HOL before they can be used in a translation from BOLT. In HOL, time is represented as a stream of natural numbers (num), the signal values are defined to be of type STATE, and circuit signals are defined to be functions of type (num → STATE).

A simplified transistor model is used defining that the signal at the source is equal to the signal at the drain if the gate is a one, else it is Nil.

	q			q
	g d n			g d n
	* * *			* * *
00001>0	0 X		00011>0	1 0
00002>0	0 X		00012>0	0 0
00003>0	1 X		00013>1	0 1
00004>0	0 X		00014>1	1 0
00005>1	0 1		00015>1	0 1
00006>1	1 0		00016>1	0 1
00007>1	0 1		00017>0	0 1
00008>1	1 0		00018>0	1 1
00009>0	1 0		00019>0	0 1
00010>0	0 0		00020>0	0 1

Table 1: Latch Simulation Data

```
⊢def NTRAN (s,g,d) =
    (∀ t.
      s t = (((g t =laa)∨(g t =lar)∨
              (g t =lrr)∨(g t =laf)∨
              (g t =lrf)∨(g t =lff)) → d t |
                                       Nil))
```

The inverter predicate definition has five arguments. The first three arguments are of type STATE and define the possible inverter output values (i.e. the ouput strength). The first is the output STATE for a true state, the second for a false output, and the third the unknown state. The unknown output value is derived from the strongest 1 and 0 strengths. The fourth and fifth arguments are signal functions of type (num → STATE). The fourth is the inverter output and the fifth is the input.

```
⊢def INVR ls Os Xs (out,in) =
    (∀ t.
      out t = (((in t =laa)∨(in t =lar)∨
                (in t =lrr)∨(in t =laf)∨
                (in t =lrf)∨(in t =lff)) → Os |
               (((in t =Oaa)∨(in t =Oar)∨
                (in t =Orr)∨(in t =Oaf)∨
                (in t =Orf)∨(in t =Off)) → ls |
                                           Xs)))
```

4.4 JOIN

The JOIN predicate performs two tasks. It determines the resulting signal value of resolving the combination of circuit outputs by applying the join function from the STATE theory. The second task is related to the sequential behavior of a charge storage node. The capacitance of a node may result in a time delay when the node is driven to a new signal level. The

delay increases as the capacitance increases or as the strength of the driving signal decreases. This sequential behavior is modeled as having a variable delay, whose length is based on the strength of the `join` function result. [5, 9].

The `JOIN` used in the latch is modeled as having two possible delays. When the pass-transistor is turned on, the storage node at the join is driven by an active strength and the delay is defined to be zero. When the pass-transistor is turned off, the storage node is driven by the resistive strength of the feed-back inverter and the delay is defined to be one.

```
⊢def JOIN (s,s',s'') =
    (∀ t. let sig = join (s' t) (s'' t) in
          (((sig = 0aa) ∨
            (sig = 1aa) ∨
            (sig = Xaa) ∨
            (sig = Xar) ∨
            (sig = Xra)) → (s t = sig)     |
                           (s (t+1) = sig)))
```

4.5 The Translation of the Structural Specification

The HOL structural specification is obtained by translating the BOLT description. The translator may be invoked to operate on a file containing the BOLT description or on BOLT text included between the keywords `BEGIN_BOLT` and `END_BOLT` within the HOL operating environment. The result of translating the cell description is:

```
BEGIN_BOLT
  MODULE qn .LATCH g d;
    BEGIN
      n1  .NTRAN g d;
      qn  .INVR  n1;
      n1  .INVR  qn  (STR='RR');
    END;
END_BOLT
```

```
⊢def LATCH (qn,g,d) =
    (∃ n1 n1' n1''.
       NTRAN (n1',g,d) ∧
       INVR 1aa 0aa Xaa (qn,n1) ∧
       INVR 1rr 0rr Xrr (n1'',qn) ∧
       JOIN (n1,n1',n1''))
```

The data structure built by the parser and construction functions is:

```
[[['LATCH'];
  ['qn'];
  ['g'; 'd'];
  ['n1'; 'qn'; 'n1''; 'n1'''];
  ['g'; 'd'; 'n1'; 'qn']];
 [['NTRAN']; ['n1'']; ['g'; 'd']; []];
 [['INVR']; ['qn']; ['n1']; []];
 [['INVR']; ['n1''']; ['qn']; ['R'; 'R']];
 [['JOIN']; ['n1']; ['n1''; 'n1''']; []];
 [[]; []; []; []]]
: string list list list
```

4.6 The Behavioral Description

When the gate of the pass-transistor is true the latch is enabled and the output, qn, follows as the inverse of d. When the gate is false the latch stores the previous data. It is desirable to simplify the description as much as possible at each level. At the behavioral level the operation no longer depends on a device's output charge sourcing ability so this specification is written in terms of boolean signal values, not the more complex STATE data type. The HOL behavioral description is:

```
⊢def LATCH_SPEC (qn,g,d) =
    (∀ t.
      (g t → (qn t = ¬d t)    |
             (qn (t+1) = qn t)))
```

4.7 The Latch Verification

The proper operation of the latch requires that the output of the pass-transistor dominate the resistive strength output of INV2. The pass-transistor is not an amplifier so there is a validity condition that the signal applied to input d must be stronger than resistive.

```
⊢def Is_bool_active (d) =
    (∀ t. (d t = 1aa ) ∨ (d t = 0aa))
```

Because the behavior of the latch is defined only for boolean value signals at the gate, there is a validity condition for the gate that it be either a 1 or 0 state. This condition yields a 12 way case analysis in the proof that is easily reduced to considering only the two cases of enabled and latching.

```
⊢def Is_bool (g) =
    (∀ t.
      (g t = 1aa) ∨ (g t = 1ar) ∨
      (g t = 1rr) ∨ (g t = 1af) ∨
      (g t = 1rf) ∨ (g t = 1ff) ∨
      (g t = 0aa) ∨ (g t = 0ar) ∨
      (g t = 0rr) ∨ (g t = 0af) ∨
      (g t = 0rf) ∨ (g t = 0ff))
```

The verification of the latch entails proving that the latch structural description and validity conditions logically imply the behavioral specification. Because the latch behavorial specification is defined in terms of boolean values the signal functions must be composed with a STATE abstraction function from the STATE theory[6]. The theorem proven is:

```
⊢ ((Is_bool_active (d) ∧
    Is_bool (g) ∧
     LATCH (qn,g,d))   ⇒
     LATCH_SPEC (STATES_ABS o qn,
                 STATES_ABS o g,
                 STATES_ABS o d))
```

5 Future Work

The BOLT to HOL translator presented in this paper represent an important step in integrating formal verification with CAD tool environments. Future steps include:

1. Expanding and validating the library of HOL definitions corresponding to the primitive components in the NOVA library.

2. Developing an abstract syntax and denotational semantics for the circuit structure level of HDLs.

3. Using the abstract syntax and denotational semantics, developing a translator generator that will, given a grammar representing the concrete syntax of a HDL, automatically create a translation program for that HDL.

4. Integrating the circuit structure level translation work with the results of other ongoing research aimed at HDL behavioral model levels to create a complete link between HDL's and verification logics.

6 Conclusion

The goal of our work is to improve CAD functional fault exclusion techniques for VLSI design by making the use of formal circuit verification at the transistor and gate level tractable. In this paper we have described and demonstrated a translator for moving circuit structure

descriptions from the realm of the CAD design tool to formal verification. This is an important step facilitating the development of correct designs as VLSI circuits become increasingly complex.

7 Acknowledgements

This research was supported in part by NASA under Space Engineering Research Grant NAGW-1406 and by the NSF under Research Initiation Grant MIP-9109618.

References

[1] AMI: A Subsidiary of Gould Inc. *BOLT Users Manual.*

[2] R. Boulton, M. Gordon, J. Herbert, and J. van Tassel. "The HOL Verification of ELLA Designs". In *1991 International Workshop on Formal Verification in VLSI Design*, Miami, January 1991.

[3] K. B. Cameron and J. C. Shovic. "Calculating Minimum Logic State Requirements for Multi-Strength Multi-Value MOS Logic Simulators". In *1987 IEEE International Conference on Computer Design: VLSI in Computers & Processors*, pages 672–675, Rye Brook, New York, October 1987. IEEE Computer Society Press.

[4] A. Camilleri, M. Gordon, and T. Melham. "Hardware Verification Using Higher Order Logic". In D. Borrione, editor, *From HDL Descriptions to Guaranteed Correct Circuit Designs*, pages 43–67. Elsevier Scientific Publishers (North-Holland), 1987. Also Technical Report No. 91, University of Cambridge Computer Laboratory, September, 1986.

[5] I. S. Dhingra. "Formal Validation of An Integrated Circuit Design Style". In G. Birtwistle and P. A. Subrahmanyam, editors, *VLSI Specification, Verification and Synthesis*, pages 293–321. Kluwer Academic Publishers, Boston, 1988. Also Technical Report No. 115, University of Cambridge Computer Laboratory, August, 1987.

[6] J. W. Gambles and P. J. Windley. "A Verification Logic Representation of Indeterministic Signal States". In *Third NASA Symposium on VLSI Design*, pages 10.2.1–10.2.12, Moscow, Idaho, October 1991. NASA Space Engineering Research Center, University of Idaho.

[7] M. J. C. Gordon. "Why Higher Order Logic is a Good Formalism for Specifying and Verifying Hardware". In G. J. Milne and P. A. Subrahmanyam, editors, *Formal Aspects of VLSI Design*, pages 153–177. Elsevier Scientific Publishers (North-Holland), 1986. Also Technical Report No. 77, University of Cambridge Computer Laboratory, 1985.

[8] M. J. C. Gordon. "HOL: A Proof Generating System for Higher-Order Logic". In G. Birtwistle and P. A. Subrahmanyam, editors, *VLSI Specification, Verification and*

Synthesis, pages 73–128. Kluwer Academic Publishers, Boston, 1988. Also Technical Report No. 103, University of Cambridge Computer Laboratory, August, 1987.

[9] J. P. Hayes. "A Unified Switching Theory with Applications to VLSI Design". *Proceedings of the IEEE*, Vol. 70(No. 10):1140–1151, October 1982.

[10] T. F. Melham. "Abstraction Mechanisms for Hardware Verification". In G. Birtwistle and P. A. Subrahmanyam, editors, *VLSI Specification, Verification and Synthesis*, pages 267–291. Kluwer Academic Publishers, Boston, 1988. Also Technical Report No. 106, University of Cambridge Computer Laboratory, May, 1987.

[11] T. F. Melham. "Using Recursive Types to Reason About Hardware Verification". In G. Milne, editor, *Design for Behavioural Verification*, Glosgow, July 1988. IFIP WG 10.2. Also Technical Report No. 135, University of Cambridge Computer Laboratory, May, 1988.

[12] B. Meyer. *Introduction To The Theory Of Programming Languages.* Prentice Hall International, 1990.

[13] J. P. van Tassel. *The HOL Parser Library.* University of Cambridge Computer Laboratory, July 1991.

[14] J. P. van Tassel and D. Hemmendinger. "Toward Formal Verification of VHDL Specifications". In L. Claesen, editor, *Applied Formal Methods For Correct VLSI Design*, pages 261–270, Houthalen, Belgium, November 1989. IMEC-IFIP WG 10.2/WG 10.5, Elsevier Scientific Publishers (North-Holland).

[15] G. Winskel. "A Compositional Model of MOS Circuits". In G. Birtwistle and P. A. Subrahmanyam, editors, *VLSI Specification, Verification and Synthesis*, pages 323–347. Kluwer Academic Publishers, Boston, 1987. Also Technical Report No. 105, University of Cambridge Computer Laboratory, 1987.

A Simple Modern Correctness Condition for a Space-Based High-Performance Multiprocessor [1]

David K. Probst and Hon F. Li
Department of Computer Science
Concordia University
1455 de Maisonneuve Blvd. West
Montreal, Quebec H3G 1M8

Abstract - **A number of U.S. national programs, including space-based detection of ballistic missile launches, envisage putting significant computing power into space. Given sufficient progress in low-power VLSI, multichip-module packaging and liquid-cooling technologies, we will see design of high-performance multiprocessors for individual satellites. In very high speed implementations, performance depends critically on tolerating large latencies in interprocessor communication; without latency tolerance, performance is limited by the vastly differing time scales in processor and data-memory modules, including interconnect times. The modern approach to tolerating remote-communication cost in scalable, shared-memory multiprocessors is to (i) use a multithreaded architecture, and (ii) alter the semantics of shared memory slightly, at the price of forcing the programmer either to reason about program correctness in a relaxed consistency model or to agree to program in a constrained style. The literature on multiprocessor correctness conditions has become increasingly complex—and sometimes confusing—which may hinder its practical application. We propose a simple modern correctness condition for a high-performance, shared-memory multiprocessor; the correctness condition is based on a simple interface between the multiprocessor architecture and high-performance, shared-memory multiprocessor; the correctness condition is based on a simple interface between the multiprocessor architecture and the parallel programming system.**

Keywords: high-performance multiprocessor in space, scalable shared-memory multiprocessor, multithreading, relaxed consistency model, multiprocessor correctness condition, parallel programming model, programming/architecture interface, local acknowledgment protocol, nonsequential consistency, hybrid model.

1 Introduction

A number of U.S. national programs, including space-based detection of ballistic missile launches, envisage putting significant computing power into space. When enabling technologies such as (i) low-power VLSI, including cold chips, (ii) multichip-module packaging, and (iii) space-based liquid cooling for hot chips—are sufficiently mature, we will see design of

[1]This research was supported in part by NSERC operating grants A3363, A0921, strategic grant MEF0040121 and the NCE Micronet network. E-mail: probst@crim.ca and probst@vlsi.concordia.ca.

space-based liquid cooling for hot chips—are sufficiently mature, we will see design of high-performance multiprocessors for individual satellites. These machines will communicate with each other and with ground-based supercomputers—for example, to support massive real-time data acquisition, up to two terabytes of data per day. Very high speed implementations of multiprocessor architectures—that is, scalable machines with short processor cycles—that are built from clusters of interconnected processor and data-memory modules have a potential performance bottleneck; they *must* tolerate the large latencies in interprocessor communication (remote memory accesses may take anywhere from 100 to 1000 processor cycles) [2,11]. Building a shared-memory parallel programming system on top of a so-called "distributed-memory" multiprocessor architecture may motivate us—after as much latency as possible has been tolerated by multithreading—to alter the semantics of shared memory slightly, at the price of forcing the parallel programmer either to reason about program correctness in a relaxed consistency model or to agree to program in a constrained style. The literature on multiprocessor correctness conditions has become increasingly complex—and sometimes confusing—which may hinder its practical application [1,3,4,9–11]. One positive note is that most relaxed consistency models (actually, these are not models of multiprocessors at all but rather restrictions on an implementation or protocol, expressed at the level of the processor/memory interface) support sequential consistency for a special class of well-behaved, i.e., well-synchronized, parallel programs. We propose a simple modern correctness condition for a high-performance, shared-memory multiprocessor; the correctness condition is based on a simple interface between the multiprocessor architecture and the parallel programming system.

2 Memory Consistency Models

The conceptual model inherited from von Neumann machines with relatively few processors is that *all* memory reads and writes appear to be atomic (indivisible). In early multiprocessors, this "default" atomicity was implemented by the indivisible read/write memory cycle. Closely related to atomicity in the traditional model is the view that each operation in a (sequential) program thread appears to execute before the next operation in the thread is issued. The high cost of waiting for each stream operation to globally perform before the next stream instruction issues is well known [7,8]. The first attempt to define a multiprocessor correctness condition that would permit assertional reasoning (essentially, the continued existence of a program state space) and still be efficiently implementable was Lamport's sequential consistency [12]. Assertional reasoning would still be possible, Lamport wrote, provided that "the result of any execution [by the multiprocessor] is the same as if the operations of all the processors [had been] executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program". In simpler language, the program behaves as if the memory accesses of all the threads were interleaved—without destroying program order—and executed sequentially.

There are at least two problems with sequential consistency as a correctness condition. First, it has been widely misinterpreted to mean that one processor's update to a shared variable must be reflected in every other processor's *view* before the updating processor may issue another memory access, thereby confusing the correctness condition—sequential

consistency—with one particular low-performance implementation [7,9]. In this interpretation, all memory accesses to all shared variables are serializable even in the absence of any explicit synchronization and/or dependence information in the parallel program. Second, (sequential) program order is an extremely opaque tool for expressing the semantics of a parallel program decomposed into threads (instruction streams) [5,11]. This is what makes DASH release consistency so conservative from a parallel-programming point of view—questions such as, does this P protect this read? are occluded, and the program interpreter must respect the (sequential) programmer synchronization as written. In the DASH protocol, this is governed by a set of rules for the permissible orderings among acquires, releases and ordinary memory accesses [9]. "Thread partial order" consistency is more forward looking—as far as possible, all such questions about the *protects* relation on program operations are answered clearly by the parallel programmer and expressed as a partial order on synchronizing and ordinary memory accesses.

When a shared-memory parallel program is presented to the "program interpreter" of a so-called "distributed-memory" multiprocessor, the machine is not—in current practice, anyway—given an abstract program specification that defines the *dependence order* on the set of program operations. In concrete terms, the dependence order includes such necessary temporal precedences among program operations as uniprocessor control and data dependences (and antidependences), interprocessor data flow- and anti-dependences, and interprocessor control dependences (including necessary temporal precedences arising from an operation that depends on the restoration of an invariant). By definition, the dependence order is the limit for optimization—assuming sequential threads as the starting point—by the program interpreter; it defines the necessary temporal precedences among program operations that must appear to be observed in any correct execution of the parallel program. This dependence order is defined over the set of *all* program operations, including synchronizing and ordinary memory accesses, control-transfer instructions and register-only instructions.

In general, it is unrealistic to expect the programmer to articulate the program dependence order to the program interpreter; this may be more realistic at lower levels where a correctness proof of a critical subalgorithm can help the programmer to discover the dependence order [13]. In practice, what the program interpreter has to work with is (i) all compiler-discoverable local order, and (ii) all programmer-supplied local order, including explicit synchronization and other dependence information. To aid the parallel programming system, the high-level parallel programmer ought to do the following: (i) identify each P-type synchronizing operation (i.e., one that potentially blocks this thread), (ii) identify each V-type synchronizing operation (i.e., one that potentially unblocks some other thread), (iii) identify each PV-type synchronizing operation (i.e., one that potentially either blocks this thread or unblocks some other thread), and (iv) indicate, for each thread, the thread partial order that specifies the necessary temporal precedences among synchronizing and ordinary program operations in that thread. By so doing, the parallel programmer defines the *protects* relation on program operations. To focus on semantics, we defer the question of how thread partial orders and scopes of synchronization primitives are expressed in parallel programming languages [2,5,11]. Instead, we adopt the "fiction" that threads are presented to the program interpreter directly as partial orders. The interpreter respects the programmer synchronization as written, with "thread partial order" replacing "program order". The

set of rules for respecting programmer synchronization in our protocol is given in Section 4. Reference [14] is an elegant introduction to partial orders in parallel systems.

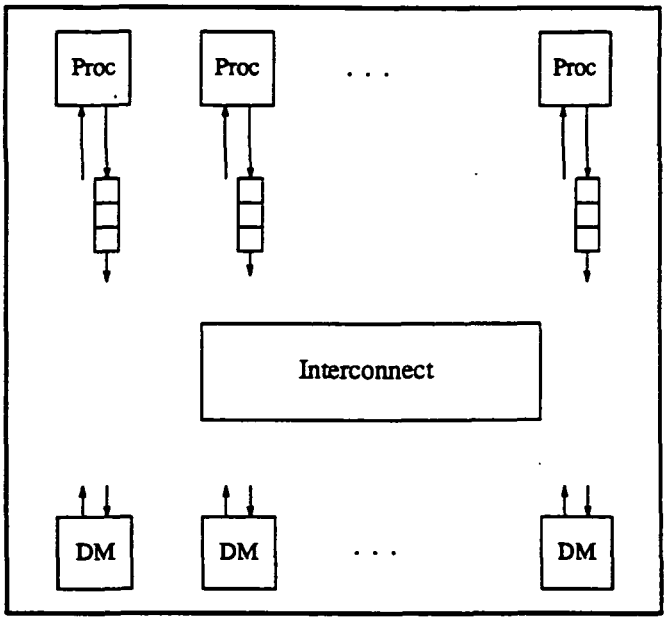


Figure 1: Abstract block diagram showing interconnection network, processor modules, data memory modules and operation buffers. No "dancehall configuration" is implied.

3 Thread Partial Order

Figure 1 shows an abstract block diagram for a multithreaded, scalable, shared-memory multiprocessor. Conceptually, the distinction between a distributed-memory and a shared-memory hardware architecture is small. In either case, any processor can access any address in the single, global address space. NUMA machines have a distinction between "cheap" local and "expensive" remote memory access, while UMA machines provide a constant-latency path between processors and memory. Multithreading means that a processor may issue a (possibly blocking—it depends entirely on the thread partial order) memory access by one thread via the operation buffer, and then—if necessary, i.e, if one has run out of concurrently-enabled operations in this stream—perform a context switch to some other thread while waiting for the first memory access to complete. Figure 1 applies equally well to architectures with various degrees of data caching, whether implemented in processor or data modules. Instruction caching is handled separately. The queues are operation buffers for memory accesses; these buffers contain multiple instructions that have been issued in parallel by processors and threads. Multithreading supports multiple pending operations by an individual processor. Relaxed consistency supports multiple pending operations by an

individual thread. Further optimization is possible when there are local acknowledgment protocols for memory accesses; this requires (data) caching of the processor's view [5].

The sharpest formulation of relaxed consistency is to state that memory accesses see "consistent" values *only* when such accesses are protected by synchronizing operations. Full consistency states that all memory accesses see consistent values (for example, reads return the most recent write) even when all synchronization is implicit. Most relaxed models include the programming constraint that every pair of conflicting memory accesses is protected by a synchronization chain. Two accesses *conflict* if they access the same location and at least one of them is a write. Abstractly, P-type operations define input synchronization points at which (i) partial or total program state is defined, and (ii) some invariants have been restored—because certain operations in other threads have completed. Similarly, V-type operations define output synchronization points at which (i) partial or total program state is defined, and (ii) some invariants have been restored—because certain operations in this thread have completed. PV-type operations (e.g., barriers) define both input and output synchronization points, and combine the semantics of P- and V-type operations [6].

When caches are present, processors (on which threads are scheduled) can have views. When local acknowledgment protocols are permitted due to (data) caching of a processor's view, "soft" V-type operations define *virtual* output synchronization points at which certain operations in this thread have sufficiently completed so that synchronization chains leaving this thread (possibly across processors) keep their usual semantics provided that we implement an additional protocol. Specifically, when semaphores are bound to the shared variables they protect, semaphore operations can be integrated with the remembering of updated values. The use of timestamps allows only the relevant portion of a processor's view to be copied to the other processor when there is a synchronization chain between threads on different processors [5]. Conceptually, this amounts to absorbing interprocessor communication into interprocessor synchronization.

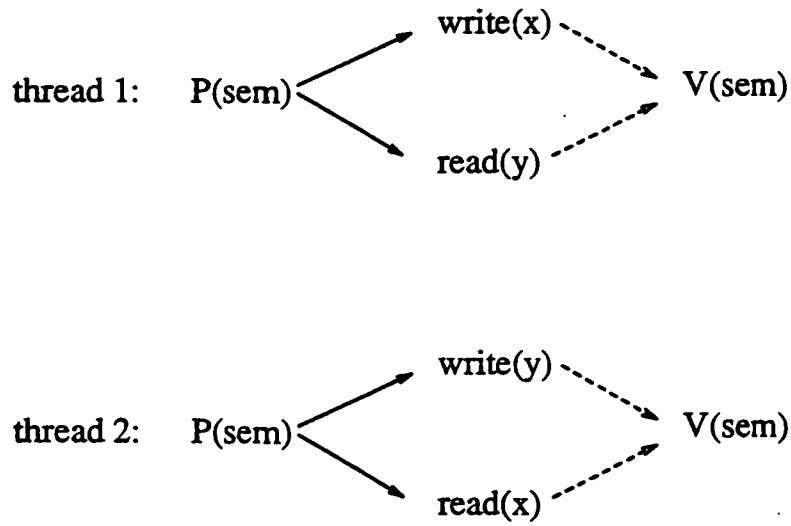


Figure 2: Two thread partial orders with local acknowledgment protocols.

Figure 2 shows the simple case of two critical sections in two threads where the operations in each critical section are concurrent. Here, the operations in each P/V-bracketed pair commute. When two concurrent operations in a thread partial order do not commute (i.e., when both orderings are permitted but have different results), a partial-order representation with branching structure may be required [15–17]. Incidentally, programmer-supplied local order is not restricted to defining the *protects* relation; it may also include specifying necessary temporal precedences between two program operations neither of which is a synchronizing operation (here, the theory becomes nontrivial). Without loss of generality, suppose that thread 1 has entered its critical section ahead of thread 2. At this point, there is a flow dependence from write(x) to read (x), and an antidependence from read(y) to write(y).

When there is no data cache, thread 1 must learn that both its memory accesses have completed with respect to thread 2 before its V operation may be issued. In this case, global acknowledgment protocols notify thread 1 of completion, while synchronizing operations guarantee completion to thread 2. When there is a data cache, thread 1 may issue its V operation as soon as local acknowledgment protocols for both its memory accesses have completed. Why? When thread 2 issues its P operation, an additional protocol can ensure that the P operation does not complete until (i) the new value of x has been communicated from thread 1 to thread 2, and (ii) the read of y has completed in thread 1 (see the next paragraph). This form of "delayed consistency" allows thread 1 to continue past its V operation before the output synchronization point has been actually reached [5].

The dashed arrow from read(y) to V(sem) in thread 1 may appear puzzling at first sight. Dashed arrows only make sense if the processors on which threads 1 and 2 are scheduled can remember their own views. In this case, why should there be an antidependence from read(y) to write(y)? If both threads are currently scheduled on the same processor, then they share the same view and the antidependence exists. If both threads are currently scheduled on different processors, then their views are disjoint and no antidependence exists. Since only the runtime system can determine which threads are currently on which processors, it is simpler for the compiler to generate code *as if* both threads currently shared the same view.

The graphics in Figure 2 can now be explained. In a thread partial order, a solid arrow $a \longrightarrow b$ means: b may not be issued until a has completed, while a dashed arrow $a \dashrightarrow b$ means: b may not be issued until a has *locally* completed (that is, has had its serialization order irrevocably determined). Abstractly, Figure 2 illustrates a 2-phase token release protocol. This is an implementation optimization; using only solid arrows in thread partial orders would not alter the correctness condition that was being implemented.

Figure 3 illustrates why the widespread emphasis on critical sections in discussions of relaxed consistency (due partly to programmer familiarity and partly to unexamined tradition, we suppose) is misleading. Here, there is a static dependence between a group of operations in one thread (protected by a V-type operation) and a group of operations in another thread (protected by a P-type operation). It seems strange to enforce a simple flow dependence between two threads by unlocking in one thread and locking in the other (although—to borrow a joke from Burton J. Smith—we could call this "generalized critical section" synchronization). The constraint issue is whether we still require that any thread's access to a shared variable be dynamically enclosed between a P-type and a V-type operation—as judged by

the thread partial order—even in cases where we are enforcing a static dependence. This constraint is satisfied in textbook solutions to the bounded-buffer problem, where both static and dynamic dependences are enforced. The answer in general is, it depends. If there is a dependence of these operations in thread 1 on "earlier" operations, then they *should* be bracketed, otherwise not.

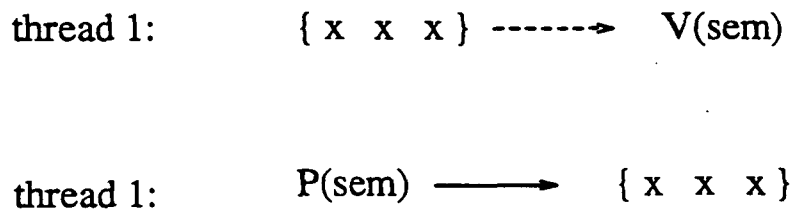


Figure 3: Synchronization primitive scope without critical sections.

4 Nonsequential Consistency

Sequential consistency and processor consistency are correctness conditions while weak consistency, release consistency, entry consistency, etc., are implementation restrictions [5,7,9]. Sequential consistency asserts that a multiprocessor execution is incorrect if it cannot be simulated by a serial (uniprocessor) execution that respects the "program order" of each thread. Serial program order is thus the test of whether the noncommutativity of two conflicting ordinary operations *matters*. The obvious modification is to replace "program order" by "thread partial order". The new condition reads as follows. First, all operations within a thread appear to execute in thread partial order, as viewed by the processor executing that thread. Second, the result of program execution is the same as if all synchronizing and ordinary memory accesses had been executed in some sequential order, and the projection of this sequence onto each thread is a linearization of that thread's partial order. Serial execution means possible (i.e., legal) execution on a uniprocessor. A multiprocessor satisfying this condition will be called *nonsequentially consistent*. But this is not quite right.

The role played by a programming constraint should be stated explicitly as a premise of the correctness condition; this is preferable to starting with an (arbitrary) implementation restriction and arguing that it implements a traditional correctness condition like sequential consistency given the programming constraint. In this spirit, coarsen the granularity to consider just the synchronizing and ordinary shared-variable accesses in a parallel program. We say that, if an ordinary shared-variable access is dynamically enclosed between a P-type and a V-type operation, then it is a *protected* access; otherwise, it is an *unprotected* access. A general correctness condition should include both types of access, and should allow for concurrent noncommutative operations within a single thread. If a parallel program is run on a nonsequentially-consistent multiprocessor, then—at this level of granularity—every correct execution on this machine has the following property:

(*i*) For each thread, there exists a serial execution of *all* the program operations that simulates the actual execution, and whose projection on that thread does not violate thread partial order.

(*ii*) Moreover, there exists a serial execution of *only* the synchronizing and protected accesses that simulates the matching portions of the actual execution, and—for each thread—the projection of this sequence on that thread does not violate thread partial order.

The easiest way to visualize the second condition is to imagine that the parallel program has been divided into properly- and improperly-synchronized *epochs*; when the final state of one epoch is intended as the initial state of another (this is the usual case), the two epochs will be separated by a barrier. During improperly-synchronized epochs, only processor consistency will hold. During properly-synchronized epochs, sequential consistency will hold. Serial simulation of a properly-synchronized epoch must take the initial and final states—which hold at the initial and final barriers, respectively—into consideration. In this way, a single, piecemeal serial execution can simulate all properly-synchronized portions of the actual execution.

We propose the following implementation of nonsequential consistency; temporarily nameless to avoid further confusion, it is one of many members of the "release consistency" family of implementation restrictions. The primary influence—apart from viewing threads as partial orders—is Tera release consistency [6,18] with an optional dash of entry consistency [5] for those people who believe that caches are the best way to implement shared memory (they are certainly not the *only* way).

(*i*) No processor may issue a shared-variable access until all P-type operations protecting it—that is, all P-type operations that precede the memory access in thread partial order—have completed.

(*ii*) No processor may issue a V-type operation until all shared-variable accesses protected by it—that is, all memory accesses that precede the V-type operation in thread partial order—have (at least locally) completed.

(*iii*) When a thread contains a PV-type operation (e.g., a barrier), all ancestor shared-variable accesses must have (at least locally) completed before the PV-type operation may issue, and the PV-type operation must have completed before any descendant shared-variable access may issue, where memory-access "ancestor" and "descendant" of a PV-type operation are defined by thread partial order.

That is, the program interpreter at each processor must run completion protocols for both synchronizing and ordinary memory accesses. When there are data caches available to processors, the optimization of "delayed consistency" is possible.

5 Conclusion

Discussions of correctness conditions and memory consistency models in shared-memory multiprocessors have grown needlessly complex in the last five years as implementation detail and excess formalism have obscured the simple language (i.e., semantics) issues. By cleanly separating the correctness condition (i.e., multiprocessor model) from the implementation

restriction (i.e., protocol model), we have brought out the simplicity of the "release consistency" implementation family. The key to simplicity is the programmer's definition of the *protects* relation on synchronizing and ordinary memory accesses. In the absence of full program dependence order, this is perhaps the most useful information that can be supplied by the programmer to the program interpreter; a good optimizing compiler can then adjust for uniprocessor dependences without altering the protects relation. Again, there will be elements of programmer-supplied local order in addition to the *protects* relation. We have proposed a new correctness condition which encompasses a wide range of parallel programming models, including ones with a mixture of shared-memory and message-passing semantics. The requirements specification of a *special-purpose* high-performance multiprocessor architecture for space applications would not include applicability to a wide spectrum of problems, but that does not make the ideas of this paper any less relevant (for example, they could be used to evolve the AT&T DSP3 Parallel Processor). We have indicated how to make an intelligent division of labor between programmer and optimizing compiler in discovering the thread partial order. One distinguishing feature of space applications is a concern for reliability; we are currently investigating a distributed-memory architecture for a *fault-tolerant* nonsequentially-consistent shared memory. Space systems are distributed systems, and—in this context—fault tolerance is as important as performance.

References

[1] S. Adve and M. Hill, *Weak ordering – a new definition*, in Proc. 17th International Symposium on Computer Architecture, May 1990, pp. 2–14.

[2] R. Alverson et al., *The Tera Computer System*, in Proc. 1990 International Conference on Supercomputing, June 1990, pp. 1–6.

[3] H. Attiya and J. Welch, *Sequential Consistency versus Linearizability*, in Proc. 3rd ACM Symposium on Parallel Algorithms and Architectures, July 1991, pp. 304–315.

[4] H. Attiya and R. Friedman, *A Correctness Condition for High-Performance Multiprocessors*, in Proc. 24th ACM Symposium on the Theory of Computing, May 1992, pp. 679–690.

[5] B. Bershad and M. Zekauskas, *Midway: Shared-Memory Parallel Programming with Entry Consistency for Distributed-Memory Multiprocessors*, Department of Computer Science, Carnegie Mellon University, Report CMU-CS-91-170, April 1991.

[6] D. Callahan and B. Smith, *A Future-Based Parallel Language for a General-Purpose Highly-Parallel Computer*, in D. Gelernter et al. (Eds.), Languages and Compilers for Parallel Computing, MIT Press, 1990, pp. 95–113.

[7] M. Dubois et al., *Memory Access Buffering in Multiprocessors*, in Proc. 13th International Symposium on Computer Architecture, June 1986, pp. 434–442.

[8] M. Dubois and C. Scheurich, *Memory Access Dependencies in Shared-Memory Multiprocessors*, IEEE Transactions on Software Engineering, **16:6**, June 1990, pp. 660–673.

[9] K. Gharachorloo et al., *Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors*, in Proc. 17th International Symposium on Computer Architecture, May 1990, pp. 15–26.

[10] P. Gibbons et al., *Proving Sequential Consistency of High-Performance Shared Memories*, in Proc. 3rd ACM Symposium on Parallel Algorithms and Architectures, July 1991, pp. 292–303.

[11] P. Gibbons and M. Merritt, *Specifying Nonblocking Shared Memories*, in Proc. 4th ACM Symposium on Parallel Algorithms and Architectures, June 1992.

[12] L. Lamport, *How to Make a Multiprocessor Computer that Correctly Executes Multiprocess Programs*, IEEE Transactions on Computers, **28:9**, September 1979, pp. 690–691.

[13] L. Lamport, *How to Make a Correct Multiprocess Program Execute Correctly on a Multiprocessor*, preliminary draft, May 1992.

[14] V. Pratt, *Modeling concurrency with partial orders*, Int. Journal of Parallel Prog., **15:1**, February 1986, pp. 33–71.

[15] D. Probst and H. Li, *Using partial-order semantics to avoid the state explosion problem in asynchronous systems*, in E. Clarke and R. Kurshan, (Eds.), Workshop on Computer-Aided Verification '90, June 1990, DIMACS Series, Vol. 3, 1991, pp. 15–24. Also Lecture Notes in Computer Science 531, Springer–Verlag, 1991, pp. 146–155.

[16] D. Probst and H. Li, *Partial-order model checking: A guide for the perplexed*, in K. Larsen and A. Skou, (Eds.), Workshop on Computer-Aided Verification '91, Proceedings, Department of Mathematics and Computer Science, Aalborg University, Report IR-91-5, July 1991, pp. 405–416. Also Lecture Notes in Computer Science 575, Springer–Verlag, 1992, pp. 322–331.

[17] D. Probst and L. Jensen, *Controlling state explosion during automatic verification of delay-insensitive and delay-constrained VLSI systems using the POM verifier*, in S. Whitaker, (Ed.), Proceedings of the 3rd NASA Symposium on VLSI Design, Moscow, ID, October 1991, pp. 8.2.1–8.2.8.

[18] B. Smith, personal correspondence.

Instruction Set Commutivity[1]

P. Windley
Laboratory for Applied Logic
Department of Computer Science
University of Idaho, Moscow, Idaho 83843

Abstract- **We present a state property called *congruence* and show how it can be used to demonstrate commutivity of instructions in a modern load–store architecture. Our analysis is particularly important in pipelined microprocessors where instructions are frequently reordered to avoid costly delays in execution caused by hazards. Our work has significant implications to safety and security critical applications since reordering can easily change the meaning and an instruction sequence and current techniques are largely *ad hoc*. Our work is done in a mechanical theorem prover and results in a set of trustworthy rules for instruction reordering. The mechanization makes it practical to analyze the entire instruction set.**

1 Introduction.

Instruction pipelining [2] is critical to good performance in modern microprocessors. Almost every microprocessor developed in the last several years contains an instruction pipeline. Significant attention has been given to the development of scheduling algorithms to reduce the the occurrence of pipeline hazards because of the performance degradation that they cause. Typically, scheduling involves reordering the instruction stream produced by a compiler.

The approaches to code reordering are largely *ad hoc* with little or no analysis showing whether the rules are correct and under what conditions they should be avoided. Indeed, when asked about the rules that he gave for avoiding semantic changes one researcher, who developed the code scheduling algorithms for the C compiler in a widely available commercial UNIX system, stated "Its all *ad hoc*. It never occurred to me that there might be any other way to do it."

Clearly, the current approach to code scheduling is unacceptable in safety and security critical applications. On one hand, modern pipelined architectures perform poorly without the aid of a compiler that is smart enough to reschedule code to avoid pipeline hazards. On the other hand, the reordering that the scheduler performs has the potential to initiate semantic changes in the code stream. We must either give up performance or live with untrustworthy code. Neither of these approaches is satisfactory.

This paper describes the analysis of a microprocessor instruction set for commutivity. We are interested in establishing, by analysis, under what circumstances the instructions can be reordered while avoiding semantic changes. The next section describes related work and the following sections present our analysis.

---

[1]This work was sponsored by the Department of Defense under University Research Program contract No. MDA904–91–C–7054

[2]See [HP90] for an excellent introduction to pipelining and pipeline hazards.

2 Related Work

The formal analysis of code reordering is related to at least three active areas of research: microprocessor verification, compiler verification, and the automatic generation of optimizers. This section discusses these three areas of research and relates them to the work presented in this paper.

Microprocessor Verification There have been numerous efforts to verify microprocessors. These efforts have been mostly for research purposes and none have included any kind of analysis of their instruction sets regarding code reordering. Only one formally verified general purpose microprocessor has been fabricated and it has so few features as to be impractical for real use. Descriptions of these efforts can be found in [Coh88, Joy89a, Joy88, Hun89]. It is important to note that none of these projects involved verification of a pipelined processor.

In [SB90], Srivas *et al.* describe the formal verification of a pipelined microprocessor called Mini Cayuga, comparable in complexity of design to that of Hunt's FM8501. However, the structure and behavior of the pipeline were hidden from the abstract specification. Only prefetching of the next instruction was incorporated into the specification. This precluded the possibility of formally reasoning about pipeline hazards and instruction scheduling.

We are designing, specifying, and verifying a microprocessor called *AVM–2*. *AVM–2* has a load store architecture and will be pipelined. The architecture of *AVM–2* is largely the same as that of *AVM–1*, but the design is substantially different. We described early results of this research in [Win91] where we demonstrated the integrity of the supervisory mode of *AVM–1*. In this paper, we describe results regarding instruction set commutivity for *AVM–2*.

Compiler Verification There has been much work on verified compilers. Space considerations do not present a full treatment here. Joyce [Joy89b] gives an excellent review. Most of the early work [Rus77, Coh80, CM86] on the compiler correctness problem used idealizations of the hardware. Recent work by Joyce [Joy89b], Moore [Moo88], and Young [You89] have looked at compiler verification under the constraints of a real instruction set. None of these efforts has addressed code reordering although Young states that initial work on an optimizer has begun. Even so, our approach is different from Young's due to the nature of the specification. The specifications in Young's work are operational while ours are denotational.

Optimizer Generation Current approaches to instruction scheduling is largely *ad hoc*. Current compiler technology utilizes rule–based, heuristic algorithms for optimizing code sequences. Representative of the state of the art, the IBM RISC System/6000 XL compiler family uses special flags associated with opcodes to indicate instructions which are "dangerous" to move [War90, War92]. To date, there has not been any published results describing the application of formal methods to pipeline scheduling.

There has been some work on the automatic generation of optimizers from specifications of one sort or another.

In [Kes84], Kessler describes a tool called Peep. Peep is an architectural description driven peephole optimizer. The description of the architecture, given in LISP, is used to generate a table of optimizable instructions that can be used in a an optimizing compiler.

In [DF84], Davidson and Fraser present a system that generates peephole optimizations called PO. PO uses productions which describe the effect of assembly language instructions in a simulator to determine substitutions for 2 and 3 assembly instruction sequences.

While this work is interesting and related to the work presented here, these efforts differ in several important ways:

- The descriptions used for generating optimizations are not related to the implementation in anyway. Our work uses a specification that is related through proof to the implementational specification.
- It is not clear whether or not any kind of theory underlies the generation of the optimizers. As we will show later, there are concepts regarding reordering that can be generalized and used as a basis for reasoning about reordering.
- it is not clear how much faith can be placed on the simulation used to determine equivalent sequences. Our work will be done in a widely accepted theorem proving environment.

3 *AVM–2*

We have designed a computer designated *AVM–2* (*A Verified Microprocessor*). *AVM–2* is a second generation design that will be implemented in CMOS. The design, specification, and verification of *AVM–1*, the predecessor to *AVM–2*, are given in [Win90] where it is used as an example to demonstrate the utility of generic models in hardware verification. *AVM–2*, like *AVM–1*, will feature a RISC–like instruction set and a large register file. Unlike *AVM–1*, *AVM–2* will have a pipelined implementation.

The Registers. *AVM–2* has a load–store architecture based on a large register file. The register file is divided into seven supervisor–mode registers and twenty–four general purpose registers.

Two additional registers are visible at the architectural level: the program counter and the program status word. The program counter (denoted `pc`) is used to sequence the computer—it indicates which instruction in memory to execute next. The program status word (denoted `psw`) is used to keep track of the status of the last ALU operation, whether or not interrupts are enabled, and the privilege level of the CPU.

The Instruction Set. *AVM–2* has 30 programming level instructions. There is a group of eight, 3–argument (source A, source B, and destination) arithmetic and logical instructions and another group of 8 arithmetic and logical instructions that use two arguments and a 16–bit immediate value. There are 4 instructions for loading and storing registers. Only the load and store instructions communicate with memory. In addition, there are instructions for performing user interrupts, jumps, subroutine calls, and shifts.

4 Instruction Set Specification

The instruction set for *AVM-2* has been formally specified as part of the design process. The specification represents a denotational description of the machine language. In this section, we present several of the state transition functions representing instructions. These examples will be used in later sections describing the analysis.

The instructions are modeled by state transition functions. In general, each function operates on a state tuple and an environment tuple. The state tuple, contains variables representing the register file, `reg`, the program status word, `psw`, the program counter, `pc`, and the memory, `mem`. The environment tuple contains variables representing the interrupt vector, `ivec`, the interrupt line, `int`, and the reset line, `reset`. Each function returns a state tuple updated to reflect the behavior of the instruction being modeled.

The `NOOP` instruction updates the state tuple by incrementing the program counter. No other actions are performed.

```
⊢def NOOP (reg, psw, pc, mem)
          (ivec, int, reset) =
      let new_pc = inc pc in
      (reg, psw, new_pc, mem)
```

Note that `NOOP` is *not* an identity function, although it is often thought of that way. The fact that `NOOP` does affect the state and resides in memory affects its commutivity.

Other instructions are quite a bit more complicated than the `NOOP` instruction. For example, the `ADD` instruction is shown below:

```
⊢def ADD (reg, psw, pc, mem)
         (ivec, int, reset) =
    let a = EL (GetSrcA pc mem) reg and
        b = EL (GetSrcB pc mem) reg and
        d = GetDest pc mem in
    let result = add (a, b) in
    let cflag  = addp (a, b, result) and
        vflag  = aovfl (a, b, result) and
        nflag  = negp result and
        zflag  = zerop result and
        sm     = get_sm psw and
        ie     = get_ie psw in
    let new_reg = UPDATE_REG psw d reg result and
        new_psw = mk_psw(sm, ie, vflag,
                         nflag, cflag, zflag) and
        new_pc = inc pc in
    (new_reg, new_psw, new_pc, mem)
```

The `ADD` instruction updates every member of the state tuple except the memory. The primary action, summing two registers and updating the register file accordingly is reflected by the updated register file, `reg`. The instruction also calculated new values for the overflow, carry, negative, and zero flags of the program status word, `psw`. The supervisory mode bit

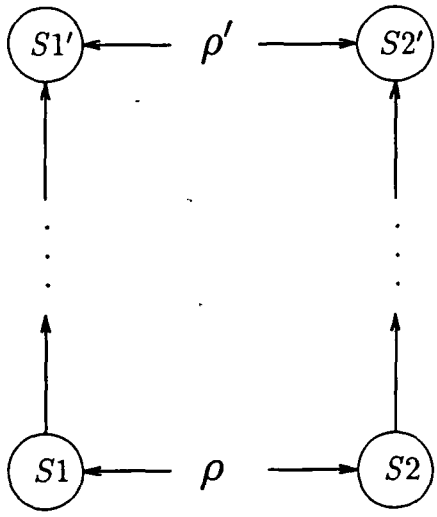


Figure 1: State Congruence

and the interrupt enable bit remain unchanged, as expected. The program counter, pc is incremented.

5 State Congruence

Our notion of state correspondence is motivated by the denotational description of the instruction set. Because the specification is denotational, we are interested in showing that state transitions made by one sequence of instructions are equivalent to the state transitions made by another sequence of instructions.

Unfortunately, the instructions sequences themselves are part of the state and so we cannot use equivalence as the relation between states. Instead, we relate the states using a property we call congruence. We call this relationship *congruence* because the states are equivalent except for (i.e. *modulo*) the ordering of the instruction sequences in memory.

Figure 1 illustrates congruence. As the figure shows, we start with two states, `S1` and `S2`, which are related by the relation ρ. After they have been transformed by a sequence of instructions, we are left with two modified states `S1'` and `S2'`. Ideally, these new states are still related by ρ, but as we will see in Section 5.2, this is not always the case and so we show them related by ρ'.

The primary congruence relation that we use in the examples is given by the following predicate:

```
⊢def Congruent loc (reg1,psw1,pc1,mem1)
                   (reg2,psw2,pc2,mem2) =
   (reg1=reg2) ∧ (psw1=psw2) ∧ (pc1=pc2) ∧
   (∀ a . ¬(a = address loc) ∧
          ¬(a = address (inc loc)) ⇒
          (fetch (mem1,a) = fetch (mem1,a))) ∧
   (fetch (mem1,address loc) =
             fetch (mem2,address (inc loc))) ∧
   (fetch (mem2,address loc) =
             fetch (mem1,address (inc loc)))
```

The predicate operates over a location and two state tuples. We say that the state tuples are *congruent* if their register files, `reg1` and `reg2`, program status words, `psw1` and `psw2`, and program counters, `pc1` and `pc2`, are the same. Additionally, we require that the memories, `mem1` and `mem2` be the same except that the words located at `loc` and `loc+1` in `mem1` are swapped in `mem2`.

5.1 Swapping NOOP

To start with, we examine the commutivity of the `NOOP` instruction. While this may seem like a trivial problem, the problem is not as straightforward as it seems since `NOOP` does affect the state. Also, the difficulties encountered in `NOOP` commutivity are typical of other commutivity proofs.

The following theorem shows that `NOOP` can be commuted with any instruction that does not modify memory or alter program flow:

```
⊢ let s1  = (reg1,psw1,pc1,mem1) and
      s2  = (reg2,psw2,pc2,mem2) and
      e = (ivec,ireq,reset) in
   ∀ inst.
   (inst = macro_inst (Opcode s1 e)) ⇒
   Congruent pc2 s1 s2 ∧
   NON_MEM_INST (Opcode s1 e) ⇒
   let s1' = (NOOP (inst s1 e) e) and
       s2' = (inst (NOOP s2 e) e) in
   Congruent pc2 s1' s2'
```

We assume that the environment does not change during execution of the two instructions (`e`, representing the environment, is used as an argument for both of them). The theorem states that for every instruction in the instruction set, if the initial states are congruent the modified states are also congruent.

The theorem is not true for instructions that alter flow control because the `NOOP` would never execute in one case and some other instruction would execute in its place. For instructions that modify memory, we can prove a more restricted version that assumes that the memory instruction does not interfere with the program (instructions and data are stored in the same memory). We will see an example using a non–interference condition in the next section.

5.2 Swapping Arithmetic Instructions

The arithmetic instructions of *AVM-2* that do not use the carry flag commute under weak congruence. Weak congruence is the same as strong congruence (defined above), except that it does not require equivalence for the entire program status word. Rather we require only that the supervisory mode bit and the interrupt enable bit be the same. Most instructions that modify the overflow, carry, negative, and zero flags of the program status word do so without regard to their previous values, so the value of the two program status words cannot be equal.

```
⊢def Weak_Congruent loc (reg1,psw1,pc1,mem1)
                        (reg2,psw2,pc2,mem2) =
     (reg1 = reg2) ∧
     (pc1 = pc2) ∧
     ((get_sm psw1) = (get_sm psw2)) ∧
     ((get_ie psw1) = (get_ie psw2)) ∧
     (∀ a . ¬(a = address loc) ∧
            ¬(a = address (inc loc)) ⇒
            (fetch (mem1,a) = fetch (mem2,a))) ∧
     (fetch (mem1,address loc) =
               fetch (mem2,address (inc loc))) ∧
     (fetch (mem2,address loc) =
               fetch (mem1,address (inc loc)))
```

In order to commute two arithmetic instructions, we require that they be non–interfering. That is, the destination registers cannot be the same as the source registers. For example, the following instructions do not commute:

```
a := b + c
d := a + e
```

Since the second instruction uses the value computed in the first, we cannot swap them without changing the resulting state. In addition, we require that the destination registers be different. The following predicate defines the non–interference property in terms of the functions used by the instruction set definition to retrieve the source and destination register indices from memory.

```
⊢def Non_Interfering pc mem  =
    ¬(GetSrcA (inc pc) mem = (GetDest pc mem)) ∧
    ¬(GetSrcB (inc pc) mem = (GetDest pc mem)) ∧
    ¬(GetDest (inc pc) mem = (GetDest pc mem))
```

Using the weak congruence predicate and the non–interference predicate, we can show, for example, that ADD and SUB commute:

```
⊢  let s1  = (reg1,psw1,pc1,mem1) and
       s2  = (reg2,psw2,pc2,mem2) and
       e = (ivec,ireq,reset) in
    Congruent pc2 s1 s2 ⇒
    Non_Interfering pc1 mem1 ∧
    Non_Interfering pc2 mem2 ⇒
    let s1′ = (ADD (SUB s1 e) e) and
        s2′ = (SUB (ADD s2 e) e)) in
    Weak_Congruent pc2 s1′ s2′
```

Note that we use strong congruence in the assumptions, but can only show weak congruence between the resulting states.

We have presented only two theorems regarding instruction commutivity in *AVM–2*. We can prove more general theorems about the commutivity of arithmetic instructions. We can also show arithmetic instructions commute with load and store instructions provided they are non–interfering.

6 Discussion

We have presented only a few small theorems regarding the analysis of the *AVM–2* instruction set. A more thorough analysis is presently underway. Even so, we believe the results to be interesting.

The fact that we can only show weak congruence when commuting arithmetic instructions is a function of the design of the instruction set. Other instruction sets would provide different results. The contribution of formal analysis is that this property is clearly and unambiguously stated in the resulting theorem.

Also, the weak congruence result affects when we can actually commute arithmetic instructions in a program. We cannot, for example, commute two arithmetic instructions that are followed by a conditional jump since the values of the flags are changed. Strong congruence is required to maintain the program meaning in this case.

We should note that our initial efforts in this area have had an affect on the architecture of *AVM–2*. In light of the results presented here regarding weak congruence, we have undertaken a modification of the instruction set semantics so that arithmetic instructions commute under strong congruence. This will allow greater freedom in code reordering to avoid pipeline hazards.

Certainly none of our specific discoveries regarding the *AVM–2* instruction set will surprise veteran compiler writers. The rules that we have demonstrated for code reordering in the *AVM–2* instruction set are well known. What is important, however, is that we are *not* veteran compiler writers. Analysis allowed us to *show* that they were correct rather than relying on years of experience and intuition. This, it seems, is the heart and soul of engineering [Sha90].

7 Future Work

We plan to extend our analysis of commutivity to explore code motion for larger code fragments. Our intent is to automate a complete analysis of instruction commutivity for all instruction pairs and use these results to determine when large instruction sequences are congruent.

As mentioned earlier, the behavioral specification of *AVM–2* will be verified against its implementation. Because of the hierarchical nature of the specification (see [Win90]), the phase–level mode of the pipeline will have a similar structure to the behavioral model of the top–level. We believe that the techniques demonstrated in this paper will allow us to perform an analysis of the pipeline to identify hazards. This work is underway.

8 Conclusion

This paper has presented examples from an analysis of commutivity in a modern instruction set. Commutivity is start at a more general notion of instruction reordering that in important to both compiler optimizations and pipeline scheduling. Analysis of instruction set reordering is important in both safety and security critical applications because of the danger *ad hoc* approaches present to the semantic integrity of the instruction stream.

Our analysis could have been performed without the benefit of a formal specification of the instruction set or a formal statement of the desired properties. However, the formal analysis has several benefits:

- The theorems about commutivity form a set of rules for commuting instructions. These rules have a demonstrated correctness.
- The formal analysis was helpful in finding interferences between instructions that were not immediately obvious.
- The assumptions and results are unambiguously stated and can be used for further reasoning about optimization and scheduling.
- In providing a set of rules about commutivity, the instructions must be analyzed on a case–by–case basis whether the analysis is done by hand or automated. The formal analysis provides a tool for quickly analyzing the instructions.

References

[CM86] Laurian M. Chirica and David F. Martin. Toward compiler implementation correctness proofs. *ACM Transactions On Programming Languages And Systems*, 8:185–214, April 1986.

[Coh80] Avra Cohn. *Machine Assisted Proofs of Recursion Implementation.* PhD thesis, University of Edinburgh, April 1980.

[Coh88] Avra Cohn. A proof of correctness of the VIPER microprocessor: The first level. In G. Birtwhistle and P. Subrahmanyam, editors, *VLSI Specification, Verification, and Synthesis*, pages 27–72. Kluwer Academic Publishers, 1988.

[DF84] Jack W. Davidson and Christopher W. Fraser. Automatic generation of peephole optimizations. In *ACM SIGPLAN 84 Symposium on Compiler Construction.* ACM, June 1984.

[HP90] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann Publishers Inc., 1990.

[Hun89] Warren A. Hunt. Microprocessor design verification. *Journal of Automated Reasoning*, 5:429–460, 1989.

[Joy88] Jeffrey J. Joyce. Formal verification and implementation of a microprocessor. In G. Birtwhistle and P.A Subrahmanyam, editors, *VLSI Specification, Verification, and Synthesis.* Kluwer Academic Press, 1988.

[Joy89a] Jeffrey J. Joyce. *Multi–Level Verification of Microprocessor–Based Systems.* PhD thesis, Cambridge University, December 1989.

[Joy89b] Jeffrey J. Joyce. Totally verified systems: Linking verified software to verified hardware. In Miriam Leeser and Geoffrey Brown, editors, *Proceedings of the Mathematical Sciences Institute's Workshop on Hardware Specification, Verification, and Synthesis*, July 1989.

[Kes84] Robert R. Kessler. Peep—an architectural description driven peephole optimizer. In *ACM SIGPLAN 84 Symposium on Compiler Construction.* ACM, June 1984.

[Moo88] J. Strother Moore. A mechanically verified language implementation. Technical Report 30, University of Texas at Austin, 1988.

[Rus77] Bruce D. Russell. Implementation correctness involving a language with `goto` statements. *SIAM Journal of Computing*, 6(3), September 1977.

[SB90] M. Srivas and M. Bickford. Formal verification of a pipelined microprocessor. *IEEE Software*, 7(5):52–64, September 1990.

[Sha90] Mary Shaw. Prospects for an engineering discipline of software. *Software Engineering*, 7(6):15–24, November 1990.

[War90] Henry S. Warren. Instruction scheduling for the IBM RISC System/6000 processor. *IBM Journal of Research and Development*, 34(1):85–92, January 1990.

[War92] Henry S. Warren. IBM T.J. Watson Research Center, Private communication, February 1992.

[Win90] Phillip J. Windley. *The Formal Verification of Generic Interpreters.* PhD thesis, University of California, Davis, Division of Computer Science, June 1990.

[Win91] Phillip J. Windley. Using correctness results to verify behavioral properties of microprocessors. In *Proceedings of the IEEE Computer Assurance Conference*, June 1991.

[You89] William D. Young. A mechanically verified code generator. *Journal of Automated Reasoning*, 5, 1989.

Session 7
Tools and Methods

Chairman: Jonathan Gibson

A Mean Field Neural Network for Hierarchical Module Placement

M. Kemal Unaltuna and Vijay Pitchumani
Department of Electrical and Computer Engineering
Syracuse University, Syracuse, NY 13244

Abstract - **This paper proposes a mean field neural network for the two-dimensional module placement problem. An efficient coding scheme with only $O(N \log N)$ neurons is employed where N is the number of modules. The neurons are evolved in groups of N in $\log N$ iteration steps such that the circuit is recursively partitioned in alternating vertical and horizontal directions. In our simulations, the network was able to find optimal solutions to all test problems with up to 128 modules.**

1 Introduction

Since Hopfield and Tank published their seminal paper [6], Hopfield-type neural nets have been used for solving many combinatorial optimization problems. One major problem with such solutions is scaleability. With increasing problem size two things happen: first, the network becomes so big that simulation times are excessively long; and second, finding good parameters becomes increasingly hard that either the network converges to invalid solutions, or the quality of the solutions is poor [14], [2].

Two-dimensional module placement is an NP-hard combinatorial optimization problem which is very important in VLSI layout synthesis. In most of the previous attempts to solve this problem with Hopfield-type networks, researchers have concentrated on small-sized problems [8], [15], [7], [1], [4]. Unfortunately, most of the interesting placement problems in VLSI are very large, with more than 20,000 modules in some cases.

To overcome the difficulty with scaleability, we propose a neural network with $O(N \log N)$ neurons (instead of $O(N^2)$ neurons used in most of the previous applications) which solves the placement problem hierarchically, in a similar way to recursive min-cut bipartitioning methods. According to a recent survey [11], min-cut based algorithms are still very popular in solving large problems and the results obtained are second only to simulated annealing.

It has been independently shown [5], [9], [13] that the neuron update equations Hopfield used can be interpreted as one way of solving the mean field equations which arise from the *mean field approximation* to simulated annealing. We call our network a *mean field neural network*, because we use the straightforward update method of solving the mean field equations [9]. This kind of update is much faster than Hopfield's update, and generally produces better solutions.

2 Mean Field Neural Network for Hierarchical Placement

2.1 Problem Formulation and Mapping

We start with an $n \times m$ array of slots and $N = nm$ equal-sized modules. We also assume two-point connectivities between the modules, given by the $N \times N$ connectivity matrix $[c_{ij}]$. Thus, multi-pin nets have to be preprocessed and mapped to two-point connectivities. The objective is to assign the modules to slots such that the estimated wiring length is minimized and the resulting placement is routable.

In [5], Fox and Furmanski formulated the load balancing problem on hypercubes in terms of optimal partitioning of a computational graph into subgraphs, and solved it using a neural network. Here, we modify and expand their ideas to the problem at hand.

Let the two-tuple (r^p, s^p) represent the position of the slot to which module p is assigned. The row number r^p, and the column number s^p are $e = \log n$ and $d = \log m$ bit binary numbers, respectively, and are given by

$$r^p = \sum_{i=0}^{e-1} r_i^p 2^i$$

$$s^p = \sum_{i=0}^{d-1} s_i^p 2^i$$

Bits r_i^p and s_i^p are mapped to neural variables y_i^p and x_i^p such that $y_i^p = 2r_i^p - 1$ and $x_i^p = 2s_i^p - 1$. Thus, y_i^p (x_i^p) represents the i^{th} bit of the row number (column number) of p's slot. All neural variables can have continuous values between ± 1. The neural variables will be evolved in groups of N in $\log N$ steps. First, $x_{d-1}^0, x_{d-1}^1, \ldots, x_{d-1}^{N-1}$ representing the highest bits of the column numbers of the modules are evolved. After this step, the circuit is effectively bipartitioned in the vertical direction. Next, $y_{e-1}^0, y_{e-1}^1, \ldots, y_{e-1}^{N-1}$ representing the highest bits of the row numbers of the modules are evolved. This means bipartitioning the circuit in the horizontal direction. After this, we return to the vertical direction and evolve variables $x_{d-2}^0, x_{d-2}^1, \ldots, x_{d-2}^{N-1}$. This way, the recursive bipartitioning of the circuit continues in alternating vertical and horizontal directions until all neural variables are evolved.

2.2 Bipartitioning Neural Network

For the first vertical bipartitioning step we will use the following energy function:

$$E_{x_{d-1}} = -\frac{A}{2} \sum_p \sum_{q \neq p} c_{pq} x_{d-1}^p x_{d-1}^q + \frac{B}{2} \sum_p \sum_{q \neq p} x_{d-1}^p x_{d-1}^q \qquad (1)$$

The first term has its minimum when the sum of the connections between modules in separate partitions is minimized, and the second term is minimized when the partitions are balanced. The mean field equations can be derived from (1) as

$$x_{d-1}^p = \tanh\left(\frac{u_{d-1}^p}{T}\right) \qquad (2)$$

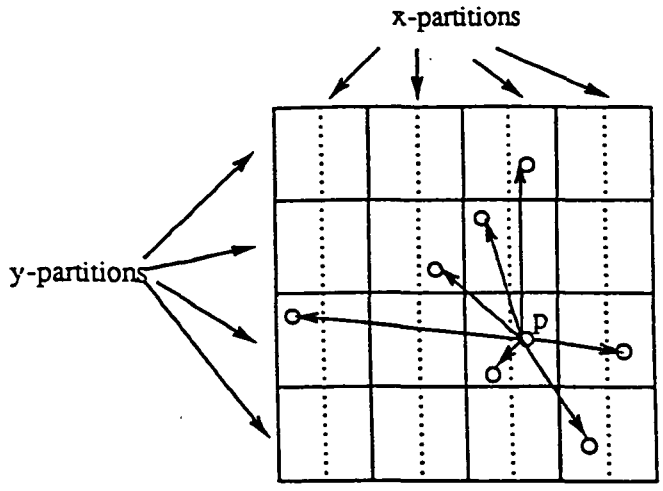


Figure 1: The situation after two vertical and two horizontal partitioning steps. The circuit is currently bipartitioned for the third time in the vertical direction. The arrows indicate attractive forces acting on module p.

$$\begin{aligned} u^p_{d-1} &= -\frac{\partial E_{x_{d-1}}}{\partial x^p_{d-1}} \\ &= A\sum_{q\neq p} c_{pq} x^q_{d-1} - B\sum_{q\neq p} x^q_{d-1} \end{aligned} \tag{3}$$

The first term in equation (3) is an attractive force where each module q connected to p tries to bring p to its side of the partition, and the second term tries to keep the partitions balanced.

2.3 Placement by Recursive Bipartitioning

In order to extend equations (2) and (3) to subsequent horizontal and vertical bipartitioning steps, it is not sufficient to consider only internal connections. We have to consider connections to modules in other partitions (at a higher level) as well.

Figure 1 shows the situation after two vertical and two horizontal bipartitioning steps when the resulting partitions are being partitioned for the third time in the vertical direction. At this stage, the modules can be grouped in the following way according to their locations relative to module p, *prior* to the current partitioning process:

1. Modules in the same x- and y-partition as p,
2. Modules in the same x-partition as p but not in the same y-partition,
3. Modules not in the same x-partition as p.

The balance force in the update equation for x^p_{d-3} should cover modules in the first group only. For the attractive forces, we need to consider two cases:

- Attractive forces from modules in the third group. Such a module q is in an x-partition either to the left or to the right of p and should bias p accordingly. However, the location of q with respect to the current bipartitioning process is irrelevant.
- Attractive forces from modules in the first or second group. Since such a module q is in the same x-partition as p, its location with respect to the current bipartitioning process should influence x^p_{d-3}.

To incorporate these different forces in the update equations, we need to make some definitions. We define X^i_{pq} to be one if modules p and q are in the same x-partition after $d-i-1$ vertical partitioning steps, and zero otherwise. In terms of neural variables this can be expressed as

$$X^i_{pq} = \frac{1}{2^{d-i-1}} \left(\prod_{k=i+1}^{d-1} (1 + x^p_k x^q_k) \right) \tag{4}$$

Similarly, Y^j_{pq} is defined to be one if modules p and q are in the same y-partition after $e-j-1$ horizontal partitioning steps, and zero otherwise:

$$Y^j_{pq} = \frac{1}{2^{e-j-1}} \left(\prod_{l=j+1}^{e-1} (1 + y^p_l y^q_l) \right) \tag{5}$$

Now consider the generalized situation where we have evolved neural variables x^p_k and y^p_l, for all $k > i$, $l > j$, and for all p. This means that the problem has been partitioned $d-i-1$ times in the vertical direction and $e-j-1$ times in the horizontal direction. Suppose we are evolving variables x^p_i, i.e. we are at the $(d-i)^{th}$ vertical partitioning step. With the help of the above definitions for X^i_{pq}, and Y^j_{pq}, we propose the following update equations for x^p_i:

$$x^p_i = \tanh\left(\frac{u^p_i}{T}\right) \tag{6}$$

$$u^p_i = A \sum_{q \neq p} X^i_{pq} c_{pq} x^q_i - B \sum_{q \neq p} X^i_{pq} Y^j_{pq} x^q_i + C \sum_{q \neq p} c_{pq} sign(Q^x_i - P^x_i) - D x^p_i \tag{7}$$

where

$$Q^x_i = \sum_{j=i+1}^{d-1} x^q_j 2^j$$

$$P^x_i = \sum_{j=i+1}^{d-1} x^p_j 2^j$$

$$sign(x) = \begin{cases} 1, & x > 0; \\ -1, & x < 0; \\ 0, & \text{otherwise} \end{cases}$$

and A, B, C, D are constants.

The first term in equation (7) represents the attractive force from modules q with $X^i_{pq} = 1$, i.e., from modules in the same x-partition as p. The second term is the balance force. This

term covers modules q for which $X^i_{pq}Y^j_{pq} = 1$, i.e., modules in the same x- and y-partition as p.

The third term represents the attractive force from modules outside the x-partition of p. Here, Q^x_i and P^x_i uniquely code the x-partitions of q and p, respectively. If module q is in an x-partition to the left (right) of p, then $sign(Q^x_i - P^x_i)$ will be -1 (+1) and the force exerted on p by q will be proportional to $-c_{pq}$ ($+c_{pq}$) which has the effect of bringing p closer to q. Note also that the definition of the *sign* function above ensures that the effect of modules in the same x-partition as p will be zero.

Finally, the fourth term in equation (7) is a computationally inexpensive way to add random noise to the neuron input u^p_i, suggested by Fox and Furmanski in [5]. Such a term with the *wrong* sign tries to flip the neuron currently being updated. The effect is negligible if the neuron output has already converged to its final value, whereas it helps the network to climb out of local minima in the early stages.

2.4 Corresponding Energy Functions

Since update equation (7) was not developed from an energy function, it is not clear if the network possesses the convergence and energy minimizing properties of Hopfield-type networks. However, a corresponding energy function can be derived from equations (6) and (7).

Let T^i_{pq} denote the connection weight between neurons (i,p) and (i,q), and I^i_p denote the bias term for neuron (i,p). The update equation for a neuron input in a mean field neural net in terms of T^i_{pq} and I^i_p is

$$u^p_i = \sum_{q \neq p} T^i_{pq} x^q_i + I^i_p, \tag{8}$$

and the corresponding energy function for a symmetric weight matrix is given by

$$E_{x_i} = -\frac{1}{2} \sum_p \sum_{q \neq p} T^i_{pq} x^p_i x^q_i - \sum_p I^i_p x^p_i, \tag{9}$$

By equating equations (7) and (8) (disregarding the noise term) we get for our network

$$T^i_{pq} = (1 - \delta_{pq}) X^i_{pq} (Ac_{pq} - BY^j_{pq}) \tag{10}$$

$$I^i_p = C \sum_{q \neq p} c_{pq} sign(Q^x_i - P^x_i) \tag{11}$$

Since $[T^i_{pq}]$ is symmetric, an energy function corresponding to equations (6) and (7) is

$$\begin{aligned} E_{x_i} = & -\frac{A}{2} \sum_p \sum_{q \neq p} X^i_{pq} c_{pq} x^p_i x^q_i + \frac{B}{2} \sum_p \sum_{q \neq p} X^i_{pq} Y^j_{pq} x^p_i x^q_i \\ & -C \sum_p \sum_{q \neq p} c_{pq} sign(Q^x_i - P^x_i) x^p_i \end{aligned} \tag{12}$$

Note that energy function (12) is very similar to the bipartitioning energy function (1), except for the last term which is the sum of the individual bias terms (11). Actually, E_{x_i}

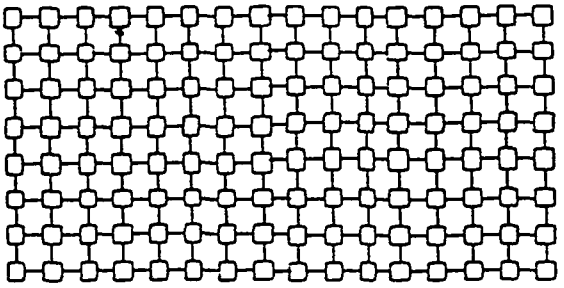


Figure 2: The 128-module example solved

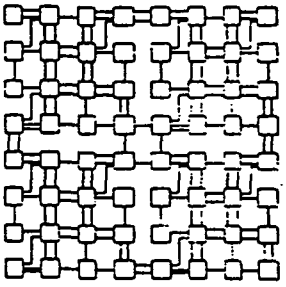


Figure 3: This 64-module test problem is the largest one in the literature, to our knowledge, solved by a Hopfield type neural network. The neural network of Sriram and Kang was able to find a solution with a cost of 182, whereas the optimal solution shown has a cost of 168. This solution was easily found by our network.

consists of 2^{d-i-1} individual (and independent) energy functions, one for each x-partition. If we limit p and q to a specific x-partition, say XP_0, then the energy function for XP_0 is given by

$$E_{x_i}^{XP_0} = -\frac{A}{2}\sum_{p\in XP_0}\sum_{\substack{q\neq p\\ q\in XP_0}} c_{pq}x_i^p x_i^q + \frac{B}{2}\sum_{p\in XP_0}\sum_{\substack{q\neq p\\ q\in XP_0}} Y_{pq}^j x_i^p x_i^q$$
$$-C\sum_{p\in XP_0}\sum_{q\neq p} c_{pq}sign(Q_i^x - P_i^x)x_i^p \qquad (13)$$

Our simulations show that the network indeed converges to local minima of the energy functions given in (13).

3 Simulation Results

The above described neural network algorithm was implemented in C on SUN4 SPARCstations and was tested on several hand-constructed examples with known optimal solutions

with up to 128 modules. In all cases, optimal solutions were found. Figure 2 shows the 128-module test problem.

We also tested the algorithm on the biggest example in the literature to our knowledge, solved with a Hopfield type neural network. This 64-module example, shown in Figure 3, is taken from [12], where the optimal solution couldn't be found. Our hierarchical algorithm needed $\log 64 = 6$ recursive partitioning steps to solve this problem. The number of iterations it took to converge at each step are: 12, 12, 24, 11, 40, and 30, respectively, where in one iteration we update all participating neurons exactly once (asynchronously). The optimal solution shown in the figure was easily obtained. The whole process took less than 10 CPU seconds.

In the simulations, we used a straightforward implementation of equation (7). This is not a very efficient way of implementing the algorithm on serial computers. The simulation algorithm can be speeded up considerably if we consider the following:

- X_{pq}^{i}, Y_{pq}^{j}, and Q_{i}^{x} are independent of the states of the neurons at the current level. They can be calculated *before* the $(d-i)^{th}$ vertical partitioning step. Thus, they can be treated as constants in equation (7).

- The connection matrix $[c_{ij}]$ can be stored as an array of linked lists where the i^{th} list consists of modules connected to module i. Thus, the first and third sums in equation (7) cover only modules q connected to p, instead of all q. This will achieve substantial speedup since the connection matrix is usually very sparse.

- The sum for the balance force stays constant from one update to the next, except for the term involving the last updated neuron. Thus, only the *change* in the balance force due to the change in the output of this neuron has to be calculated from one update to the next, which can be done in constant time.

These speedup techniques were incorporated in the first vertical bipartitioning step and a more than 10-fold speedup was observed. They will be implemented fully in the future versions of the algorithm.

4 Discussion and Conclusion

In this paper, we proposed a $O(N \log N)$ mean field neural network for the module placement problem. The problem is solved in a divide-and-conquer fashion by recursive bipartitioning where at each bipartitioning step exactly N neurons are evolved. The neural algorithm is similar to min-cut methods, yet maintains a level of globality, since all participating neurons evolve *simultaneously.*

The performance of the algorithm in our simulations has been very encouraging and merits further investigation. With the speedup techniques discussed in the previous section implemented, we can expect that the algorithm will take less than one CPU second to solve the 64-module problem shown in Figure 3 on a SUN4 SPARCstation. Since the speeded up version of the algorithm will be very fast, we will tackle larger and more realistic placement problems, including benchmark circuits.

The algorithm can be extended to popular layout styles like standard cell and sea-of-gates by integrating the area constraints. For example, the balance term in the update equation (7) can be modified to

$$- B \sum_{q \neq p} X^i_{pq} Y^j_{pq} x^q_i area_q \tag{14}$$

such that the bisection process produces partitions of approximately equal area.

One very important aspect of the algorithm is finding good parameters, and a good simulation temperature for each bipartitioning step. It is well known that mean field nets posses a *critical temperature*, T_c, [10], [5], [13], [4], where the bulk of the optimization occurs. Estimating T_c, and annealing the network around it not only shortens the running times, but also improves the solutions. In our simulations so far, we used a single temperature and empirically determined parameters. Future research will include the estimation of the parameters, the critical temperature, and the use of annealing.

Acknowledgement

We would like to thank Dr. G. C. Fox and Dr. W. Furmanski for useful discussions.

References

[1] H. Date, M. Seki, and T. Hayashi, "LSI module placement methods using neural computation networks", *Proceedings of the IEEE International Joint Conference on Neural Networks*, Vol. III, pp. 831-836, Jun. 1990.

[2] Gerald W. Davis, Jr., "Sensitivity analysis in neural net solutions", *IEEE Transactions on Systems, Man, and Cybernetics,* Vol. 19, No. 5, pp. 1078-1082, 1989.

[3] A. E. Dunlop and B. W. Kernighan, "A procedure for placement for standard cell VLSI circuits", *IEEE Transactions on Computer Aided Design CAD-4*, Vol. 1, pp. 92-98, 1985.

[4] L. Fang, W. H. Wilson, and T. Li, "Mean field annealing neural net for quadratic assignment", *International Neural Networks Conference, INNC 90 Paris,* pp. 282-286, 1990.

[5] G. C. Fox and W. Furmanski, "Load balancing loosely synchronous problems with a neural network", *Caltech Concurrent Computation Project,* Report No. C3P-363b, 1988.

[6] J. J. Hopfield and D. W. Tank, "Neural computation of decisions in optimization problems", *Biological Cybernetics,* Vol. 52, pp. 141-152, 1985.

[7] J. Naft, "NEUROPT: Neurocomputing for multiobjective design optimization for printed circuit board component placement", *Proceedings of the IEEE International Joint Conference on Neural Networks*, Vol. I, pp. 503-506, 1989.

[8] G. Persky, "Experiments in cell placement with a simulated network", *Proceedings of International Workshop on Placement and Routing, Research Triangle Park,* North Carolina, May 10-13, 1987.

[9] C. Peterson, J. Anderson, "Neural networks and NP-complete optimization problems", *Complex Systems*, Vol. 2, pp. 59-89, 1988.

[10] C. Peterson, B. Soederberg, "A new method for mapping optimization problems onto neural networks", *International Journal of Neural Systems,* Vol. 1, No. 1, pp. 3-22, 1989.

[11] K. Shahookar and P. Mazumder, "VLSI Cell Placement Techniques", *ACM Computing Surveys*, Vol. 23, No.2, 1991.

[12] M. Sriram and S. M. Kang, "A modified Hopfield network for the two-dimensional module placement", *IEEE International Symposium on Circuits and Systems*, pp. 1664-1667, 1990.

[13] D. E. Van den Bout and T. K. Miller, "Graph partitioning using annealed neural networks", *IEEE Transactions on Neural Networks*, Vol. 1, No. 2, 1990.

[14] G. V. Wilson and G. S. Pawley, "On the stability of the traveling salesman problem of Hopfield and Tank", *Biological Cybernetics,* Vol. 58, pp. 63-70, 1988.

[15] M. L. Yu, "A study of the applicability of Hopfield decision neural nets to VLSI CAD", *26th ACM/IEEE Design Automation Conference,* pp. 412-417, 1989.

Detection of Feed-Through Faults in CMOS Storage Elements [1]

Waleed K. Al-Assadi, Yashwant K. Malaiya*, and Anura P. Jayasumana
Electrical Engineering Department
* Computer Science Department
Colorado State University
Fort Collins, CO 80523

Abstract- **In testing sequential circuits, internal faults in the storage elements (SEs) are sometimes modeled as stuck-at faults in the combinational circuits surrounding the SE. The detection of some transistor-level faults that cannot be modeled as stuck-at are considered. These *feed-through* faults, cause the cell to become either *data-feed-through*, which makes the cell combinational, or *clock-feed-through*, causing the clock signal or its complement to appear at the output. Under such faults, the cell does not function as a memory element. Here it is shown that such faults may or may not be detected depending on delays involved. Conditions under which *race-ahead* occurs are identified.**

1 Introduction

Testing of sequential circuits has long been known to be a very difficult problem. Unlike the combinational logic, a test sequence is required to detect a fault in a sequential circuit. The test sequence has to include an initialization sequence and a propagation sequence. A common approach is to convert the problem of testing synchronous sequential circuits into the simpler problem of testing combinational circuit. This is accomplished by using testable design approaches like LSSD which provide direct access to inputs and outputs of combinational blocks [1, 2, 3]. If one can assumes that most faults within a SE can be modeled as stuck-at-0/1 faults on the inputs or outputs, then these faults do not need to be explicitly considered. This is because such faults are equivalent to the stuck-at faults in the combinational logic surrounding the SEs.

Considering SEs as primitives for the purpose of fault simulation and test generation for sequential circuits can significantly reduce computational complexity. This paper considers the problem of detecting faults in the CMOS synchronous SEs that cannot be modeled as stuck-at-0/1. Such faults, termed *feed-through* faults, cause a SE to become either *data-feed-through* or *clock-feed-through* and cause the cell to lose the sequential behavior [4]. These faults generally occur due to some internal bridging faults and are independent of transistor sizing. As an example consider the D-latch in Figure 1. Bridging faults between nodes D and $D1$ causes the cell to become *data-feed-through*, i.e. $Q = D$. Bridging faults between nodes CLK or $\overline{CLK}$ and D1 cause the cell to be *clock-feed-through*, i.e. $Q = CLK$ or $\overline{CLK}$. These faults can lead to timing problems because of coupling between combinational blocks normally separated by SEs. The formal definition for these two behaviors is given below.

Definition 1: Let T=$\{t_1,....,t_n\}$ be the set of all possible input combinations for an elementary synchronous SE with input D and a control signal CLK. Here t_i is a 2-tuple

[1]This work was supported by a SDIO/IST funded project monitored by ONR.

corresponding to (D, CLK) and n=4. Let $R(s, t_i)$ be the response of the cell to the input vector t_i applied to the cell when the cell is at state s. A faulty SE cell is said to have a *feed-through* fault if it becomes either *data-feed-through* or *clock-feed-through.*
(i) A faulty SE cell is said to be *data-feed-through* when its behavior becomes combinational such that $R(s, t_i) = f(y)$ for each $t_i \in T$, where y is the data part of t_i.
(ii) A faulty synchronous SE cell is said to be *clock-feed-through* if $R(s, t_i) = CLK$ or $\overline{CLK}$ where CLK is the control signal.

In a master-slave or a two-phase clocking circuit, the *clock-feed-through* fault may cause the succeeding SEs to always latch a 1 or a 0. This will cause the *clock-feed-through* faults to appear as stuck-at faults. Here we will show that in some cases *data-feed-through* faults cannot be detected because they can be masked by combinational propagation delays. If the timing allows, the *data-feed-through* faults can result in *race-ahead*, causing SE to reach the next state one clock period too early. The following section examines the problem of detecting *data-feed-through* faults in detail.

2 Feed-through Faults in Sequential Circuits

A latch is in the *transparent phase* when the clock is high. The falling edge of the clock serves as the sampling edge when the latch locks in the input value. and the latch enters the *latch phase.* It is common to use a pair of latches such that they are triggered by non-overlapping clock phases $\phi 1$ and $\phi 2$. This avoids the problem of race-ahead when feedback is present, because at any time only one of the two latches can be in the transparent phase.

When one of the two latches in a pair is *data-feed-through*, it is possible for a transition to race-ahead in one clock period through two combinational blocks (Figure 2a). If the circuits involves feedback (Figure 2b), then the presence of a *data-feed-through* fault may cause transitions normally corresponding to two successive clock-periods to occur within a single clock-period. This can give rise to race-ahead as defined below.
Definition 2: A *race-ahead* occurs when a SE goes from state s_i to s_{i+2} in one clock period, whereas normally a transition from s_i to s_{i+1} should occur, followed by a s_{i+1} to s_{i+2} transition in the next clock period.

It can be shown that a race-head may not occur in some cases and thus a *data-feed-through* fault may not be detected. Let us consider the diagram in Figure 2a consisting of two pipelined combinational blocks. For the normal circuit the propagation of a transition (new logical values) can be described by the following sequence:
1. $\phi 1 \downarrow$: a transition latched in $L11$.
2. $\phi 2 \downarrow$: corresponding transition (CT) latched in $L12$.
3. $\phi 1 \downarrow$: CT latched in $L21$ (after passing through $C1$).
4. $\phi 2 \downarrow$: CT latched in $L22$.
5. $\phi 1 \downarrow$: CT latched in $L31$ (after passing through $C2$).
6. $\phi 2 \downarrow$: CT latched in $L32$.

Where $\downarrow$ indicates the falling edge.

The two phases constitute one clock period. To specify the requirements for correct operation, let us adapt the following notation:
d_{c1}, d_{c2} = propagation delays through $C1$ and $C2$, respectively.

d_{LCQ} = clock-to-output delay through a single latch.
d_{LDQ} = data-to-output delay through a single latch (in transparent phase or when data-feed-through fault is present).
t_s = latch set-up time with respect to the falling edge.
$t_{\phi 1\phi 2}$, $t_{\phi 2\phi 1}$ = duration between falling edge of $\phi 1$ and $\phi 2$, and $\phi 2$ and $\phi 1$ respectively.
$g_{\phi 1\phi 2}$, $g_{\phi 2\phi 1}$ = the gap between $\phi 1 \downarrow$ and $\phi 2 \uparrow$, and between $\phi 2 \downarrow$ and $\phi 1 \uparrow$ respectively.
The major requirements for correct operation are,

$$t_{\phi 1\phi 2} \geq max(g_{\phi 1\phi 2}, d_{LCQ}) + t_s \quad (1)$$

$$t_{\phi 2\phi 1} \geq max(g_{\phi 1\phi 2}, d_{LCQ}) + d_{c1} + t_s \quad (2)$$

and similarly

$$t_{\phi 2\phi 1} \geq max(g_{\phi 1\phi 2}, d_{LCQ}) + d_{c1}max + t_s \quad (3)$$

Now let us consider the case when latch $L21$ has data-feed-through fault, when $\phi 2$ signal is active, there exists a combinational path $L12 - C1 - L21 - L22$. The following sequence is possible:
1. $\phi 1 \downarrow$ = a transition latched in $L11$.
2. $\phi 2 \downarrow$ = CT latched in $L22$, provided the inequality (4) below is satisfied:
3. $\phi 1 \downarrow$: CT latched in $L31$.
4. $\phi 2 \downarrow$: CT latched in $L32$.

Thus between two successive falling edges of $\phi 2$ (i.e. within a single clock period) both $C1$ and $C2$ are traversed. This can occur only if:

$$t_{\phi 1\phi 2} \geq max(g_{\phi 1\phi 2}, d_{LCQ}) + d_{LDQ} + d_{c1} + d_{LDQ} + t_s \quad (4)$$

where d_{c1} is the delay through a sensitized path.

In some situation, the condition in inequality (4) may not be satisfied and the following sequence may occur.
1. $\phi 1 \downarrow$: a transition latched in $L11$.
2. $\phi 2 \downarrow$: CT does not arrive at $L22$ in time but is latched in $L12$.
3. $\phi 1 \downarrow$: has no effect on $L21$.
4. $\phi 2 \downarrow$: CT arrives at $L22$ and is latched.
5. $\phi 1 \downarrow$: CT latched in $L31$.
6. $\phi 2 \downarrow$: CT latched in $L32$.

The above sequence will occur if:

$$T \geq d_{LCQ} + d_{c1} + d_{LDQ} + t_s \quad (5)$$

Where T is the clock period ($t_{\phi 1\phi 2} + t_{\phi 2\phi 1}$), and if (4) does not hold. In this case the operation is normal and race-ahead is not observed. This suggest that in some cases higher propagation delays can mask *data-feed-through* faults.

Similar conditions can be obtained if the second latch, for example $L22$ has a *data-feed-through* fault. Race-ahead can also occur in circuits with feedback [4].

3 Conclusion

Some defects in storage elements can cause *feed-through* faults which need to be considered when high fault coverage is required. The *clock-feed-through* faults will generally appear as stuck-at-0/1 faults. We have shown here that *data-feed-through* faults can cause race-ahead in synchronous sequential circuits. In some cases, the *data-feed-through* faults may be masked. The conditions for these have been presented.

References

[1] M. K. Reddy and S. M. Reddy, "Detecting FET Stuck-Open Faults in CMOS Latches and Flipflops," *IEEE Design and Test,* pp. 17-26, October 1986.

[2] D. L. Liu and E. J. McCluskey, "A CMOS Cell Library Design for Testability," *VLSI Systems Design,* pp. 58-65, May 4, 1987.

[3] R. Anglada and R. Rubio, "Functional Fault Models for Sequential Circuits," *Research Report DEE-3*, Electronic Engineering Department, Polytechnical University of Catalunya, Barcelona 1987.

[4] W. K. Al-Assadi, Y. K. Malaiya, and A. P. Jayasumana, " Use of Storage Elements as Primitives for Modeling Faults in Sequential Circuits," To appear in Proc. Int. Conference on VLSI Design, January 1993.

[5] Y. K. Malaiya and R, Narayanaswamy, "Modeling and Testing for Timing Faults in Synchronous Sequential Circuits," *IEEE Design & Test of Computers,* vol. 1, pp. 62-74, November 1984.

[6] M. R. Dagenais and N. C. Rumin, "On the Calculation of Optimal Clocking parameters in Synchronous Circuits with Leve-Sensitive Latches," *IEEE Transaction on Computer-Aided Design,* vol.8, no. 3, pp. 268-278, March 1989.

[7] S. H. Unger and C. Tan, "Clocking Schemes for High-Speed Digital Systems," *IEEE Transaction on Computers,* vol. C-35, no. 10, pp. 880-895, October 1986.

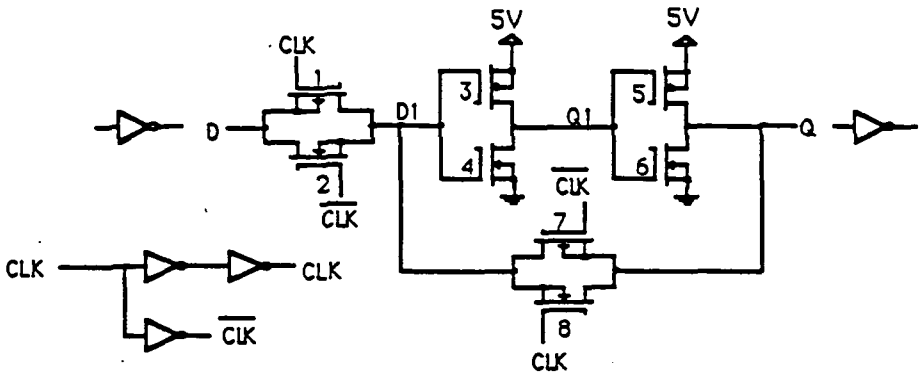
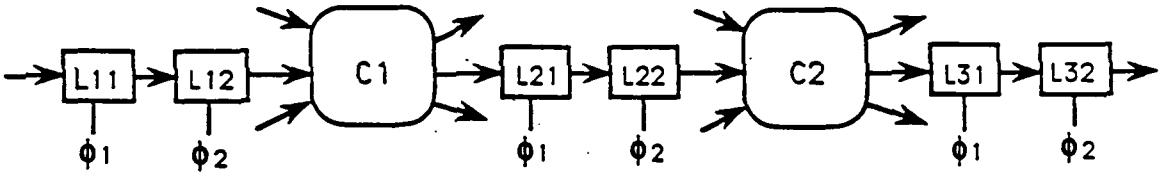
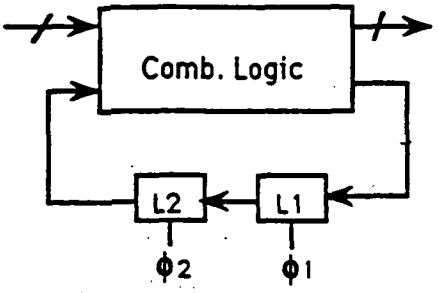


Figure 1 The D-latch



a. Two pipeline stages



b. Synchronous sequential circuit with feedback

Figure 2 Use of latches with 2-phased clock

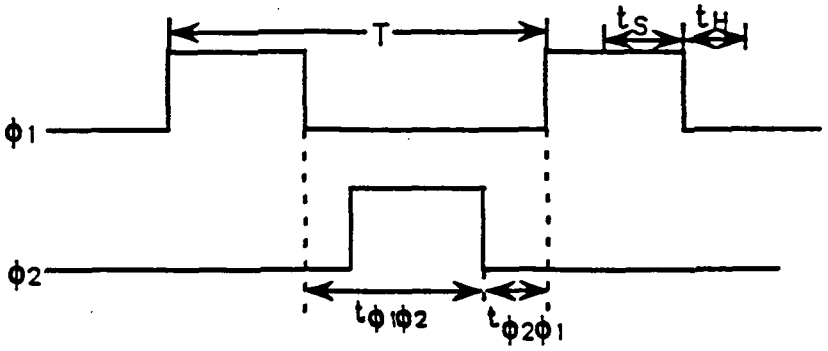


Figure 3 The two clock phases

Schematic Driven Layout of Reed Solomon Encoders

Kari Arave, John Canaris, Lowell Miles and Sterling Whitaker
NASA Space Engineering Research Center
University of New Mexico
2650 Yale SE, Suite # 101
Albuquerque, New Mexico 87106

Abstract – **Two Reed Solomon error correcting encoders are presented. Schematic driven layout tools were used to create the encoder layouts. Special consideration had to be given to the architecture and logic to provide scalability of the encoder designs. Knowledge gained from these projects was used to create a more flexible schematic driven layout system.**

1 Introduction

The layout of custom VLSI circuits frequently requires the generation of very regular, densely packed functional blocks. These blocks require extreme attention to area consumed as well as attention to local and global routing considerations. It is often appropriate to perform all routing inside the base cells of these sections. In this case, no routing channels are needed and all routing is done with *connection by abutment.* The complex wiring task is addressed by the layout designer during the planning and cell layout stage. After the base cells are layed out they must be placed together to form the larger functional block. As all interconnections are made by abutment, little mental effort is required at this stage.

This placement stage, in the layout of regularly structured blocks, is time consuming and tedious. A schematic driven, smart tiler, coupled with a schematic capture system which allows connections by abutment, provides a significant time savings in layout design.

The two Reed Solomon (RS) error correcting encoders presented in this paper were developed for Hewlett-Packard (HP) Disk Mechanism Division using a $0.75\mu m$ triple metal CMOS technology. These encoders were layed out using a schematic driven tiler and were used, by HP, as macrocells on HP designed custom chips. This paper will first present a general review of Reed Solomon encoding followed by a description of the two specific encoders designed for HP. The paper will conclude with a description of the the schematic driven layout and the tools used to create it.

2 Reed Solomon Encoders

A Reed Solomon (RS) code is a cyclic symbol error correcting code for correcting errors introduced into data during transmission through a communication channel [1]. Information is represented as m bit binary symbols forming a finite field $GF(2^m)$ containing 2^m-1 nonzero symbols. The elements of the field are specified by an irreducible, primitive polynomial $p(x)$.

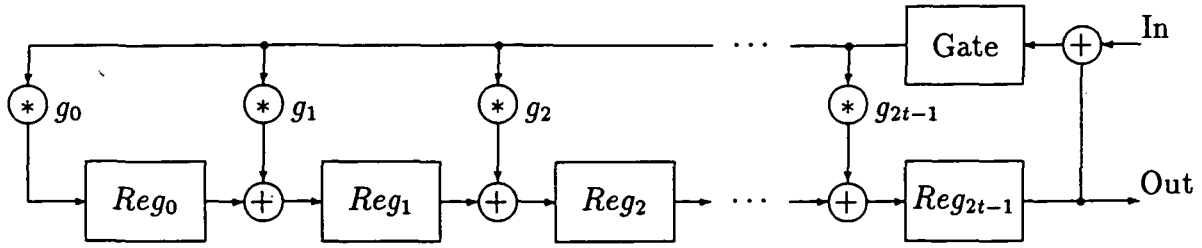


Figure 1: Parity generator block diagram.

A code block consists of $N \leq 2^m - 1$ symbols. Each code block includes both information symbols and parity symbols. In order to correct t symbol errors, $2t$ parity symbols are calculated and appended to a group of information symbols during an encoding process. An RS code is classified as a (N,k) code where k is the number of information symbols in a code block and N is the total number of information plus parity symbols. $N = 2^m - 1$ for a full length code. If $N < 2^m - 1$, the code is said to be shortened.

The RS code block can be defined as a polynomial $c(x)$, of order $N - 1$, formed from an order $k - 1$ information polynomial, $m(x)$, and an order $N - k$ generator polynomial, $g(x)$, as in Equation 1. A shift in time, due to the addition of the parity symbols, is represented by x^{2t}.

$$c(x) = x^{2t}m(x) + x^{2t}m(x) \bmod g(x). \tag{1}$$

Consequently, every valid code block is a multiple of the generator polynomial

$$g(x) = \prod_{i=s+1}^{s+2t} (x - \beta^i) = \sum_{j=0}^{2t} g_i x^i \tag{2}$$

where $\beta = \alpha^h$ is a primitive element of the field, s is an offset term and t is the error correcting ability of the code. A transmitted code block, $c(x)$, is modified by the introduction of errors in a noisy channel. The appended parity from the encoding process allows t such errors to be corrected during a decoding process.

An encoder can also represent data in a dual basis such that

$$[z_0, z_1, \ldots, z_{m-1}] = [u_{m-1}, u_{m-2}, \ldots, u_0]\, T \tag{3}$$

where $[z_0, z_1, \ldots, z_{m-1}]$ is the symbol represented by the dual basis, the symbol representing the normal basis is $[u_{m-1}, u_{m-2}, \ldots, u_0]$ and where T is an m by m transform matrix. Normal data can be derived from data represented in the dual basis using an inverse transform, T^{-1}.

A dual field is simply a different representation of the original field. The coefficients of $g(x)$ are linear operators. An operator O in the original representation of the field can be used in the dual representation by applying the following transform.

$$O_{dual} = TO_{original}T^{-1} \tag{4}$$

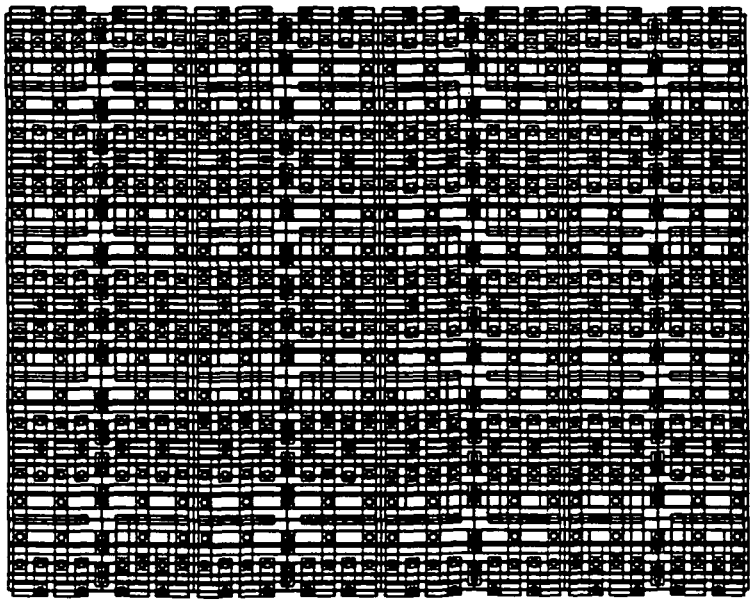


Figure 2: Layout of constant multiplier.

Figure 1 shows a block diagram of the logic required to perform the parity generation function. The parity generator organization can be viewed as a set of slices. Each slice consisting of a multiply/add structure and a register. Each multiplier multiplies data by a constant, g_i, which is a coefficient of the generator polynomial. Interleaving of two or more encoded messages allows higher burst error correction capabilities by spreading out the effect of burst errors over more than one codeword. The desired interleaving depth, I is controlled by the depth of the register shown in Figure 1. Each register is therefore an m-bit wide, I-bit deep shift register.

The two encoders designed for HP operated on 8-bit symbols ($m = 8$) forming a field GF(2^8). The first, RS1, corrected a single symbol error ($t = 1$) and had no interleaving ($I = 1$). The second, RS3, corrected three errors ($t = 3$) and had a variable interleave depth of $I = 2, 3, 5$ or 9. The code blocks were of variable length, $2t + 1 \leq N \leq 255$ where 255 is the full length code ($2^m - 1$). These encoders operated in the normal basis and required no data transformation. The primitive polynomials were identical, but the two used different generator polynomials since the correcting capabilities were different. Both were designed for a sustained 30 MHz (240 Mbit/sec) data rate and included circuitry in the parity generator to detect zeros.

3 Layout

One restriction on most schematic driven layout is the requirement that the schematic exactly match the layout. The schematic becomes a floor plan for the tiler. Rotations, mirrorings, hierarchy, intermediate node labels, port positions, names and locations of all cells must exactly match the layout. Since schematics are usually captured early in the design process, so that schematic driven simulation can be used to verify the logic design, physical layout

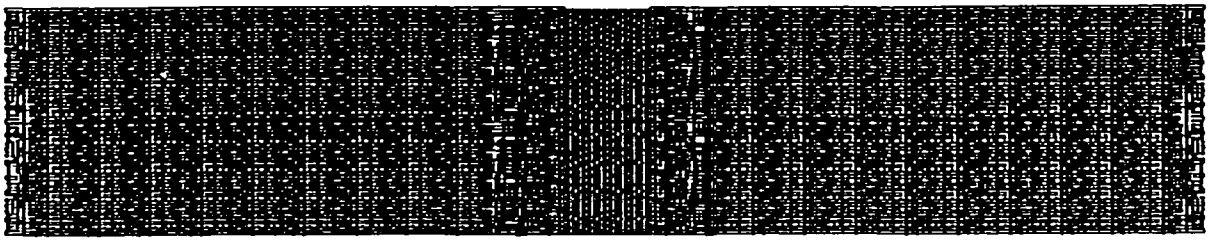


Figure 3: Layout of two adjacent RS3 slice details.

information is not available at the time schematics are originally captured. This results in additional schematic capture iterations, following the design phase, when schematics are redrawn to match the layout, and logic simulations are rerun to verify that no errors have been introduced during the modifications. To overcome these restrictions, tiling parameters were added to the schematic to avoid this rework and to improve the readability of the encoder schematics. A slice in the encoder parity generator will be used to illustrate this improvement.

A slice consists of a multiplier, adder and register. The multiplier cell is a precharged exclusive or (XOR) chain. Eight of these chains form a constant multiplier. The input data word is multiplied by a constant, g_i, programmed into the multiplier as XOR cells or interconnect (ZERO) cells. The XOR cell consists of 4 NMOS transistors. The ZERO cell is a modified XOR cell which acts as an interconnect block.

The multiplication constant can be programmed with a single mask layer defining the pattern of XOR and ZERO cells in the XOR chains. The final layout of a constant multiplier is shown in Figure 2. For maximum operating speed the XOR chain is precharged from both ends. The addition function is folded into the evaluate structure for the multiplier. The XOR cell was designed in layout to consume minimum area and the registers were designed to match the pitch of two XOR cells. Half the registers were placed above the multiplier and half below.

Since the registers are twice as wide as the XOR chain, the outputs of the columns in the multiplier alternate between top and bottom. The higher order nibble is output on one end of the constant multiplier and the lower order nibble on the opposite end. This requires the columns of the multiplier matrix to be rearranged such that the columns in the matrix are $[C_0C_4C_1C_5C_2C_6C_3C_7]$. Also, in order to avoid long interconnect runs between registers on the top and bottom to drive the adder inputs in the multiply/add structure, a second slice detail was drawn such that the top and bottom sections were reversed. These two slice details were then alternated in the parity core allowing connection of the adjacent slices by abutment. Figure 3 shows the final layout of two adjacent RS3 slices including routing between the slices. There is no interconnect required between any of the leaf cells. The entire structure is connected by abutment and thus is an ideal candidate for tiling.

Figure 4 is the logic diagram for the top of the slice drawn during logic design. Figure 5 is the same diagram modified to drive the tiler. Notice that the cell mirrorings and rotations cause a loss of readability and that the hierarchy changes hide the logical functionality,

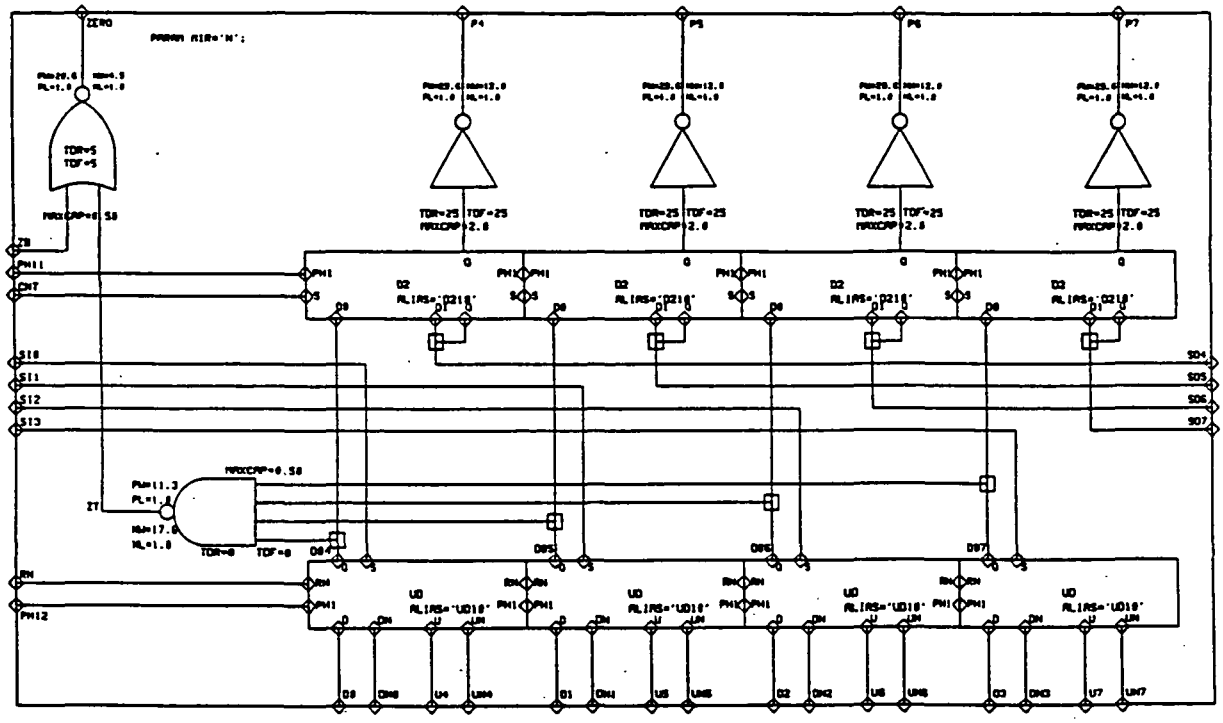


Figure 4: Original schematic diagram for top of RS1 slice.

however, the block can drive the tiler and labels are automatically placed on the artwork from the schematic. This makes labeling a one time manual task.

Figure 6 is the original logic diagram for the multiplier core. Figure 7 is the multiplier diagram modified to drive the tiler. The cell mirrorings and rotations again cause a loss of readability. The schematic entry program was then modified to allow tiling parameters to be added. This allowed the original logic diagram to drive the tiler directly without the need to redraw the schematic. Figure 8 is the original logic diagram modified at the time of layout with parameters to drive the tiler.

4 Acknowledgements

This research was supported in part by NASA under Space Engineering Research Grant NAGW-1406 and by the NSF under Research Initiation Grant MIP-9109618.

References

[1] G. C. Clark and J. B. Cain *Error Correcting Coding For Digital Communications*, New York NY, Plenum Press, 1981

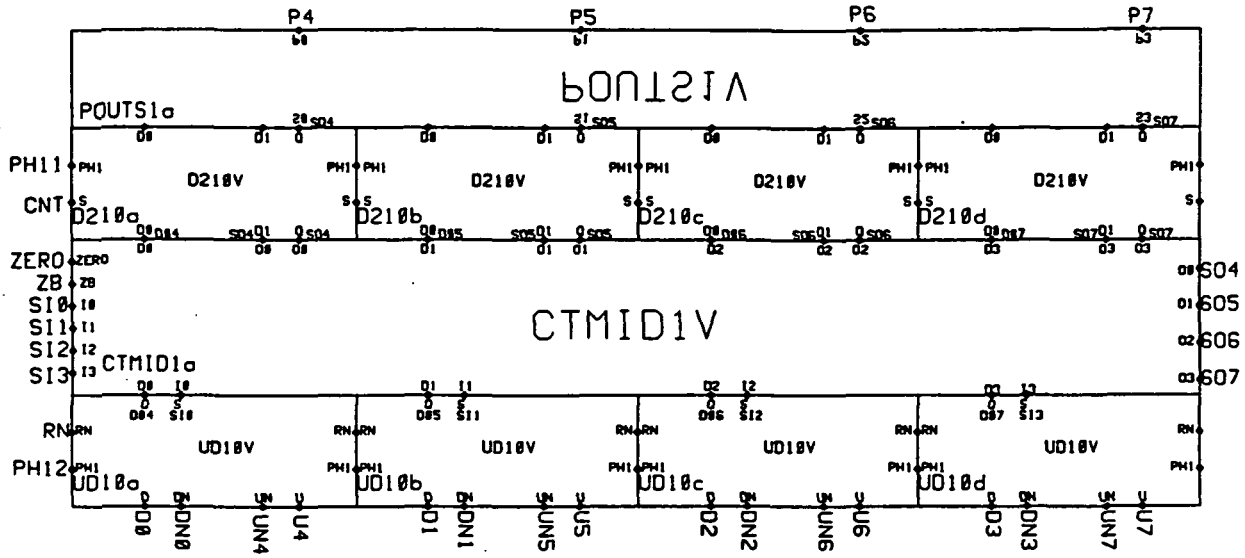


Figure 5: Schematic diagram for tiling.

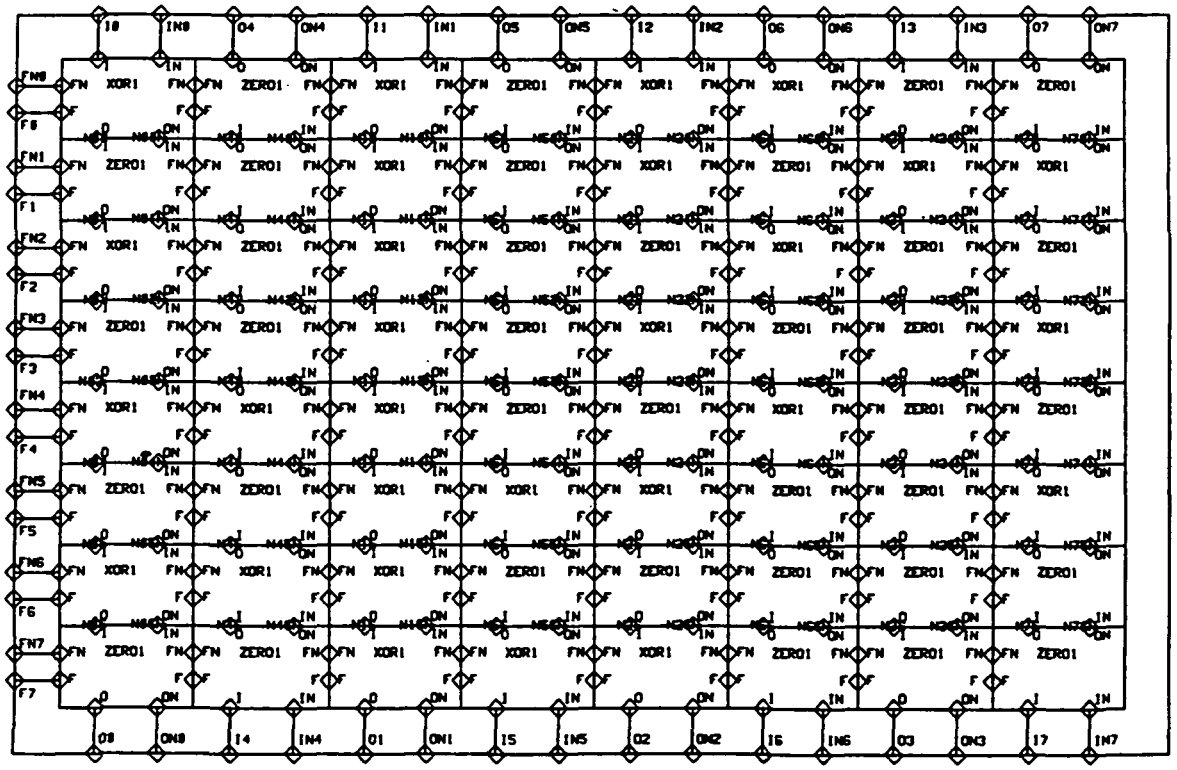


Figure 6: Original schematic diagram for multiplier core.

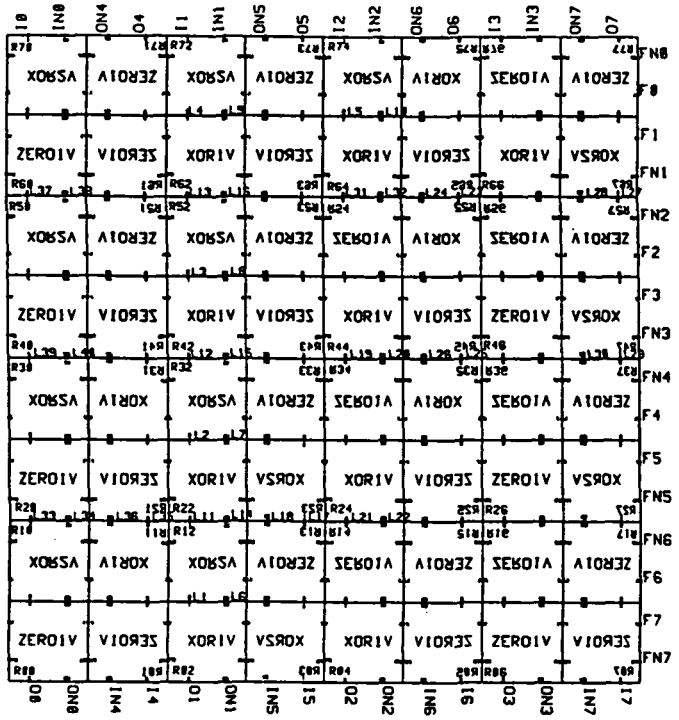


Figure 7: Multiplier diagram for tiling.

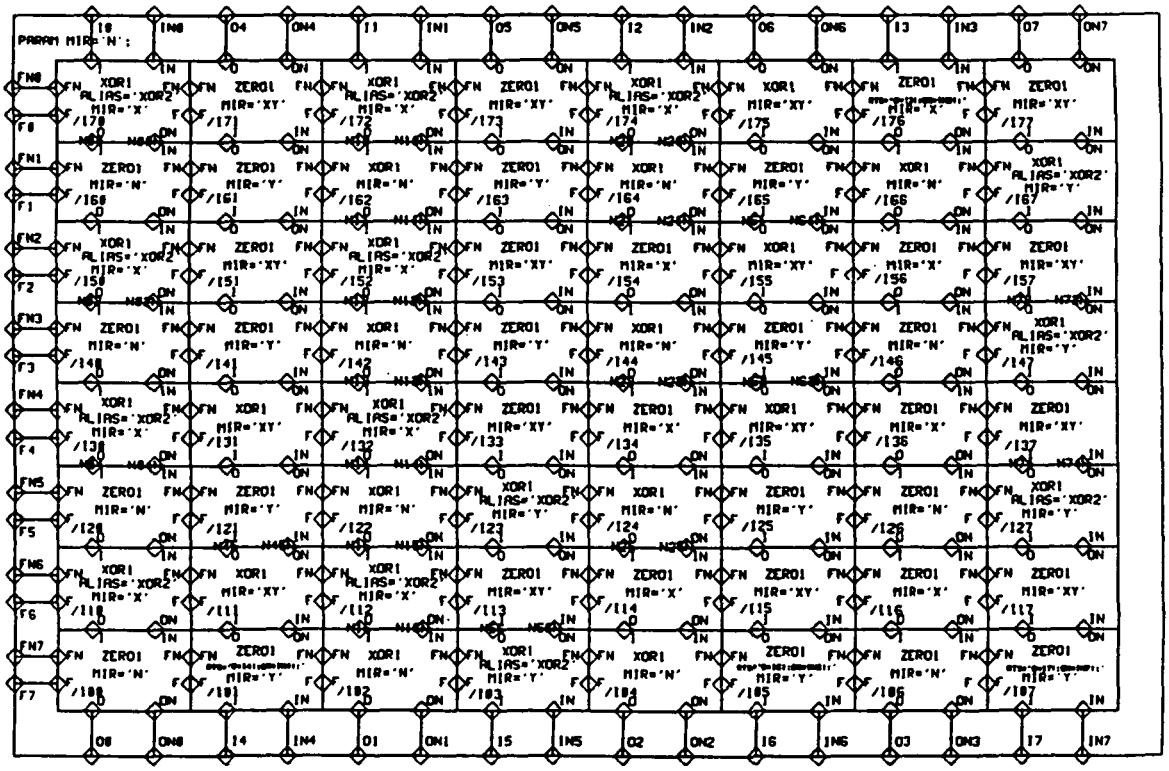


Figure 8: Parameterized schematic diagram for multiplier core.

Interpolative Modeling of GaAs FET S-parameter Data Bases for Use in Monte Carlo Simulations

L. Campbell and J. Purviance
NASA SERC for VLSI System Design
University of Idaho, Moscow, Idaho 83843

Abstract - **A statistical interpolation technique is presented for modeling GaAs FET S-parameter measurements for use in statistical analysis and design of circuits. This is accomplished by interpolating among the measurements in a GaAs FET S-parameter data base in a statistically valid manner.**

1 Introduction

Statistical analysis and design of high frequency GaAs circuits requires accurate statistical models of the variation of the GaAs FETs' performance. In this paper we develop a method for modeling a GaAs FET S-parameter data base that is concise, efficient, accurate, and which can generate a simulated data base which is statistically indistinguishable from a measured data base. Two sets of data samples will be said to be indistinguishable if their statistical properties do not differ. This goal is met by introducing and developing the statistical interpolation model which was first presented in this context in Campbell's dissertation [4, 5]. The term "statistical interpolation model" was used there to refer to the density estimation techniques used in this work. These density estimation techniques are based on kernel density estimation and data clustering. "Interpolative model" will be used from here on as a shortened form for statistical interpolation model. The interpolative model is developed here for the purpose of modeling probability density functions (PDFs) for use in statistical modeling of GaAs FETs. A probability density function is defined in Definition 1. The weighted sum of two or more PDFs is also a PDF.

Definition 1 *Probability Density Function.*

$$f(x) \ni: \int_{-\infty}^{+\infty} f(x)dx = 1, f(x) \geq 0 \; \forall \; x$$

2 Modeling Assumptions

As we have already stated above, our objective is to create a sample data base that has the same statistics as a measured data base. To put this in more precise terms, we must find the statistical distribution (PDF) of the population from which the measured samples were taken. Such a PDF cannot simply be directly calculated. There are an infinite number of possible densities from which a data set may have been sampled. For example, it is possible that the PDF is a set of peaks centered at each of the measured data points, a simple uniform distribution, or the PDF might be a series of peaks and valleys similar to the Mandelbrot set [2]. In order to model the data PDF, we must make educated assumptions about its nature.

The assumptions made to construct the model require knowledge about the kind of data expected. In order to model the PDF of a data set we need to ask the following questions:

- What are we modeling and what are the known properties of its PDF?
- What models have others used and could they be improved upon?
- For what are we going to use the model?
- What modeling assumptions should be made based on the answers to the previous questions and what will be the effect of these assumptions?

In this work we are modeling the statistical properties of a set of manufactured transistors. While other kinds of devices could be modeled, our particular application is to GaAs integrated circuits. The knowledge about the statistical properties of the GaAs FET parameters is limited. This is because the GaAs FET manufacturing processes in general are new and because each set of data to be modeled will come from new fabrication lines. It has generally been accepted that the univariate marginal distributions and the joint probability density functions of the parameters are continuous [2]. Their multivariate distributions can be expected to have short tails and have single or multiple modes clustered in a local region [2, 15]. We list these properties below:

- Continuous univariate marginal distributions.
- Continuous joint probability density functions.
- Short tailed multivariate distributions.
- Single or multiple modes clustered in a local region.

Others have used unimodal, univariate and multivariate trimmed and nontrimmed Gaussian distributions as parameter models. This is based on the assumption that the individual parameters are statistically independent. As shown in [2], this assumption is very simple and highly unlikely. Others have also modeled the parameter densities by their marginal distributions and covariance matrices [15]. As shown in [10], the marginal distribution and covariance matrix method is not adequate for an accurate model. The main reason is that the others' techniques do not model the higher order statistical structure of the data. That is to say they do not properly model the local modes and valleys of the joint probability distribution. Since the parameters will be used as the input to a simulator, error from modeling the parameters will affect the accuracy of the simulation [10]. In light of the above stated nature of the data to be modeled, the general direction taken in *this work assumes that the data PDF is a finite mixture of multivariate Gaussian distributions.*

A finite mixture $p(x)$ is a sum of a set of subdistributions $K_i(x)$ where the subdistributions may take any form [16]. Gaussian subdistributions are chosen because they have desirable statistical properties [13]. In addition, the data will be assumed to be time invariant over the time period of its use [2]. It will be assumed that new data will be measured if at any point the underlying process changes. A finite mixture distribution is defined as follows:

Definition 2 *Finite Mixture Distribution*

$$p(x) \doteq \sum_{i=1}^{n} \pi_i K_i(x)$$

In theory, a finite mixture distribution can model closely any distribution since for example as n goes to infinity the distribution has an increasing number of subdistributions. This ultimately becomes an infinite set of points if the Gaussians have zero variance. An infinite collection of points will accurately model any distribution. In practice, n will be a small number, 10 for example. For small n, these modeling assumptions do not model all the variation to be found in the PDFs of all possible data. It is however a substantially more robust model than the previously used techniques. Detailed analysis of the accuracy of the model was presented in Campbell [4]. How this model is constructed from the measured data is the subject of the next sections.

3 Variable Kernel-Based Method

The first technique we will use for statistical interpolative modeling is based on variable kernel density estimation [3]. The variable kernel approach to density modeling is the best suited of the standard nonparametric density estimation techniques for the kinds of data we wish to model. This will be discussed in some detail in the first subsection. Also in the first subsection, we will describe the basic variable kernel density estimation and say why it is useful. Then we will describe what its limitations are, and how we extended it to better suit our particular kind of data.

3.1 Variable Kernel Density Estimation

The variable kernel density estimation method was invented by Brieman *et al.* and presented in the paper[3]. It combines the advantages of the kernel density estimation technique, and the nearest neighbor technique. That is, the data dependency of the kernel estimate, and the local density dependency of the nearest neighbor estimate. In kernel density estimation, the position of data samples is used as the basis for defining the shape of a density estimate. In variable kernel density estimation, the spacing of the data samples is used in addition to their position for defining the shape of the density estimate.

Kernel density estimation is based on the idea that each point of a data set contributes an equal amount of information about the density from which it is sampled. If the density is locally Gaussian, an estimate of the real density may be constructed by putting a Gaussian distribution around each data point. The sum of these Gaussian distributions forms an estimate of the real density. The choice of Gaussian kernel distributions is reasonable since it matches the assumption made previously that the data PDF is a finite mixture of Gaussian distributions.

If we take a PDF as in Figure 1a and sample it as in Figure 1b, then a fixed kernel estimate would take a kernel like the PDF on the right-hand side in Figure 1c and put it around every data point. The normalized sum of these kernels in Figure 1d forms the estimate of the original PDF in Figure 1a. In the variable kernel method, the kernels vary

in shape according to the local density of data points in a neighborhood of the data point around which the kernel is put. A data point with a high local density would have a PDF like the one on the right-hand side in Figure 1c, and a data point with a low local density would have a PDF like the one on the left-hand side in Figure 1c. That is to say that the variance of the Gaussian kernels is higher for regions of low density and the variance is lower for regions of high density. The normalized sum of these variable kernels forms the estimate of the original PDF as in Figure 1e.

The variable kernel method combines the kernel estimator and the nearest neighbor method to produce an estimator that is smooth and varies according to the local density of the data [11]. Kernel density estimation can also be thought of as a moving average which averages the points within the kernel window. The nearest neighbor method is based on the idea of smoothing data according to the local density of the data. The shape of the kernel changes according to the width of the window needed to contain a fixed number k of points. The width of the box is found by finding the kth nearest neighbor. If we order the data points near a given data point x, then the kth neighbor at distance $d_k(x)$ is the kth nearest neighbor. Model parameters are chosen so that the PDF of the model smooths or interpolates the data, in order to match the statistics of the data PDF.

3.2 Extension of Variable Kernel Density Estimation

Variable kernel density estimation is limited by the fact that the kernels are not correlated with the local region. The Gaussian distributions used as the kernel PDFs may not accurately reflect local trends in the data. For example, if all the data are in a line then, in order to reflect the local trend, the kernel PDFs should be too. In the variable kernel method however, the kernel distributions will have excess probability off the local trend. This is illustrated in Figure 2 where the data samples are in the middle, the variable kernel method is at the bottom, and the desired result is on top. In order to correct for this deficiency, we developed the concept of a localized nearest neighbor.

The Kernels need to be oriented in the same manner as the orientation of the local trend. The localized nearest neighbor matches the local trend by restricting the choice of nearest neighbor to a "local region" around the anchor point of the kernel. *Each dimension* of the kernel is normalized to reflect the direction of and distance to the localized nearest neighbor in the local region. The shape of the local region is a rectangular box whose sides are proportional to the standard deviation in that dimension for all the data. The size of the local region is a model parameter added to those already required for variable kernel density estimation. These model parameters are optimized to fit the data.

The nearest neighbor found within this local region is then used to orient the kernel PDF so as to reflect the local trend. Consistent with our assumption about the data PDF, the kernel PDF is a multidimensional standard Gaussian distribution with uncorrelated components and zero mean. In order to modify the kernel PDFs to reflect the local trend, each dimension of the kernel PDF is multiplied by the distance in each dimension to the local nearest neighbor. This requires modification to the distance calculation in the variable kernel estimate formula. These modifications are shown below where m is the number of dimensions of the data and each dimension of the kernel is made proportional to the localized

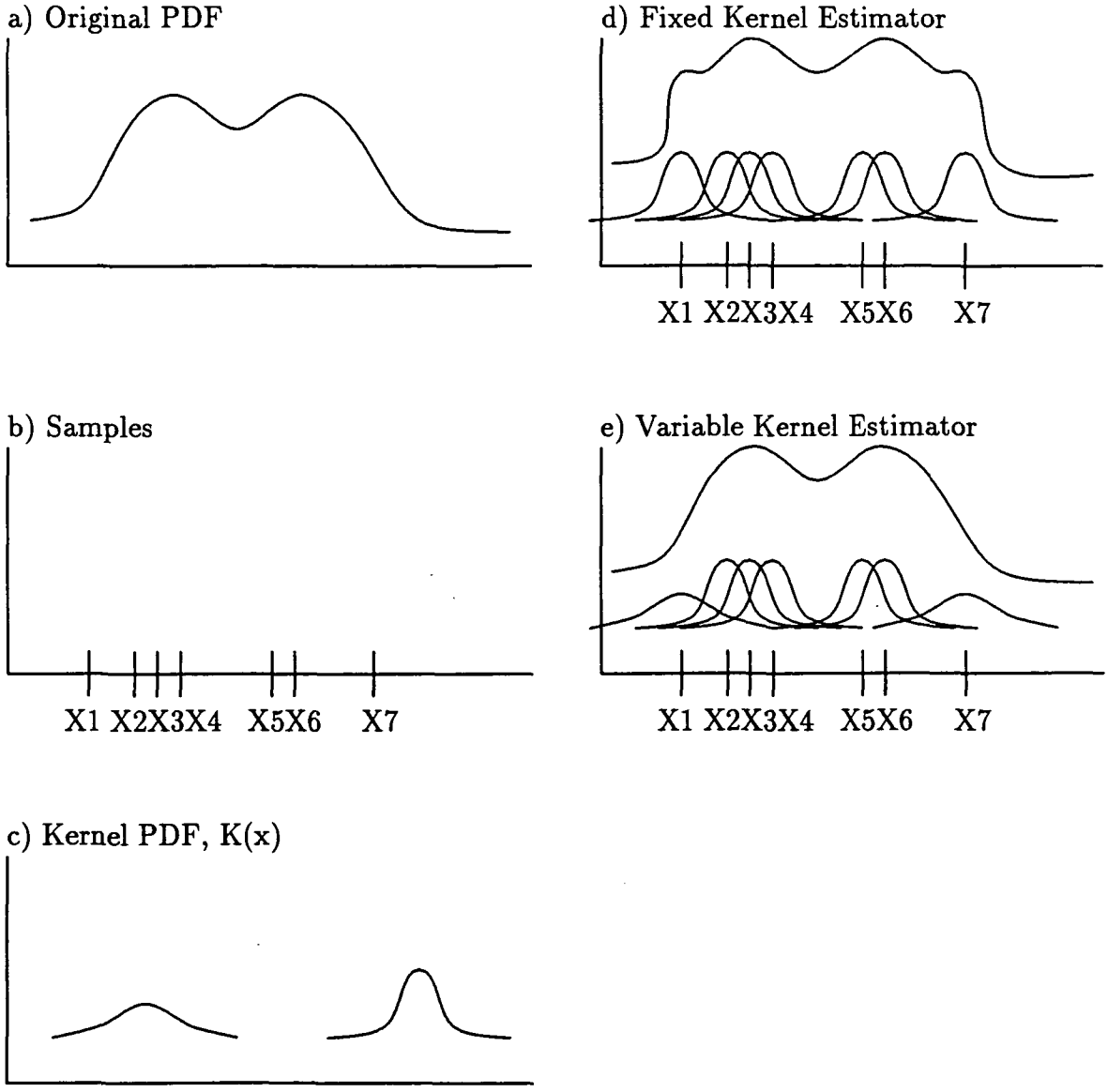


Figure 1: Kernel Variation

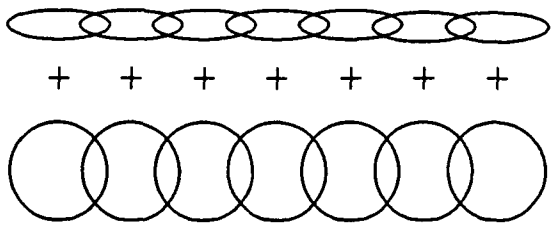


Figure 2: Illustration of Local Trends

nearest neighbor $d_{i,j,k}(x)$.

Definition 3 *Extension of the Variable Kernel Method.*

$$\hat{f}(x) = \frac{1}{n}\sum_{i=1}^{n}\prod_{j=1}^{m}\frac{1}{hd_{i,j,k}(x)}K\left(\frac{x - X_i}{d_{i,j,k}(x)}\right)$$

The sum of all the kernel PDFs forms the model PDF used to generate the simulated data. Figure 3 illustrates the idea, where kernels are shown as ovals around the data points. The generation of points using the extended variable kernel method works as follows. The localized nearest neighbor is found for all the points in the data base of samples from the original PDF. A data point P_i is chosen at random from the data base of samples from the original PDF. The spread around each data point P_i, is determined by the model parameters a and K_i, where K_i is a function of the kth nearest neighbor in the local region determined by q. The choice of a, q, and k is done by an optimization process that is discussed in [4]. The generation of points is done by the following equation where data points are vectors in the data space:

$$\hat{P}_j = P_i + a\Delta P_j diag(K_i(k,q)) \tag{1}$$

Where:

$\hat{P}_j$	is a data point vector generated from this model;
P_i	is a measured data point vector chosen at random from the measured data;
a	is a constant model parameter;
ΔP_j	is a vector chosen at random from the kernel PDF;
$K_i(k,q)$	is a scaling vector containing the distance from the chosen P_i to the kth nearest neighbor in each of the data's dimensions [6].

4 Cluster-based methods

The previous technique works well for large data sets in low dimensional spaces [13]. The analysis in [13] suggests that it will not work as well for larger numbers of dimensions. This was investigated in Campbell [4]. This section presents an alternative way of reconstructing the large dimensional PDFs of an S-parameter data base. It has the added benefit that it requires less memory for storage and thus may be used for data reduction. The assumption made in order to model S-parameters was that the PDF was a finite mixture of Gaussian distributions (Section 2 and repeated below). This new process works by finding the groups of data that make up the individual Gaussian distributions $K_i(x)$ of the finite mixture.

Definition 4 *Finite Mixture Distribution*

$$p(x) \doteq \sum_{i=1}^{n}\pi_i K_i(x)$$

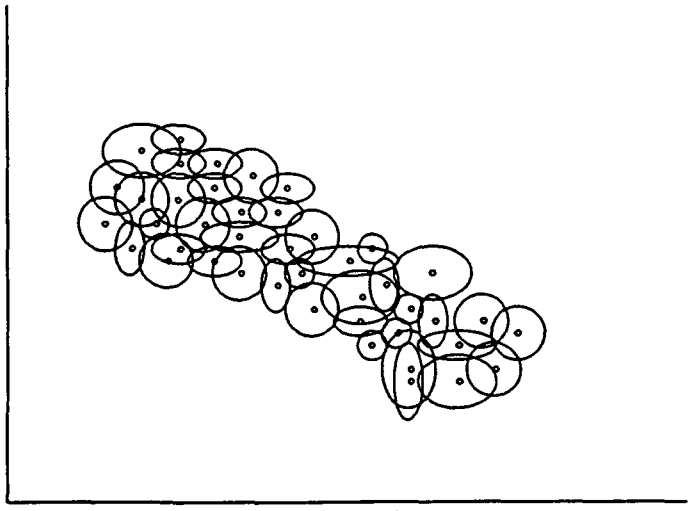


Figure 3: Illustration of Kernels

This presents the problem of deciding from which subdistribution a given data point was sampled. We solve this problem by grouping the data into clusters which represent the subdistributions. A cluster is a collection or set of data points. Each data point is defined as a set of coordinates in a multidimensional Euclidean space [7]. The Euclidean space defines the possible operating properties that may be held by the device the data represents. Data points are assigned to clusters according to which cluster they are closest to.

Below we will briefly describe the methods used for clustering. A more detailed examination is given in Campbell [4] in which we examined the methods of forming groups of clusters from a given set of data and explain our choice of clustering technique. We will examine the various methods for measuring the distance between two clusters in order for the clustering methods to determine the best possible grouping of data points to form clusters. Then we will use an example to describe how the finite mixture distributions are constructed from the clusters.

4.1 Clustering

All methods for clustering data decide which cluster a data point belongs to by the distance between data points. Clustering may be thought of as the process of joining two smaller clusters to form larger clusters, the simplest example being that of forming a cluster from two data points (each of which may be thought of as a cluster of one). How points are chosen to be members of different clusters is what distinguishes the different cluster distance measures.

To find the most compact clusters, the best cluster merging method is complete linkage. In complete linkage, the distance between two clusters is measured by the longest distance between any two points in the two clusters. The two clusters in the data set which have the shortest complete linkage distance are joined to make a larger cluster. Complete linkage tends to find very tight clusters [8]. Cluster distance measuring techniques are the basis for the cluster forming methods which are discussed next.

Cluster analysis is a technique for finding grouping patterns in data [1, 14, 8]. There are a number of clustering techniques, but they are mostly variations on the following techniques. There are two hierarchical clustering methods which produce a hierarchical tree of clusters. The most commonly used techniques are listed below.

- Agglomerative hierarchical clustering
- Divisive hierarchical clustering
- Nonhierarchical K-means clustering

The agglomerative clustering method is the most desirable method since it is efficient and merges outlyers only at the top of the clustering hierarchy. Agglomerative techniques work by starting with all the data points as separate clusters, finding the clusters that are closest together, and merging them one at a time. Ultimately, there is only one cluster. The user of the program must decide by his own criteria how many clusters are desired. Thus the chosen method for finding a Gaussian cluster from a finite mixture is agglomerative clustering using complete linkage.

4.2 Cluster-Based Density Estimation

Next we will discuss how finite mixtures are reconstructed using the clustering-based methods. We will do this first using a one-dimensional example which will illustrate the basic method. Then we will show how new simulated data points can be generated from a finite mixture. We will also discuss possible variations to the approach including a method for efficiently storing the finite mixture.

For the example, if seven data points are chosen from a PDF (Figure 4a) at random as illustrated in Figure 4b. These points are labeled $X_1, X_2, ..., X_7$, and are defined by their position values. The next step, as shown in Figure 4c, is to identify the clusters. For this example the data will be grouped into 4 clusters. For the one-dimensional case, a cluster is the average of the values of data points in the cluster.

Definition 5 *1-d Cluster*

$$c_j = \frac{1}{n_j} \Sigma_{i=1}^{n_j} X_i^j$$

For the one-dimensional case, the Gaussian kernel PDF (f_k below) for a cluster is centered at the average of the cluster, and the variance of the Gaussian kernel PDF is proportional to the variance of the data points in the cluster. This is illustrated in Figure 4d. The basic technique is to cluster the data, then model each cluster by a kernel density. Because the clusters can contain different numbers of data points, the kernels will be stored with numbers (π_j below) indicating the proportion of the total number of points each kernel's cluster contained. When points are generated from the kernels, each kernel is allotted a generated point with a probability that is proportional to the number of points in the kernels divided by the total number of points in the original data set. The density estimate for the data is shown in Figure 4e and in the equation below.

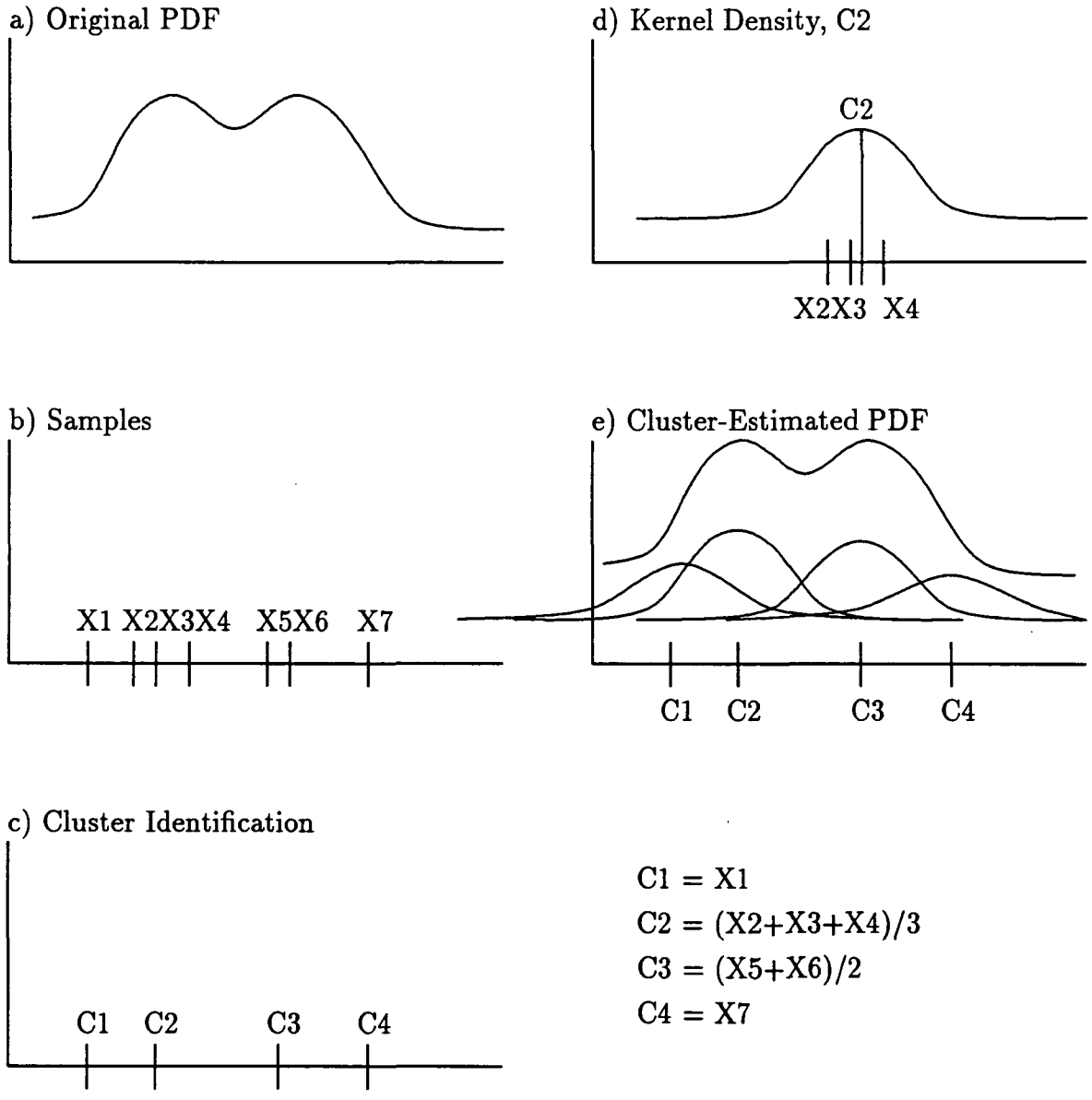


Figure 4: Cluster-Based Density Estimation

Definition 6 *1-d Example of Cluster-Based Density Estimate*

$$\hat{F}(x) = \Sigma_{j=1}^{n} \pi_j f_k(x - c_j)$$

In the multidimensional case, there are a number of additional considerations. The coordinates of a cluster kernel are found by the geometric average of the data points in the cluster. The cluster kernel PDFs may then be correlated to the data by using the square-root method [12]. The square-root method uses a square root of the correlation matrix of the cluster data points to correlate vectors generated from the kernel PDFs. The required matrix square root is computed using the Cholesky decomposition [17].

One of the problems in the simulation of circuits is *representing* the distribution of parameters for the devices of a system. The Truth Model [10] proposes to use measurements of actual devices as the data to model their parameter distribution. This has the problem of requiring a considerable amount of storage for the data. In order to reduce the required amount of data stored for each kernel, the kernels may be uncorrelated Gaussian distributions with the variance in each dimension proportional to the cluster data points. This is a considerable storage savings since for correlated kernels the entire correlation matrix must be stored. The resulting model requires far less storage than the Truth model [4].

5 Summary

In this paper, we examined the existing density modeling techniques available, and we gave the details of the density-estimation techniques developed in this work for modeling the data. We considered the assumptions that can be made about the nature of the data to be modeled, and it was assumed that the data PDF is a finite mixture of multivariate Gaussian distributions. In addition, the data were assumed to be time invariant [2] over the time period of its use. We introduced two density estimation techniques to model this data:

1. Extended Variable Kernel Density Estimation.
2. Cluster-based method.

There is a relation between the two density estimation techniques presented here. The extended variable kernel density estimation technique is simply the cluster-based method with a cluster size of one. Together they constitute the statistical interpolative GaAs FET models.

6 Acknowledgements

This research was supported in part by NASA under Space Engineering Research Grant NAGW-1406 and by the NSF under Research Initiation Grant MIP-9109618.

References

[1] M. Anderberg, *Cluster Analysis for Applications*, Academic Press 1973.

[2] P. Becker, and F. Jensen, *Design of Systems and Circuits for Maximum Reliability and Yield*, McGraw Hill 1977.

[3] L. Breiman, W. Meisel, and E. Purcell, "Variable Kernel Estimates of Multivariate Densities," Technometrics, vol 19, no. 2, May 1977, 135-144.

[4] L. Campbell, *Interpolative Modeling for GaAs Simulation*, Ph.D. Dissertation, University of Idaho 1992.

[5] L. Campbell, J. Purviance, and C. Potratz, "Statistical Interpolation of FET Data Base Measurements," Proc. of the 1991 IEEE MTT-S International Microwave Symposium, Boston MA, June 1991.

[6] J. Friedman, F. Bashett and L. Shustek, "An Algorithm for Finding Nearest Neighbors," IEEE Trans. on Computers, Oct 1975, 1000-1006.

[7] K. Hoffman, *Analysis in Euclidean Space*, Prentice Hall 1975.

[8] L. Kaufman and P. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*, Wiley 1990.

[9] J. Purviance and M. Meehan, "CAD for Statistical Analysis and Design of Microwave Circuits," International Journal of Microwave and Millimeter-Wave Computer-Aided Engineering, vol 1, no. 1, 1991, 59-76.

[10] J. Purviance, M. Meehan, and D. Collins, "Properties of FET Statistical Data Bases," Proc. of IEEE MTT-S International Microwave Symposium, May 1990.

[11] J. Raategver and R. Duin, "On the Variable Kernel Model for Multivariate Nonparametric Density Estimation," COMPSTAT 78 (eds L. Corsten and J. Hermans), Physics Verlag 1978.

[12] E. Scheuer and D. Stoller, "On Generation of Normal Random Vectors," Technometrics, May 1962.

[13] B. Silverman, "Density Estimation for Statistics and Data Analysis," Chapman and Hall 1986.

[14] H. Spath, *Cluster Analysis Algorithms for Data Reduction and Classification of Objects*, Ellis Horwood 1980.

[15] R. Spence and R. Soin, *Tolerance Design of Electronic Circuits*, Addison-Welsey 1988.

[16] D. Titterington, A. Smith, and U. Makov, *Statistical Analysis of Finite Mixture Distributions*, Wiley 1985.

[17] J.H. Wilkinson and C. Reinsch, *Linear Algebra*, Springer-Verlag Berlin, Heidelberg 1971.

Teaching VLSI in High School CHIPS Cirriculum Project

L. Volkening

Moscow High School
Moscow, Idaho 83843

Abstract - **Current educational reform methods do not allow for students to become true partners of applications technology, especially in the context of microelectronics. Students are rarely exposed to the fundatmentals of digital logic design.**

1 The Discussion of the Problem

One main goal of educational reform is to infuse into curriculums problem solving logic from conceptualization to solution. Yet, many educational reforms are versions of the 'updating' concept. Problem solving methods are being taught in the frame of mind that hands-on use of microelectronics will naturally develop higher thinking skills. Students and teachers are taught how to use microelectronics without knowing why microelectronics works or how logic is applied in designing a microcircuit. And as long as there is room for one error in analyzing a problem, then the whole process of conceptualizing and solving a problem becomes invalid.

Higher thinking skills should involve the teaching of students to become visionary thinkers, creative problem solvers, logical analyzers of knowledge, creators of new knowledge and individuals able to appreciate applications technology. If students and teachers are to fully realize these goals, they must become part of the process of technology beyond the application of a black box. They must not be held hostage to the directives of a microcircuit black box or a computer interface. This major concern still exits in regards to the argument of teaching math using calculators.

2 Analyzing the Problem

The problem is the result of years of allowing the education process to lag far behind technology applications process. Dependence on solving problems has become so far ingrained in the black box mentality, logical reasonable thinking skills have become a lost art in many curriculums and at many age levels (K-12).

The education process needs a quantum jump into the 1990's just to begin teaching students methods to use to meet the needs of the next 20+ years. The jump will require three questions to be answered.

1. How does an educator bridge the gap between yesterday's technology applications and today's technology applications, especially in areas that are seldom referenced in textbooks or typical lab activities?

2. How does an educator acquire current equipment, materials, and training?

3. How does the classroom become a place where the materials, methods, and instructor are constantly revitalized to new concepts at the same pace as technology is being applied to the world outside the classroom?

These questions will never be resolved until the rates of educational change match the rates at which technology is applied to solve problems beyond the classroom. And, if the two rates continue to diverge at an *increasing* rate, the technology gap may never be closed. If teachers and students are to appreciate the dreams, desires and goals of mankind in general, changes must be made and they must be made in a manner to ensure continuity.

3 Developing a Solution to the Problem

The methods and logic used in microelectronics research and design can act as an ideal host vehicle to better understand conventional basics as well as implement new concepts and new technologies into the curriculum. The process needs not to be age level based either. Logical approaches to problem solving can be applied through out the total K-12 curricula.

This is a systems approach to education. Inherent in this approach a concept of revitalization becomes a natural outcome. A 'living system' develops where old concepts are explored and utilized via the logic of microelectronics applications. Example, a new form of student involvement in auto mechanics is evolving. Through redesigning computer chips applications, it is possible to alter automotive performance. Years ago, similar classroom content evolved based on the operation of carburetors.

3.1 The Problem Solution

Teachers and students must be made to feel it is an natural to design a microelectronic device to help solve a problem as it is to pick up a pencil or search out a formula. Specific examples of the basic fundamentals of microelectronics as listed below must fit into the curriculum and be applied as needed towards the solving of problems.

Digital Notation
Gates and Transistor Logic Sequence
Chemistry Technology Considerations
Quantum Level Considerations
Clocking and Timed Event Applications
Computer Simulation Programming
CAD Technology
Fabrication Methods
System or Chip Uses
Data Interpretation
Scientific Methods Applications
Truth Tables
Beginning Basic Bread Boarding
Key Debouncing (Debouncers)
Pulse Train Analysis

3.2 Implementation of the Problem Solution

Applications of the fundamentals listed above might involve the use of the following concepts. And if looked at closely, many of these concepts do not require new educational methods or major educational reforms to be understood.

- Transistor logic
- Logic design elements (NAND to NOR Gates)
- Logic skills development through games and riddles
- Logic skills applications to calculator uses (shortening steps in calculator math)
- Logic skills applications in estimation techniques
- Inquiry based digital logic labs
- Boolean algorithms to explain series and parallel circuits
- Math curve fitting abilities to develop pattern recognition skills
- LEGO type interfacing to the computer
- Data Analysis (Rapid scan IR, UV/VIS, NMR, Field Theory, etc.)
- Recognition of 'Hubble' on-line information systems.
- Square Wave Generation/Key Debouncers and the like
- Pulse Train Analyzers' can be the result of truth table analysis and/or bar code analysis

As this approach of integration into existing curricula continues, students will graduate with skills capable to solve problems without being hostage to the black box mentality. Additionally, they will be more likely to become sincere supporters of an education that requires the design, development and application of logic design elements in classrooms.

3.3 Problem Solution Applications

The job of the teacher becomes one of facilitator to help the student tie the fundamentals of digital logic design to application concepts which would evolve into educational methods not yet designed. Some examples might be:

1. Boolean algebra can be used to develop fundamental circuit analysis and logic sequencing.
2. Chips could be designed to help a student understand the analysis of optics from interference patterns to bar codes.

3. Analog to digital conversions can be used to help students develop a better understanding of the concepts of acceleration and gravity.

4. Rotational dynamics can be better visualized if a chip were designed to look at clock pulse rates.

5. FET technology is an exceptional technology for helping the student understand and conceptualize the problems inherent in the study of field theory.

6. Quantum theory concepts are easily illustrated through the use of explaining the chemistry of the production of a microcircuit.

7. A scheme could be designed for a student to visualize and appreciate a work of art through the eyes of an artist and through the eyes of a scientist.

8. Truth table logic techniques could be used to help make informative decisions.

9. Data analysis and pattern recognition are natural processes in the analysis of clock pulses.

10. Student designed I.C.s' can become a focal point for those willing to take the next step.

3.4 Problem Solution Implementation Bonuses

Bonuses from integrating fundamental logic design elements into existing curricula would be:

1. Use of microelectronics helps the student prepare for the future with a clear vision.

2. The integration of math and science technology becomes routine in the classroom.

3. Each strand of a curriculum can be strengthened with the threads of microelectronics applications. (Microelectronics applications can be used to help unify various aspects of the curriculum.)

4. The student will gain self confidence as the course progresses, since common concepts can be explained using higher thinking skills.

5. Becoming a better citizen can be realized since microelectronic circuit designing techniques require team efforts.

6. Logic skills can easily be transferred and applied towards understanding other curricula.

7. Full problem conceptualization occurs as students use hands on activities developing critical thinking skills.

4 How to Make the Solution to the Problem Work

The solution to the problem could easily be achieved by setting-up a resource of highly trained individuals which will readily share high level technical information at a level easily integrated into the classroom. This resource of teachers could also become liaisons for sharing and helping concepts be developed into educational methods. In addition, this resource of teachers could also develop a resource of students that other students could identify with in their attempts to understand the problem solving process as well as applications of microchip technologies. Digital logic designs might also be shared through various sites and schools as students realize they could become part of a team beyond the classroom.

5 Summary

Classrooms should be places where students are taught to look ahead using visionary and realistic mentalities. Classrooms should also be places where the problem solving process is taught from the starting point of *conceptualizing* a problem to understanding how to move through the steps of *solving* a problem. Microelectronics logic is unique in that the same inherent skills are required by the circuit designer when developing a circuit for application to a particular problem or when developing a circuit that allows a user to analyze a problem. These skills are also unique since they can readily be adaptable to most areas of physics, math, other sciences, as well as cross curricula.

6 A Starting Point

Phase Three Teachers of the Microelectronics Research Center at the NASA Institute for Secondary Science and Math Teachers met for three weeks in the Summer of 1992. The group decided to pool various ideas to act as a continuation of the starting points developed during phases I and II. A 10 day plan was developed to act as a common starting point for the Fall of 1992. This was agreed to in order to have in the future a 'base line' for reference in further curriculum development work. Depending on equipment, the group would be delivering the same background materials at the same time in the year. Teachers could then be free to develop throughout the year units integrating microelectronic technology into the other basics offered in the course. When funding becomes available the teachers plan to meet again during the year to share success and failure stories.

In addition, this group shared ideas and concerns with the Phase 1 and Phase 2 teachers also meeting at the institute during the Summer. As a result, several new curriculum content materials were discovered. Phase 1 and Phase 2 teachers requested to continually be updated as the project grows. The 10 day plan was also shared.

At this time , a task force of highly trained teachers at various teaching levels, backgrounds, regional sites and work loads are involved at implementing curriculum that has its roots in the fundamentals of digital logic design. It is expected that many more will also benefit as ideas are shared.

Session 8
VLSI Design

Chairman: Jody Gambles

Bit-Systolic Arithmetic Arrays Using Dynamic Differential Gallium Arsenide Circuits

Grant Beagles
Texas Instruments
Lewisville TX 75067

Kel Winters
Advanced Hardware Architectures
Moscow ID 83843

A. G. Eldin
University of Toledo
Toledo OH 43606

Abstract - **A new family of gallium arsenide circuits for fine grained bit-systolic arithmetic arrays is introduced. This scheme combines features of two recent techniques of dynamic gallium arsenide FET logic and differential dynamic single-clock CMOS logic. The resulting circuits are fast and compact, with tightly constrained series FET propagation paths, low fanout, no dc power dissipation, and depletion FET implementation without level shifting diodes.**

1 Introduction

The advantages of parallel arrays of serial arithmetic processing modules have been recognized since the 1950s. These arrays were initially seen as a way to speed up signal and information processing. With the advances in VLSI technology, it is now possible to easily realize these highly modular architectures.

A number of bit-systolic arithmetic arrays have been developed with the intent of maximizing the clock rate for a given CMOS process, including a 2-D convolver for image processing, an integer polynomial solver and, a finite-field polynomial solver. These arrays were modeled using SPICE in a 2-micron (minimum feature size) CMOS process offered through MOSIS. Even using this relatively inexpensive low-performance process and worst case models, cells were reliably modeled well in excess of 100MHz [9,13].

A new dynamic gallium arsenide logic circuit family was proposed in 1991 [1]. These circuits, resembling GaAs D-RAM circuits, have several advantages over previously used dynamic GaAs logic circuits. Prior dynamic GaAs circuits use both depletion and enhancement mode devices on the same die and typical process yield is about 50%. A very major advantage of the new family of dynamic circuits used for this project is that only depletion mode devices are required to realize any function. Using only one type of device should give significantly higher process yields.

A second advantage is the absence of any DC power supply. This characteristic completely eliminates the DC power dissipation problems that have severely limited the use of dynamic GaAs logic circuits. This characteristic is not without its downside, however, since the clock driver must be hefty enough to rapidly charge the capacitors used in the circuits and clock skew is critical.

Another characteristic which sets these circuits apart from other dynamic GaAs circuits is their regularity and lack of level shifting diodes, a messy requirement of previous dynamic GaAs families. These circuits may be very compact and possess the traits desirable in systolic architectures.

The systolic arithmetic cells described earlier were redesigned using this new dynamic GaAs logic. The new circuits were modeled using P-SPICE and process parameters charac-

teristic of the Vitesse depletion mode GaAsFET. The models show an order of magnitude improvement in clock speed when compared to the CMOS cells described by Winters et.al.

2 Dynamic GaAs Circuits

The complexity of GaAs FET VLSI circuits is limited by the maximum power dissipation while the uniformity of the device parameters determines the functional yield. For a given process yield, the functional yield can vary significantly. The variation is due, in part, to the use of different circuit structures that may be operated in either dynamic or static modes. Also, the sensitivity of the proper functionality to variations in the process parameters is highly dependent on the selection of the circuit structure and the mode of operation.

Static GaAs circuits are ratioed circuits. This is one of the reasons why their functionality and speed of operation are strongly dependent on the variations in the device threshold voltage. The fundamental requirement for dynamic circuits is that the ratio of the device current in its ON state to that in the OFF state should be sufficiently large (several orders of magnitude). This basic requirement for dynamic logic circuits can be met using GaAs circuits.

Dynamic circuits are ratioless, which makes their functionality completely insensitive to threshold voltage variations. The dynamic GaAs circuit's speed is also significantly less affected by these variations than in static logic designs.

The family of dynamic GaAs circuits used for the designs in this project do not dissipate DC power. Dynamic circuits have smaller device counts when compared to static circuits having similar functionality. These features are very attractive for the implementation of ultra high speed VLSI architectures as demonstrated in this paper.

The evolution of the new GaAs dynamic circuits [1], used in the implementation of the systolic arithmetic arrays is an extension of the ideas employed in the JCMOS DRAM Cell and the related work on BiCMOS dynamic memory and logic structures [10]-[12].

The operation of the new circuits can be easily explained using the diagrams in Figure 1. An intermediate stage of a dynamic shift register and its idealized functionality are illustrated.

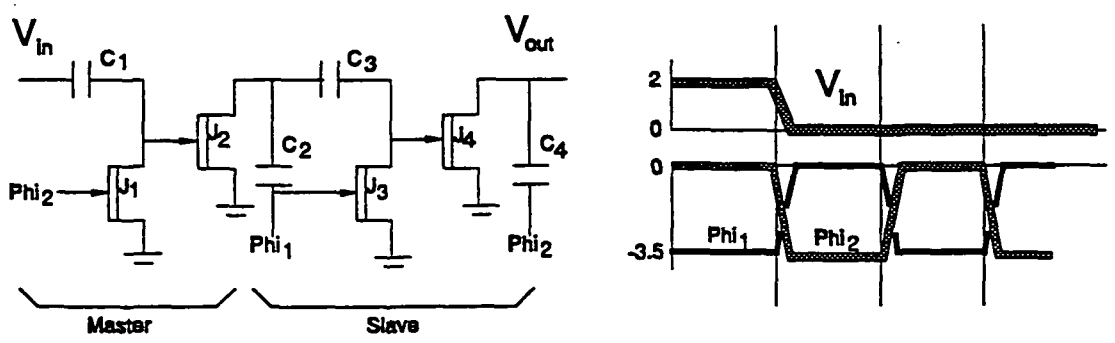


Figure 1: The D-type flip-flop uses depletion mode transistors having a threshold voltage of -0.7 volts.

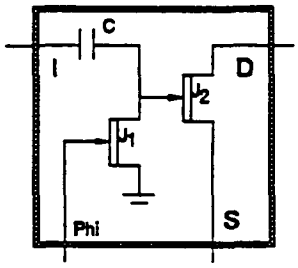
During T_1, $Phi_2 = 0$ volts. The master section is in the sample phase. The input data is stored on C_1. C_1 is charged to 2 volts or discharged to 0 volts for a logical 1 or 0 respectively. C_2 is precharged to approximately 3.5 volts through $J_2(Phi_1 = -3.5$ volts).

The slave section is in the evaluation phase. Transistor J_3 is in cut off and the drain voltage of J_2, $V_{D2} = 0$ volts, thus providing a reference voltage for evaluating the data stored on C_3. If C_3 is charged, J_4 is turned off and the precharged capacitor C_4 retains its voltage (logic 1). However, if C_3 is discharged, J_4 is turned on and C_4 is discharged to represent a logic 0. During T_2, the roles of the master and slave sections are interchanged.

Simulations of this circuit using a device model which accounts for second order side effects and is accurately calibrated to a 1 micron HFET process, verifies the operation of the D-Type flip-flop at 2GHz [1]. A comparison with DCFL implementations show that the dynamic circuit requires 30% less area and dissipates only 10% of the power (dynamic only) of the DCFL flip-flop.

The basic dynamic circuit can also be used to implement the AND, OR and complex logic functions. The operation of the basic circuit is similar to that of the dynamic flip flop.

Fig. 2 summarizes the operation. When the input is logic 0, the capacitor C is discharged during the sampling phase and J_2 will turn on during the evaluation phase. Similarly, if the input is a logical 1, the capacitor is charged during the sampling phase causing J_2 to turn off during the evaluation phase.



I	C	J_2
0	DISCHARGED	ON
1	CHARGED	OFF

Figure 2: Basic dynamic logic circuit

Figure 3 shows a complex dynamic logic gate.

Bit-systolic systems are defined here as synchronous digital systems whose combinatorial timing paths involve the computation of no more than one bit of data. Moreover, these systems are constrained to be locally connected, that is, modules at each level of hierarchy are connected exclusively to their nearest neighbors in the physical artwork. Therefore, bit-systolic systems are very fined grained pipelined architectures with tightly restricted fanout and interconnect capacitance. Such systems may also be said to be bit-extensible, implying that a computational word width may be extended to an arbitrary number of bits without affecting the clock speed (excluding clock and control signal loading).

For example, Figure 4 illustrates a bit-systolic serial-parallel multiply-accumulator introduced in 1990 as a building block for 2-D image convolver and single-instruction multiple data path (SIMD) processor arrays [13]. Here, the multiplier, x, is pipelined through n stages, and the multiplicand, y, is input in parallel form. The maximum fanout in the multiplier pipeline is 2 gate inputs. Summing is performed in an accumulator pipeline consisting of full-adder modules and pipeline delays aligning the product-sum to the multiplier. This

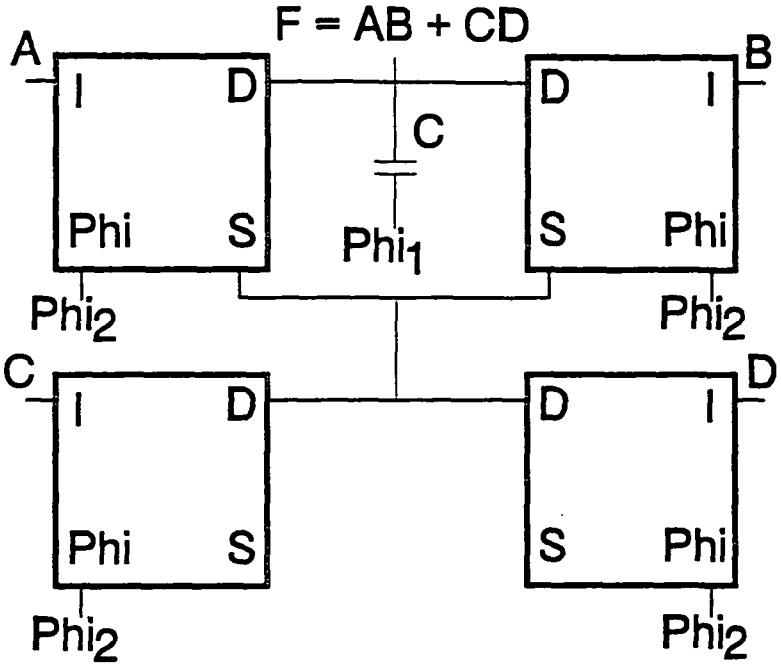


Figure 3: The dynamic complex logic gate

systolic multiplier requires 2n clock cycles to multiply two n-bit unsigned integers. The least significant bit of the product, x0y0, appears at the output n clock cycles into the multiplication sequence. The module may be easily extended to accommodate signed inputs. Operation of the bit-systolic multiplier is illustrated in Table 1.

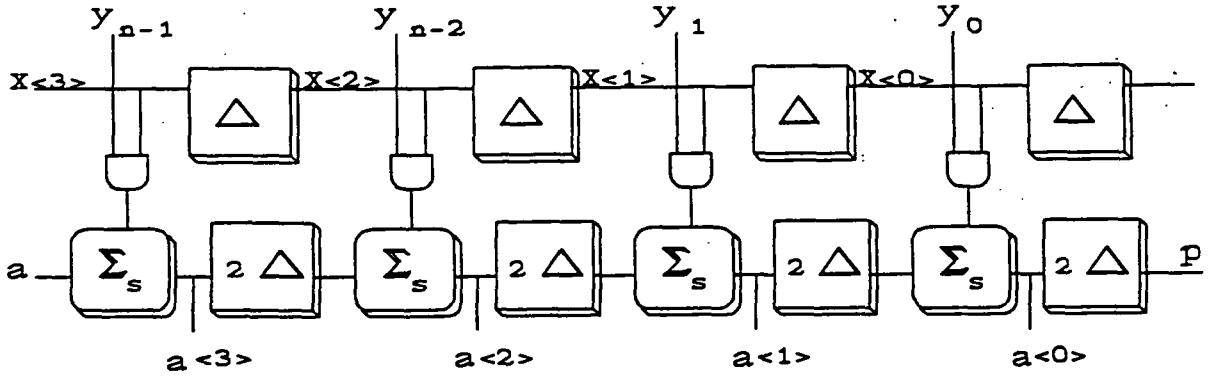


Figure 4: Bit-systolic serial-parallel multiplier

The useful property of this configuration is that it contains n bits of storage for the multiplier and $2n$ bits for the product. An array of these modules would have the proper ratio of operand versus product storage. For instance, a product sum could be accumulated by a single multiplier whose output is fed back to its addend input in $2n$ clock cycles per multiplication.

The product pipeline can accumulate external addends with its partial products without additional adder logic. The external addend is shifted serially into the a input and must be pre-shifted n bits into the product pipeline before the LSB of the multiplier is entered. Thus, the lower n bits of the addend occupy the high n bits of the product pipeline at the beginning of a multiply sequence. Then, n multiplier bits are shifted into the multiplier

Clock Cycle	y3	y2	y1	y0
	x<3>	x<2>	x<1>	x<0>
	a<3>	a<2>	a<1>	a<0>
1	x0			
	x0y3			
2	x1	x0		
	x1y3	x0y2		
3	x2	x1	x0	
	x2y3	x0y3+x1y2	x0y1	
4	x3	x2	x1	x0
	x3y3	x1y3+x2y2	x0y2+x1y1	x0y0
5	0	x3	x2	x1
		x2y3+x3y2	x0y3+x1y2+x2y1	x0y1+x1y0
6	0	0	x3	x2
		x3y3	x1y3+x2y2+x3y1	x0y2+x1y1+x2y0
7	0	0	0	x3
			x2y3+x3y2	x0y3+x1y2+x2y1 +x3y0
8	0	0	0	0
			x3y3	x1y3+x2y2+x3y1
9	x0	0	0	0
	x0y3			x2y3+x3y2
10	x1	x0	0	0
	x1y3	x0y2		x3y3
11	x2	x1	x0	0
	x2y3	x0y3+x1y2	x0y1	

Table 1: Bit-systolic multiplier operation

pipeline, leaving the LSB of the accumulated product in the LSB of the product pipeline. During the next n clock cycles, the multiplier is shifted out (replaced by zeroes), the low n bits of the product are shifted out of the product pipeline, the high n bits of the product are left in the low half of the product pipeline, while the low n bits of the next addend are pre-shifted into the high product pipeline half. This is illustrated in Table 2 for a four bit Systolic S-P multiplier. Once again, the shaded rows represent the multiplier pipeline. The unshaded rows represent the accumulator pipeline, where only the contents of the first flip-flops in each bit-cell are shown.

Clock Cycle	y3	y2	y1	y0
	x<3>	x<2>	x<1>	x<0>
	a<3>	a<2>	a<1>	a<0>
1	x0			
	a3+x0y3	a1		
2	x1	x0		
	a4+x1y3	a2+x0y2	a0	
3	x2	x1	x0	
	a5+x2y3	a3+x0y3+x1y2	a1+x0y1	
4	x3	x2	x1	x0
	a6+x3y3	a4+x1y3+x2y2	a2+x0y2+x1y1	a0+x0y0
5	0	x3	x2	x1
	a7	a5+x2y3+x3y2	a3+x0y3+x1y2+x2y1	a1+x0y1+x1y0
6	0	0	x3	x2
	a0	a6+x3y3	a4+x1y3+x2y2+x3y1	a2+x0y2+x1y1+x2y0
7	0	0	0	x3
	a1	a7	a5+x2y3+x3y2	a3+x0y3+x1y2+x2y1 +x3y0
8	0	0	0	0
	a2	a0	a6+x3y3	a4+x1y3+x2y2+x3y1
9	x0	0	0	0
	a3+x0y3	a1	a7	a5+x2y3+x3y2

Table 2: Multiply/Accumulate Operation

Each bit-cell of the multiplier module may be constructed from four basic cell types: XOR, Carry, Latch, and Product, which will be described later. In the CMOS version, these were implemented in differential dynamic circuits that, like the array architecture, were very constrained in propagation path, fanout, and connectivity. Mapping the basic differential circuit structure to dynamic GaAs circuits was straightforward.

3 GaAs Systolic Cells

The first step in this project was to design the circuits for each cell required. The CMOS circuits were available and their performance characteristics well documented [9]. The CMOS design required 5 cells; a latch, a product cell, a carry cell and both a P-logic and N-logic XOR cell. These cells could then be tiled into arrays with alternating P and N stages. The GaAs cells are composed of only depletion mode devices and so only 4 cells were necessary. Pipelining in the GaAs arrays is accomplished through the two clocks whereas the CMOS relies on the alternating P and N stages with a single clock signal. Figures 5 and 6 show the CMOS circuits and the corresponding GaAs analogs respectively.

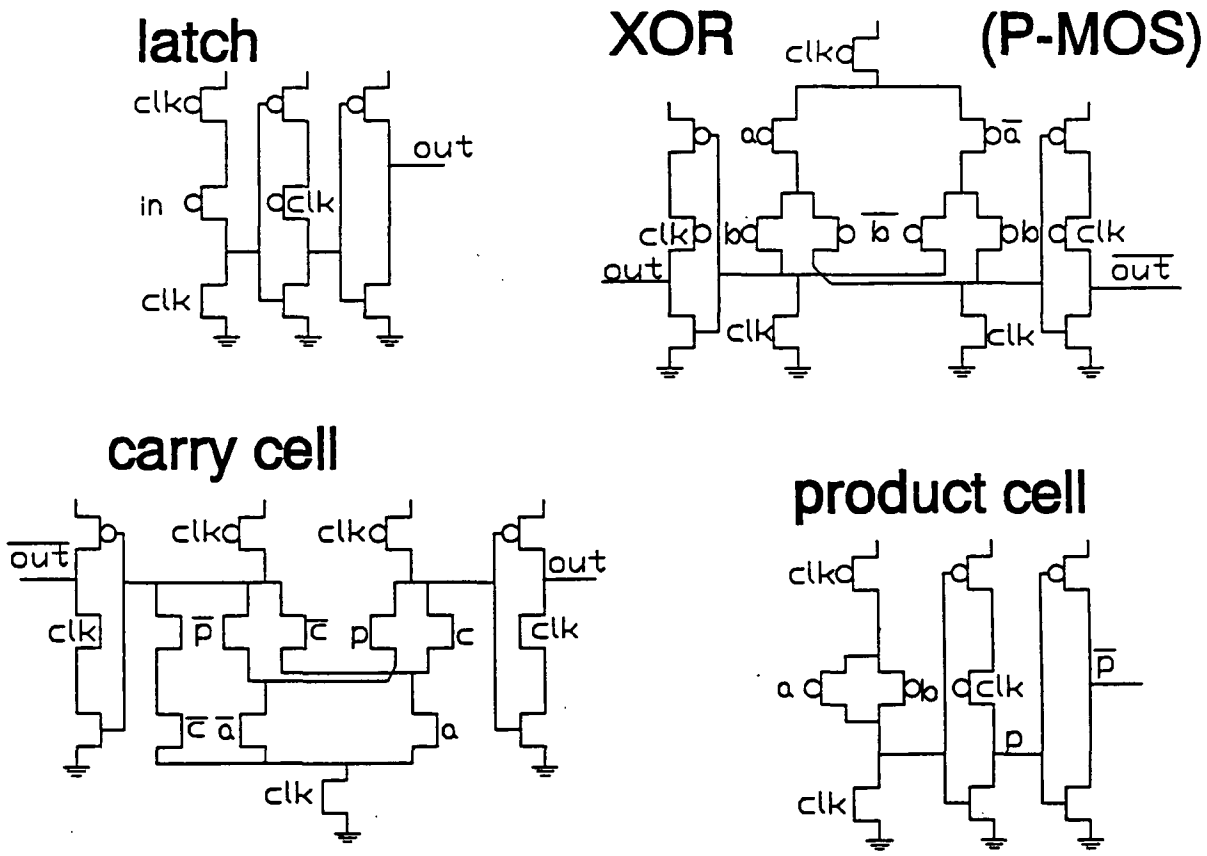


Figure 5: CMOS logic cells for systolic arrays

A quick comparison of the CMOS and GaAs shows that the device count (including capacitors) is quite comparable. The CMOS process offered by MOSIS is a $2\mu m$ (minimum feature size) 2 metal layer process. MOSIS offers a $.7\mu m$, 3 metal layer GaAs process (Vitesse). The smaller feature size along with an additional metal layer should equate to more densely packed circuits and hence reduced parasitic capacitances. Smaller parasitics, coupled with gallium-arsenide's higher charge mobility, mean that the cells should perform at a significantly faster clock speed than the CMOS circuits.

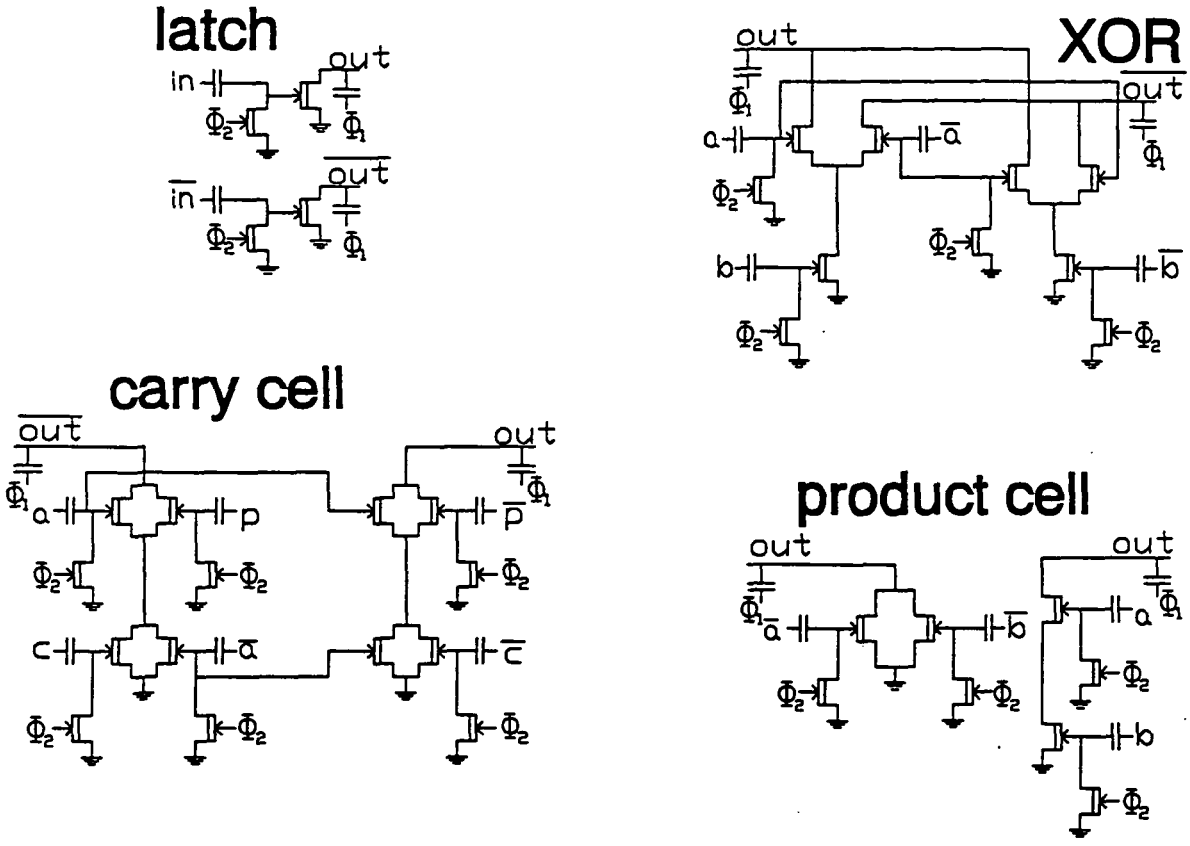


Figure 6: GaAs logic cells for systolic arrays

4 Simulation

The GaAs circuits were simulated using P_SPICE (professional version). Since a circuit layout was not implemented, the parasitic capacitances were conservatively estimated. All of the GaAs circuits were simulated at 1 and 2 GHz and all of the cells showed reliable operation at 2 GHz.

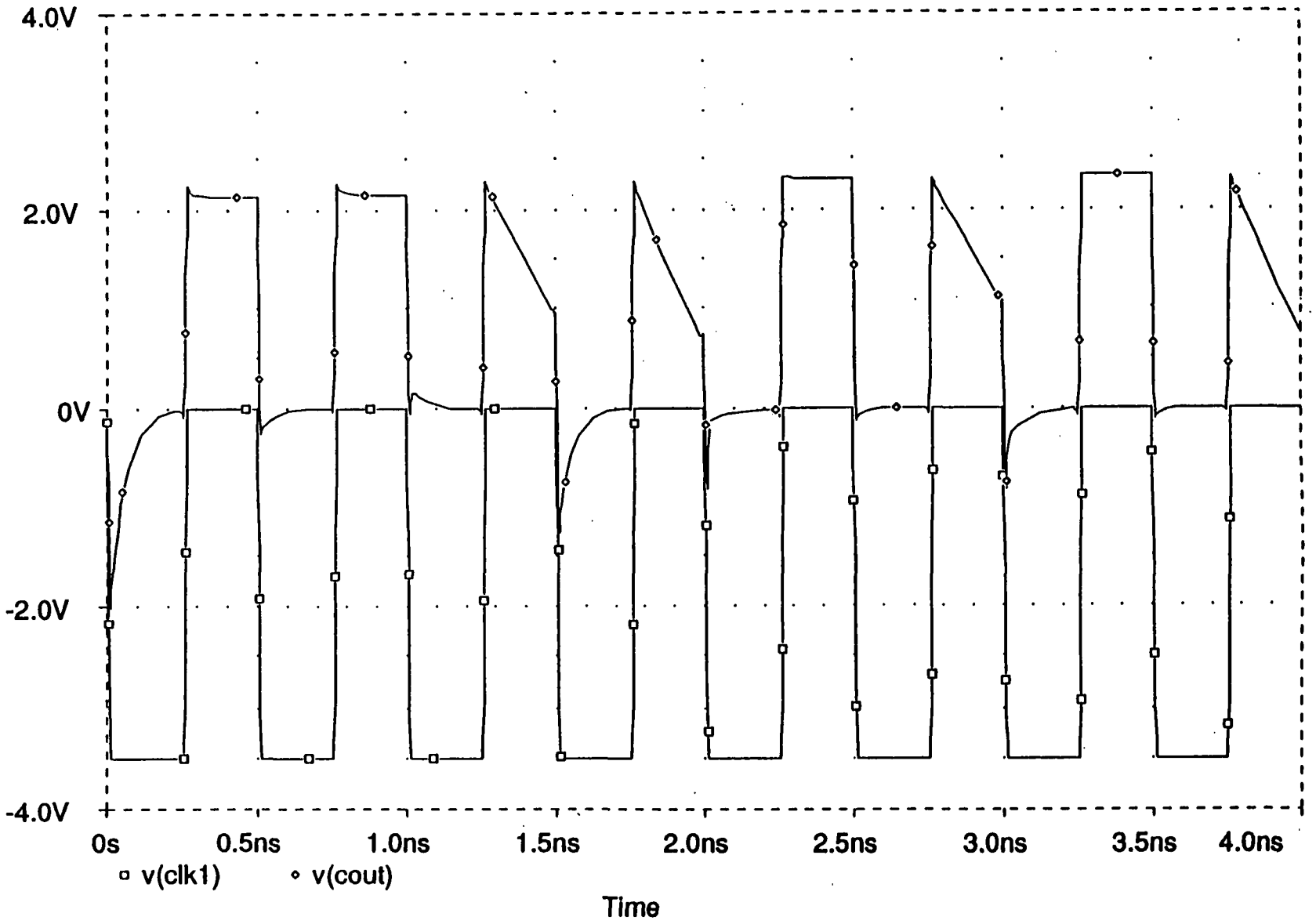
5 Conclusions

The speed of Gallium Arsenide technology implemented in differential dynamic circuits in bit-systolic arrays would enable very high performance solutions to a broad range of problems. The basic cell architectures themselves have been tested and their performance verified in CMOS [9]. The GaAs cells have the same functionality as the CMOS cells, however, since no layout of the circuits has been done, the performance evaluation is preliminary and conservative. It is estimated that, given a good layout , these cells would perform at least 2 times faster than the models used for this paper.

GaAs Carry Cell (2GHz.)

Date/Time run: 05/20/92 14:25:14

Temperature: 27.0



8.1.10

References

[1] A. G. Eldin, , "New Dynamic FET Logic and Serial Memory Circuits for VLSI GaAs Technology", 3rd NASA SERC Symposium on VLSI Design, Moscow, Idaho, 1991.

[2] D. K. Ferry, Gallium Arsenide Technology, Howard Sams & Company, Indianapolis, Indiana, 1985.

[3] N. Kanopoulos, Gallium Arsenide Digital Integrated Circuits A Systems Perspective, Prentice Hall, Englewood Cliffs, New Jersey, 1989.

[4] S. I. Long and S. E. Butner, Gallium Arsenide Digital Integrated Circuit Design, McGraw-Hill Publishing Company, New York, New York, 1990.

[5] C. Mead and L. Conway, Introduction to VLSI Systems, Addison-Wesley Publishing Company, Reading, Massachusetts, 1980.

[6] N. Sclater, Gallium Arsenide IC Technology Principles and Practices, TAB Professional and Reference Books, Blue Ridge Summit, Pennsylvania, 1988.

[7] M. Shur, GaAs Devices and Circuits, Plenum Press, New York, New York, 1987.

[8] P. W. Tuinenga, SPICE A guide to Circuit Simulation & Analysis Using P-Spice, Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[9] K. Winters, D. Mathews, and T. Thompson, "Application Specific Serial Arithmetic Arrays", 2nd NASA SERC Symposium on VLSI Design, Moscow, Idaho, 1990.

[10] A. G. Eldin and M. I. Elmasry, "VLSI Dynamic Memory" United States Patent #4,791,611 Dec 13, 1988.

[11] A. G. Eldin and M. I. Elmasry, "New Dynamic Logic and Memory Circuit Structures For BICMOS Technologies" IEEE Journal of Solid State Circuits, VOL. SC-22, pp 450-453, June 1987.

[12] A. G. Eldin and M. I. Elmasry, "A Novel JCMOS Dynamic RAM Cell For VLSI Memories" IEEE Journal of Solid State Circuits, VOL. SC-15, pp 715-723, June 1985.

[13] K. Winters, "Serial Multiplier Arrays for Parallel Computation," NASA SERC Symposium on VLSI Design, Moscow, Idaho, 1990.

A VLSI Implementation of DCT Using Pass Transistor Technology

S. Kamath, Douglas Lynn and Sterling Whitaker
NASA Space Engineering Research Center
for VLSI System Design
University of Idaho
Moscow, Idaho 83843

Abstract –**A VLSI design for performing the Discrete Cosine Transform (DCT) operation on image blocks of size 16×16 in a real time fashion operating at 34 MHz (worst case) is presented. The process used was Hewlett-Packard's CMOS26—A 3 metal CMOS process with a minimum feature size of $0.75\mu m$. The design is based on Multiply-Accumulate (MAC) cells which make use of a modified Booth recoding algorithm for performing multiplication. The design of these cells is straight forward and the layouts are regular with no complex routing. Two versions of these MAC cells were designed and their layouts completed. Both versions were simulated using SPICE to estimate their performance. One version is slightly faster at the cost of larger silicon area and higher power consumption. An improvement in speed of almost 20% is achieved after several iterations of simulation and re-sizing.**

1 Introduction

Modern image processing techniques find applications in television broadcasting as well as transmission of pictures from space based artificial satellites. Since a single image requires a tremendous amount of storage or communication channel bandwidth for it's transmission, data compression is employed to reduce this requirement. The Discrete Cosine Transform (DCT) [1] is an important step in many data compression algorithms. The DCT is a frequency domain transformation of a time domain signal which performs very close to the statistically optimal transform—the Karhunen-Loeve Transform (KLT).

In the digital VLSI domain, Complementary Metal Oxide Semiconductor (CMOS) technology is well renowned for it's low power consumption when compared to other logic families. Traditional CMOS technology uses both PMOS and NMOS transistors for forming logic and hence the name 'Complementary'. In CMOS pass technology [7, 8, 9, 12, 13] however, NMOS transistors alone are used for forming the logic. Both PMOS and NMOS transistors are employed only in the inverters which serve the dual purpose of inverting logic as well as buffering signals. The inverter buffers are needed to restore the quality of the signal passed by an NMOS transistor since it passes a degraded '1'. The most important reason for using pass transistor technology is the area savings that result when compared to the traditional CMOS [8, 9]. The pass transistor based VLSI layouts are dense and compact and in general most of the chip area tends to be the area devoted to forming the logic itself.

This paper presents the design and VLSI implementation of a major portion of a two dimensional DCT chip proposed to NASA to be incorporated in one of their space based

image compression applications. Since the emphasis is on speed, we have tried to improve the speed of the chip while maintaining power and area requirements within acceptable limits. The design steps are reviewed in the following sections on a block diagram level. The transistor level circuit diagrams are available in [4]. Some of the important layout aspects are mentioned with details and actual layouts being available in [4]. A section on design verification is included with explanations on how an improvement of nearly 20% in speed is achieved without any consequent increase in area. To conclude, the salient features of the chip designed are summarized in a table along with notes on what remains to be done to complete the chip before it can be fabricated.

2 Circuit Design

The design of 2D DCT chip is explained on a block diagram level in the following three subsections. The transistor level circuit diagrams used in the design are available in [4].

2.1 Definition of DCT and overall chip description

The Discrete Cosine Transform (DCT) [1] of a finite data sequence which in this case is the real time image data $a[j]$ of length N is defined as

$$C(u) = \frac{2\ m(u)}{N} \sum_{j=0}^{N-1} a[j]\ cos\left\{\frac{(2j+1)u\pi}{2N}\right\} \quad where\ u = 0, 1, \ldots, (N-1) \qquad (1)$$

$$and\ \ m(u) = \begin{cases} \frac{1}{\sqrt{2}} & if\ u = 0 \\ 1 & if\ 1 \leq u \leq (N-1) \end{cases}$$

The above definition applies to one dimensional (1D) data sequence.

For a two dimensional data sequence, like the two dimensional (2D) image data $a[j,k]$ having the dimensions of $N \times N$, a 2D DCT is defined as

$$C(u,v) = \frac{4\ m(u)\ m(v)}{N^2} \sum_{k=0}^{N-1} \sum_{j=0}^{N-1} a[j,k]\ cos\left\{\frac{(2j+1)u\pi}{2N}\right\}\ cos\left\{\frac{(2k+1)v\pi}{2N}\right\} \qquad (2)$$

$$where\ u = 0, 1, \ldots, (N-1)\ \ \&\ \ v = 0, 1, \ldots, (N-1)$$

Here the values of the scale factors $m(u)$ and $m(v)$ are defined as was $m(u)$ in equation (1).

Assuming the image is divided into blocks of 16×16 pixels, $a[j,k]$ in equation (2) becomes a 16×16 matrix. Equation (2) can be partitioned so that the 2D DCT is performed essentially as two 1D DCTs, the first operating on the rows of the image data and the second on the transpose of the matrix resulting from the 1D DCT operations. Assuming that the pixel data is input to the chip row-wise one pixel (word) at a time, the proposed method performs a 1D DCT on each row of the input data. This takes 16 clock cycles per row assuming data is presented at the rate of one word per clock cycle and assuming that a multiply-accumulate operation can be performed in one clock cycle. The multiplication referred to here is the multiplication of the pixel data with the *cos* terms in equation (2) appropriately scaled.

Since the scaled *cos* terms can be pre-computed they are treated as constant coefficients and stored in memory. For the DCT to be computed in real time, assuming the pixel data is a continuous never-ending stream of data, we need 16 multiplier-accumulators (MACs). Each MAC is responsible for computing one of the 16 transform domain terms using equation (1). After 16×16 cycles, all the row DCTs will be completed. Then the DCTs on the columns can begin. Assuming again that real time operation is required, a second set of 16 MACs is needed. When the column DCTs are being computed, row DCTs can be referred to the second 16×16 image block. This requires a 16×16 buffer memory (called the transposition memory) between the two sets of MACs. Each set of MACs can be configured as either a 4×4 array or an 8×2 array depending on the size of the transposition memory. The overall block diagram of the chip is shown in Figure 1.

The ROM cells alongside each of the MACs are arrays of Read Only Memories with each cell containing an unique column of the coefficient matrix (scaled *cos* terms from Equation 2). The transposition memory performs a transpose operation and provides the transposed 1D DCT output to the second dimension processing MACs one word at a time as shown in Figure 1. Image data enters the chip from the top left one word at a time and the 2D DCT output from the chip leaves the chip one word at a time through the shift register at the bottom right. The shift register is a parallel input (word) serial output (word) register.

The multiply operation for the DCT is accomplished in the MAC cell by using modified Booth recoding. The function of the MAC can be expressed mathematically by the following recursive equation.

$$c[i] = a[i] \cdot b[i] + c[i-1] \quad for\ i = 0, 1, \ldots, 15 \tag{3}$$

Here the $a[i]$ input is the image data and $b[i]$ input is the scaled cos term in equation (1). At the beginning of the process when $i = 0$, $c[i-1]$ is set to 0 with $a[0]$ and $b[0]$ being the first element of the first row and the first column of the image matrix and the coefficient matrix respectively. For the next 15 clock cycles when $i = 1, \ldots, 15$, the c output from the previous multiply-accumulate operation becomes the present c input. Thus, each MAC accumulates the inner product of one row of image data and one column of the coefficient matrix.

This method was chosen mainly due to it's speed and regularity of implementation. The VLSI layouts for two versions of the MAC were completed and both were found to be highly area efficient and neither version required any complex routing. In the proposed design due to accuracy considerations, a 16×16 bit multiplier with accumulate capacity was chosen. Both versions of the multiplier cells are considered in detail in the next subsection.

2.2 Design of Modified Booth Recoded MAC

The process of parallel multiplication of two multi-bit numbers, say A and B, consists of multiplying all the bits of A by each of the bits of B starting from the Least Significant Bit (LSB) and shifting the resulting partial products one bit to the left for every operation before performing the final addition of the properly shifted partial products to form the final answer. The modified Booth algorithm reduces the number of partial products that must be added together, thereby speeding up the computation [2] by recoding the bits of B. The exact magnitude of the speed up is a function of the number of bits the algorithm considers

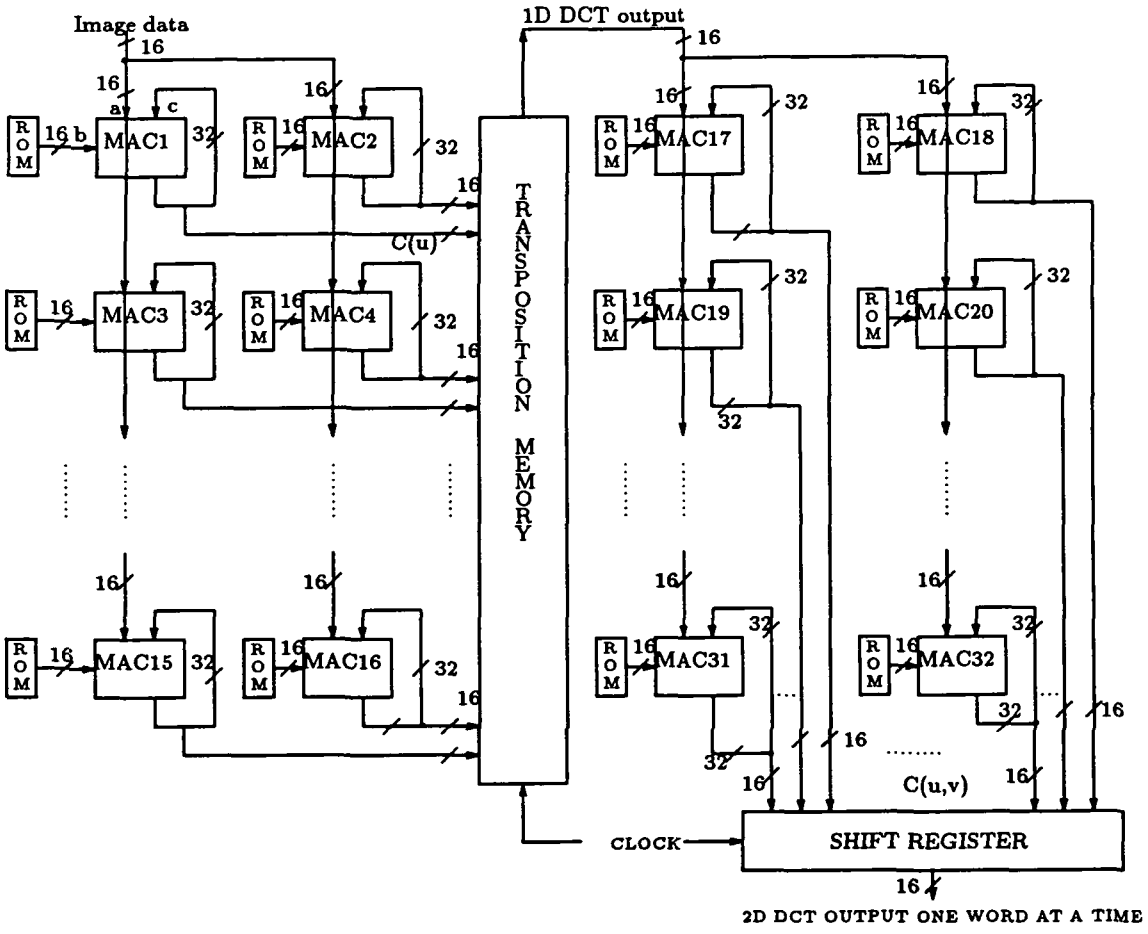


Figure 1: Overall block diagram of the 2D DCT chip

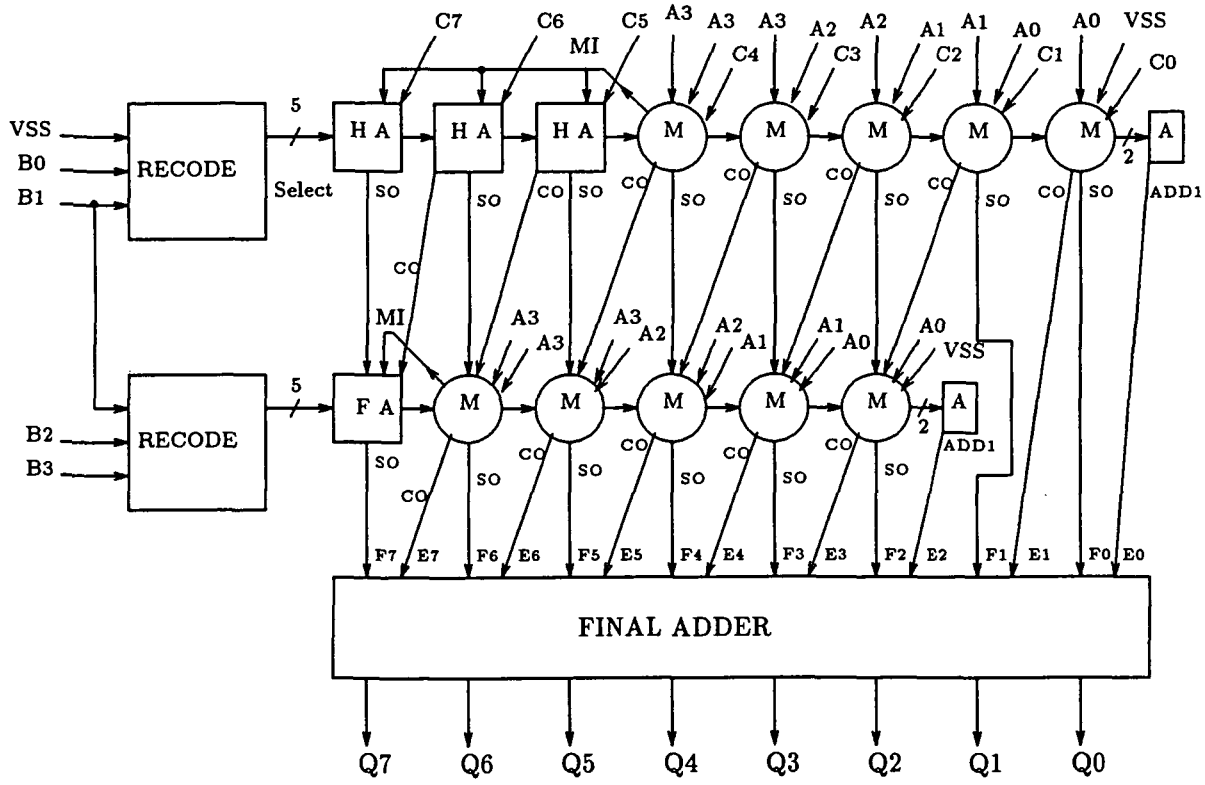


Figure 2: Block diagram of a 4 × 4 bit Booth MAC

in each step. In this design, a bit-pair recoding is employed as a result of which the number of rows of partial products to be added together is cut in half thereby yielding a speed up factor of nearly 50%. The recoded digits take on values ranging from -2 to +2 depending on the combination of three bits of B including one overlap bit for every step. The details about this recoding scheme are available in [2] and design steps for it's implementation in pass transistor technology are summarized in [4].

The block diagram of 4 × 4 bit Booth MAC is shown in Figure 2. Since the recoding algorithm requires that the partial products be shifted 2 positions to the left for every row of partial product, the array takes a trapezoidal shape. The circular blocks labeled M are the multiplier cells, the HA blocks are simple Half-Adders, the FA block shown is a Full-Adder, the blocks denoted as A are the Add 1 cells and finally the RECODE blocks are the recoding circuits for the B operand.

The multiplier cell is responsible for selecting the appropriate partial product bit and adding this to two other bits namely, the Sum-In (SI) and the Carry-In (CI) as shown in Figure 3. The SI and the CI are the Sum-Out (SO) and Carry-Out (CO) signals from appropriate cells of the row above the multiplier cell in question. The partial product bits presented to each multiplier cell are A_i (A) and A_{i-1} (AM1) with their complements being provided by inverters within the multiplier cell. The outputs of the multiplier cell namely, the SO and the CO signals become the SI and CI inputs to the appropriate cells in the row below it. This carry save technique [11], is depicted in Figure 2. The multiplier cells in the top row contain Half-Adders where as the multiplier cells in all the other rows use

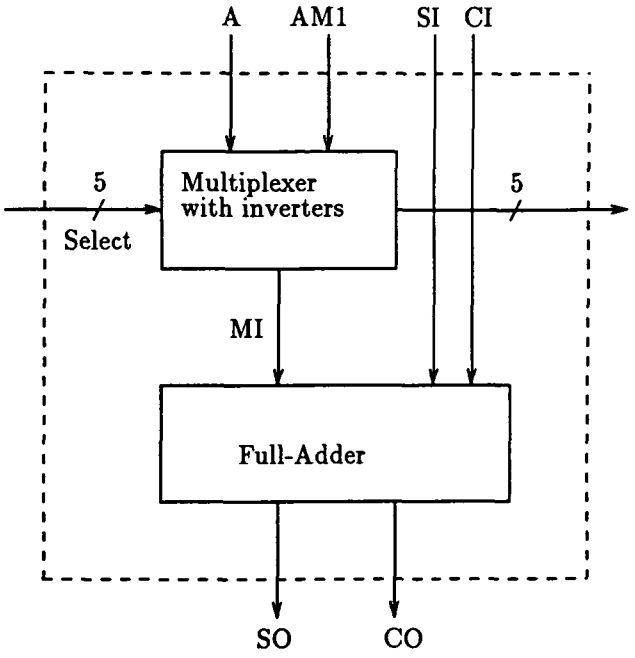


Figure 3: Block diagram of a multiplier cell

Full-Adders. The signal labeled MI takes care of the sign extension.

The design steps for implementing the above cells in pass transistor technology are available in [4]. In pass transistor implementation we need both the signal and it's complement. There are two ways in which this can be done within the MAC array. One option is to simultaneously generate the complements of signals using redundancy in hardware. This option (option A), increases the individual cell areas in the MAC, thereby increasing the area of the MAC cell itself. Option B reduces hardware redundancy thereby saving some of the area of a multiplier cell by using additional inverters to obtain complements of signals. Figure 4 illustrates the multiplier cells designed using both options. The area of these cells and their comparative speed etc. are given in subsequent sections. In this figure the block named BTS is a Binary Tree Structured [6] network and INV(1) as well as INV(2) represent a single and double inverters respectively. The block labeled MUX is a 5:1 multiplexer.

2.3 Conditional Sum adder for the final addition

An interesting scheme called the conditional sum adder scheme [10] has been investigated by K. Cameron and J. Canaris [3] and changes in the design to suit CMOS pass transistor version have been proposed by them. This type of adder is found to be extremely fast particularly for large word sizes. This adder has since been incorporated in a VLSI chip fabricated using a 1.6μm double metal CMOS process. The same approach has been adopted in this design with appropriate changes in transistor sizing and VLSI layouts to suit the 1μm drawn (0.75μm internal) three metal CMOS process that has been used here. As expected, this not only saved chip area but after re-sizing some of the transistors reduced the worst case delay through the adder as compared to the earlier design. These details are included in the subsequent sections.

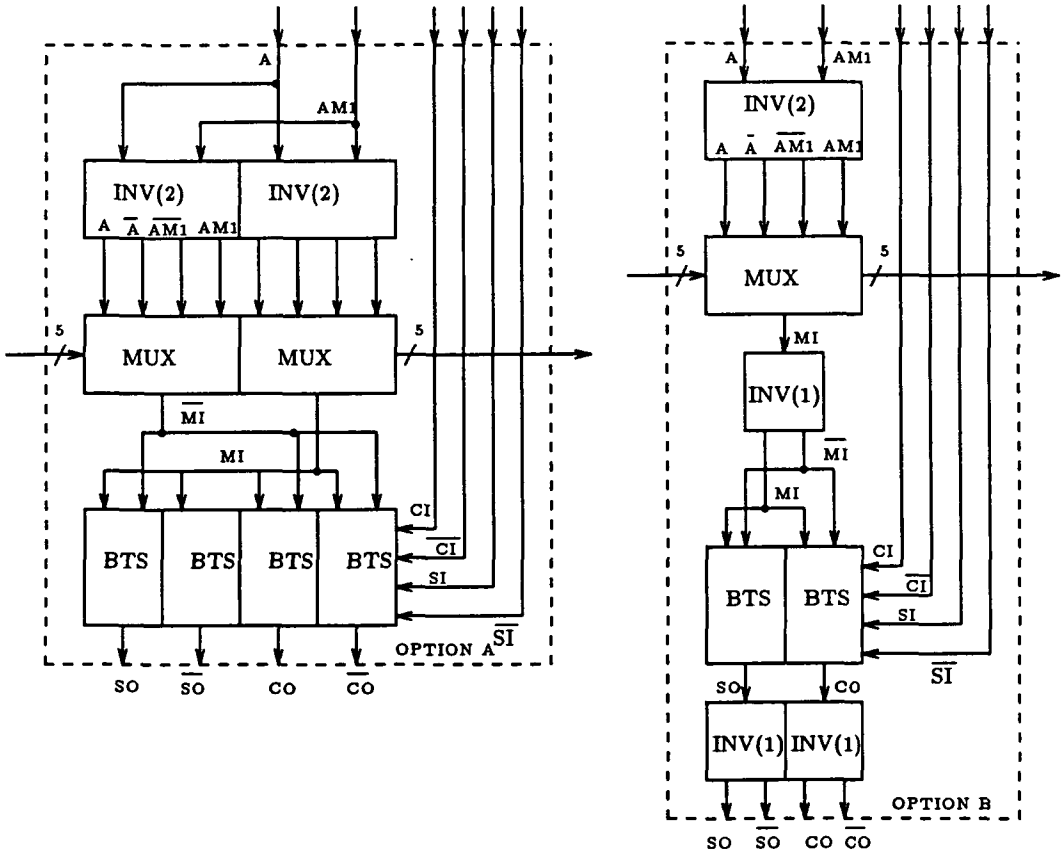


Figure 4: Block diagrams of the multiplier cell for option A and option B

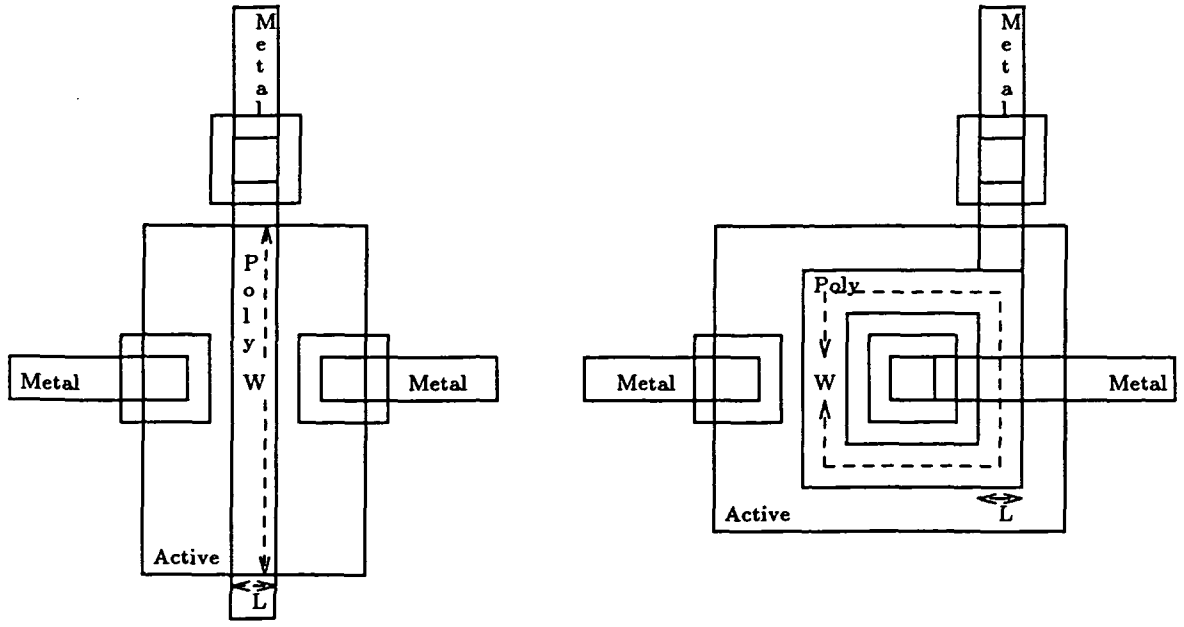


Figure 5: Typical layouts of a regular and square transistor

3 Layouts

The following two subsections deal briefly with the layouts of some of the cells required for the multiplier section and the conditional sum adder section.

3.1 Layouts of multiplier cells

After taking into account the various VLSI issues discussed in detail in [4], the layouts for multiplier cells for option A and option B were drawn. The areas of the cells in option A and option B are $69\mu m \times 69\mu m$ and $42.2\mu m \times 76.6\mu m$ respectively which translates to an area saving of 32% for option B when compared to option A.

3.2 Layouts of conditional sum adder

As mentioned earlier, the conditional sum adder was previously designed using a $1.6\mu m$ double metal process [3]. Since the process used for this design had a minimum feature size of $1\mu m$ (drawn) which after the shrinkage was reduced to $0.75\mu m$, effectively there was a reduction in the minimum lengths of transistors by a factor slightly more than two. This implied that the widths of transistors could be reduced by the same factor without any appreciable change in the performance of the transistor. Doing this resulted in the adder taking about 25% less area when compared to the earlier design.

Square transistors were used in certain inverter buffer cells to take advantage of the resulting reduction in capacitance thereby improving the speed. Figure 5 contrasts the regular and square transistors. More details about these two ways of drawing a transistor along with their merits and demerits are available in [4].

$\overline{SO}$ signal	Delay for option A (ns)	Delay for option B (ns)
First row	4.27	4.5
Second row	5.88	6.5
Third row	7.65	8.5
Fourth row	9.41	10.4
Fifth row	11.32	12.8

Table 1: The worst case delay through the multiplier section

The approximate areas of the multiplier section in option A and option B are $2300\mu m \times 800\mu m$ and $1600\mu m \times 850\mu m$ respectively. The area of the conditional sum adder is approximately $1150\mu m \times 210\mu m$. Thus, when the multiplier and conditional sum adder sections are combined, option B is not only more area efficient, but also less area is wasted as compared to option A.

4 HPSPICE simulation

Representative portions of the multiplier section for both options and the 32 bit conditional sum adder are simulated using HPSPICE to estimate their worst case propagation delay as well as the approximate dynamic power dissipation in them. The worst case conditions for this design are assumed to be 4.3V VDD and 0.3V VSS after accounting for the noise on the power supply and a temperature of 110C.

4.1 Simulations on the multiplier section

Representative portions of the multiplier sections for both options were simulated for several worst case input patterns. The worst case timing patterns were determined by [5] for a similar structure. Since the sum path through the multiplier array turned out to be the critical delay path, the input pattern which resulted in only the sum signals transitioning as a result of input signals transitioning was found to be the worst among the worst case patterns analyzed. This pattern was A = 1111 1111 1111 1111 and B = 1000 0000 0000 0001 and $C = 0$. None of the carry signals transitioned during this simulation. Table 1 summarizes the delays for $\overline{SO}$ signals for the top five rows of the multiplier section for both the options. This table shows that option A is slightly faster than option B by about 1 ns for four rows which can be extrapolated to 2 ns for the entire multiplier section.

An approximate estimate for the average dynamic power dissipated in the multiplier section of the 32 bit MAC is obtained as 47.4 mW for option A and 29.8 mW for option B at the maximum operating speed.

4.2 Simulation of conditional sum adder

The conditional sum adder was earlier designed using pass transistor technology using a $1.6\mu m$ double metal CMOS process with a worst case propagation delay of 24 ns [3]. In this design the corresponding number after the first-cut simulation using standard transistor

Signal name	conditional sum adder delay (ns)	MAC delay (ns)
Q3	6.83	10.96
Q7	9.85	15.56
Q15	13.82	24.19
Q31	18.9	29.39

Table 2: The worst case delays through the conditional sum adder and the MAC

layouts everywhere was 23 ns assuming a load capacitance of 1 pF. Using square transistors in two of the inverter buffer cells namely, MUXBUFF2(D) and MUXBUFF2(E), the worst case delay was reduced to 22 ns. Further analysis of the simulation results and after resizing the transistors in MUXBUFF2(E) to reduce the capacitive load on the critical delay path, the delay was reduced to 18.9 ns which translated into a speed improvement of nearly 20%.

In order to estimate the performance of the MAC, information from two separate simulations namely, the multiplier section alone and the 32 bit conditional sum adder alone was used. Since the worst case output timing pattern for each output of the multiplier section is predictable as per Table 1, the timing of the inputs to the conditional sum adder were all shifted in time by the same amounts as they would be if they were to be the outputs of the multiplier section. The simulation that followed yielded a worst case delay through the conditional sum adder and due to the time shifting of inputs, the delay through the entire MAC itself as 29.39 ns. This translated into a clock frequency of 34 MHz for the DCT chip. Table 2 summarizes the delay for the various bit positions of the conditional sum adder outputs alone and with it's inputs shifted in time to mimic the performance of the MAC.

An estimate for the average dynamic power consumption in the conditional sum adder was 10.25 mW. Thus the approximate figure for the MAC turned out be 57.7 mW for option A and 40 mW for option B.

5 Conclusions

Critical blocks for a pass transistor based CMOS 2D DCT chip were designed and performance estimated. An improvement of nearly 20% in speed was obtained over previous work. The processing of the image data was performed parallelly using 16 MACs for 1D processing and an equal number of MACs for the second dimensional processing. The two 16×16 matrices representing the real-time image data and the coefficients were multiplied using MAC cells the layouts of which were regular and there were no complex routings. The MAC employed the modified Booth recoding algorithm and performed the multiplication in 2's complement arithmetic. A 32 bit conditional sum adder was used inside each MAC for the purpose of final addition. This adder was found to be extremely fast with the minimum number of series transistors in it's critical path thereby making it one of the fastest adders for the word size among various types of adders investigated. This was verified by simulations on the actual layouts using a software tool for VLSI design wherein after the resizing of some of the inverter buffers to improve the speed by nearly 20%, the final adder overhead for one of the worst case delays turned out to be only 10.5 ns. Table 3 summarizes the salient features of the chip. Although none of the blocks in Figure 1 other than the MAC

Technology	Digital CMOS
Process	CMOS26 (3 metal)
Minimum length	1μm (drawn) 0.75μm (internal)
Core area for 32 MAC cells	6.4mm $\times$ 6.8mm (option B)
Transistor count for 32 MAC cells	247,904 (option B)
16 $\times$ 16 bit worst case multiply time	29.39 ns (option B)
Worst case clock frequency	34 MHz. (option B)
Ave. dynamic power consumption (32 MACs)	1280 mW (option B)
Number of pins	32 + power, ground, clock etc.
DCT rate	$\frac{34MHz}{16x16}$

Table 3: The salient features of the chip designed

has been exclusively designed and simulated for this design, some estimates are available for the design and implementation of these blocks. The ROM cells placed along side each of the MAC cells need to store 256 bits of data each. From an earlier design in which ROM cells were used [14] it was possible to estimate the area required for each ROM after adjusting for the process as $200\mu m \times 140\mu m$. This was found to be a very small percentage of the area of the MAC cell. Likewise it is envisaged that it would not take too much of silicon area to implement the transposition memory, shift register etc shown in the overall block diagram in Figure 1 and it is expected that it would be easy to pitch match these blocks to the existing MAC layouts. It would then remain to provide buses for the power supplies, input output pads, pad drivers, clock, clock drivers etc. before the chip could be fabricated. Even if all these items meant doubling the core area for 32 MACs as a rough estimate for the entire 2D DCT chip, the chip size would still remain a very practical and realizable entity.

6 Acknowledgements

This research was supported in part by NASA under Space Engineering Research Grant NAGW-1406 and by the NSF under Research Initiation Grant MIP-9109618.

References

[1] N. Ahmed, T. Natarajan and K. Rao, " Discrete Cosine Transform", IEEE Transactions on Computers, Jan. 1974, pp. 90-93.

[2] M. Annaratone, *Digital CMOS Circuit Design*, Kluwer Academic Publishers, Boston, MA, 1986.

[3] J. Canaris and K. Cameron, "A Comparison of Two Fast Binary Adder Configurations", NASA SERC 1990 Symposium on VLSI Design, Moscow, ID, pp. 78-86.

[4] S. Kamath, "A VLSI implementation of Discrete Cosine Transform using Pass Transistor Technology", M.S Thesis, University of Idaho, Moscow ID.

[5] Y. Oowaki et al., "A 7.4ns CMOS 16 × 16 Multiplier", 1987 IEEE International Solid-State Circuits Conference, pp. 52-53.

[6] G. Peterson and G. Maki, "Binary Tree Structured Logic Circuits: Design and Fault Detection", Proceedings of IEEE International Conference on Computer Design: VLSI in Computers, Port Chester, NY, Oct., 1984, pp. 671-676

[7] D. Radhakrishnan and G. Maki, *Design of Pass Transistor Switching Circuits*, Moscow, ID, University of Idaho, 1983.

[8] D. Radhakrishnan, S. Whitaker and G. Maki, "Formal Design Procedures for Pass Transistor Switching Circuits", Proceedings IEEE Custom Integrated Circuits Conference, Rochester, NY, May, 1984, pp. 139-144

[9] D. Radhakrishnan, S. Whitaker and G. Maki, "Formal Design Procedures for Pass Transistor Switching Circuits", IEEE JSSC, vol. SC-20, April, 1985, pp. 531-536

[10] A. Rothermal et al., "Realization of Transmission Gate Conditional Sum (TGCS) Adders with Low Latency Time", IEEE Journal of Solid State Circuits, vol. 24, No. 3, June 1989, pp. 558-561.

[11] S. Steinlechner and G. Spahlinger, "Carry-Save Adders and their Application for a Multiplication with Factored Multiplicands", Proceedings of IEEE International Conference on Computer Design: VLSI in Computers, Port Chester, NY, Oct., 1985, pp. 359-362

[12] S. Whitaker, *Design of Combinational Logic with Pass Transistors*, Moscow, ID, University of Idaho, 1982.

[13] S. Whitaker, "Pass Transistor Networks Optimize NMOS Logic", Electronics, McGraw Hill, New York, NY, Sept. 22, 1983, pp. 144-148

[14] S. Whitaker, J. Canaris and K. Cameron, "Reed Solomon VLSI Codec For Advanced Television", IEEE Transactions on Circuits and Systems for Video Technology, vol. 1, No. 2, June 1991, pp. 230-236.

A High Speed CMOS A/D Converter[1]

Don R. Wiseman and Sterling R. Whitaker
NASA Space Engineering Research Center
University of New Mexico
2650 Yale SE, Suite # 101
Albuquerque, New Mexico 87106

Abstract – **This paper presents a high speed A/D converter. The converter is a 7 bit flash converter with one half LSB accuracy. Typical parts will function at approximately 200 MHz. The converter uses a novel comparator circuit that is shown to out perform more traditional comparators, and thus increases the speed of the converter. The comparator is a clocked, precharged circuit that offers very fast operation with a minimal offset voltage (2 mv). The converter was designed using a standard 1 micron digital CMOS process and is 2,244 microns by 3,972 microns.**

1 Introduction

Today, large integrated circuits (ICs) are involved in solving complex signal processing problems. A mixture of digital and analog signals often must be integrated into a single system [1]. Modern digital-to-analog (D/A) and analog-to-digital (A/D) converters have become common components for integrated signal processing. Of the many styles of A/D converters available [1]-[2], only three [3, 4], the flash, half flash, and successive approximation, take full advantage of the speed that CMOS technologies can provide.

Of the three types of converters, the flash converters are clearly the fastest, but they are also the largest. For high performance systems, the flash converter is, however, the choice. To minimize the area required this coverter was designed for 7 bits of resolution, thus limiting the comparator chain to the length of 128 comparator cells. Secondly, the comparator cell itself is highly optimized for both performance and size [5]. Finally, the state of the art process used to manufacture this converter will allow 1 micron gate lengths, thus helping to keep the size of the individual transistors to a minimum.

The basic configuration of a *flash* A/D converter is shown in Figure 1. This type of A/D converter is fully parallel, since it processes each of the n output bits simultaneously thus accounting for the speed of operation [6]. However, the price for this speed is the requirement for 2^n comparators. Thus, for large values of n this can make flash A/D converters prohibitively large.

This style of converter works by first taking a reference voltage (Vref) and dividing it down a chain of resistors. The voltage at each resistor is fed to a comparator, and is compared with the input voltage (Vin). Thus the output down the chain of comparators will be a logic zero until the divided reference voltage is less than the input voltage. At that point, and further down the comparator string, the outputs will be a logic one. The outputs of the

[1]This research was supported in part by NASA under Space Engineering Research Center Grant NAGW-1406.

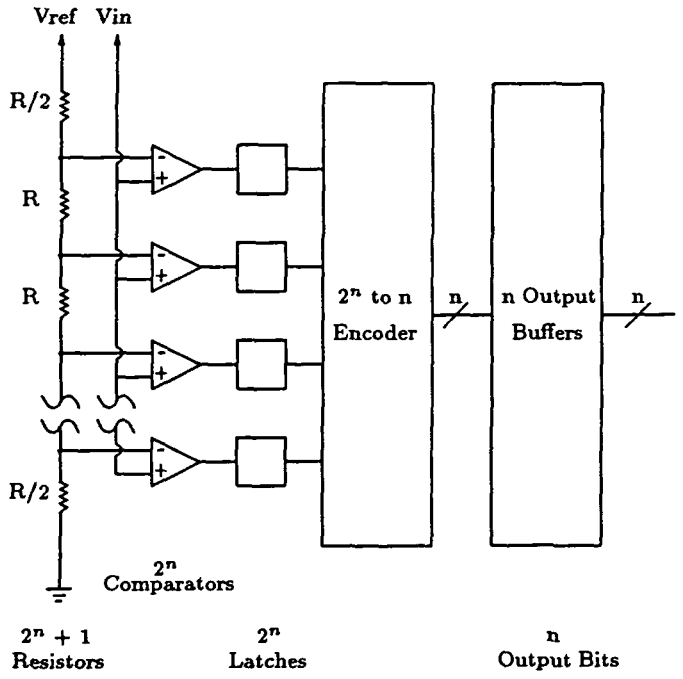


Figure 1: Flash A/D converter

comparators are then latched into flip-flops so that comparison of the next input sample may begin, without destroying the results of the previous conversion. The outputs of the flip-flops are fed into a 2^n to n encoder. This encoder is essentially a set of XOR (Exclusive Or) gates and a small ROM (Read Only Memory) that determines the n binary output bits given the sequence of m zeros and $2^n - m$ ones. Finally, the output from the encoder is sent to a set of output buffers, and the digital representation of the input sample is output from the converter [3, 7].

2 Circuit Design

MOSIS will be used to fabricate this part using Hewlett-Packard's CMOS-34 process. CMOS-34 is a single poly, double metal layer process with 1 micron minimum gate lengths. Using CMOS-34 design rules, this A/D converter was designed to perform at a worst case of 100 MHz. To make the digital logic sections of the converter function as fast as possible, pre-charged logic was used in the memory section of the converter. Additionally, a pipelined architecture was implemented throughout the part to assist in increasing circuit speed. A diagram of the basic structure a flash A/D converter can be seen in Figure 1.

This A/D converter was designed for a 6 volt analog voltage supply and a 5 volt digital supply. This results in a maximum analog input voltage swing of 5 volts. With a 5 volt swing possible at the analog input, each successive count of digital output represents approximately 40 mV of change on the input voltage.

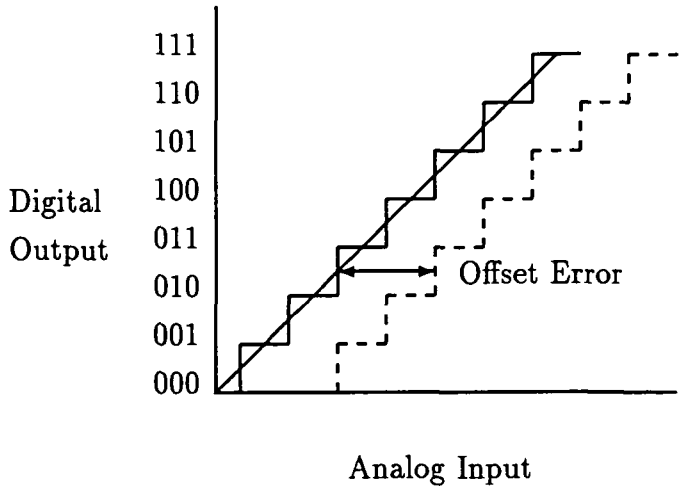


Figure 2: Offset Error Example

2.1 Accuracy Considerations

Before designing an analog integrated circuit, it is necessary to have an understanding of the types and sources of possible inaccuracies in the analog to digital conversion. In the worst case, a sum of all these errors should not result in either circuit failure, or skewed output of more than one half of a least significant bit (LSB), 20 mV.

The first type of error that can occur is called *offset error*. Offset error occurs when the analog input voltage for each digital step output is in error by a fixed amount. Pictured in Figure 2, offset error is most commonly caused by a uniform offset voltage being present on the inputs of all of the comparators. The comparators used in this design have a simulated offset voltage of less than 2 mV. Therefore, offset voltage problems should be reasonable. To determine the offset voltage of the comparators simulations were created that decreased the differential input voltage until the comparator failed.

The analog input voltage difference between two digital output steps is ideally one LSB. The difference from this ideal is called *differential linearity error*. This type of error, pictured in Figure 3, can be caused by systematic or random offsets in the comparators, or by resistance mismatches in the resistor chain. If differential errors are large enough, the A/D converter can lose its *monotonicity*. Monotonicity is the characteristic that there is a unique input voltage for each and every digital output word, and that the digital output always increases for increasing analog input voltages [3].

Another error that can occur in flash A/D converters is *integral linearity error*. Also picture in Figure 3, this error represents a summation of the differential errors up to any point in the digital output. Because of this, integral linearity errors generally have the same sources as differential linearity errors [3]. The measures taken to reduce these errors are discussed throughout the remainder of this paper.

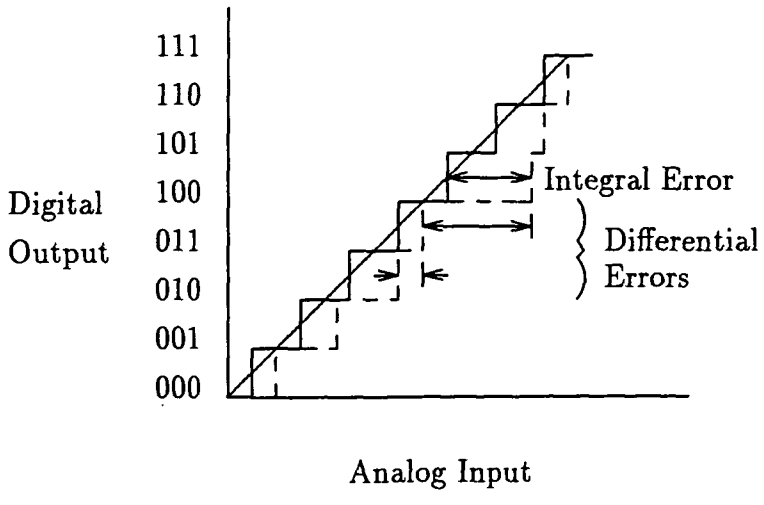


Figure 3: Integral and Differential Errors

2.2 Resistor Ladder

The chain of resistors shown at the left of Figure 1 was constructed from a long strip of polysilicon. The actual resistance of the the line, and thus each individual resistor value, is not critical. What is critical is that each resistor have the same value, such that the ratios of resistance at any point is fixed [8].

Additionally, in order to maintain a regular current density down the line of polysilicon, contacts (equipotential surfaces) are not placed in the current flow. Rather the voltage is tapped on "dog-leg" type structures that come from the polysilicon at regular intervals. Also, in order to keep the reference voltages as stable and noise free as possible, decoupling capacitors were integrated at each reference node. These capacitors help to greatly reduce noise that is coupled through the differential input on the comparator during the pre-charge phase of comparison.

2.3 Comparators

Three viable comparators with typical styles were selected and compared. After the analysis, the best comparator style for this project was selected, and then optimized using HP Spice. The analysis was performed by attempting to size the comparators so that they were of approximately equal speed and accuracy. The comparators were then simulated and their characteristics were compared. These characteristics included size, power, noise, offset, clocking, and input and output voltage levels.

The first comparator analyzed was a clocked, precharged CMOS comparator. This comparator is shown in Figure 4. The circuit functions as follows. First, while the clock, ($\phi 1$), is low, nodes one and two (N1 and N2) are precharged to VDD. Then when the clock goes high the input and reference voltages will turn on transistors Q5 and Q6. If there is a voltage difference between Vin and Vref, Q5 and Q6 will conduct different currents. This causes a potential difference between N3 and N4. This in turn provides positive feedback such that the current difference through Q3 and Q4 increases, and very rapidly, either N1 or N2 will

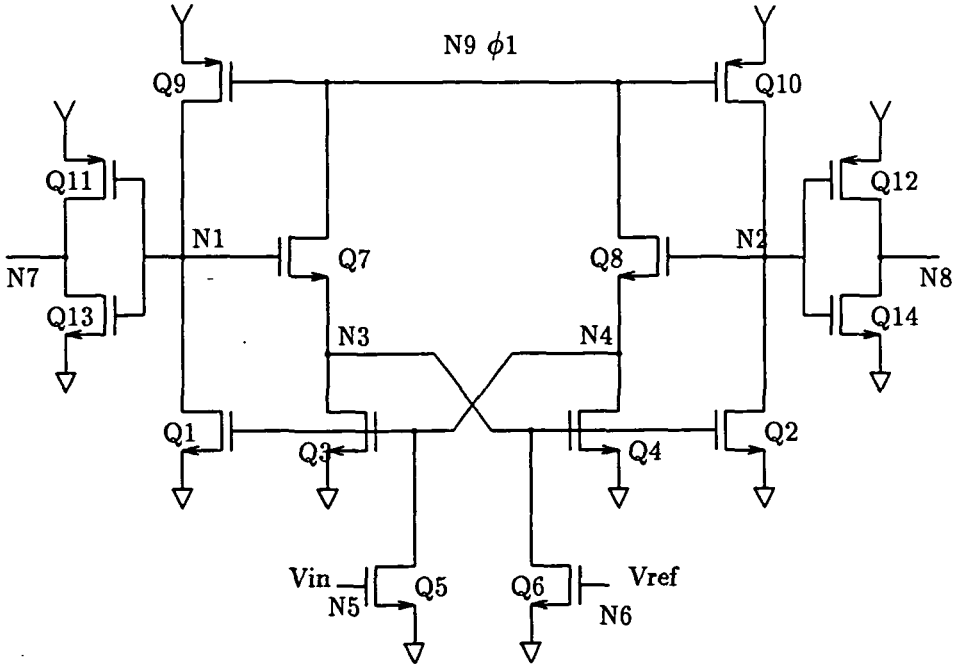


Figure 4: Clocked, Precharged, CMOS Comparator

be discharged to VSS. The output voltage may be read at either N7 or N8 depending on the logic level that is desired. Whichever of the two nodes is used for the output, the other inverter stage is left in place to create a balanced circuit.

The next comparator is slightly simplier than the previous comparator. It uses capacitively stored logic levels to increase its speed, and it requires two synchronized clocks. As seen in Figure 5, during $\phi 1$ the input and reference voltages are stored on C1 and C2. Then, during $\phi 2$, the comparator turns on and the two voltages on the capacitors are compared. The output of the comparator may be taken at either N1 or N2, or from both locations if a differential output is desired [9].

The final comparator was a three stage, autozeroing comparator. This comparator has the advantage of allowing the input voltage to swing from rail to rail. It also uses a clocked scheme to increase its speed. Figure 6 shows this comparator circuit. During $\phi 1$, the reference voltage is placed on N1 and the first two stages are autozeroed. Then during $\overline{\phi 1}$ the input voltage is placed on N1. The difference between the reference voltage and the input voltage is coupled into the three stages of inversion and gain with the output appearing at N6.

The three comparators were simulated and compared. Tables 1 through 3 represent a synopsis of the comparisons that were performed and the results. A simple ranking scheme was used to determine which comparator performed the best. Each attribute for each comparator is marked with either a 1, 2, or 3, depending on how the comparator performed. A 1 was given to the best, a 3 to the worst.

As can be seen from Tables 1 to 3, clearly the clocked, precharged, comparator of Figure 4 out performs the other two. Therefore, this comparator was used in the design of this A/D converter.

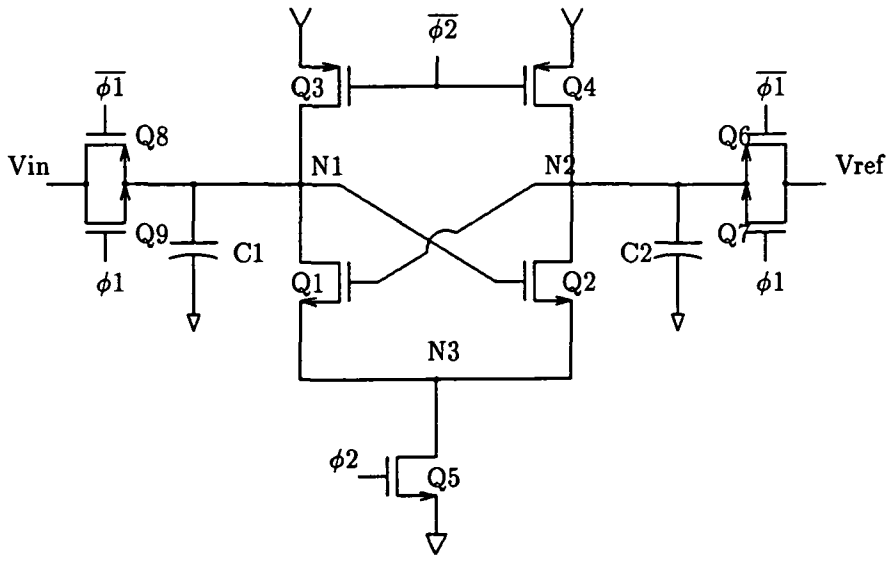


Figure 5: Dual Clocked, Stored Level, CMOS Comparator

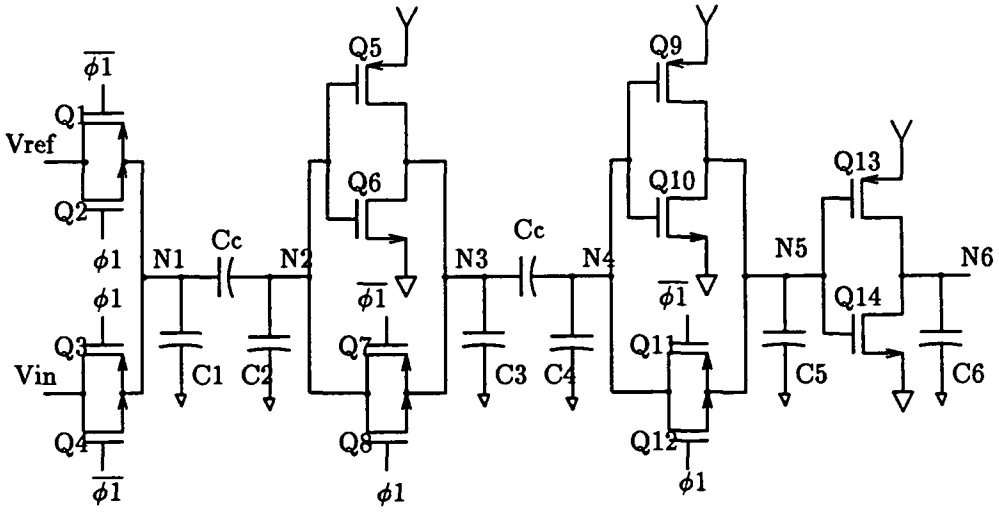


Figure 6: Three Stage, Autozeroed, CMOS Comparator

Attribute	Ranking	Comments
Size	2	14 somewhat large transistors.
Power	1	No DC paths to VSS or bias currents.
Noise Generation	1	Input signals have only minimal coupled noise.
Offset Errors	2	Offset reduced to 10mv fairly easily.
Clocking	1	Requires only one, single phase clock.
Input Levels	2	Input levels from VDD to VSS plus a threshold.
Output Levels	1	Outputs from VDD to VSS.

Table 1: Summary of Evaluations for Clocked, Precharged, Comparator

Attribute	Ranking	Comments
Size	1	Small number of transistors keeps this cell small.
Power	2	Draws power from inputs and bias current during $\phi 2$.
Noise Generation	2	Some noise is transferred to input signals by C1 and C2.
Offset Errors	3	Offset harder to control.
Clocking	3	Requires two, double phase clocks.
Input Levels	3	Smallest valid input voltage range.
Output Levels	3	Reduced output voltage range.

Table 2: Summary of Evaluations for Dual Clocked, Stored Level, Comparator

Attribute	Ranking	Comments
Size	3	14 somewhat large transistors plus 5 capacitors.
Power	3	High power; large current while zeroing first two gain stages.
Noise Generation	3	Large noise transferred to inputs and to the power supplies.
Offset Errors	1	Because of autozeroing, no offset problems.
Clocking	2	Requires one, double phase clock.
Input Levels	1	Valid inputs from VDD to VSS.
Output Levels	1	Outputs from VDD to VSS.

Table 3: Summary of Evaluations for Three Stage, Autozeroed, Comparator

2.4 Other Cells

There are two other cells of some interest in the digital section of the A/D converter. First is an exclusive-or (XOR) gate. There are many XOR styles available, but a 6 transistor one was chosen primarily for its speed and low transistor count [10]. The only XOR gates with fewer transistors, require both complemented and uncomplemented input signals. This can reduce the transistor count by two in the XOR gate, but it actually increases the number of transistors in the flip-flop cell that feeds the XOR by two fairly large transistors since they are drivers. These not only add to the size of the cells, but also to the capacitance in the cell. Therefore, for overall size and speed of the converter, this XOR gate was chosen.

The other cell of interest is the encoding ROM. This ROM takes the 128 signals from the XOR gates and encodes them into to proper seven bit representation. In order to make this ROM as fast as possible, precharged logic is used. Basically, the cell functions by precharging all of the seven bit lines while the clock is low. During this time, all of the discharge transistors are turned off. Then during the high phase of the clock, the appropriate bit lines are discharged, and the correct digital representation is sent on to a bank of output drivers.

3 Layout Considerations

Because, of the speed and analog nature of this chip, the layout of the circuitry is just as critical as the circuit design. Any small oversight could introduce an error term that would make the part much less usable at higher speeds. In particular, the comparator cell was

drawn several times before a balance of all the important design criteria was reached. With the high speed digital side of the chip, there were several timing and noise challenges, as well.

The final layout of the chip core is shown in Figure 7. On the left side of the chip is the comparator chain and resistor ladder. A guard ring surrounds this section to isolate it from the digital side of the chip. The analog side is also fed from a power supply separated from the digital side, again to keep the sensitive analog circuitry as isolated as possible.

The dimensions of the core are 2,244 microns by 3,972 microns. The core was intentionally left in this rectangular fashion for several reasons. First, to square up the chip would either require several turns in the resistor ladder or running several critical signal lines over susceptible circuitry. Both of these options are very undesirable because of the potential error terms that would be introduced. Additionally, since the long term purpose of this converter is to be used as a macrocell on a larger chip, there is really no need for the converter to be square. Therefore, it was decided to leave the core slightly rectangular.

3.1 Comparator Layout

The comparator cell layout was the most critical layout on the chip. This is because the accuracy of the A/D converter is primarily set by performance of the comparator. Also, the speed of the converter is limited by the speed of the comparator. The layout for the cell is shown in Figure 8. (Refer back to Figure 4 for a schematic of this comparator.)

Because there is a stack of 128 of these cells, it was necessary to layout the comparator horizontally. Also, to minimize the comparator's offset, it was necessary to draw the cell symmetrically. This symmetry keeps the transistors as balanced as possible. The final size of the comparator cell is 317.2 microns by 24.8 microns.

It is difficult to simulate or calculate the exact amount of noise, offsets, and other error factors that may occur in this comparator, because they are so dependent on manufacturing tolerances. This fact is complicated by the lack of analog process modeling done for the CMOS-34 process. This process is primarily intended to be a digital, not analog, process. Therefore, every attempt was made in layout to optimize the comparator cell so that the most accurate cell possible in CMOS-34 would be created while retaining the desired speed characteristics.

Offset and noise effects were modeled into the comparator cell simulations in an attempt to ensure the accuracy of the circuit. As shown earlier, the offset voltage for the comparator is less than 2 mV. One half LSB accuracy is desired for the converter, so 17.5 mV of noise could appear on the inputs of the comparators and the circuit would still be within acceptable limits. Simulations that include noise sources like power supply noise, substrate noise, clocking noise, etc. show that the total error voltage that could be present on the inputs to the comparators is less than 13 mV. Therefore, this A/D converter will have better than 1/2 LSB accuracy.

On the right side of the cell is the poly-silicon strip that creates the resistor ladder. The nominal resistance of the poly-silicon strip is 670 ohms per cell. Next to the resistor is a gate capacitor structure which yields a nominal capacitance of 0.8 pF per cell. This capacitor, along with the resistor, form an RC time constant that helps to keep the reference input of

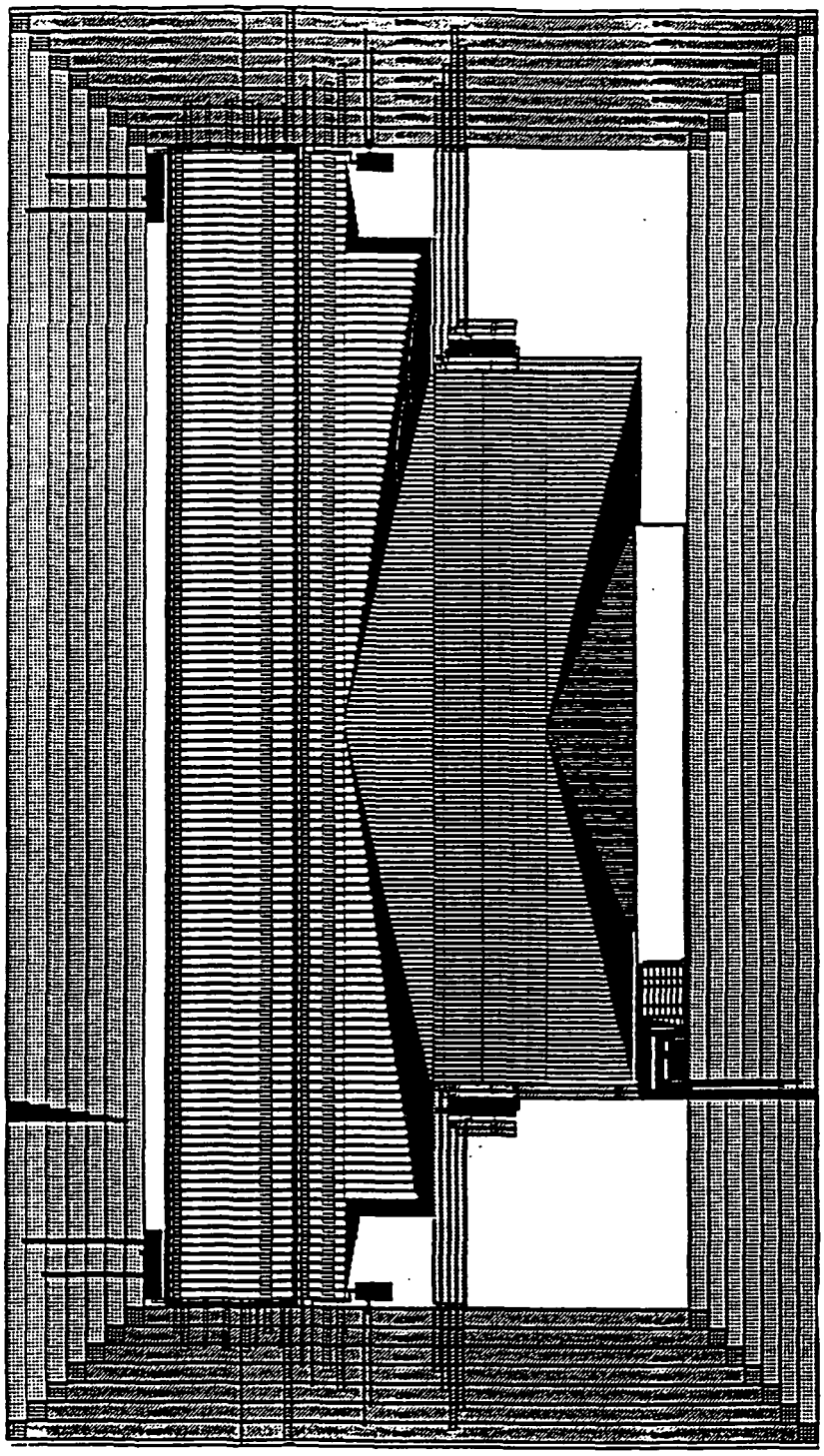


Figure 7: Layout Core of the A/D Converter

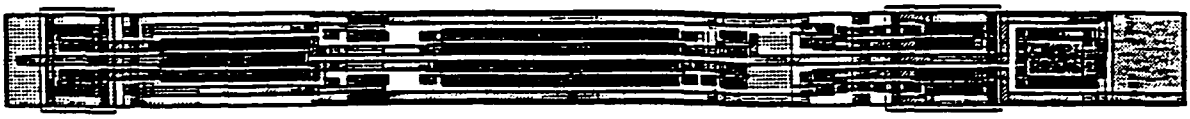


Figure 8: Comparator Cell Layout

the comparator (Vref) noise free. This in turn helps to minimize the differential errors that could occur during the conversion process from reference node noise.

Many other enhancements to the layout have been made to keep the cell as accurate as possible. First, the gates of large transistors are driven at both ends. This helps to minimize the resistance of the gate poly-silicon along the length of the transistor. Next, the source and drain diffusions of all the transistors are tied to metal down the entire length of the diffusion. This keeps source and drain resistance to a minimum, and thus increases the speed of the devices. Thirdly, the power supply lines that run throughout the cell are sized such that the noise on them (and thus the noise they couple into the cells) is kept to under 0.25 volts. If the lines were sized too small, large voltage spikes would appear on them and potentially couple into the comparator circuitry. Additionally, the cell was drawn so that signal lines do not cross over any sensitive nodes in the comparator. In certain instances, signals crossing over certain nodes could capacitively couple enough charge onto the node to corrupt or distort the comparator's operation. Also, the differential input pair of the comparator was sized minimally so that the noise that is coupled into the comparator through the parasitic capacitors of the transistor pair would be small.

3.2 Digital Cells

The stack of comparator cells feed directly into a stack of flip-flops. Recall that the comparator uses half of the clock phase for pre-charge, and the other half to evaluate the current analog input. Because of this, the output of the comparator is only valid during half the clock. This flip-flop will take the output of the comparator, amplify it to full digital levels, and turn it into a signal that is stable for an entire clock. The layout for the cell is shown in Figure 9.

In addition to accepting an analog input, these flip-flops must be able to function with worst case parameters at 100 MHz. This is extremely fast for current CMOS technologies, but simulations show that these cells work at that speed. The flip-flop has a setup time of less than 0.5 ns and a hold time of 0 ns. With these timings the flip-flop will accept 3.0 V for an input high voltage and 1.5 V for an input low voltage. It then drives the 0.6 pF of capacitance that the next cells (the exclusive or cells) presents to it in 3.0 ns.

The next stack of cells in the path of the data flow are the exclusive or (XOR) cells. The

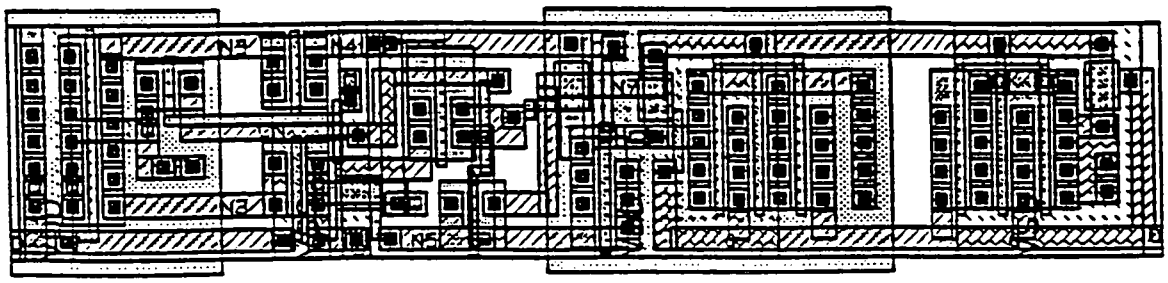


Figure 9: Layout of First Flip-Flop

output of each flip-flop, and thus each comparator, is XOR'd with its neighbor to find the point in the comparator stack where the reference voltage becomes smaller than the analog input voltage. Once this point is known, the digital output can be found through a simple memory lookup. The layout for the XOR cell is shown in Figure 10.

The remaining digital cells consist of a pulse type flip-flop, a precharged ROM, and an output flip-flop are not shown. Each cell has enlarged clock and power supply distribution networks to handle the high frequencies for which this circuit is designed. Additionally, the precharged ROM makes use of an unusual transistor configuration. The transistors in the ROM are constructed in a square or "waffle" type fashion. By doing this, the node at the center of the "waffle" has an absolutely minimal parasitic diffusion capacitance. This keeps the capacitance on the data lines as small as possible, and thus allows the ROM to function as quickly as possible. In fact, the capacitance on the data lines in the ROM cell is about 1.2 pF. This is quite small considering there are more than 64 transistor diffusions on each line. A single transistor drawn in this "waffle" fashion is shown in Figure 11.

References

[1] E. A. Vittoz, "The Design of High-Performance Analog Circuits on Digital CMOS Chips", *IEEE Journal of Solid-State Circuits*, Vol. SC-20, No. 3, June 1985, pp. 657-665.

[2] D. A. Hodges, P. R. Gray, and R. W. Brodersen, "Potential of MOS Technologies for Analog Integrated Circuits", *IEEE Journal of Solid-State Circuits*, Vol. SC-13, No. 3, June 1978, pp. 285-294.

[3] P. E. Allen, and D. R. Holberg, *CMOS Analog Circuit Design*, New York, N.Y., Holt, Rinehart and Winston, 1987, Chap 10.

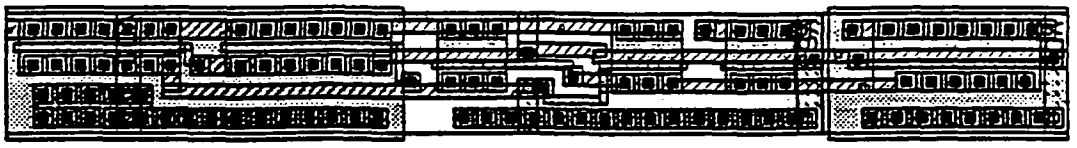


Figure 10: Layout of XOR Cell

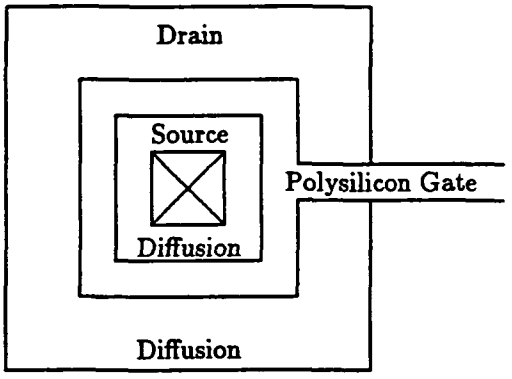


Figure 11: Waffle Type Transistor Configuration

[4] B. M. Gordon, "Linear Electronic Analog/Digital Conversion Architectures, Their Origins, Parameters, Limitations, and Applications", *IEEE Transactions on Circuits and Systems*, Vol. CAS-25, No. 7, July 1978, pp. 391-418.

[5] A. Yukawa, "A CMOS 8-Bit High-Speed A/D Converter IC" *IEEE Journal of Solid-State Circuits*, Vol. SC-20, No. 3, June 1985, pp. 775-779.

[6] T. Kumamoto, M. Nakaya, H. Honda, S. Asai, Y. Akasaka, and Y. Horiba, "An 8-bit High-Speed CMOS A/D Converter", *IEEE Journal of Solid-State Circuits*, Vol. SC-21, No. 6, Dec. 1986, pp. 976-982.

[7] A. G. F. Dingwall, "Monolithic Expandable 6 Bit 20 MHz CMOS/SOS A/D Converter", *IEEE Journal of Solid-State Circuits*, Vol. SC-14, No. 6, Dec. 1979, pp 926-932.

[8] D. A. Hodges "Analog Switches and Passive Elements in MOSLSI", *Analog MOS Integrated Circuits*, New York, N.Y., John Wiley and Sons, 1980, pp. 14-18.

[9] R. Gregorian, and G. C. Temes, *Analog MOS Integrated Circuits For Signal Processing*, New York, N.Y., John Wiley and Sons, 1986, pp. 425-437.

[10] N. Weste, and K. Eshraghian, *Principles of CMOS VLSI Design: A Systems Perspective*, Reading, MA., 1985, Chap 4.

Behavior of Faulty Double BJT BiCMOS Logic Gates [1]

Sankaran M. Menon Yashwant K. Malaiya† Anura P. Jayasumana
Dept of Electrical Engineering and † Computer Sceince
Colorado State Univerisity
Fort Collins, CO 80523

Abstract - **Logic Behavior of Double BJT BiCMOS device under transistor level shorts and opens is examined. In addition to delay faults, faults that cause the gate to exhibit sequential behavior were observed. Several faults can be detected only by monitoring the current. The faulty behavior of Bipolar (TTL) and CMOS logic families is compared with BiCMOS, to bring out the testability differences.**

1 Introduction

Combining the advantages of CMOS and Bipolar, BiCMOS is emerging as a major technology for many high performance digital and mixed signal applications. The main advantages of CMOS technology over bipolar are lower power dissipation and higher packing density. Bipolar technology offers better output current drive, switching speed, I/O speed and analog capability. Combining the advantages of bipolar and CMOS, BiCMOS offers the following advantages [1]; improved speed over CMOS, lower power dissipation compared to bipolar, flexibility in I/O (TTL, ECL, CMOS compatibility), high performance analog capability and latch up immunity. Compared to the CMOS counterparts, BiCMOS circuits can be faster by a factor of upto two for the same level of technology. Access times of less than 10ns have been reported for 0.8 μm BiCMOS ECL input/output 256K and 1M-bit SRAMs [2]. BiCMOS is even being considered for high performance microprocessors and dynamic RAMs, and it is felt that it will be one of the main technologies to drive almost all functions in the decade ahead [3].

In the present day integrated circuits, most of the defects and failures can be abstracted to shorts and opens in the interconnects and degradation of devices [4]. Transistor level shorts and opens model many of the physical failures and defects in ICs [5]. A study by Gailiay [6] on 4-bit MOS microprocessor chips revealed that many of the faults were shorts and opens at the transistor level. Analysis of faults in elementary static storage elements suggest that transistor level testing provides a higher coverage of faults compared to that at the gate level [7]. Thus, it is necessary to study the effects of failures at the transistor level and develop accurate fault models at this level [5].

The major fault models at transistor level are stuck-at faults, and shorts and opens of transistors and interconnects [8]. It has been shown [9, 10] that the stuck-at model does not cover many of the manufacturing defects in BiCMOS devices and that most open faults manifest themselves as delay faults. In this paper, we present the behavior of double BJT BiCMOS device under stuck-ON and stuck-OPEN failures for all transistors and bring

[1]This research was supported by a SDIO/IST funded project monitored by ONR.

out the testability differences between the three logic families, namely; TTL, CMOS and BiCMOS. Levitt et. al. [9] presented inadequacy of stuck-at fault models for BiCMOS. Testing of BiCMOS circuits and a design for testability scheme for current testing of BiCMOS circuits was presented in [11]. In this study, we present detailed behavior of BiCMOS device under various faults along with a comparison with CMOS and TTL logic families.

Several different designs of BiCMOS circuits exist. The most common type of BiCMOS circuits employ bipolar transistors to perform the function of driving output loads and CMOS to perform logic functions. In this paper, we briefly review the operation of a double BJT (D-BJT) BiCMOS NAND device. Logic behavior of D-BJT BiCMOS NAND devices are examined under different faults and their comparison is presented with other logic families (TTL and CMOS).

This paper is organized as follows. The operation of D-BJT BiCMOS NAND devices is described in Section 2. Section 3 deals with the analysis of physical failures in D-BJT BiCMOS devices, where the logic behavior of D-BJT BiCMOS devices are examined under different faults. Comparison of the three logic families (TTL, CMOS and BiCMOS) are done in Section 4. Finally, conclusions drawn from the study are given in Section 5.

2 BiCMOS Devices

BiCMOS circuits employ one or two Bipolar Junction Transistors (BJTs) to perform the function of driving output loads and CMOS to perform logic functions. In this section, the operation of a double BJT (D-BJT) NAND device is presented.

A Double BJT BiCMOS NAND realization is shown in Figure 1. There are many other realizations possible and we have investigated the above realization. The BiCMOS NAND realization shown in Figure 1 uses two output BJTs (Bipolar Junction Transistor) and there are other implementations that use single BJT. In this study, we deal with only double BJT BiCMOS devices. The functioning of the BiCMOS NAND can be explained by first applying logic '0' to one or both of the inputs which would cause at least one P-device to be ON and at least one N-device in each serial N-pairs to be OFF. With at least one N-device in each serial N-pairs being OFF, no current is supplied to the base of Q_2 resulting in transistor Q_2 being OFF. With the P-devices (P_1 and/or P_2) ON, the base of the bipolar NPN (Q_1) transistor would be about 5V supplying base current and turning ON the bipolar transistor (Q_1) providing logic '1' at the output. With either of the inputs being at logic '0' and the other input at logic '1' would still cause either of the parallel connected P-devices to be ON and either of the series connected N-devices to be OFF. This would still supply base current to the bipolar transistor Q_1 causing logic '1' at the output. With both the inputs at logic '1', the P-devices (P_1 and P_2) would be turned OFF, and the N-devices N_1, N_2, N_3 and N_4 would be turned ON, supplying base current to Q_2 which discharges the load. Transistor N_1 and N_2 draw current from the base of Q_1 thus rapidly turning this device OFF. This will cause the output to be a logic '0'. Thus the circuit realizes the NAND function. It may be noted that during output High to Low transition, transistor N_5 turns OFF as a result of transistors N_1 and N_2 discharging Q_1 base, causing the gate of N_5 to be low [12], this results in all the current through N_3 and N_4 to be provided as base current to transistor Q_2. During output Low to High transition, transistor N_5 turns ON to discharge the base of Q_2 quickly

to speed up the transition. It may also be noted that the static power consumption of the circuit is negligible neglecting reverse biased leakage currents. Block diagram of a general D-BJT BiCMOS device is shown in Figure 2. A D-BJT BiCMOS gate consists of CMOS p- and n-parts to perform logic function, and two output BJTs for driving the output node.

D-BJT BiCMOS devices do not have the full V_{DD} to Ground logic swing of CMOS devices. The output High voltage (V_{OH}) is limited to V_{DD}-$V_{BE(Q1)}$ and output Low voltage (V_{OL}) is limited to Gnd+$V_{BE(Q2)}$. V_{ILmax} and V_{IHmin} were determined to be 2.2V and 2.7V respectively [13]. The logic levels for D-BJT BiCMOS are 0.6V to 2.2V for logic level '0' and 2.7V to 4.4V for logic level '1' [13]. Any voltage between 2.2V and 2.7V is considered indeterminate. The device characteristics given for Fujitsu BiCMOS gate array devices [14] are V_{IHmin}=2V, V_{OHmin}=2.4V, V_{ILmax}=0.8V and V_{OLmax}=0.5V.

3 Analysis of Physical Failures in D-BJT BiCMOS devices

The response of the D-BJT BiCMOS NAND shown in Figure 1 is evaluated for hard failures of the bipolar & MOS transistors and their results are presented in this section. Possible failures considered are stuck-ON and stuck-OPEN of transistors. The output of the BiCMOS gate is obtained by simulating one failure at a time for all possible stuck-ON and stuck-OPEN failures for all transistors. Stuck-ON and stuck-OPEN were simulated by turning ON and turning OFF the respective transistors. Open (OP) in bipolar transistor terminals (emitter, base & collector) were simulated by connecting a resistance of R>1MΩ in series with the respective node and short (SH) were simulated by connecting a hard short of R<0.01Ω between the respective terminals. The BiCMOS gate outputs obtained analytically have been compared with SPICE simulation outputs to ensure correctness.

The fault-free and faulty behavior of BiCMOS NAND is summarized in Table 1. The length and width of pMOS (L_p,W_p) and nMOS (L_n,W_n) transistors used for BiCMOS devices in this study are (L_p=1.5μm,W_p=30μm) and (L_n=1.5μm,W_n=26μm), similar to the values used in [9], for consistency. Simultaneous current monitoring was performed during SPICE simulation and the observed I_{DDQ} values are listed in the Table along with the output logic levels. In Table 1, the subscript represents the transistor number for the BiCMOS circuit shown in Figure 1 and superscript represents the type of hard failure under consideration where ON indicates stuck-ON failure and OP indicates stuck-OPEN failure. For example, N_1^{ON} indicates transistor N_1 stuck-ON, N_1^{OP} indicates transistor N_1 stuck-OPEN, Q_{1c}^{OP} indicates transistor Q_1 collector open and Q_{1c-e}^{SH} indicates transistor collector to emitter short.

In order to make the analysis a true representative of circuit conditions, CMOS inverters were used to drive the BiCMOS device and CMOS inverters were used as load to the BiCMOS device as shown in Figure 3. Gates G_1 and G_2 are CMOS inverters used to drive the BiCMOS NAND gate G_3. CMOS inverter G_4 is used as load to the BiCMOS NAND. The length and width of pMOS (L_p,W_p) and nMOS (L_n,W_n) transistors used as CMOS driver devices in this study are (L_p=5μm,W_p=60μm) and (L_n=5μm,W_n=20μm). The sizes for the CMOS load devices used are (L_p=5μm,W_p=40μm) and (L_n=5μm,W_n=15μm). To study the effects of

output fan-out on BiCMOS devices, analysis was conducted with one CMOS load alone and also with an RC (Resistor Capacitor) load along with a CMOS load as shown in Figure 3. R=100Ω and C=1pF were chosen for this study and RC load referred to henceforth in this paper refers to the above values.

3.1 Stuck-ON faults in D-BJT BiCMOS NAND

Referring to the D-BJT BiCMOS NAND shown in Figure 1, for the physical failure P_1^{ON}, input vector '11' causes the N-devices (N_1, N_2, N_3 and N_4) to be ON. This causes transistor Q_2 to be ON, providing a conduction path from V_{DD} to V_{SS}(Gnd), resulting in enhanced I_{DDQ}. The current drawn by the device with this vector for the fault under consideration is 2mA instead of the normal 0.2μA. Current testing technique can be employed to detect this fault. Similar result is observed for transistor P_2 stuck-ON failure (P_2^{ON}). SPICE simulation indicates the output voltage level to be ≈1.63V, which is logic '0' level for BiCMOS devices indicated as '0' in Table 1. For transistor N_1 stuck-ON and input vector 01, transistors P_2, N_2 and N_4 would be turned ON and with transistor N_1 stuck-ON leads to transistor Q_2 to be ON. This provides a conduction path between V_{DD} and V_{SS}(Gnd), resulting in an increased current flow (enhanced I_{DDQ}). SPICE simulation indicates the output voltage to be 1.86V, which is logic '0' for D-BJT BiCMOS devices. For transistor N_2 stuck-ON, input vectors 00, 01 or 11 would exhibit fault-free behavior. With input vector 10, transistors P_1 and N_1 would be turned ON and due to the transistor N_2 stuck-ON under consideration, causes transistor Q_2 to be ON. This results in a conduction path to exist between V_{DD} and V_{SS}(Gnd) resulting in enhanced I_{DDQ}. SPICE simulation indicates the output voltage level to be ≈1.86V, which is logic '0' level for D-BJT BiCMOS devices. Stuck-ON failures of transistors N_3 and N_4 result in enhanced I_{DDQ} for input vectors 01 and 10 respectively. The fault-free and faulty logic levels for N_3 and N_4 stuck-ON failures exhibit logic '1' at the output. Since the fault-free and faulty logic levels are the same, current testing alone can detect the failures. Transistor N_5 stuck-ON failure does not cause any appreciable effect for output Low to High transitions. However, during output High to Low transitions, with input vector '11', the output finds a low resistance path through transistors N_3, N_4 and N_5. Due to this low resistance path, transistor Q_2 does not turn ON and hence, High to Low transition gets delayed. This delay is dependent upon the output load. For RC load, the High to Low transition delay was observed to be 1.45ns instead of the normal 0.89ns. It may be noted that due to the low resistance path through transistors N_3, N_4 and N_5, output goes all the way to ground instead of $Gnd+V_{BE(Q2)}$. Transistor Q_{1c-e} and Q_{1b-c} shorts result in enhanced I_{DDQ} and causes a faulty output logic level '1' for input vector 11. Transistor Q_{2c-e} and Q_{2b-c} shorts also result in enhanced I_{DDQ} and causes a faulty output logic level '0' for input vectors 00, 01 and 10. Transistor Q_{1b-e} and Q_{2b-e} shorts result in delay faults for Low to High transition and High to Low transitions respectively, as the base to emitter junction of the transistors do not get forward biased and hence do not get turned ON. The Low to High transition delay observed for Q_{1b-e} short with RC output load is 1.98ns compared to the fault-free delay of 1.07ns. The High to Low transition delay observed for Q_{2b-e} short with RC output load is 1.47ns compared to the fault-free delay of 0.89ns.

Table 1. Behavior of D-BJT BiCMOS NAND with Stuck-ON and Stuck-OPEN faults.

BiCMOS NAND Stuck-ON and Stuck-OPEN results														
Input	*ff*	P_1^{ON}	P_2^{ON}	N_1^{ON}	N_2^{ON}	N_3^{ON}	N_4^{ON}	N_5^{ON}	Q_{1c-e}^{SH}	Q_{1b-c}^{SH}	Q_{1b-e}^{SH}	Q_{2c-e}^{SH}	Q_{2b-c}^{SH}	Q_{2b-e}^{SH}
A B	X i	X i	X i	X i	X i	X i	X i	X i	X i	X i	X i	X i	X i	X i
0 0	1 n	1 n	1 n	1 n	1 n	1 n	1 n	1 n	1 n	1 n	D_{0-1} n	0 a	0 a	1 n
0 1	1 n	1 n	1 n	0 a	1 n	1 a	1 n	1 n	1 n	1 n	D_{0-1} n	0 a	0 a	1 n
1 0	1 n	1 n	1 n	1 n	0 a	1 n	1 a	1 n	1 n	1 n	D_{0-1} n	0 a	0 a	1 n
1 1	0 n	0 a	0 a	0 n	0 n	0 n	0 n	D_{1-0} n	1 a	1 a	0 n	0 n	0 n	D_{1-0} n
Input	*ff*	P_1^{OP}	P_2^{OP}	N_1^{OP}	N_2^{OP}	N_3^{OP}	N_4^{OP}	N_5^{OP}	Q_{1e}^{OP}	Q_{1b}^{OP}	Q_{1c}^{OP}	Q_{2e}^{OP}	Q_{2b}^{OP}	Q_{2c}^{OP}
A B	X	X	X	X	X	X	X	X	X	X	X	X	X	X
0 0	1	1	1	1	1	1	1	D_{0-1}	R	R	D_{0-1}	1	1	1
0 1	1	1	Q^n	1	1	1	1	D_{0-1}	R	R	D_{0-1}	1	1	1
1 0	1	Q^n	1	1	1	1	1	D_{0-1}	R	R	D_{0-1}	1	1	1
1 1	0	0	0	D_{1-0}	D_{1-0}	D_{1-0}	D_{1-0}	0	0	0	0	D_{1-0}	D_{1-0}	D_{1-0}

X = Output, i = Current drawn by the device, Q^n = Previous State,
ON = Stuck-ON, OP = Stuck-Open, SH = Short,
ff = fault free, I^* = Indeterminate (2.2-2.7Volts), (e, b, c = emitter, base, collector),
n (Normal Current) = 2e-7A, a (Abnormal Current) > 2.00e-2A, R = Stuck-at-0 after initialization,
D_{0-1} = Low to High transition delay, D_{1-0} = High to Low transition delay.

Current testing can be very effective for testing failures which result in elevated I_{DDQ} from a normal $\approx 0.2\mu$A to enhanced $\approx$2mA, an increase by a factor of $\approx 10^4$. Since stuck-ON failures P_1^{ON}, P_2^{ON}, N_3^{ON} and N_4^{ON} provide same logic level for faulty as well as fault-free operations, current testing alone can detect the failures. Transistor N_1 and N_2 stuck-ON as well as Q_{1c-e}^{SH}, Q_{1b-c}^{SH},Q_{2c-e}^{SH}, and Q_{2b-c}^{SH} failures exhibit dissimilar outputs under faulty and fault-free conditions, conventional logic testing can detect the failure. However, current testing would detect this failure mode. Transistors Q_1 & Q_2 base to emitter shorts manifest as Low to High and High to Low transition delays respectively and hence delay test alone would detect the failure modes.

3.2 Stuck-OPEN faults in D-BJT BiCMOS NAND

Two faults in the D-BJT BiCMOS NAND exhibit sequential behavior (Q_n), similar to the behavior seen in CMOS circuits. Presence of the fault P_1 stuck-OPEN with input vector '10' causes the previous state to be retained resulting in sequential behavior. Similar sequential behavior is observed for P_2 stuck-open with input vector '01'. Two pattern tests can be applied to detect these stuck-open failures.

S-OPEN failures of transistors N_1 and N_2 exhibit unique delay faults, as observed in S-BJT BiCMOS NAND. A first glance would lead one to expect that with input vectors '11', the output parasitic capacitance would be discharged by turning ON of transistors N_3, N_4 and Q_2. However, due to the OPEN fault of N_1 or N_2 under consideration, the vectors 00, 01 or 10 would charge up the parasitic capacitors at the base as well as the emitter nodes of the bipolar transistor Q_1. With the application of input vector 11, the series path of N_3 and N_4 will be turned ON but the series path of N_1 and N_2 will not be turned ON due to the

fault. This will cause transistor Q_1 to remain ON for sometime because of the charge stored at the base of the bipolar transistor Q_1. Transistor Q_1 would be discharged slowly through N_3, N_4 and Q_2 path alone causing delay in the output response. The slow to fall delay fault is shown in Figure 4a,b,&c. This type of fault has been observed in [9]. Figures 4a&b show the response of the D-BJT BiCMOS NAND to N_1^{OP} failure with one CMOS output load and input pulse width t_{pw} of 10ns and 4ns respectively. Figure 4a shows the response of the BiCMOS NAND to N_1^{OP} with only one CMOS connected to the BiCMOS output. The inputs shown in this figure are the inputs applied to the BiCMOS NAND and the input pulse width (t_{pw}) is 10ns wide. The response of the BiCMOS NAND with N_1^{OP} fault with input t_{pw}=10ns shows slow to fall delay (t_{hl2}) of 4.5ns instead of the normal propagation delay (t_{hl1}) of 0.526ns. Response of the BiCMOS output with the same one CMOS load and with t_{pw} of 4ns causes a High to Low delay of 7.2ns instead of the normal propagation delay of 0.58ns. As the clock period is small, it should be noticed here that the output barely reaches the switching threshold and the logic level does not have a chance to drop to logic '0' range. If the clock period is further reduced or if the output load is increased, the output level will not have a chance to reach even the switching threshold. An example of which is shown in Figure 4c where an RC load is used in addition to a CMOS load. It can be seen that the output does not have a chance to reach the switching threshold. The response of the fault-free BiCMOS NAND with t_{pw}=4ns and RC load gives a propagation delay of 0.79ns.

Response of stuck-OPEN failure of N_4 are shown in Figures 5a,b&c. For the stuck-OPEN failure of N_4 shown in Figure 5a, with input vector '11' exhibits a delay (t_{hl2}) of 1.204ns and output logic level of 1V instead of the normal propagation delay of 0.526ns and logic level of 0.6V, with one CMOS load and input pulse width t_{pw} of 10ns. Reducing the pulse width t_{pw} to 4ns exhibits a delay (t_{hl2}) of 1.25ns in place of the normal t_{hl1} of 0.58ns as shown in Figure 5b. The output level is observed to be $\approx$1.5V instead of the normal output level of $\approx$0.6V, however, the output logic level is still a valid logic '0' level of D-BJT BiCMOS devices. With RC load and input pulse width t_{pw} of 4ns, the output logic level does not fall below the logic threshold as shown in Figure 5c. Hence, the fault appears as stuck-at-1 for logic testing purposes.

Bipolar transistor Q_1 emitter and base open faults manifest as stuck-at-0 after initialization (shown as R in Table 1). It can be seen that with either of the above faults, output cannot go to logic '1' (other than during power up) as no path exists between output and V_{DD}. With Bipolar transistor Q_1 collector open, the output exhibits Low to High transition delay (D_{0-1}) as shown in Figures 6ab&c. Figure 6a shows the response of BiCMOS NAND to Q_1 collector open with one CMOS load connected to the BiCMOS output and with input pulse width t_{pw} of 10ns shows Low to High transition (t_{lh2}) delay of 1.08ns instead of the normal Low to High transition (t_{hl1}) of 0.823ns. Response with the same one CMOS load and with input pulse width (t_{pw}) of 4ns exhibits Low to High transition delay (t_{lh2}) of 1.105ns instead of the normal propagation delay (t_{lh1}) of 0.72ns as shown in Figure 6b. Figure 7c shows the response of the BiCMOS NAND with input t_{pw}=4ns and RC load where the faulty output exhibits a larger delay for the Low to High transition. The faulty Low to High transition delay is seen to be t_{lh2}=2.3ns instead of the normal Low to High transition delay of t_{lh1}=0.91ns.

Bipolar transistor Q_2 emitter, base & collector open faults manifest as High to Low delay

faults. Base and emitter opens are shown in Figures 6a,b&c. With one CMOS load and input t_{pw} of 10ns shows a delay (t_{hl2}) of 1.059ns instead of the normal propagation delay (t_{hl1}) of 0.526ns for Q_2 base open. However, Q_2 emitter exhibits a lower delay compared to Q_2 base open. With input t_{pw} of 4ns and with one CMOS load, the output exhibits a delay (t_{hl2}) of 1.68ns instead of the normal propagation delay (t_{hl1}) of 0.58ns. With RC load at the output of the BiCMOS NAND input t_{pw} of 4ns, the output exhibits a delay (t_{hl2}) of 3.22ns instead of the normal propagation delay (t_{hl1}) of 0.79ns, for transistor Q_2 base open. Transistor Q_2 emitter open with input t_{pw} of 4ns exhibits stuck-at-1 behavior since before the input can make a transition to output low, the input undergoes transition to opposite logic level. If the input pulse width (t_{pw}) is made wider, the output would go to the other side of the logic threshold.

Transistor N_5 serves the purpose of discharging the base of Q_2 quickly to speed up the output Low to High transition [12]. Stuck-OPEN failure of transistor N_5 can be expected to result in delayed Low to High transition.

There is an interesting observation during output High to Low transition which needs mentioning. During output High to Low transition and with N_5 stuck-OPEN, it is observed that the output transition speeds up and causes enhanced dynamic current (I_{DD}) as shown in Figure 7. This can be explained as follows. Under normal operation with N_5 fault-free, any of the input vectors 00,01 or 10 causing output high(1), turns ON transistor N_5, thereby base of Q_2 remains discharged, keeping transistor Q_2 OFF. With transistor N_5 stuck-OPEN, any of the input vectors 00, 01 or 10 causing output high(1), will not be able to turn ON transistor N_5 and discharge base of Q_2. This causes some base bias to exist at the base of transistor Q_2. The base bias existing at the base of Q_2 may be sufficient enough to turn ON the device partially. Since the input vectors 00, 01 or 10 are intended to turn ON transistor Q_1 to provide output High, with Q_2 also partially ON, enhanced dynamic I_{DD} is observed. It may be noted that given sufficient time the current may decay to Zero. Hence, this probably cannot be termed as enhanced I_{DDQ} fault and so this is being termed as enhanced dynamic I_{DD} fault. I_{DDQ} testing may detect this fault as the enhanced dynamic I_{DD} current is about 2 orders of magnitude greater than fault-free current, just after the initial transient. Due to the existence of residual base bias on transistor Q_2, with input vector 11, turns transistor Q_2 ON faster than fault-free where no residual base bias exists. Hence, the speed up for output High to Low transition. The fault-free output High to Low transition delay is observed to be 0.90ns and with transistor N_5 stuck-OPEN 0.43ns, resulting in 0.47ns early transition than that of fault-free.

4 Comparison of the three logic families (TTL, CMOS, BiCMOS)

Summary of faulty behavior of double BJT BiCMOS NAND, TTL NAND and CMOS NAND is provided in Table 2, where transistor stuck-ON and stuck-OPEN faults have been examined. It was shown [9] that some of the open failures manifest themselves as delay faults. Apart from delay faults, we have observed sequential behavior, stuck at 0/1 and some faults that cause degraded signal levels. Most of the stuck-ON faults can be tested in a definite way

using power supply current monitoring. Some of the stuck-OPEN failures exhibit sequential behavior which require two pattern tests to detect the failure.

Table 2. Summary of faulty behavior in BiCMOS, CMOS and Bipolar NAND Gates.

Summary of BiCMOS, CMOS & TTL			
Type of faulty Behavior	BiCMOS NAND (D-BJT)	CMOS NAND	TTL NAND
Stuck-at-0 or 1	7.7 %	$\approx$ 25 %	> 90 %
Sequential Behavior	7.7 %	$\approx$ 25 %	—
Current Testable Fault	38.5 %	$\approx$ 50 %	—
Delay Fault	46.1 %	—	—

Conclusions that are drawn from the above summary is that almost $\approx$38.5% of the physical failures in D-BJT BiCMOS devices manifest as current testable faults, which does not include N_5 open fault manifesting as enhanced dynamic I_{DD} current. Hence, a good number of physical failures can be detected using current monitoring techniques [15, 16, 10]. A scheme for current monitoring in BiCMOS devices is presented in [11]. Apart from faults manifesting as abnormal I_{DDQ}, majority of failures manifest as delay faults.

5 Conclusions

Physical failures causing transistor stuck-OPEN in Double BJT BiCMOS devices were examined. In addition to sequential behavior observed in CMOS devices, BiCMOS devices also exhibit delay faults. Some of the stuck-ON faults can be detected by observing voltage level, however, power supply current (I_{DDQ}) monitoring would definitely detect the fault. Faulty behavior of the three different families, namely, TTL, CMOS and BiCMOS were compared to bring out the testability differences between the three logic families.

References

[1] A. R. Alvarez, *BiCMOS Technology and Applications*, Kluwer Academic Publishers, 1989.

[2] R. Haken, *'Process technology for submicron BiCMOS VLSI'*, IEEE Intl. Symp. on Circuits and Systems, pp. 1971-1974, 1990.

[3] B. C. Cole, *'Is BiCMOS the next technology driver?'*, Electronics, pp. 55-57, Feb. 1988.

[4] T. E. Mangir, *'Sources of failures and yield improvement for VLSI and restructurable interconnects for RVLSI and WSI: Part I-Sources of failures and yield improvement for VLSI'*, Proc. IEEE, Vol. 72, pp. 690-708, June 1984.

[5] J. A. Abraham and W. K. Fuchs, *'Fault and error models for VLSI'*, Proc. IEEE, Vol. 74, pp. 639-653, May 1986.

[6] J. Gailiay, Y. Crouzet and M. Vergniault, *'Physical versus logical fault models in MOS LSI circuits: Impact on the testability'*, IEEE Trans. Computers, Vol. C-29, pp. 527-531. June 1980.

[7] Y. K. Malaiya, B. Gupta, A. P. Jayasumana, R. Rajsuman, S. M. Menon and S. Yang, *'Functional Fault modeling for elementary static storage elements'*, Technical report, Dept. of Computer Science, Colorado State University, April, 1989.

[8] C. C. Beh, K. H. Arya, C. E. Radke and K. E. Torku, *'Do stuck fault models reflect manufacturing defects?'*, Proc. IEEE Test Conf., pp. 35-42, Nov. 1982.

[9] M. E. Levitt, K. Roy and J. A. Abraham, *'BiCMOS fault models: Is stuck-at adequate?'*, Proc. ICCD., pp. 294-297, Sep. 1990.

[10] S. M. Menon, Y. K. Malaiya and A. P. Jayasumana, *'Behavior of Faulty Single BJT BiCMOS Logic Gates'*, Proc. of the 10th IEEE VLSI Test Symposium, pp. 315-320, April 1992.

[11] A. E. Salama and M. I. Elmasry, *'Testing and Design for Testability of BiCMOS Logic Circuits'*, Proc. of the 10th IEEE VLSI Test Symposium, pp. 217-222, April 1992.

[12] Deierling, K., *'Digital Design'*, in BiCMOS Technology and Applications, A. R. Alvarez, Editor, Kluwer Publishers, pp. 165-200, 1989.

[13] S. M. Menon, A. P. Jayasumana and Y. K. Malaiya *'A Detailed Analysis of faults in Single and Double BJT BiCMOS Logic Gates'*, Technical Report, Dept. of Electrical Engineering/Computer Science, Colorado State University.

[14] *Fujitsu ECL & BiCMOS ASIC Selector Guide*, pp. 20-21, 1990

[15] Y. K. Malaiya and S. Y. H. Su, *'A New Fault Model and Testing Technique for CMOS Devices'*, Proc. Intl. Test Conference, 1982.

[16] Y. K. Malaiya, A. P. Jayasumana, Q. Tong and S. M. Menon, *'Enhancement of Resolution in Supply Current Based Testing for Large ICs'*, Proc. of IEEE VLSI Test Symposium, pp. 291-296, April 1991.

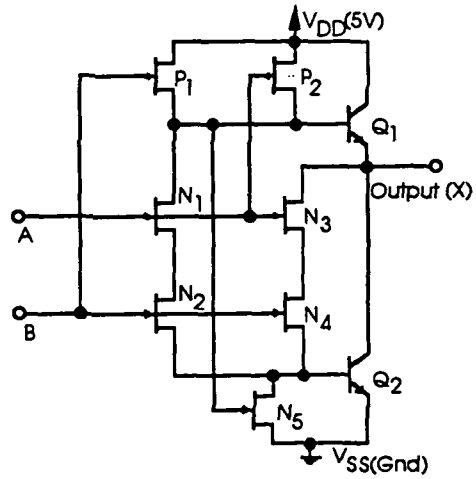


Figure 1: A D-BJT BiCMOS NAND.

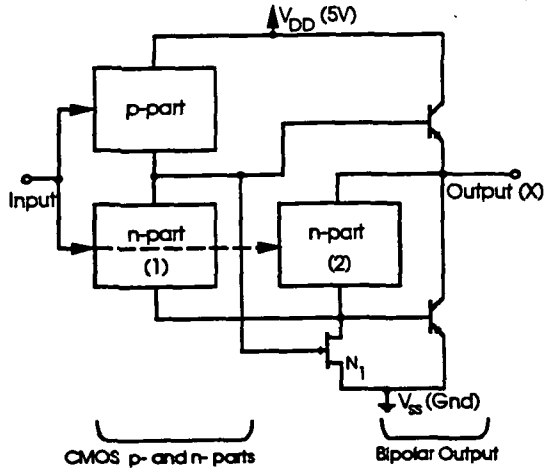


Figure 2: A general D-BJT BiCMOS device.

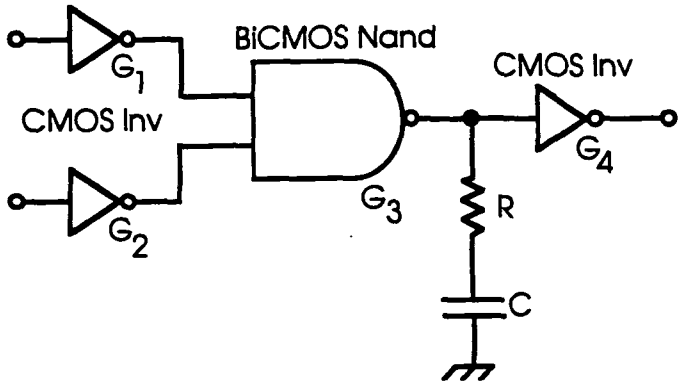
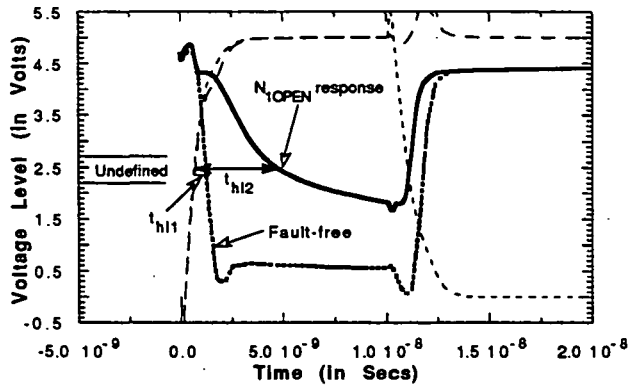
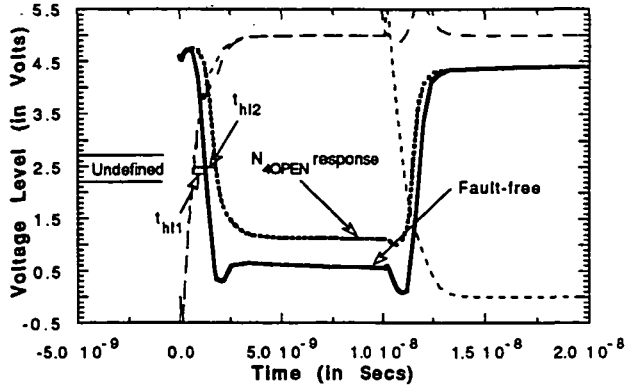
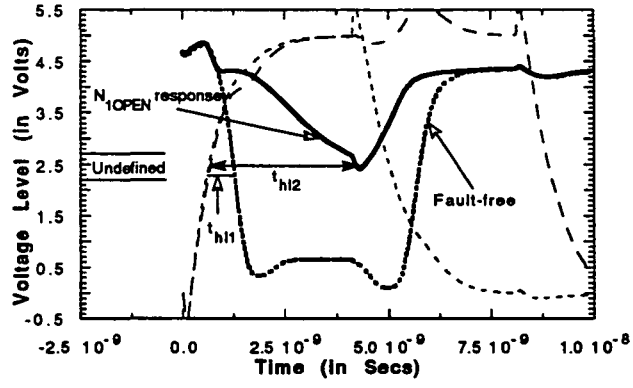
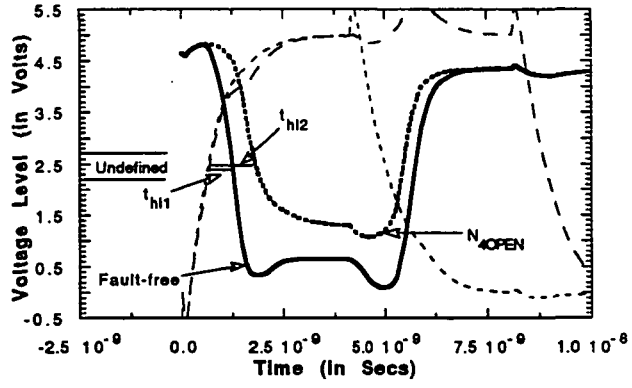


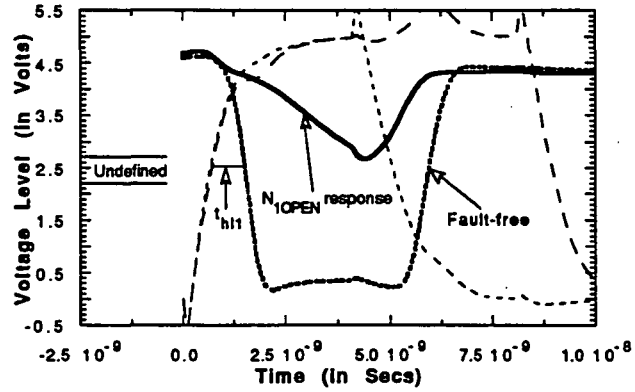
Figure 3: S-BJT BiCMOS NAND with CMOS inverter load and driver











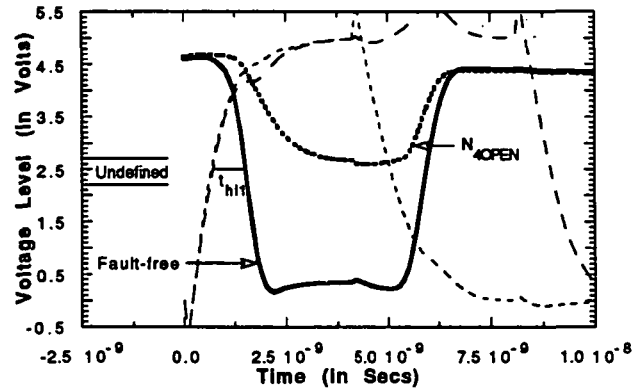
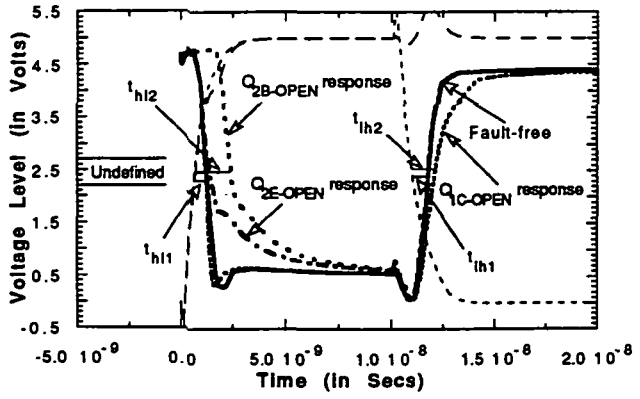
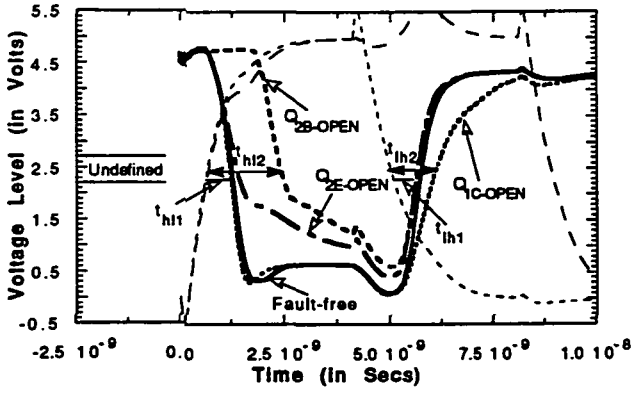


Figure 4: (a) BiCMOS response to N_1^{OPEN} with t_{pw}=10ns & One CMOS Load (b) t_{pw}=4ns & One CMOS Load (c) t_{pw}=4ns & RC Load.

Figure 5: (a) BiCMOS response to N_4^{OPEN} with t_{pw}=10ns & One CMOS Load (b) t_{pw}=4ns & One CMOS Load (c) t_{pw}=4ns & RC Load.





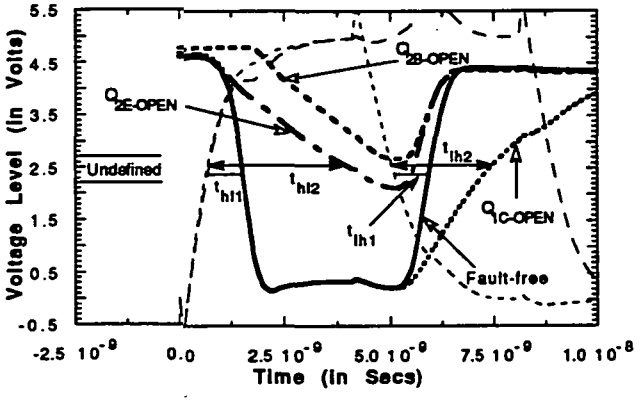
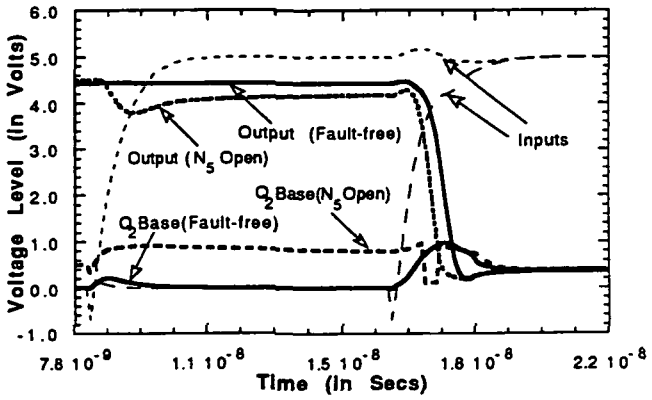


Figure 6: (a) BiCMOS response to $Q1_C^{OPEN}$ and $Q2_{BE}^{OPEN}$ with t_{pw}=10ns & One CMOS Load (b) t_{pw}=4ns & One CMOS Load (c) t_{pw}=4ns & RC Load.



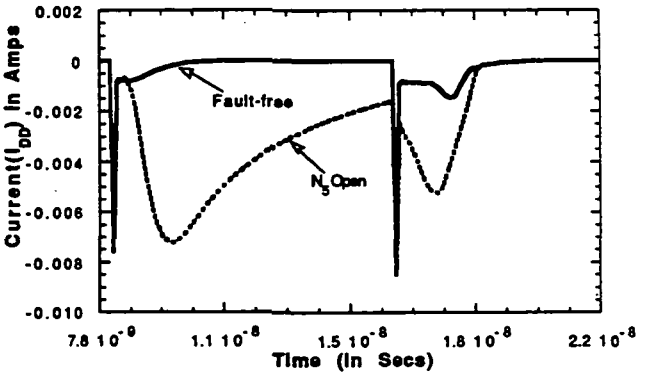


Figure 7: BiCMOS response to N_5^{OPEN} and Fault-free with RC Load (a) Voltage levels at BiCMOS output and Q_2 base (b) Plot illustrating enhanced dynamic I_{DD}.

On Fast Carry Select Adders

M. Shamanna and S. Whitaker
NASA Space Engineering Research Center for VLSI System Design
University of Idaho
Moscow, Idaho 83843

Abstract –**This paper presents an architecture for a high-speed carry select adder with very long bit lengths utilizing a conflict-free bypass scheme. The proposed scheme has almost half the number of transistors and is faster than a conventional carry select adder. A comparative study is also made between the proposed adder and a Manchester carry chain adder which shows that the proposed scheme has the same transistor count, without suffering any performance degradation, compared to the Manchester carry chain adder.**

1 Introduction

The trend in Very Large Scale Integrated (VLSI) circuits is toward ever increasing complexity at faster clock rates. This necessitates the use of high-performance arithmetic circuits. The computing performance of many systems is limited by the propagation delay through the arithmetic processing units. Very long bit length adders are becoming the norm. For example, in order to conform to IEEE standards dictates that a double-precision floating-point parallel multiplier needs at least a 108-bit adder[1].

Recently, a 112-b transmission gate adder has been proposed [1] using a Manchester scheme with a conflict-free bypass circuit. The authors concluded that their adder was 20% faster with less than the half the number of transistors compared to a conventional carry select adder. This paper presents a novel carry select adder architecture with the same transistor count and propagation delay as Sato's adder[1].

A discussion on conflict-free bypass circuits has also been presented in [1]. One of the bypass circuits discussed therein suffers from a potential failure. This issue has been addressed in Section 2 of this paper. The design of the proposed adder is presented in Section 3. Section 4, includes a comparative study of the proposed adder with the Manchester adder and the conclusions are presented in the last section.

2 Conflict-Free Bypass Circuits

Manchester adders have been primarily implemented in dynamic and domino logic families. It was shown in [2] that the incorporation of a bypass circuit decreases the carry propagation delay time in a Manchester adder. This circuit was also implemented in dynamic logic. The same bypass circuit design seems to have been adopted in [3], but transmission gates have been used in order to ensure that the circuit remains static. This is illustrated in Figure 1.

It has been stated by Sato et al[1] that "*this circuit solves the power consumption problem. However there are some transition phases in which the bypass circuit does not provide any*

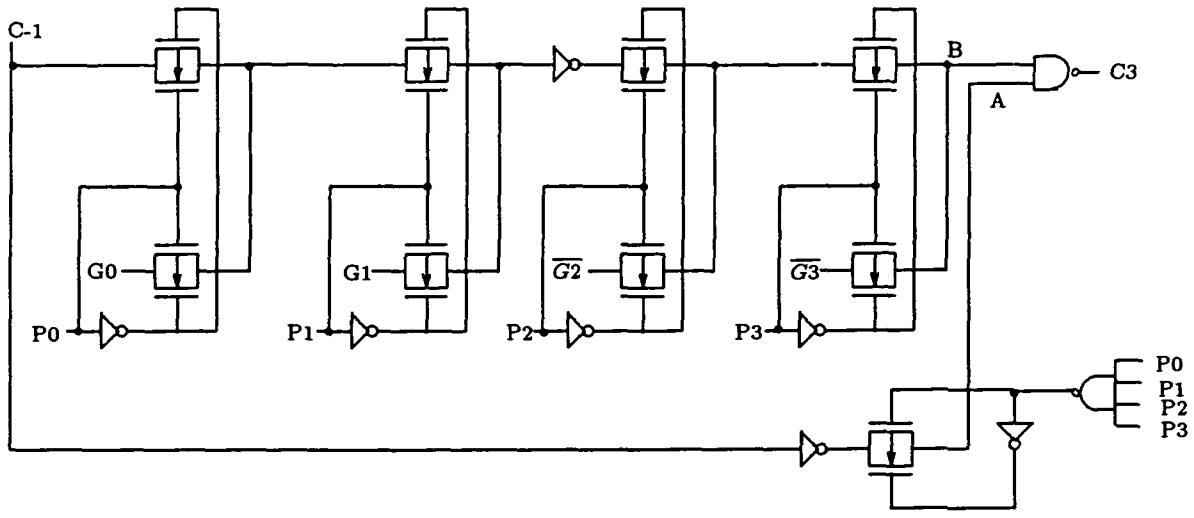


Figure 1: Manchester adder bypass circuit.

performance improvement". The authors further explain that, when the C-1 signal changes from low to high, the signal passes through the bypass circuit to node A and the signal C3 immediately responds. However when C-1 signal changes from high to low, the transition of C3 does not occur until the ripple carry signal reaches the NAND gate at B thereby yielding no performance enhancement.

In addition, under certain conditions the circuit may fail to function as desired. Referring to Figure 1, consider the case when C-1=1 and $P0 \ldots P3 = 1 \ldots 1$. Then A=0 and C3=1. For the next addition, if $P0 \ldots P3 \neq 1 \ldots 1$, the transmission gate which outputs A is disabled and so node A is left storing the logic level in the form of charge on the input capacitance of C3. Over time, if the condition $P0 \ldots P3 \neq 1 \ldots 1$ continues, this charge is affected by leakages through the reversed biased junctions associated with the transmission gate. Leakage paths exist to both VDD and VSS. The dominant path is determined by both layout and processing parameters. If the node A leaks to a logic 0 then C-3 would be erroneously evaluated as a logic 1 irrespective of the value of B. If node A settles midway between VDD and VSS, then there is excessive power consumption due to the dc path set up by turning on both the p-channel and n-channel transistors in C3 driven by A. The circuit would work as desired only if the node A remains at a logic 1. This is clearly undesirable since the length of time $P0 \ldots P3 \neq 1 \ldots 1$ is data dependent during operation leading to possible failure.

To remedy this problem, the circuit needs to be modified as shown in Figure 2, so that when $P0 \ldots P3 \neq 1 \ldots 1$, the P type transistor turns ON, forcing node A to a logic 1. It is noted that although this modification guarantees the circuit to function as desired but fails to provide any performance improvement. The transition of C3 still must wait for the ripple carry signal to reach the final NAND gate, when C-1 goes from high to low with $P0 \ldots P3 = 1 \ldots 1$.

Another solution which yields performance improvement without posing any circuit problems at the expense of extra hardware is shown in Figure 3 [1]. The three transmission gates controlled by T1, T2 and T3 perform an OR operation. The signals T1, T2 and T3 are given

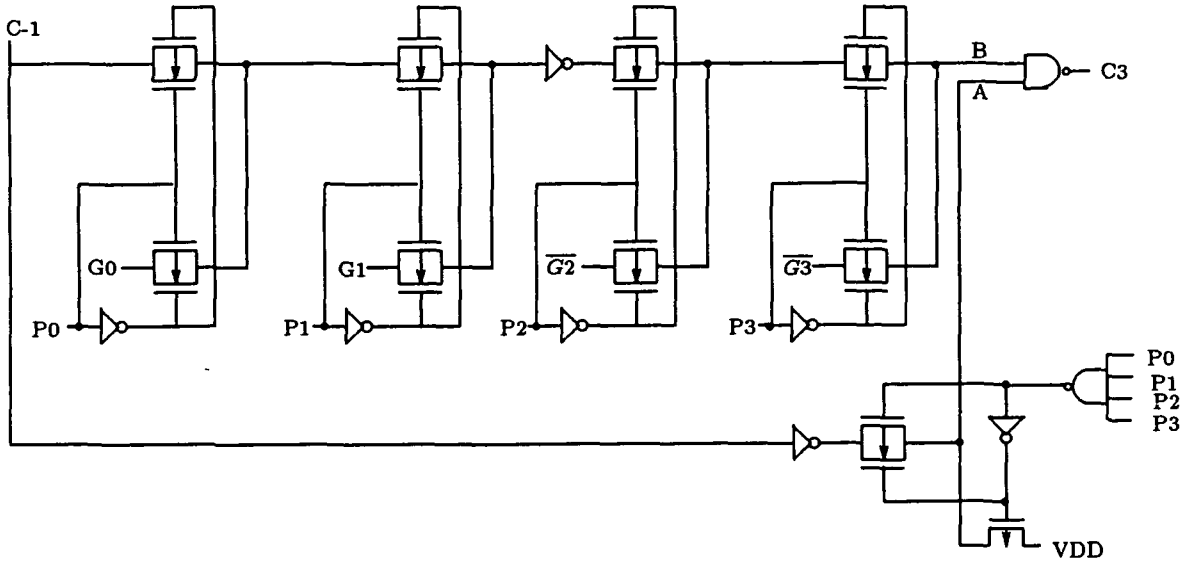


Figure 2: Improved bypass circuit.

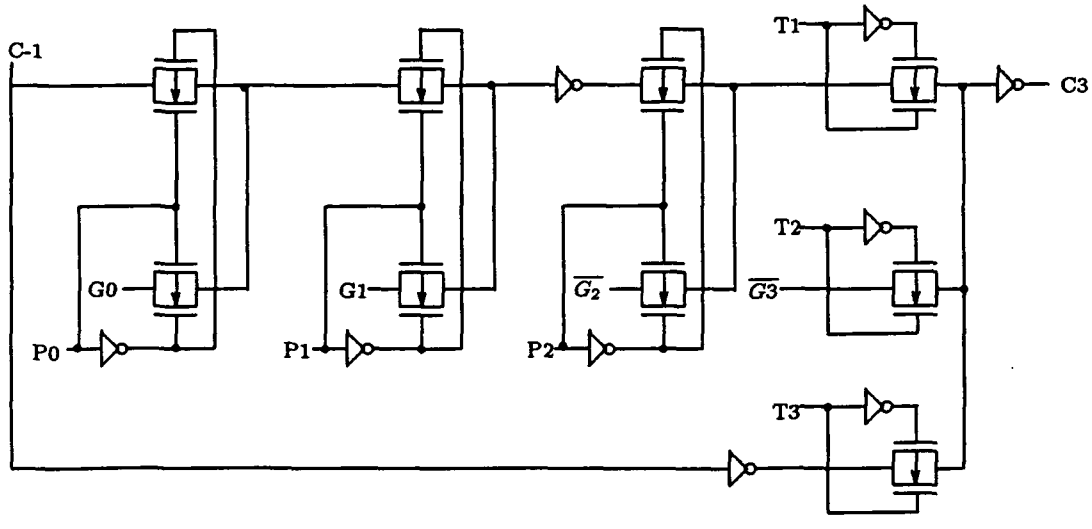


Figure 3: Performance enhanced bypass circuit.

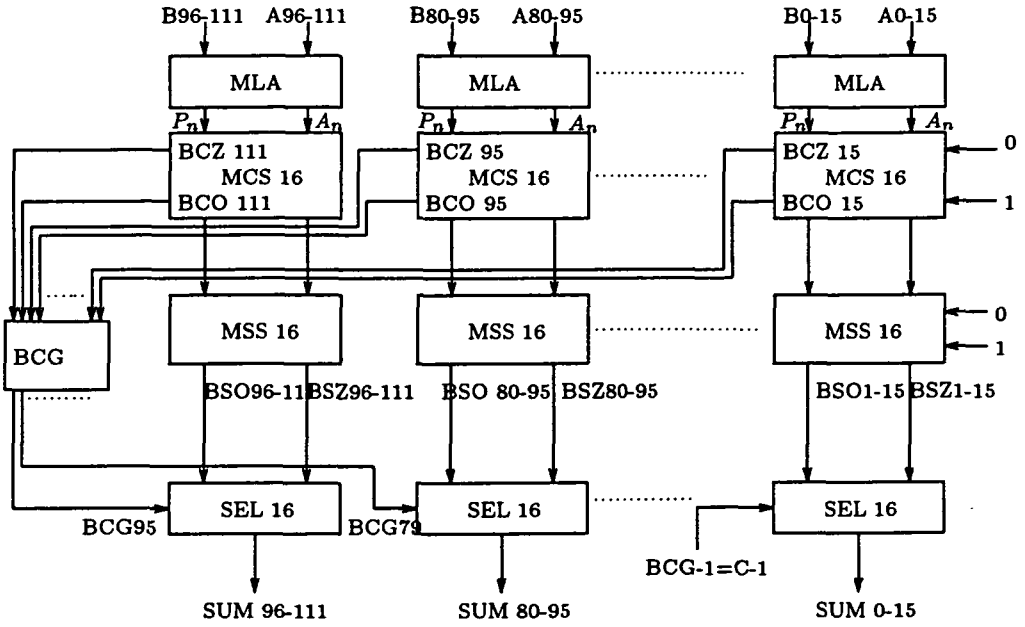


Figure 4: Block diagram for a 112-bit adder.

by :

$$T1 = P0' * P1' * P2' * P3$$
$$T2 = P3'$$
$$T3 = P0 * P1 * P2 * P3.$$

The principle used is one of the underlying principles of the BTS pass design methodology [4]. This scheme not only resolves signal conflict but also reduces power consumption.

3 Architecture

This section discusses the architectural details of the proposed Carry Select Adder. In order to provide an effective comparison between the proposed scheme and the Manchester carry adder scheme suggested in [1], we will follow the same design style and Module organization as in [1]. The proposed carry select adder is different from the conventional carry select adder since the carry path has been optimized for minimum delay. As a result of the optimization, the generate signal is no longer used in our scheme. Also, the adder cells, which usually have two transistor delays between carry in and carry out, have been redesigned to have only one transistor delay.

The 112-bit adder is divided into seven 16-b adder blocks. Figure 4 shows the block diagram of the adder. Figure 5 shows a 1-b circuit diagram of MS-16; It comprises of MLA, MCS16, MSS16 and SEL16 blocks (SEL 16 block is not shown in Figure 5). Transmission gates are employed to reduce area, power consumption and to reduce delay times. The MLA consists of an XOR circuit producing the carry propagate signal (P_n). MCS16 produces two carry-out signals BCZ_n and BCO_n associated with carry-in = 0 and 1 respectively. MSS16

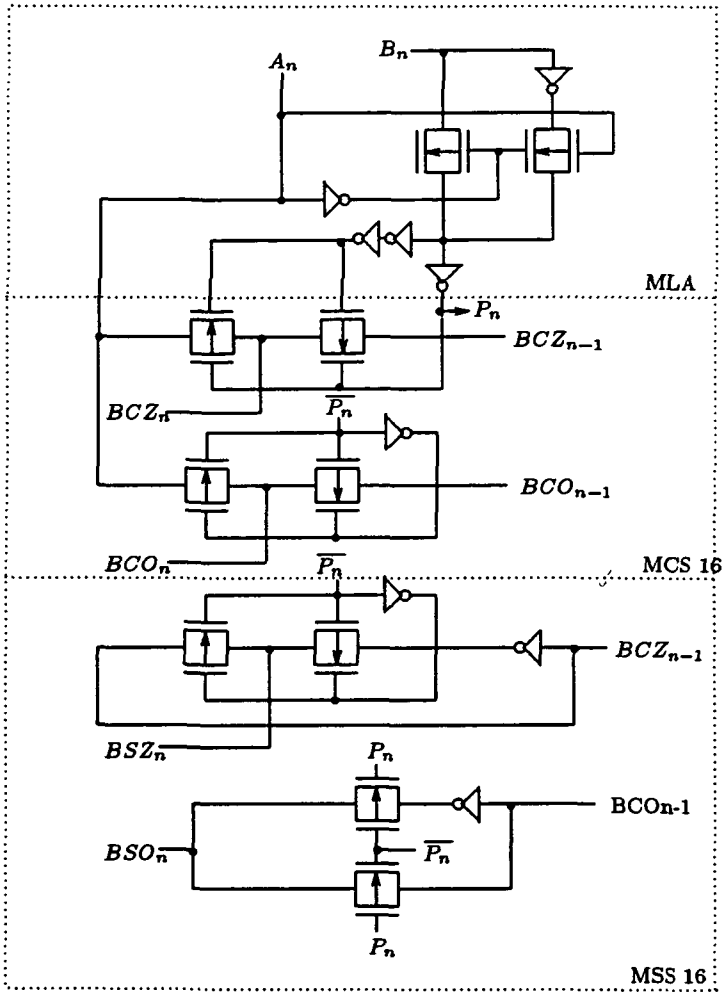


Figure 5: MS-16 circuit diagram.

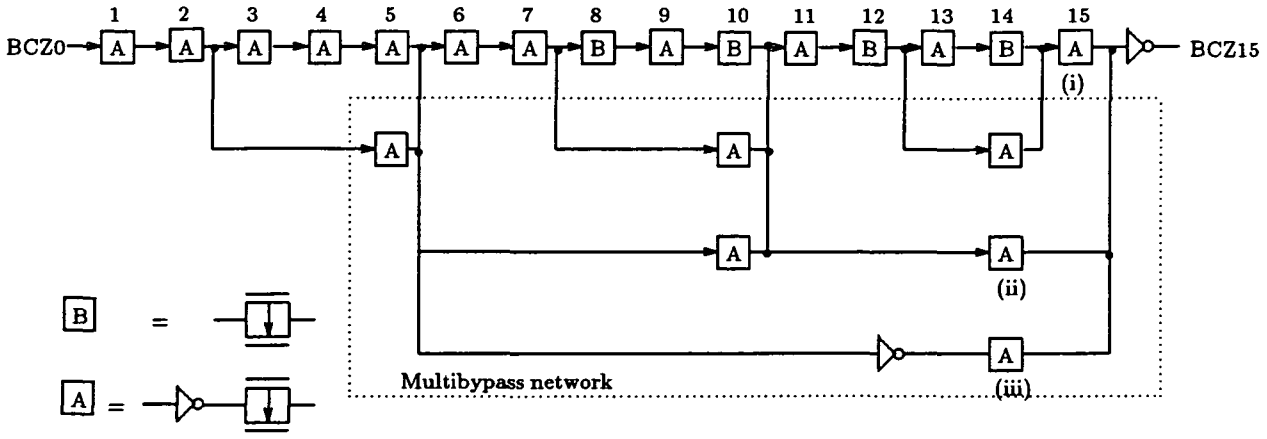


Figure 6: 16-bit multiple bypass circuit.

produces two sum signals BSO_n and BSZ_n corresponding to carry-in= 1 and 0 respectively. SEL16 selects either BSO_n or BSZ_n depending on the value of BCG_m. The SEL16 is a 16-b, two to one, Multiplexor constructed entirely from transmission gates. This is not shown in the Figure 5 for the sake of brevity. The critical paths for the MCS16 block are formed by the paths through which the signals BCO_n and BCZ_n propagate. These critical paths have sixteen transmission gates in a chain. In order to linearize the quadratic nature of the relationship between the length of the chain and the propagation delay, inverters are placed at suitable points. The placement of the inverters depends on the process parameters and is a fit candidate for global optimization. However, to achieve the high performance expected, the use of bypass circuits in the critical paths become a necessity. Figure 6, shows the 16-b multiple bypass circuitry generating the BCZ15 signal. This is similar to the bypass circuit in [1]. It is noted that the speed could be increased further by optimal placement of inverters. However for the sake of comparison, we shall restrict ourselves to the same inverter positioning as the scheme in [1]. It is seen that the propagation delay of BCZ15 consists of four or less transmission gates (excluding the inverter delays). The by-pass circuitry shown in Figure 6, provides conflict-free by-pass performance.

Similar circuitry ensures that a delay of less than four transmission gates for BCO15 signal. In Figure 6, the signal BCZ15 has three paths which are wired-OR (Input of the final inverter). The three transmission gates are controlled by the following three signals:

$$\begin{array}{ll} \text{(i)} & T2' * P15 \\ \text{(ii)} & T1' * T2' * P15 \\ \text{(iii)} & T1 * T2 * P15 \end{array}$$

$$\begin{array}{ll} \text{where} & T1 = P6 * P7 * P8 * P9 * P10 \\ & T2 = P11 * P12 * P13 * P14 \end{array}$$

Only one of the transmission gates is enabled at a time enabling conflict-free performance.

The BCG block provides the BCG signals for the terminal multiplexors (SEL16) in the 16-b MS16 slices. These BCG signals act as the control signals for the MUX'es to select the proper sum outputs. Depending on whether BCG=1 or 0, BSO1–15 or BSZ1–15 is output as

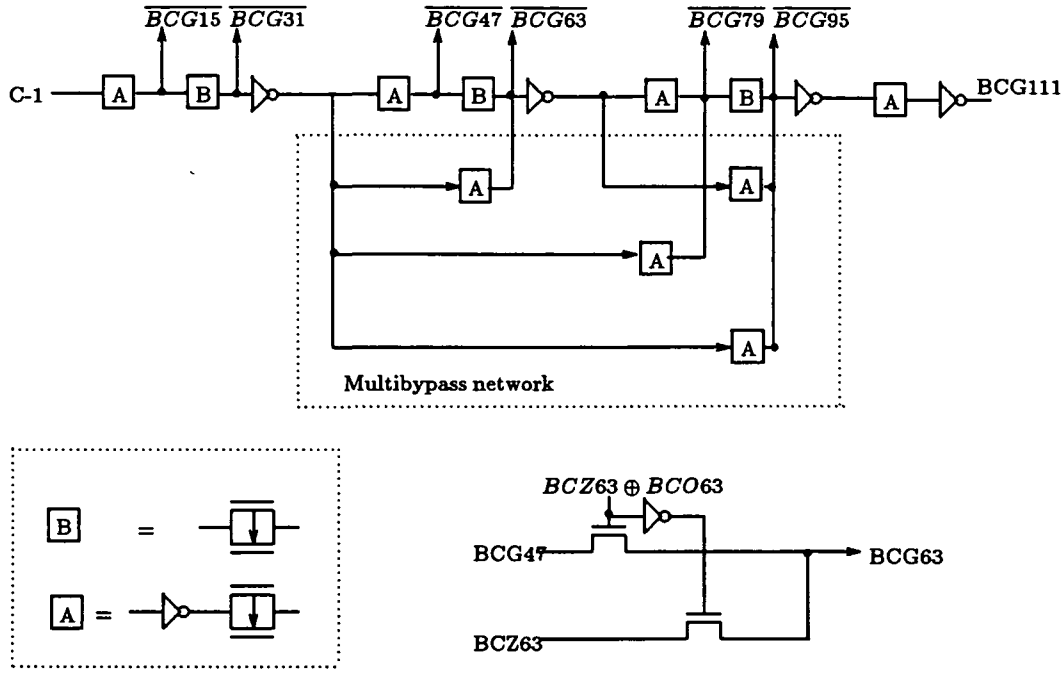


Figure 7: BC63 and BCG111 logic.

SUM0–15 respectively. The logic generating the BC63 signal is shown in Figure 7. Similar logic generates the rest of the BCG signals. The BCG block also uses a multi-bypass circuit as shown in Figure 7 to reduce the propagation delay.

4 Comparison

The transistor count of the one bit slices of the proposed adder and Sato's adder will be compared and it is shown that both have the same transistor count and propagation delays. Since the only comparison vehicle is Sato's adder, it will be referred to as the "existing adder".

The existing adder consists of seven 16 bit MS16 slices and a BCG block. Each MS16 slice, comprises of MLA, MB16 and a ML16 block. The proposed adder follows the same module organization and consists of seven 16 bit MSS16 slices and one BCG block. Each MSS16 slice is made up of MLA, MCS16, MSS16 and SEL16 subblocks.

The proposed MLA is the same as the MLA block in the existing adder. The MB16 of the existing adder has three transmission gates and one P transistor. The proposed MCS16 block has four transmission gates. The ML16 of the existing adder also has four transmission gates plus one NAND gate (four transistors). The MSS16 and SEL16 blocks have four and two transmission gates respectively. Running a global BCG'_m through the SEL16 block saves an inverter (two transistors). Accounting for the extra transistor used in our MCS-16 block, we are still left with one transistor less than the existing adder configuration.

Sato et al. concluded that their adder has half the number of transistors as that of a conventional carry select adder and is about 20% faster than the same. The proposed carry

select adder design has the same advantage in the number of transistors. Additionally, the critical paths of the proposed adder have the same number of the transistors as that of the existing adder. Hence, it should be possible for the proposed adder to be at least as fast as Sato's scheme. Since in the existing adder, the critical signal BC_m must pass through a NAND gate and an inverter before the final transmission gate is enabled to output SUM while in the proposed scheme, a similar signal BCG_m is fed directly to the final transmission gate the proposed scheme should have a speed advantage. The delay of both the signals BCG_m and BC_m are the same because both of them are produced by similar circuitry.

5 Conclusion

In summary, we have presented a new design for a carry select adder with conflict-free bypass circuit with optimal hardware. An existing bypass circuit was shown to suffer from potential failure and was modified to ensure satisfactory performance. A novel 1-b adder design was also proposed with an optimized carry path. Incorporating this design in the long bit adder along with the bypass circuit results in a saving of about 50% in transistor count and also increases the speed by about 20% compared to the conventional carry select adder designs.

References

[1] T. Sato et al., "An 8.5 ns 112-b Transmission gate adder with a conflict free bypass circuit," *IEEE Journal of Solid State Circuits*, Vol. 27, April 1992, pp. 657–659.

[2] P. K. Chan et al., "Analysis and Design of CMOS Manchester adders with variable carry skip," *IEEE Transactions on Computers*, Vol. 39, Aug. 1990, pp. 983–992.

[3] T. Kudo and T. Tokumaru, "Design of 32 bit high speed adder," *IEICE 70th Anniv. Nat. Conv. Rec.*, No. 284, 1987, p. 2–88.

[4] G. Peterson and G. Maki, "Binary Tree Structure Logic Circuits: Design and fault detection," *Proc. IEEE Int. Conf. Comp. Design: VLSI in Computers*, Oct. 1984, pp. 139–144.

This research was supported in part by NASA under Space Engineering Research Center Grant NAGW-1406.

Session 9
Cache Architectures

Joseph Feeley

A DRAM Compiler Algorithm for High Performance VLSI Embedded Memories

A. G. Eldin
Electrical Engineering Department,
The University of Toledo
Toledo, Ohio, 43606

Abstract - **In many applications, the limited density of the embedded SRAM does not allow integrating the memory on the same chip with other logic and functional blocks. In such cases, the embedded DRAM provides the optimum combination of very high density, low power and high performance. For ASIC's to take full advantage of this design strategy, an efficient and highly reliable DRAM compiler must be used. The embedded DRAM architecture, cell and peripheral circuit design considerations and the algorithm of a high performance memory compiler are presented .**

1 Introduction

The storage functions of a digital system can be implemented by one of two approaches. The first approach is illustrated in Fig. 1, where standard off-the-shelf memory chips are used. Chip A contains the logic functions whereas chip B provides the memory. This approach is generally adopted if the application does not justify the cost of developing an ASIC chip while the logic and processing functions can be economically and conveniently implemented by off-the-shelf components or semi-custom design using Gate Arrays that do not usually contain or enable the implementation of any sizable storage function efficiently. Additionally, the use of the off-the-shelf memory chips is the only choice in cases where the size of the

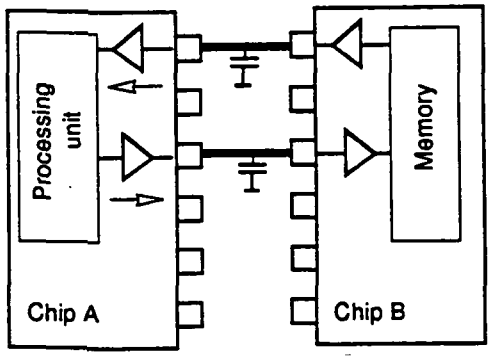


Figure 1: Additional delay due to chip capacitance

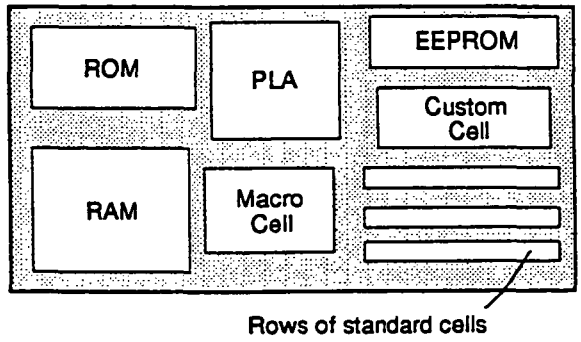


Figure 2: ASIC chip with embedded memory modules

required memory is too large to be integrated with the logic and processing functions on the

C-5

same ASIC chip. In this approach, there is an additional 40% delay in accessing the memory due to driving large off chip capacitive loads. The second approach is illustrated in Fig. 2 where the memory and logic functions are integrated on the same chip. Embedding memories increases the system speed by eliminating the Input/Output delays resulting from driving large off chip loads. Embedded memories have larger memory bandwidth (*number of bits accessed per second*) as there is no restriction on the number of I/O pins due to packaging requirements. This approach also increases the level of system integration. It provides lower power dissipation and system cost. To take advantage of the embedded memory approach, a reliable and efficient memory compiler (Fig. 3) takes the user's specifications such as type, size, organization, speed and aspect ratio and provides the verified design including the mask data for fabrication, the circuit model for accurate simulation, the logic model, delay parameters and logic symbol for logic and functional simulation and the layout boundary, input/output and signal tables for automatic placement and routing. Embedded/read write Random Access Memories are either static (SRAM) or dynamic (DRAM). A comparison

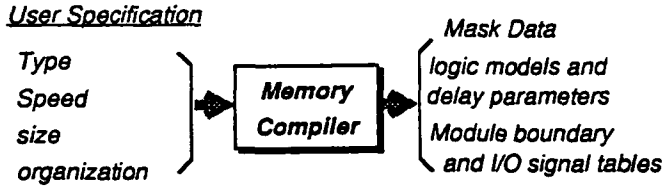


Figure 3: Memory compiler

between the embedded SRAM using the 6-Transistor (6-T) cell and the 1-Transistor (1-T) cell DRAM shows that the DRAM is approximately four times denser than SRAM while the SRAM is twice as fast as the DRAM [1]. The density advantage of the DRAM is due to the smaller cell size. The DRAM is inherently slower because of the charge sensing of the 1-T cell that requires a more critical timing to ensure that the different phases of the read/write cycle are completed in the proper sequential order. The embedded DRAM is critical for implementing such applications in a cost effective way. There are major differences between embedded DRAM blocks and standard off-the-shelf DRAM chips in terms of fabrication, memory size and organization, design strategy, testing, yield enhancement and applications. The fabrication of standard DRAM chips uses a process that is tailored to provide the maximum number of bits per chip. It offers limited choices in the organization (number of words x number of bits per word). The process provides either the trench or stacked capacitor DRAM cell which are three dimensional structures to realize the maximum capacitance per unit area of the chip. The increase in the process complexity is justified because of the very large production volume of such devices. At the present time, embedded DRAM's are fabricated using standard CMOS process. Three dimensional capacitors are not readily available and therefore the cell area of the embedded DRAM is significantly larger than that

of the standard DRAM. This is the reason that standard DRAM chips can have a capacity of up to 25 In this paper, The embedded DRAM design and the compiler's synthesis, verification and characterization algorithms are presented.

2 Embedded DRAM Design

The 1-T DRAM cell is shown in Fig. 4. The storage capacitance Cs = 50 fF. The cell is placed in an N-well to reduce the leakage and alpha particle induced soft error rate. The access transistor is a P-channel transistor. The maximum ratio of the bit line parasitic capacitance Cb to the storage capacitance is (Cb/Cs) = 6.5 which provides a minimum

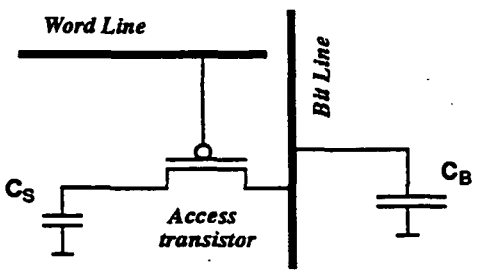


Figure 4: 1-T DRAM Cell

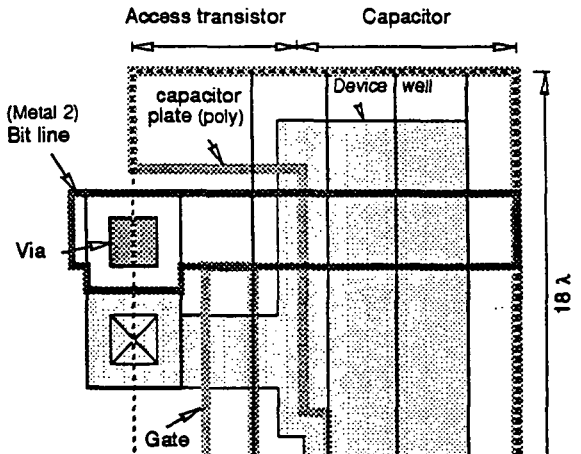


Figure 5: 1-T DRAM Cell layout

sense signal on the bit line of 200 mv. This ensures reliable fast operation. The capacitor plate (polysilicon) is connected to 0 volts and covers the whole array except where the access transistors are formed. This provides low resistance effective shielding of the storage nodes from different noise sources. The metal lines are made of the second layer of metal (Metal2) to reduce the parasitic bit line capacitance and the word lines use the first layer of metal (Metal1). The layout of the cell is illustrated in Fig. 5. Two cells share one drain contact to the bit line and one poly contact to the word line.

The basic block architecture of the embedded DRAM is shown in Fig. 6. It consists of two arrays. Only one array (when selected) remains connected to the sense amplifiers through the bed. All amplifiers switches are lumped together and assembled at the top of the array to improve the layout symmetry of the sense amplifier around the sense lines. This results in higher sensitivity. A new reference voltage generator design is based on the threshold voltage

of the CMOS inverter. The circuit is shown in Fig. 7. The reference voltage generator precharges the bit lines to 3.6 volts through the equalization switches. These switches and

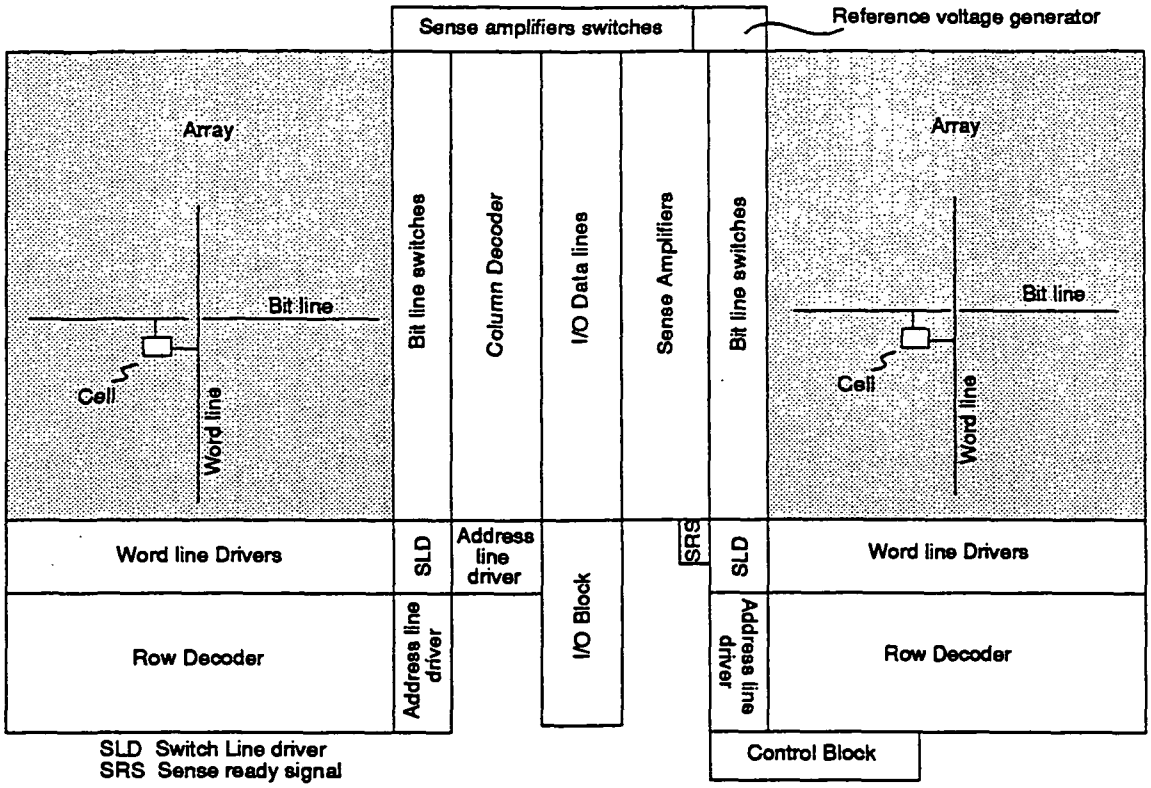


Figure 6: DRAM basic Block diagram

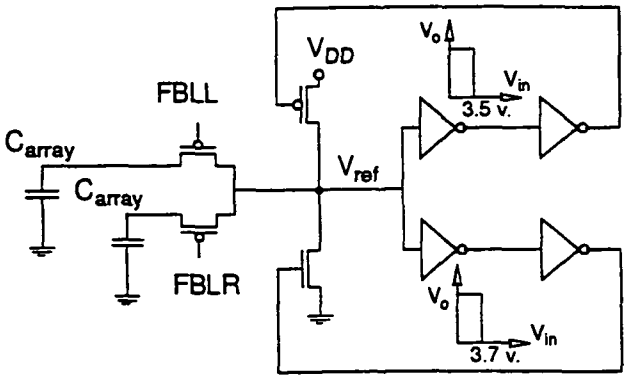


Figure 7: Reference Voltage Generator

the bit line capacitances are represented in Fig. 7 by one lumped transistor and one capacitor for each array. The generator has always at least one array connected to its output. These switches are normally ON except when the array is selected for reading or writing. The output voltage V_{ref} is compared to the threshold voltages of the two inverters which are 3.5 v. and 3.7 v. Either the P-ch or the N-ch transistor will turn on depending on whether V_{ref} is lower than 3.5 v or larger than 3.7 v. to charge or discharge the output to the desired

reference level respectively. This circuit accurately produces the desired reference voltage $V_{ref} = 3.6v$. with minimum power dissipation. This reference voltage is approximately the average value of the minimum and maximum voltage levels of the storage nodes which are 2 v. and 5 v. respectively. the input/output circuit uses a gated static latch to realize a 17 ns access time using the 1.5 μm CMOS process. The control block uses self timed circuits to generate the control and timing signals required to ensure that the different phases of the read/write cycle are completed in the proper sequential order for proper DRAM operation.

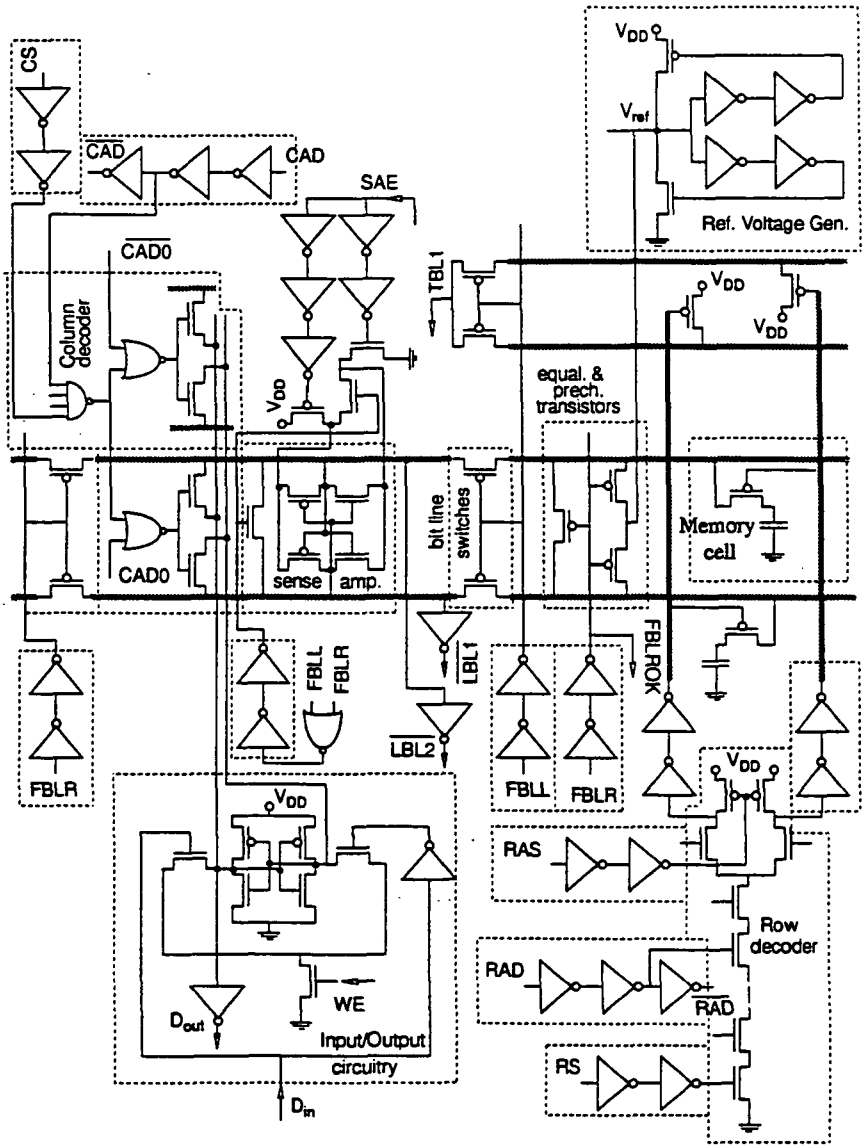


Figure 8: The circuit schematic of the basic DRAM block

The circuits of different blocks of the architecture are shown in Fig. 8 while Fig. 9 illustrates the circuit of the control block. The operation of the DRAM is summarized as follows.

First phase (RAS = 1)

The precharge transistors of the row decoder are turned off. Either FBLR or FBLL is logical 1 depending on the value of MSB being 1 or 0 respectively. This disconnects the precharging and equalization transistors of the selected array and floats the bit lines. The bit line switches of the unselected array are turned off thus disconnecting the bit lines of the array from the common sense amplifiers inputs (sense lines). The floating of the bit lines of the selected array is detected by the FBLROK or FBLLOK depending on selecting the right or left array respectively. Assuming that the right array is selected, FBLROK=1 results in RS=1 which enables the row decoder of the right hand side array. The selected word line potential is pulled down to 0 v. This selects all the cells on that word line. The charge transfer takes place between the cell capacitance and the associated bit line capacitance. The selection of the word line is detected by one of the P-chQQQ transistors on the top column of the array. The turned on transistor on the top of the array causes the sense amplifier enable signal to be SAE=1 after sufficient delay to ensure that the charge sharing mechanism is completed before the sense amplifier is enabled. The turning on of the sense amplifier is detected by a pair of inverters connected to the bottom pair of bit lines. This signal enables the column decoder (CS=1) after making sure that the sense amplifiers have latched to the correct states stored in the memory cells of the selected word line. The column decoder connects the output of the selected sense amplifiers of the I/O data lines through the pass transistors of the decoder. The data of the sense amplifiers overwrite the previous data in the weak latches of the I/O circuitry and in the same time restore the data in the selected memory cells. In the write cycle, the column decoder is enabled immediately after RAS=1. The data to be written (D_{in}) is routed to the selected sense lines and override the much weaker signal from the memory cells. When the sense amplifiers are enabled the D_{in} are written into the selected memory cells.

Second phase (RAS = 0)
The row decoder is disabled (RS=0) and the precharge transistors are turned on to unselect the word line. Once all the access transistors of the memory cells are turned off, the sense amplifiers are disabled (SAE=0). The bit lines are precharged end equalized (FBLR=0) and the bit line switches of the unselected array (left array) are turned on to reconnect the bit lines to the common sense lines. In the same time CS=0 and the column decoder is disabled. This disconnects the I/O data lines from the sense lines. A new cycle can be initiated after the bit lines are equalized at the same reference potential V_{ref}.

3 The DRAM Compiler Algorithm

The compiler performs several functions that include DRAM module synthesis, design verification through DRC and simulation, accurate DRAM module characterization and models for higher level simulation, placement and routing. The correctness of the synthesis algorithm is critical to ensure the proper functionality and high performance of the variable size DRAM while realizing the highest density. This is because the DRAM operation requires accurate timing of sequential control signals and a certain maximum limit of the ratio between the bit line capacitance and the storage capacitances as was detailed in Section 2. The signals delay and the value of the sense signal are strong functions of the different line

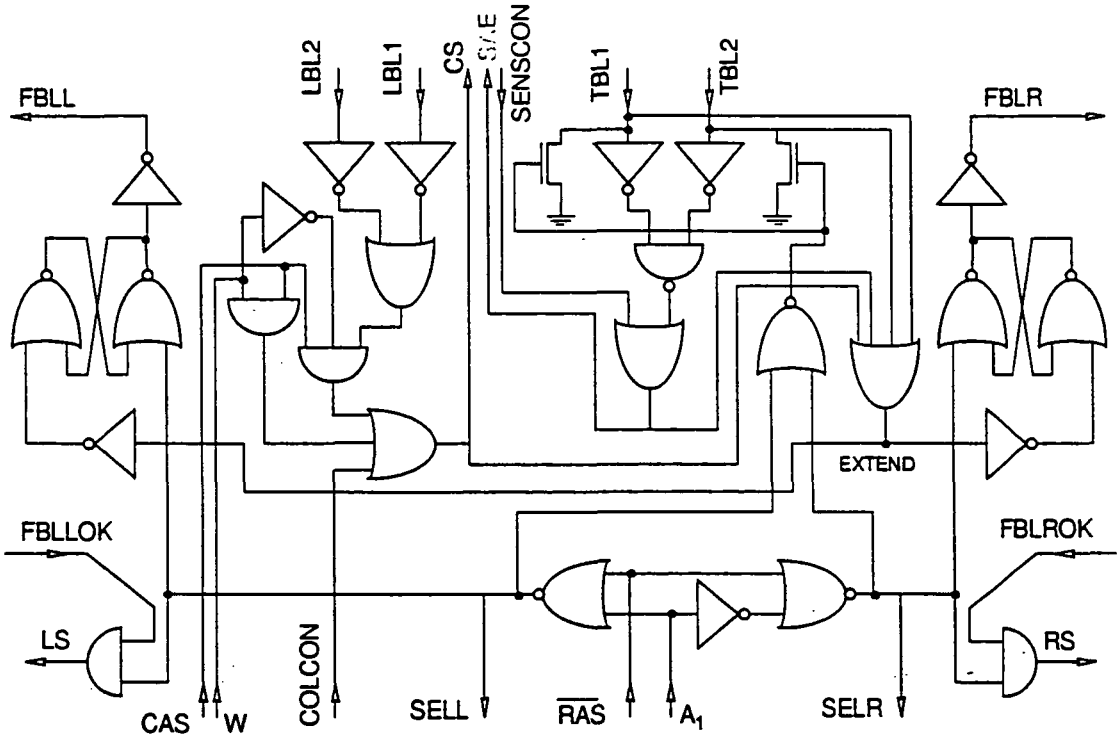


Figure 9: The circuit schematic of the control block

capacitances which in turn heavily depend on the synthesis algorithm.

The algorithm generates DRAM modules that have a size range between 256 bits and 262144 (256 kbits). The maximum limit is based on the assumption that the maximum chip size is 1 cm2 and that the DRAM occupies 80% of the chip area using a 1.2 μm CMOS process. The maximum size also fulfills the storage requirements of most ASIC applications. Below the minimum DRAM size of 256 bits, the DRAM advantage of area savings is diminished and the use of the SRAM is a better choice. The DRAM is built of basic DRAM blocks. Each basic block (Fig. 6) has a maximum of 32768 bits (32 kbits). A block decode logic is used to select on out of a maximum of 16 identical basic blocks. The relative placement of these blocks is determined by the required overall DRAM aspect ratio, the number of generated blocks and the aspect ratio of each block. Table 1 shows the minimum and maximum size and organization parameters of the DRAM module. Table 2 shows the maximum and minimum values of the parameters of the basic DRAM block.

The method of calculating the number of blocks, rows and columns per block, can be explained with reference to Fig. 10. The x and y axes represent the minimum number of basic DRAM blocks required to realize the specified number of words (WORDS) and bits per word (BITS) respectively. The numerical entities in the table indicate the minimum number of identical basic DRAM blocks required for the implementation of a particular DRAM design (module). The number of blocks (BLK) is obtained by multiplying the number of bit blocks (BLK1) by the number of word blocks (BLK2). The word block is 256 words and

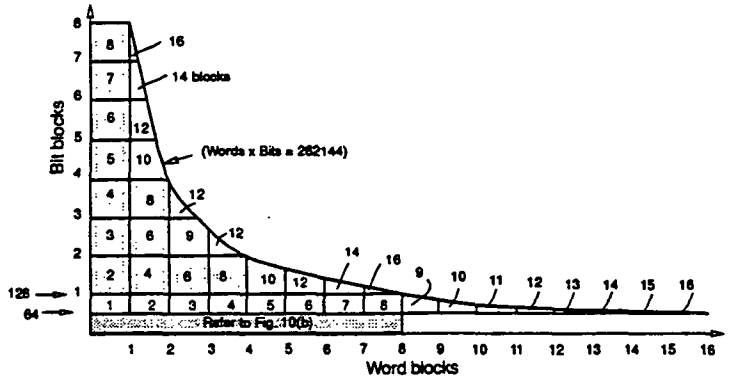
Table 2

	Maximum	Minimum	Increment
Number of basic blocks	16	1	1
Number of rows per block	256	4	2
Number of columns per block	128	16	2

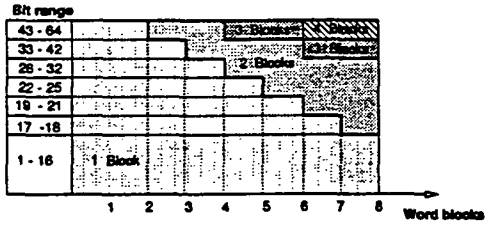
Table 1

	Total Size	Total number of words (WORDS)	Total number of bits per word (BITS)
Maximum	262144	262144	1024
Minimum	256	4	1

the bit block is 128 bits. Fig. 10a is used if (BITS $>$ 64) while Fig. 10b for (BITS $\leq$ 64). For example if the required DRAM module is organized as (280 words $\times$ 300 bits), then BLK1=3, BLK2=2 and the total number of blocks is BLK=6. Each block is organized as (140 words $\times$ 100 bits). The DRAM module is built as shown in Fig. 11 where one block select address line (BS) connects the bit lines either from the blocks labeled (A) or (B) to the I/O data bus. BS is also used to activate the selected blocks. When the number of bits per word (BITS) is larger than 64, no column decoder is required. An other organization example that uses the table of Fig. 10b is (1300 words $\times$ 28 bits) which requires 2 blocks (BLK=2).



(a)



(b)

Figure 10: Number of basic blocks for all possible DRAM modules

The algorithm of calculating the number of basic blocks, number of rows (ROWS) and columns (SA) per block is illustrated in the flow chart of Fig. 12. All variables are integers. The algorithm first calculates the minimum number of blocks (BLK1) required to satisfy the

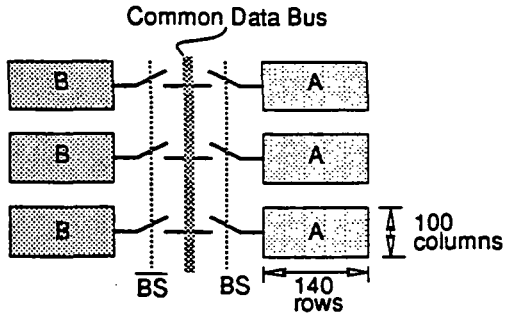


Figure 11: DRAM example module organization

specified number of bits per word (BITS). If (BITS > 64), no column decoder is required (K=1). The minimum number of blocks (BLK2) required to satisfy the specified number of words (WORDS), is calculated. Thus the total number of blocks (BLKS) is determined. The number of rows per block (ROWS), number of sense amplifiers, **which is equal to the number of columns**, per block (SA) and the number of bits per block (IO), **which is equal to the number of I/O data lines**, are calculated. If BITS $\leq$ 64, the value of k is calculated where

$$k = number\ of\ column\ decoder\ address\ lines\ (n) + 1 \tag{1}$$

This initial value of k gives the maximum number of sense amplifiers and the minimum number of rows per block. To optimize the aspects ratio of the basic block, the algorithm attempts to satisfy the condition (2 $\times$ SA $\leq$ ROWS). This means that the total number of memory cells of one array in the x-direction equals to that in the y-direction. if this condition is satisfied, the array's aspect ratio equals that of the memory cell. This optimizes the speed of operation by providing balanced capacitive loads on the word and bit lines and avoids the generation of DRAM blocks with uncommon dimensions that may be difficult to fit in ASIC chip. This condition is satisfied by iteratively reducing the number of the sense amplifiers while increasing the number of rows. Then number of total blocks is then calculated. A simplified flow chart is shown in Fig. 13 that highlights the basic synthesis and verification steps of the compiler algorithm. The specified size and organization are checked against the values of Table 1 to determine whether they are within the supported limits. The size and number of basic blocks, number of rows and columns per block and the size of the column decoder are determined as explained earlier with reference to Fig. 12. The size of the basic block row decoder in bits (m) is related to the number of rows per block as,

$$2^{m-1} \leq \text{ROWS} \leq 2^m \tag{2}$$

The number of I/O data lines and associated circuits (IO) per block is given by ($10 = \frac{SA}{K}$). The sizes of the transistors of the reference generator are scaled according to the total number of bits per block. this for area efficiency and circuit stability. Similarly, the sizes of the sense amplifiers switches, word line drivers, column address drivers and drivers of the bit line switches and equalization and precharge switches are scaled proportionally with the number of sense amplifiers per block. The size of the row address driver is scaled proportional to the

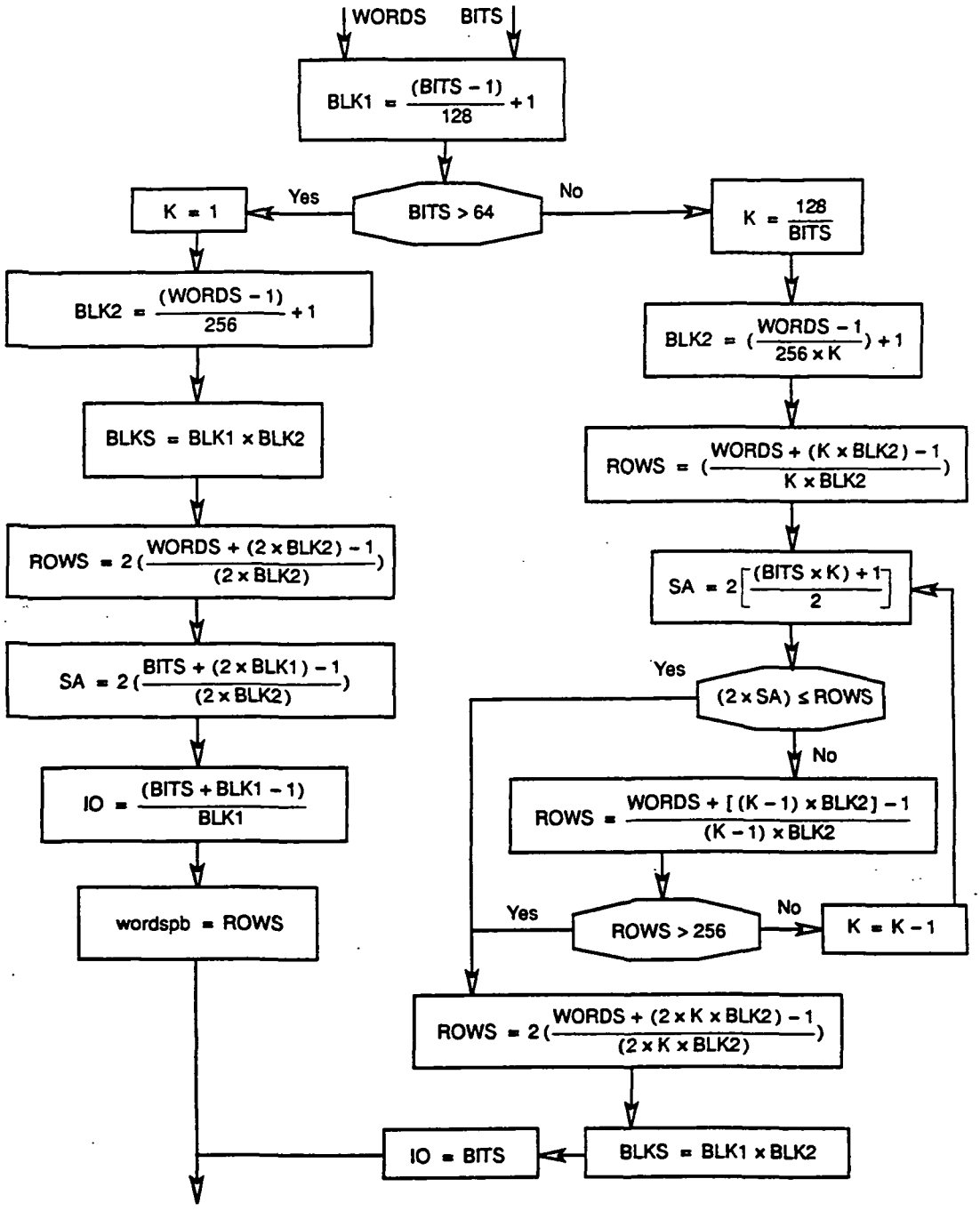


Figure 12: Algorithm for determining the number of basic blocks and its organization

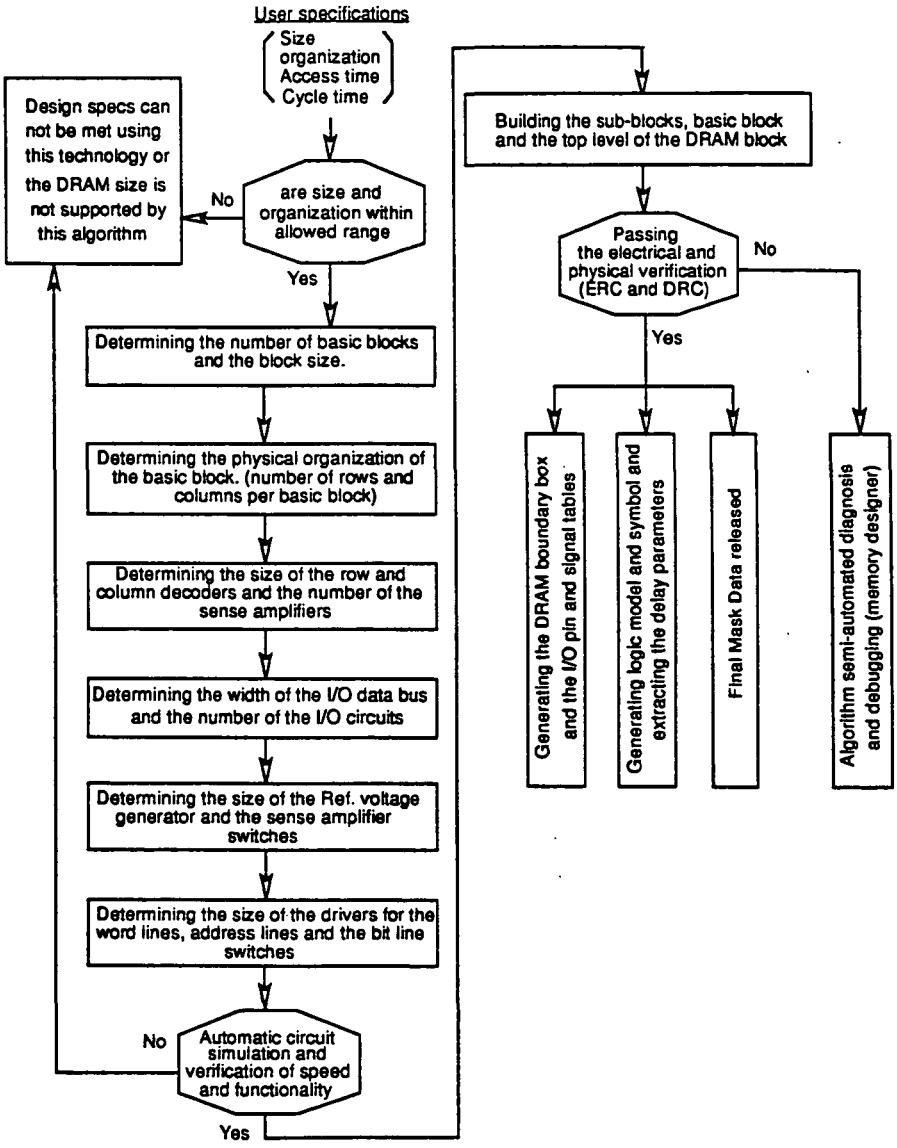


Figure 13: Summary of the DRAM compiler algorithm

number of rows per array. For each of those variable size circuits, a lower level compiler is used. The scaling of such sub-circuits achieves area efficiency, performance consistency and reliable operation of different DRAM sizes and organizations.

An automatic circuit simulation (using SPICE) and analysis are performed on the resulting DRAM module. The skeleton spice file contains the circuits of Fig. 8 and Fig. 9. The sub-circuits are only extracted once from the layout to reflect parasitic capacitances. The loading effect due to the variable size is modeled using scalable load cells on the word and bit lines. The results of the simulation are automatically analyzed to verify the completion of different phases of the cycle in the proper sequential order and the proper operation of the generated module. It also measures the access and and cycle times and other internal critical delay times. The compiler is capable of generating 20,971,360 different DRAM modules. Therefore, this accurate method of circuit design verification is essential. The following step is to proceed with the building of the sub-blocks, basic DRAM block and the top level of the DRAM module. The generated layout is automatically checked hierarchically for electrical and design rule violations The interior bulk of the cell arrays is excluded to save in processing time and storage requirements. Finally, the mask data are released. The logic model and symbol are generated and delay parameters are accurately extracted from the spice simulation data. The DRAM module boundary box and I/O pin and signal tables are generated. Table 3 summarizes the results obtained using this synthesis algorithm for several DRAM modules. It compares the user's specific specifications with the actually generated modules.

Table 3

User specifications			Generated DRAM Module											
Size (bits)	Organization		BLK1	BLK2	BLKS	SA sense amps /block	ROWS rows per block	k	words per block	WORDSG Total words	IO Bits per block	Generated size	efficiency (η) %	Array Aspect Ratio
	WORDS	BITS										SizeG (bits)		
8073	351	23	1	1	1	46	176	2	352	352	23	8096	99.72	0.87
262144	2048	128	1	8	8	128	256	1	256	2048	128	262144	100	1.67
127167	1311	97	1	6	6	98	220	1	220	1320	97	129360	98.3	1.49
259328	256	1013	8	1	8	128	256	1	256	256	127	262144	98.9	1.67
27675	1025	27	1	2	2	82	172	3	516	1032	27	28208	98.1	1.59
128000	250	512	4	1	4	128	250	1	250	250	128	128000	100	1.7

The efficiency (η) of the generator is defined as $\eta = (\frac{size}{sizeG} \times 100\%)$ where $size$ and $sizeG$ are the total specified and generated DRAM module sizes in (bits) respectively. It reflects the compiler's ability to generate area efficient DRAM modules. It is a measure of the compiler's flexibility and the silicon area utilization of the DRAM. The table also lists the aspect ratio of the array for an aspect ratio of the DRAM cell of 1.67.

4 Conclusions

The DRAM compiler presented is capable of generating 20,972,360 different DRAM modules that cover a wide range of sizes, organizations and support very large number of applications.

The use of the described architecture, memory cell, peripheral circuitry, self-timed control signals, variable size sub-circuits, very flexible synthesis algorithm, automatic verification and accurate characterization results in highly area efficient, reliable, dense and high performance DRAM modules.

References

[1] A. G. Eldin "High Speed VLSI SRAMs and DRAMs Evolution and Trends" Proceedings, The International Conference on Microelectronics, Cairo, December 1991.

A Novel Cache Mechanism

J. A. Gunawardena
Faculty of Engineering
University of Peradeniya, SRI LANKA
+94 (08) 88185

Abstract - **This cache mechanism is transparent but does not contain associative circuits. It does not rely on locality of reference of instructions or data. No redundant instructions or data are encached. Items in the cache are accessed without address arithmetic. A cache miss is detected by the simplest test; compare two bits. These features would result in faster access, higher hit rate, reduced chip area and less power dissipation in comparison with associative systems of similar size.**

1 Introduction

Most computer programs exhibit the property of temporal locality; i.e.,

- Much of the code consists of repetitive loops.
- The same data is accessed several times within a short interval of time.

Often, they also exhibit the property of spatial locality; i.e.,

- Instructions executed in sequence occupy a contiguous area of the main memory.
- Successive instructions access data in a contiguous area of the main memory.

All cache systems rely on temporal locality to meet the basic objective of reducing accesses to the main memory. Conventional cache systems that rely on temporal locality alone must employ fully associative access of the cache. But associative hardware is complex and costly. Hence, most conventional cache systems depend on the property of spatial locality as well. As a result the hit rate decreases when the program exhibits poor spatial locality.

The system proposed in this paper uses no associative hardware at all. Yet, its performance is totally independent of spatial locality of code and data in the main memory. Hence its performance would be as good as or better than that of a fully associative cache of similar capacity.

If a cache memory system is to be completely transparent to software it should cater to the following two attributes of programs:

- A datum may be operated on by several distinct instructions.
- An instruction may receive control from that assigned to the preceding location in the main memory or from a branch instruction.

The following sections show how a hypothetical computer utilizing the proposed cache mechanism provides for these attributes while minimizing accesses to the main memory. For clarity here, the hypothetical computer is assigned a simple architecture. However, with suitable modifications, the proposed mechanism will complement most other features of modern computers including those of multi-processor systems.

2 General Features

This cache system is completely transparent to software. Programs are compiled or assembled and loaded onto a random access main memory as if there were no cache. In contrast to conventional systems, there is no restriction on the location in the cache at which an item may be entered. The Execution Unit (EU) does not perform address arithmetic for instructions or for scalar data. This work is done by a Cache Management Unit (CMU) and is required only when a miss occurs. The miss detection test is the simplest possible; compare two bits.

The cache is accessed as a random access memory. When decaching and encaching, it is operated as a normally full cyclic FIFO buffer. All decached items are returned to the main memory. Thus, only one copy of each item is maintained. Instructions and scalar data are placed in the same cache. Arrays may be placed in the same cache provided that each array is taken in its entirety. Arrays may also be placed in a separate cache. These conditions are necessary for the proper functioning of the miss detection test.

3 Structure

Fig. 1 shows the functional components of the proposed system. For clarity at this stage we consider a computer in which all instructions and data are of fixed length p bits, with each instruction or datum occupying one **word** of the main memory.

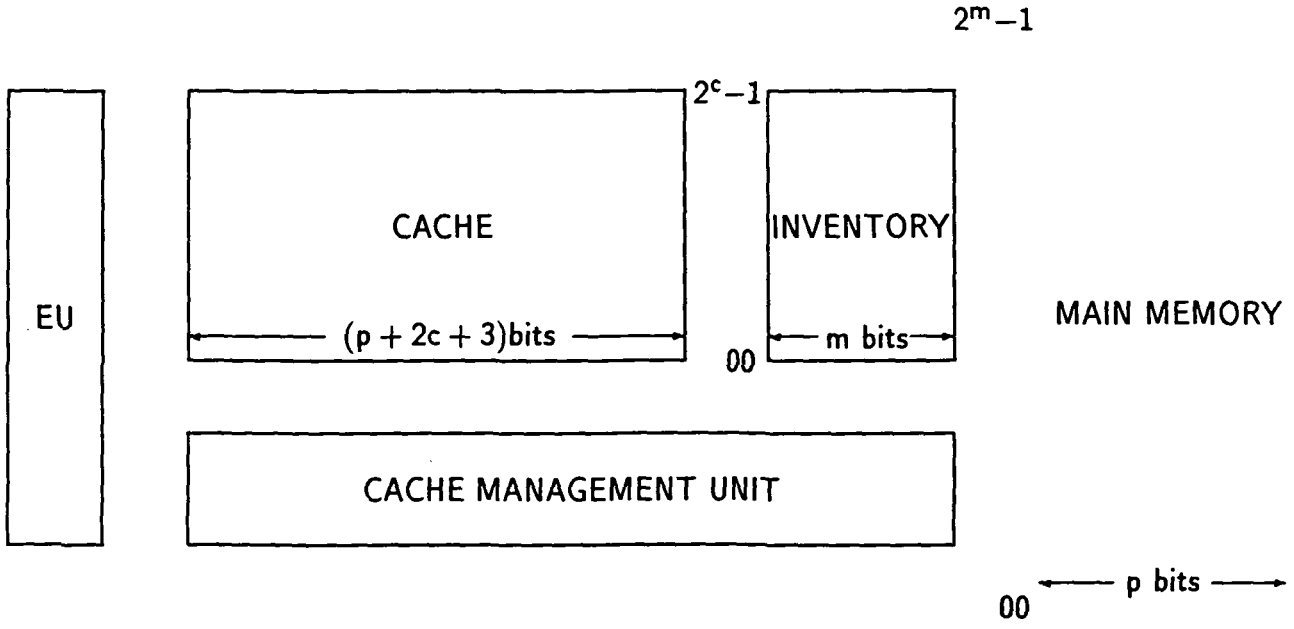


Fig. 1 Functional components of the cache system.

The cache consists of 2^c **lines**; i.e. a line has a physical address of c bits. We use the term "physical address" because, at a later stage we need to introduce the term "logical address" with respect to cache lines.

The cache contains both instructions and data. Each cache line contains one **item**, viz., one instruction or one datum. The cache is normally full. Hence the current item in a line must be decached before a new item can be encached there. A register contained in the CMU holds the address of the line from which an item will be decached next. This is referred to as the Decache Address (DA). The cyclic FIFO rule is implemented by incrementing DA after each encache operation, with wraparound when necessary.

For this simple model it will be shown that each line should contain $(p + 2c + 3)$ bits. Of these, p bits are required for the encached item. The additional $(2c + 3)$ bits comprise fields that hold information required for non-associative operation of the system. The function of each of these fields will be described as the need arises.

The inventory holds the main memory address of each encached item. Thus a cache address refers to a line in the cache as well as the corresponding **entry** in the inventory. If the main memory contains 2^m words then each entry in the inventory takes m bits.

The CMU encaches each item when it is first required by the EU. Hence, the EU needs to communicate with the cache only, unless a miss occurs. In this event it signals the CMU which in turn accesses the cache, inventory and main memory to take corrective action.

4 The Contents of the Cache

An instruction must be in the cache before its execution may commence. Similarly, a datum must be in the cache before it may be operated on.

Consider the execution of a register/memory instruction in which the register is implied in the opcode. Assume that the instruction has been just encached at CACI, the Cache Address of the Current Instruction, and that the datum and the next instruction are in the main memory.

The EU is unaware that the datum is not in the cache and tries to access it using the content of a field from the extra $(2c + 3)$ bits at CACI. The exact line accessed is irrelevant at this point; it is sufficient to note that, because the datum is not in the cache, the miss detection test must return a miss.

The current instruction which is in the line at CACI consists of the opcode and the operand as obtained from the main memory (Fig. 2). The operand is the Main memory Address of the Datum (MAD). The CMU gets MAD from the line at CACI, accesses the main memory, obtains the datum and encaches it in the line at DA. Note that MAD and DA are not related.

The CMU enters DA in a field set apart for it from among the extra $(2c + 3)$ bits in the line at CACI. This field is referred to as the Cache Address of the Datum (CAD). The EU hereafter uses CAD to access the datum in the cache.

For this particular example, since the current instruction and the datum were encached in immediate succession, we have

$$\mathrm{CAD} = \mathrm{CACI} + 1 \tag{1}$$

This relation will not hold true in general because the datum may have been encached earlier for it to be operated on by a preceding instruction. Hence, the pointer CAD in a field in the line at CACI is a necessary requirement.

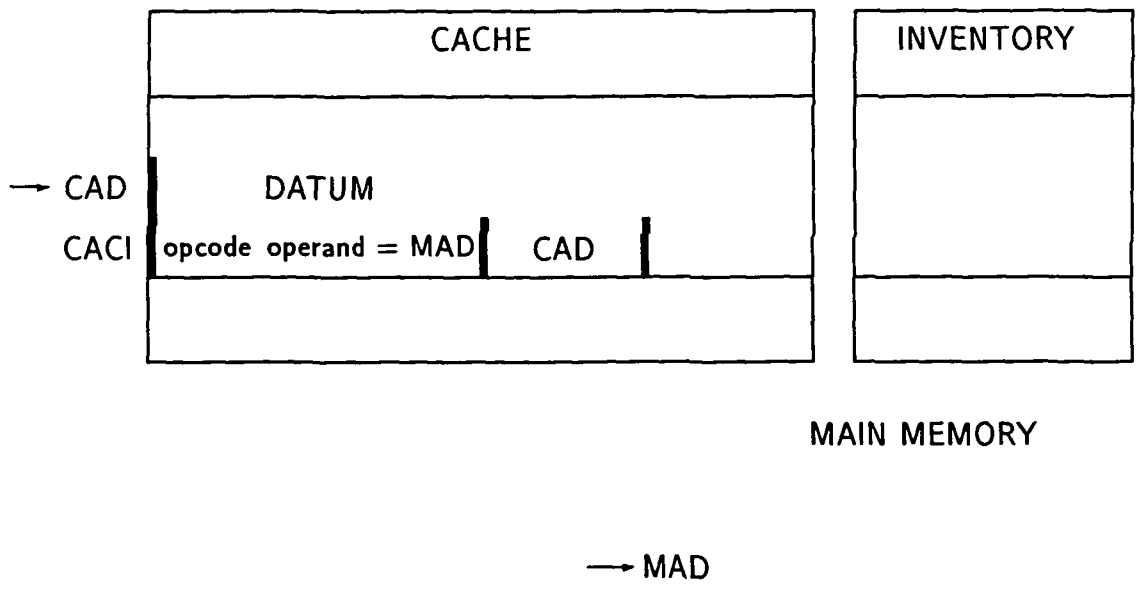


Fig. 2 Current instruction and its datum encached in succession.

In the case of a datum which has been encached earlier, the CMU must first locate the line containing the datum in order to update the CAD field at CACI. The procedure to locate this line will be explained in a later section.

After the datum is operated on, the EU tries to pass control to the instruction at the next sequential location in the main memory. But, since this instruction is not in the cache the EU scores a miss. The CMU then takes corrective action as described below.

At the time that the current instruction was encached the inventory would have been updated. Thus the inventory entry at CACI contains MACI, the Main memory Address of the Current Instruction (Fig. 3). Let the Main memory Address of the Next Instruction be denoted by MANI. Then,

$$\mathrm{MANI} = \mathrm{MACI} + 1 \tag{2}$$

The CMU accesses the inventory at CACI and obtains MACI. It then increments MACI and accesses the main memory at MANI to obtain the next instruction which it encaches at CANI, the Cache address of the Next Instruction. It then enters CANI in a field in the line at CACI.

From the FIFO rule, for this particular example

$$\mathrm{CANI} = \mathrm{CAD} + 1 = \mathrm{CACI} + 2 \tag{3}$$

Again, this relation does not hold true in general because what is referred to here as the "next instruction" may have been executed earlier, after which a branch may have transferred control to the current instruction. In that case the next instruction would have been in the cache before execution of the current instruction commenced. Hence the pointer CANI in the line at CACI is a necessary requirement.

The two pointers CAD and CANI account for $2c$ bits at CACI. The remaining 3 bits are employed for the miss detection test which will be described later.

If the instruction at CACI receives control at a subsequent time and if its datum and the next instruction have not been decached in the meantime, the pointers CAD and CANI will remain valid. The instruction at CACI will then be executed with no access to the main memory or to the CMU. The only additional work for the EU is the miss detection test.

Now, consider the execution of a branch instruction. Let the branch condition be implied in the opcode. Here, the operand MAD points to the "datum" which is in fact the branch destination in the main memory. If the branch is not taken the EU does not need the "datum" and hence does not access the cache at CAD. As before, CANI passes control to the next instruction. When the branch is taken for the first time, the EU scores a miss when accessing the "datum" in cache. At this point the CMU locates the "datum" and updates the CAD field of the line at CACI. The EU then branches to CAD. Thus, a loop containing one or more branch instructions will, after the first pass, be executed with no reference to the CMU, the inventory or the main memory. The only condition is that the code and data of the loop do not require more lines than the total number in the cache.

If the EU exits the loop more items may need to be encached. By the FIFO rule some items will be decached. If the EU enters the loop again it may attempt to access an item which has been decached. On attempting access using the now erroneous content of a CAD field or a CANI field it scores a miss. The CMU then encaches the item or finds where it is in the cache and updates CAD or CANI.

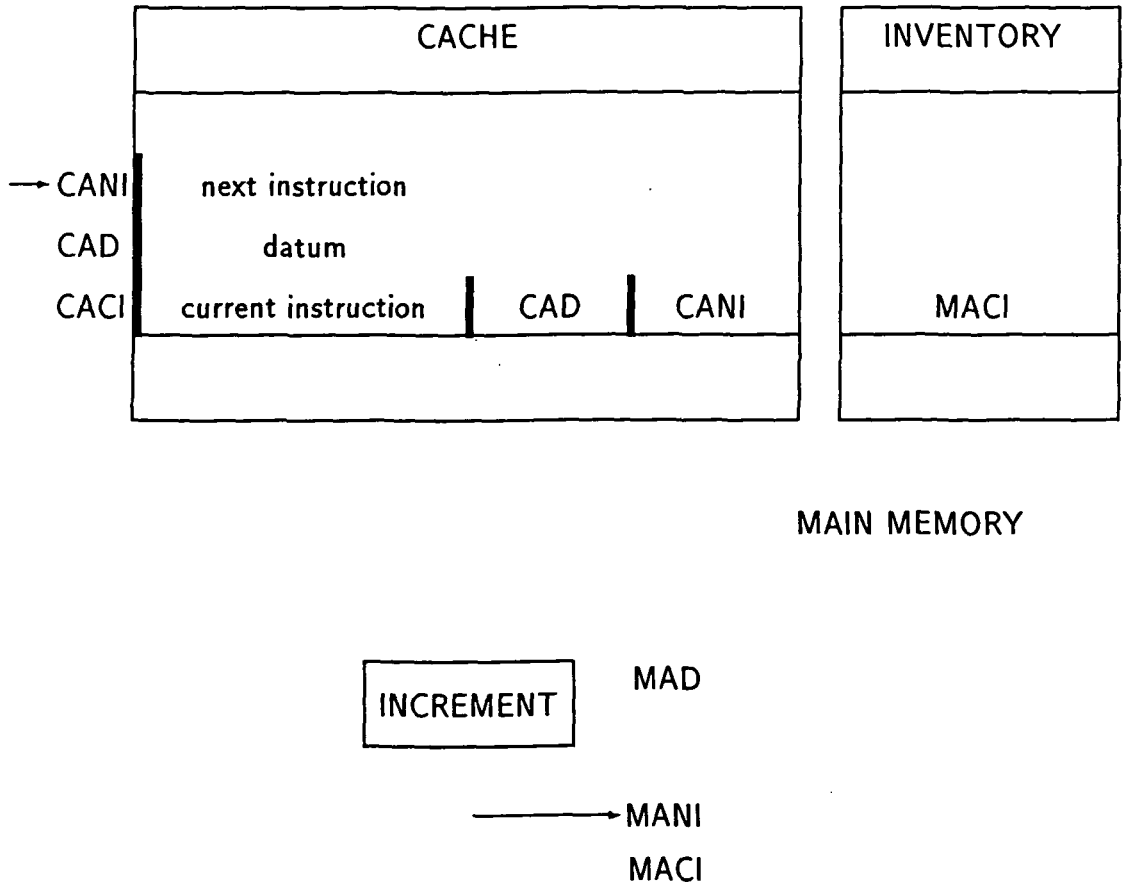


Fig. 3 Current instruction, its datum and the next instruction encached in succession.

In summary, after encaching an instruction the CMU augments it with two cache addresses. The EU runs the program with no address arithmetic.

5 The Contents of the Main Memory

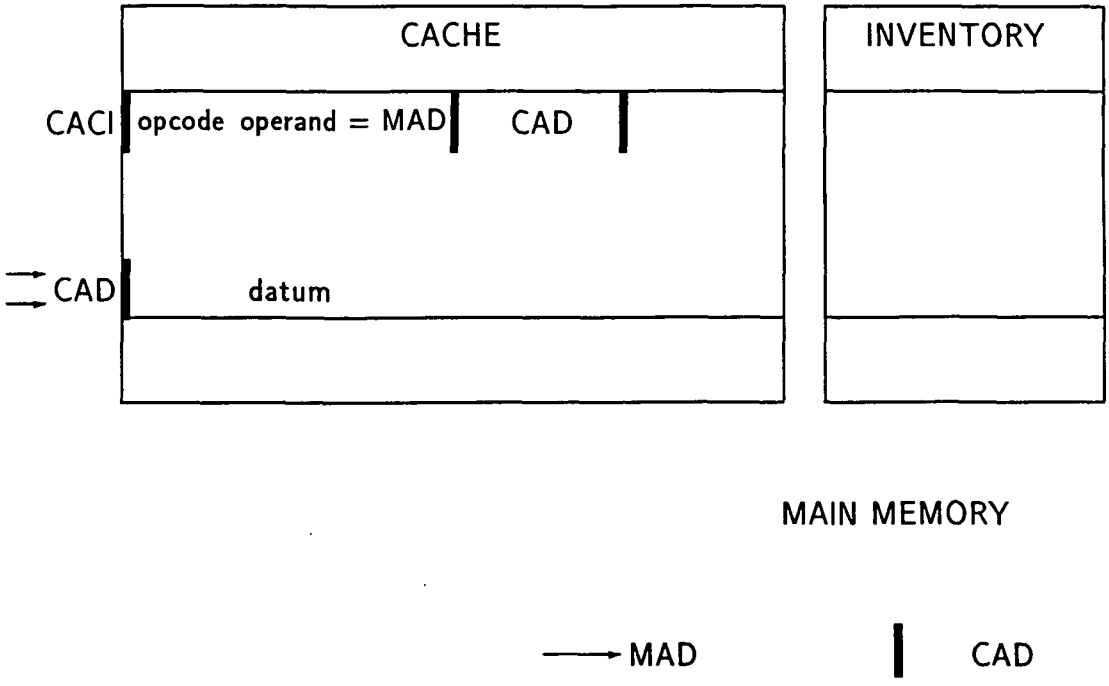


Fig. 4 Datum encached by a preceding instruction.

In trying to access a datum in the cache the EU scores a miss whenever the pointer CAD is invalid. CAD can be invalid under two conditions:

- The datum is in the main memory.
- The datum has been encached by a preceding instruction.

In the former case, the CMU encaches the datum and updates CAD by inserting DA as explained in the previous section. In the latter case the CMU must find where the item is before it can update CAD. In order to provide for this, whenever a datum is encached DA is written in the word in the main memory from which the datum is taken. Thus only one copy of the datum exists at any instant; after a datum is encached the word that originally carried it will contain a pointer to it. MAD becomes an indirect pointer to the datum and CAD becomes a direct pointer. This procedure guarantees that the CMU can link a datum

to any number of instructions. Thus, for the case where a miss occurs while the datum is in the cache, the CMU obtains MAD from the operand field of the instruction itself and accesses the main memory at MAD to get the cache address of the datum (Fig. 4). A similar procedure is carried out whenever the EU scores a miss in trying to access an instruction.

The main memory now contains a mix of items and cache pointers. The CMU must distinguish between these two types. An obvious way to provide for this would be to increase the length of a main memory word to $(1+p)$ bits, with the extra bit carrying a boolean field INMAIN. An alternative technique which maintains the length of a main memory word at p is described in a later section.

6 The Encache Procedure

Whenever a miss occurs the EU supplies CACI to the CMU. It also indicates whether the fault was in CAD or in CANI. The CMU then accesses the main memory at MAD or MANI as appropriate. If the required item is in the cache it updates the faulty field at CACI. If the item is in the main memory the CMU encaches it before updating the field.

The steps of the decache/encache procedure and the reason for carrying out each step are listed below.

1. Get the inventory entry at DA. This entry is the main memory address of the word from which the current item in the line at DA was taken.

2. Return the current item in the line at DA to the main memory word at the address given by the inventory entry. Only one copy of an item is maintained.

3. Invalidate the content of the CAD and CANI fields at DA. If the new item to be encached is an instruction it must be forced to score misses when first attempting to access the datum and the next instruction.

4. Place the new item in the line at DA. The EU will access this directly hereafter.

5. Copy DA into the main memory word. The CMU will use this as a pointer to the item if it becomes necessary to link it to other instructions.

6. Enter the main memory address of the encached word, in the inventory at DA. This entry is required for proper decaching of this line. Decaching will become necessary once DA has been incremented through one wraparound. This entry is also necessary to find MANI and explicitly link an instruction to its successor in the cache.

7. Insert DA in the appropriate field, i.e., CAD or CANI, in the line at CACI. This ensures a hit if the current instruction is executed again.

8. Increment DA with wraparound if necessary. This implements the cyclic FIFO rule.

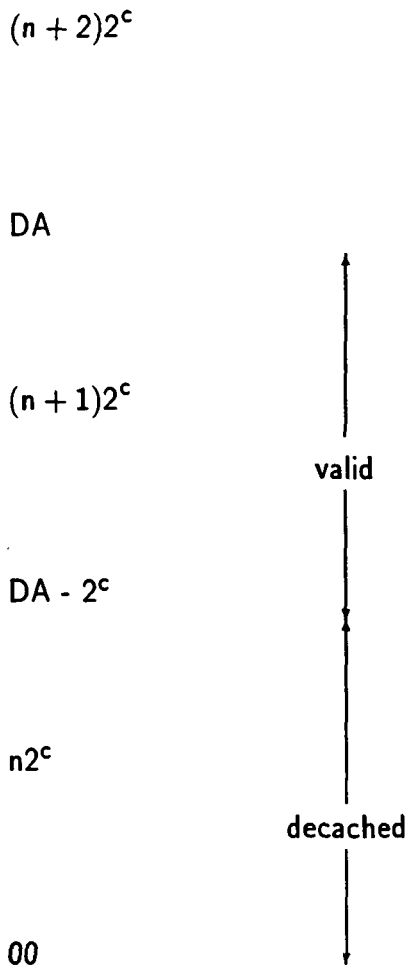


Fig. 5 Logical addresses of cyclic FIFO buffer represented as a quasi-finite stack.

7 The Miss Detection Test

For decache/encache operations, the CMU treats the cache as a FIFO buffer with physical addresses ranging from 0 to $(2^c - 1)$. Logically, this FIFO buffer may be treated as a quasi-infinite stack in which only the top 2^c locations contain valid items (Fig. 5). In this logical representation, to encache an item the CMU pushes it on to the stack. Since only the top 2^c items are valid a decache operation is not required.

The logical address of a line in the stack lies in the range 00 to (DA - 1) and is given by

$$logical\ address = physical\ address + n2^c \tag{4}$$

where n is the count of the number of wraparounds since power-up. n is held in a field of MSBs in DA. This field may be referred to as the wraparound part of the logical address.

Whenever an item is encached the wraparound field of DA is stored in a field set apart for this purpose at each location in the cache. The logical address of a cache location is determined by appending the physical address to the content of the wraparound field at the accessed location, treating the former as LSBs.

Recall that whenever an item is encached, its cache address CAD or CANI as the case may be, is stored in a field at CACI. If CAD and CANI consist of logical addresses a miss may be detected by comparing the wraparound part of CAD or CANI with the content of the wraparound field at the accessed location. Equality indicates a hit.

As the program runs, more and more items get encached and n increases steadily. It appears that the miss detection test would be effective only if the wraparound field is large enough to accommodate all wraparound from power-up to shutdown. The upper bound to the length of the wraparound field may be determined as follows.

At the instant that CAD is updated to a value say CADi, the datum and the current instruction are both present in the valid portion of the stack. Hence,

$$0 <| \text{CADi} - \text{CACI} |< 2^c \tag{5}$$

Let the contents of the physical location represented by CADi be replaced zero or more times. Let the resulting logical address of that location be CADj. If the same instruction receives control again then CACI and CADj must fall within the valid portion of the stack. Hence,

$$0 <| \text{CADj} - \text{CACI} |< 2c \tag{6}$$

Also CADj > CADi Therefore, from equations (4) and (5),

$$0 < \text{CADj} - \text{CADi} < 2^{1+c} \tag{7}$$

Lemma If K, L, M are non-negative integers such that $K - L < 2^M$
then, $K = L$ iff $[K = L]$ modulo 2^M

Thus, CADi = CADj iff [CADi = CADj] modulo 2^{1+c}

Hence, the maximum length required for logical addresses is (1+c) bits. **The wraparound field need not be more than 1 bit long.**

Now, since CADi and CADj both refer to the same physical cache location

$$[\text{CADi} = \text{CADj}] \text{ modulo } 2^c. \tag{8}$$

Therefore the miss detection test reduces to comparing two bits; the MSB of CAD or CANI and the wraparound bit stored at the accessed line.

In summary, DA is maintained as a register of $(1 + c)$ bits. The current MSB or wraparound bit is recorded at each location when an item is encached. When the datum is encached the $(1 + c)$ bits of DA are entered in the CAD field. The EU accesses the datum using the c LSBs of CAD and compares the MSB of CAD to the wraparound bit stored at the accessed location. Equality indicates a hit. The procedure in the case of the next instruction is similar.

8 Miscellaneous

8.1 Word Length of Main Memory

As explained earlier, whenever an item is encached its cache address is placed in the main memory word. As a result the main memory contains a mix of items and pointers. In order to distinguish between these two types an additional bit may be incorporated in every word of the main memory. The following procedure eliminates the need for an additional bit.

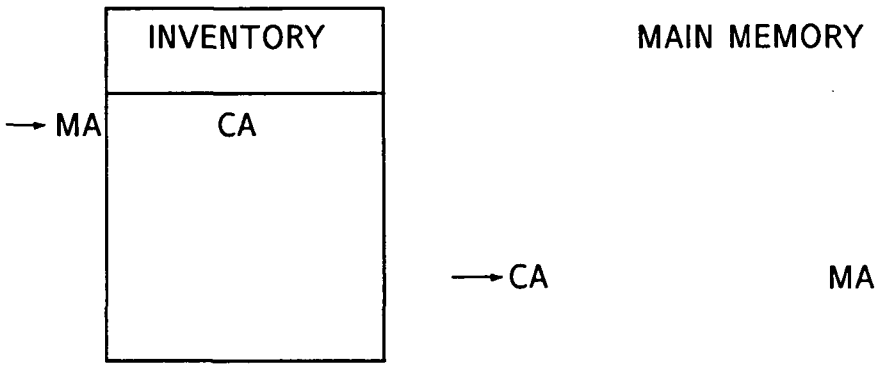


Fig. 6 Pointers at an encached word of the main memory and in the inventory.

Whenever the CMU accesses the main memory, say at MA, it interprets the c LSBs of the content of the accessed word as a cache address, say CA (Fig. 6). It then obtains the inventory entry at CA and compares it with MA. Equality indicates that MA contains a cache pointer. Inequality shows that MA contains a program item.

8.2 Decaching the Current Instruction

It is possible that the EU scores a miss when CACI is equal to DA. This would result in the current instruction being decached. This possibility should be taken into account in organizing the data transfers within the CMU.

9 Conclusions

A novel method of implementing a cache memory is proposed. Its primary features are the absence of associative hardware and complete independence from spatial locality of code or data. The absence of associative hardware makes large caches possible at relatively low cost. Conventional single-address instructions get automatically augmented to a two-address format. This enables the program to be executed from the cache without address arithmetic. A cache miss is detected by the simplest possible test; compare two bits. The correctness of the test has been proved with mathematical rigor.

10 Acknowledgements

The work described in this paper was done by the author during a sabbatical year in 1974/75, at the Department of Computer Science, University of Reading, U.K. The author expresses his gratitude to the Royal Society and the Nuffield Foundation for the grant that enabled him to spend a year in the U.K. and to Dr. J.D. Roberts who helped and encouraged him throughout the course of this work. The author wishes to acknowledge the generous assistance by way of suggestions and comments given by Dr. T. Srikanthan and Dr. A.B. Premkumar during the writing of the present draft. Thanks are also due to Mr. H.G. Wijewansa, Mr. T. Kirubarajan and Miss. S.S. Madahapola who assisted the author in preparing the final version of this paper.

11 References

[1] J.A. Gunawardena, "Improvements in or Relating to Computer Store Mechanisms," United Kingdom Patent No. 1531261 of 1976. Filed by the National Research Development Corporation, London.
[2] J.A. Gunawardena, "Improvements in or Relating to Computer Store Mechanisms," United States Patent No. 4212058 of 1980. Filed by the National Research Development Corporation, London.

A Set-Associative, Fault-Tolerant Cache Design [1]

Dan Lamet and James F. Frenzel
Department of Electrical Engineering
University of Idaho, Moscow, ID 83843
208-885-7888 jfrenzel@groucho.mrc.uidaho.edu

Abstract - **The design of a defect-tolerant control circuit for a set-associative cache memory is presented. The circuit maintains the stack ordering necessary for implementing the LRU replacement algorithm. A discussion of programming techniques for bypassing defective blocks is included.**

1 Introduction

The dramatic increase in cache memory size and diminishing geometries has resulted in lower yields. Today's high performance processors often have on-chip cache and consequently the yield of these memories can be a significant factor in determining the ultimate cost of the processor. One way of increasing yields is to provide defect-tolerance through the use of redundant resources. The two methods for achieving defect-tolerance most commonly employed, error correcting codes and spare rows and columns [4], require additional hardware and introduce additional access delays. This paper describes a cache design which uses the redundancy inherent in set-associative caches to operate correctly in the presence of manufacturing defects with minimal additional hardware or propagation delays.

Caches are small memories which operate between the processor and main memory. Their goal is to reduce average memory access time. This is accomplished by using very fast memory, storing only the most used bytes (instructions and/or data) in the cache, and copying multiple word units of main memory called blocks. Block usually contain 4 to 16 bytes from main storage and are organized into groups called sets. In a direct-mapped cache, each set consists of only one block, whereas in an n-way, set-associative (SA) cache, each set contains n blocks, where n is the associativity. The total cache size is the product of the block size, the number of sets, and the associativity [2].

Sohi observed that a cache memory does not have to be defect free to meet its objective, namely reduce the average memory access time of a hierarchical memory system [6]. A direct-mapped cache memory with a defective block will never be able to hold items from main memory which map to that set in the cache. For a cache to operate properly under this condition two things are necessary: one, the cache must be able to recognize a defective block and generate a miss and two, must have the capability of performing a load-through, so that the processor can access the item. An associative cache has alternate locations within a set which can be used when there is a defective block present. Ideally, the circuitry which implements the replacement algorithm would be modified at test time to exclude defective blocks from selection during replacement. Provided each set has at least one good block, all items from main memory can map to a good location in the cache. This exclusion would be implemented in the replacement algorithm circuitry.

[1]This research was supported by NASA under Space Engineering Research Center Grant NAGW-1406

	A	B	C	D
A	X	X	X	X
B	X	X	X	X
C	X	X	X	X
D	X	X	X	X

a) The original matrix.

	A	B	C	D
A		X	X	X
B	X		X	X
C	X	X		X
D	X	X	X	

b)Without the diagonal.

	A	B	C	D
A		X	X	X
B			X	X
C				X
D				

c) The non-redundant information.

Figure 1: The Reference Matrix

When the processor requests a piece of data not currently in the cache, then a miss is said to occur. The cache must decide which block's data to remove, making space for the new data. For a direct-mapped cache, the decision is trivial, as each block from main storage maps to a single block in the cache. However, with a set-associative cache, assuming the referenced set is full, there are n possible blocks to replace. One of the best replacement algorithms is referred to as least recently used (LRU), where the set is treated as a stack and accessing a particular block moves that block to the top of the stack. The least recently used block is always at the bottom of the stack and a miss will load the data into this block and move it to the top of the stack. A block is also moved to the top of the stack on a cache hit because it now the "most recently used." Efficient implementations of the LRU replacement algorithm require $n(n-1)/2$ bits of storage per set to maintain the $n!$ possible stack configurations. Consequently, a 4-way SA cache requires 6 bits of storage per set, while an 8-way SA cache requires 28 bits per set [3]. Alternative replacement strategies are first in, first out (FIFO), and random [2].

2 LRU Replacement Circuit Operation

The set-associative, fault-tolerant cache uses the algorithm described by Maruyama to implement LRU replacement [3]. This method revolves around the concept of a reference matrix, which stores the stack order. The matrix is square, with a column and row for each block. Each bit represents whether or not one block has been used more recently than another block. A 1 indicates that the row block has been used more recently than the column block. For example, if there is a 1 in the B row and C column, then C block has been used *less* recently than B block.

The entire square matrix is not needed to store the stack order. First, the diagonal can be removed, because B column and B row has no meaning. B block can only occupy one space in the stack. Also, one of the remaining triangular sections may be removed since stack order is symmetrical, i.e., if A row, C column indicates the relationship between A and C blocks, then we don't need C row, A column to do the same thing. This leaves only the upper right hand half of the matrix, a triangle with $n(n-1)/2$ elements, where n is the associativity of the cache. Figure 1 shows the reduction of the matrix. Note that in part c), the first block has no column and the last block has no row.

Figure 2 shows the circuit implementation of the defect-tolerant LRU replacement algorithm. The circuit has n inputs and n outputs, one of each for every block. Of the n outputs,

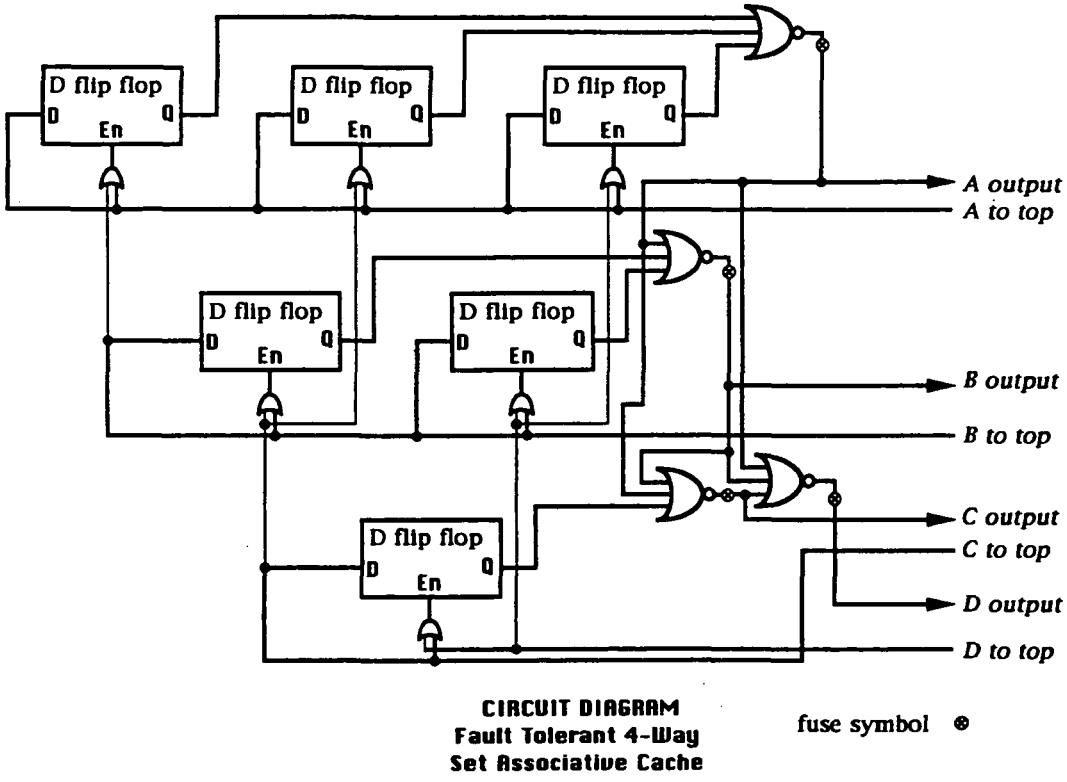


Figure 2: LRU Control Circuitry

one and only one will be high at any one time. This output indicates the block which has been used least recently. The order of the other three blocks is stored correctly within the circuit, but cannot be discerned from the outputs. When a block is faulty, a fuse is blown at its output, permanently fixing that output low. In the case where all blocks have been marked, all outputs will remain 0. This is the only case where at least one output is not high.

The inputs are used to notify the control circuit when a certain block has been referenced. In other words, when the CPU makes a memory call and the cache receives it, either the address being called is already in the cache or it needs to be loaded into the cache. In both cases, that block must now be placed at the top of the stack, indicating that it is the most recently used. This is done by bringing the input line called "'block' to top" high. For example, suppose that after normal operation the stack comes to rest in the order BCDA, where A is at the bottom of the stack (least recently used) and B is at the top of the stack. The CPU sends a request for an address which happens to be located in block D. The cache circuitry routes the appropriate data to the CPU, but at the same time sets the "D to top" line to a 1, causing the stack to place D at the top. The final state of the stack is DBCA. Note that although the stack order has changed, A is still at the bottom of the stack. The "D to top" line must be set regardless of D's position in the stack as it is now the most recently used block. This causes D to go to the top of the stack.

There are several patterns which describe the internal operation of the LRU control circuit. Each row has a certain permanent priority among the blocks. The A block has the

	A	B	C	D
A		1	1	0
B			0	1
C				0
D				

a) The initial state.

	A	B	C	D
A		1	0	0
B			0	1
C				1
D				

b)After C block accessed.

	A	B	C	D
A		1	0	0
B			0	0
C				0
D				

c)After D block accessed.

Figure 3: Normal Circuit Operation

lowest priority and priority increases to the highest, which is D for a 4-way cache. This priority is invoked only when there are two or more blocks with zero rows. Normally, if all the bits in a row are 0, then that block is on the bottom of the stack. If there is more than one row with all zeros, then priority breaks the tie. An example of this is the initial state. At startup all bits are set to zero and because the A block has the lowest priority, it is selected as being on the bottom of the stack.

When a block is selected, that is a memory reference to it is made, it must be moved to the top of the stack. This is accomplished by setting all the bits in the block's row to 1 and all the bits in the block's column to 0. For example, take the stack state as shown in Figure 3, part a). This is for the stack order ABDC, where C is at the bottom. When C is referenced, C row (only a single bit) is set to 1 and C column is set to 0. This case is shown in part C. D then comes to the bottom. Once D is referenced, B comes to the bottom.

To use the fault-tolerant properties of the circuit, fuses must be installed at the output of each NAND gate. The fuses have the characteristic that once they are blown, the incoming signal is left floating and the outgoing signal is drawn to ground. This forces the faulty block to always indicate that it is above the bottom of the stack even though it will eventually float to the top and its row bits will saturate to zeros. Consequently, the faulty block will never be selected for replacement during a cache miss.

3 Programming Techniques

There are several methods for implementing memory reconfiguration in the presence of defects. These are electrically programmable links, electron-beam programmable fuses, and laser cutting/welding [5]. These techniques are employed to bypass faulty resources and activate spare units in common applications, but special considerations must be made for the circuit described in this paper.

Typically, spare rows are initially connected as well as the primary rows. Whether or not any repair takes place, there must be some fuses blown. This is acceptable when the number of rows and columns is limited. In our design, however, there is a fuse for every block. For an 8K cache with a block size of 4, 2K fuses are implemented. It is not feasible to burn all of them on every chip. For this reason, in the absence of defects the circuitry must operate properly with the fuses in-place, without requiring laser programming. Past redundancy techniques typically use only one type of programming. For example, many RAM chip designs are programmed by opening certain connections. A few close certain connections, but none use both technologies. This is acceptable for their purposes, but not

for the LRU control circuit.

The LRU circuit requires that for one output line, two input lines are originally connected so that one is dominant until the fuse is burnt, in which case the second line becomes dominant. Circuitry to implement this requirement is troublesome. One of the simplest solutions is to put a resistor between the second input and the output so that originally, the difference in signals becomes a voltage drop across the resistor and after the fuse is opened, the second signal can be passed through the resistor. This solution is costly. Resistors, or their VLSI equivalent, must be added to 2K fuse locations in the cache, consuming area and some considerable power will be dissipated through the resistors whether or not the fuse is blown. Other more complicated solutions become even more cumbersome. The ideal solution would be to simply open the first line and close the second. This may be possible.

P.W. Cook discovered that connections could be made through an oxide layer on silicon using the same technology which is used to open connections [1]. He documents this in a 1975 paper using a 6 micron technology.

4 Future Work

As stated above, P.W. Cook's method of combining open-fuse and closed-fuse technology may be the best possibility for the cache design presented. One area of research will be to determine if it is still feasible in today's smaller geometries. Electron-beam methods will also be examined. More testing will continue on the operation of the LRU circuit itself as well as defining its surrounding circuitry.

References

[1] P.W. Cook, S.E. Schuster, and R.J. von Gutfeld. Connections and disconnections on integrated circuits using nanosecond laser pulses. *Applied Physics Letters*, February 1975.

[2] J. F. Frenzel. Performance of defect-tolerant set-associative cache memories. In *3rd NASA Symposium on VLSI Design*, pages 3.2.1–3.2.9. University of Idaho, October 1991.

[3] K. Maruyama. mLRU page replacement algorithm in terms of the reference matrix. *IBM Technical Disclosure Bulletin*, pages 3101–3103, March 1975.

[4] Will R. Moore. A review of fault-tolerant techniques for the enhancement of integrated circuit yield. *Proceedings of the IEEE*, pages 684–698, May 1986.

[5] R. Negrini et al. *Fault Tolerance Through Reconfiguration in VLSI and WSI Arrays*, chapter 4. The MIT Press, June 1983.

[6] Gurindar S. Sohi. Cache memory organization to enhance the yield of high-performance VLSI processors. *IEEE Transactions on Computers*, April 1989.

Session 10
Correlators

John Canaris

A 1 GHz Sample Rate, 256-Channel, 1-Bit Quantization, CMOS, Digital Correlator Chip

C. Timoc, T. Tran and J Wongso
Spacebourne, Inc.
742 Foothill Blvd., Suite 28
La Canada, CA 91011

Abstract - **This paper describes the development of a digital correlator chip with the following features:**

- **1 Giga-sample/second**
- **256 channels**
- **1-bit quantization**
- **32-bit counters providing up to 4 seconds integration time at 1GHz**
- **very low power dissipation per channel**

The improvements in the performance-to-cost ratio of the digital correlator chip are achieved with a combination of systolic architecture, novel pipelined differential logic circuits, and standard 1.0 um CMOS process.

1 Introduction

Digital correlation spectrometers have been in use in radio astronomy since Weinreb introduced a 100-channel 1-bit autocorrelator built with transistors to measure the spectrum of radiation collected by a single antenna [8]. The power spectrum of radio astronomy signals is obtained after a Fourier transform, performed with a general purpose computer, of the correlation function.

A number of digital correlation spectrometers have been built over the past few years with either CMOS or bipolar ECL (Emitter Coupled Logic) technologies. However, existing correlators are either limited in bandwidth or in the number of channels (lags) per chip.

At Caltech, B. Von Herzen has developed a CMOS digital correlator for the Submillimeter Observatory at Hilo, Hawaii. The correlator comprising 320 channels is fabricated with a 1.0 um CMOS process and operates at a maximum clock frequency of 400 MHz with a power dissipation of 13W per chip [7]. Also at Caltech, S. Padin and S. L. Scott have developed an ECL digital correlator chip which operates with a sample rate of 250 MHz. This correlator has been installed at the Owens Valley Radio Observatory [5].

Another group of researchers from the University of Idaho led by J. Canaris and S. Whitaker have developed for NASA/JPL a 32-channel, CMOS correlator which operates with a maximum clock frequency of 25 MHz when it will be fabricated with a 0.8 um CMOS process [3]. Also at NASA/JPL a group of researchers lead by K. Chandra and W. Wilson have developed a 26-channel, ECL correlator chip capable of operating at a maximum clock frequency of 300 MHz [4].

Yet another group of researchers very active in the development of digital correlators is lead by Alexander Bos from the Netherlands Foundation for Radio Astronomy. They have constructed several digital correlators [1], [2].

At the NRAO (National Radio Astronomy Observatory) R. P. Escoffier and others have been developing different types of digital correlator spectrometers for the past two decades. Many of these correlators are installed in the VLBA Array Operation Center in Soccoro, NM.

Traditionally, the CMOS technology has dominated high-density integrated circuits and bipolar ECL has been used to fabricate high-performance products. However, theoretical studies of computation in VLSI show that CMOS offers excellent opportunities for fabricating, not only moderate performance VLSI circuits, but also high performance products. Retiming can be used to transform combinational networks into systolic (maximum pipelined) structures with order-of-magnitude improvement in performance and throughout. Compared to other competing technologies (e.g., silicon bipolar, GaAs), CMOS is a technology in which transistors and wires (chip area) are extremely plentiful and inexpensive. Therefore, significant improvements in the performance-to-cost ratio could be obtained by using systolic systems fabricated with a CMOS technology.

In a prior research and development effort, we developed a novel CMOS circuit, called PDL (Pipelined Differential Logic), which is capable of operating with short propagation delays, low power dissipation and low switching noise. A total of 18 different designs using PDL circuits were fabricated for the past 6 years through the MOSIS service. A 256-bit shift register was designed with PDL circuits and it was fabricated with a 1.0 um CMOS process. The shift register operated at a maximum clock frequency of 1 GHz [6].

2 Digital Correlator Spectrometer

The architectural organization of the correlator is illustrated in Fig. 1. The correlator calculates the correlation function of input signals A and B. To perform an auto correlation function inputs A and B are connected together. The A signal is delayed while the B signal is broadcast undelayed to all channel. Each correlator channel comprises a multiplier and an accumlator. One input of a channel receives one of the delayed signals while the other input receives one of the delayed the undelayed signal. The correlation products for each delay stage (channel) are accumulated in counters. The contents of all counters are read from the correlator by a computer for calculating the power spectrum by a Fourier transform.

A cell library using novel PDL (Pipelined Differential Logic) circuits has been developed for each logic function required by the correlator. This involved the sizing of transistors, by circuit simulation (SPICE), in order to achieve 1 GHz operation with minimum power dissipation. Since no layout was available when the first transistor sizing was performed, the loading capacitance of each interconnecting wire was estimated based on previous experience with a 1.0 um CMOS process. In addition to the design of logic circuits, input and output circuits have also been designed to operate with 1 GHz clock signals, with standard ECL logic levels, on 50 ohm (characteristic impedance) transmission lines.

An example of PDL circuit [6], performing a 2-way exclusive-OR function, is illustrated in Fig. 2. A set of differential inputs (A, -A, B, -B) are connected to a cascade switch (tran-

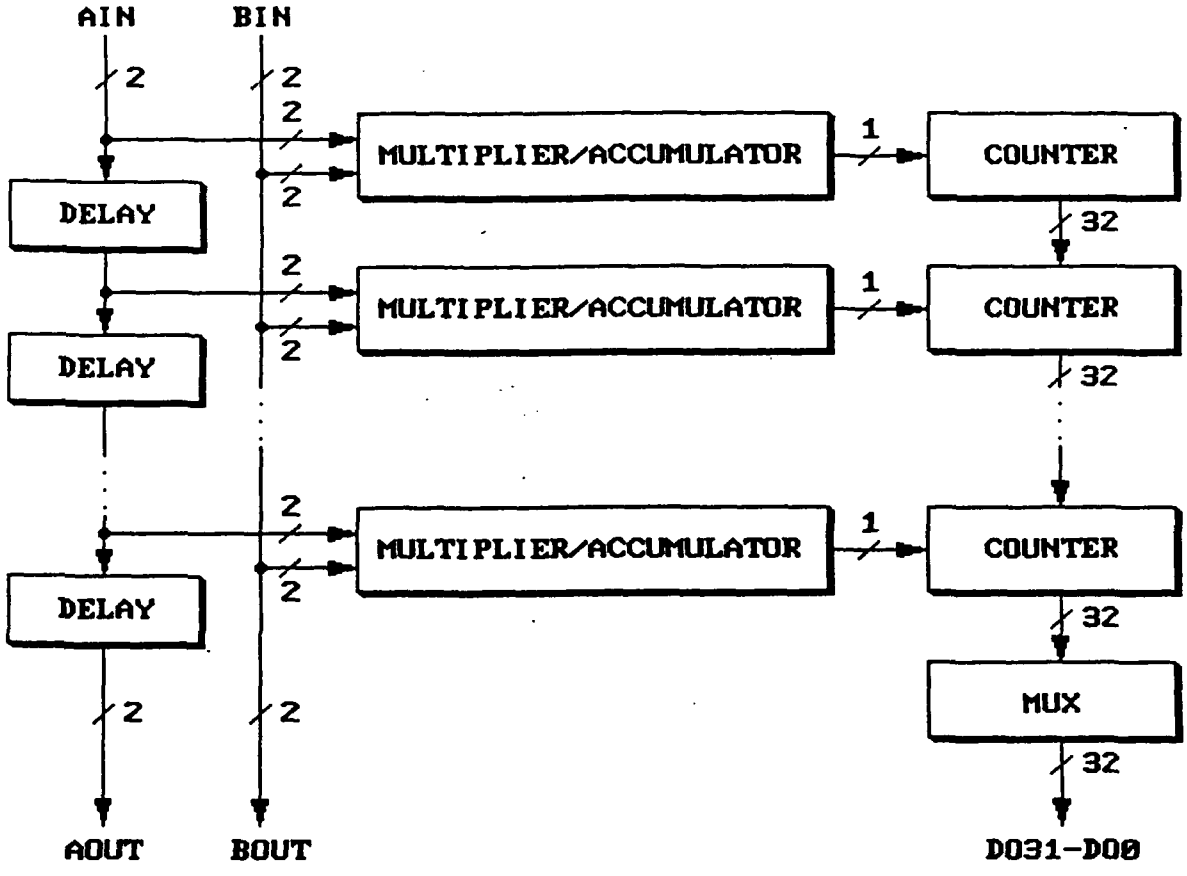


Figure 1: A block diagram of the proposed digital correlator showing the major functional elements

sistors N3-N8). A sense amplifier comprising transistors N1, N2, P1, P2 and equalization transistor N9 is connected on top of the cascade switch.

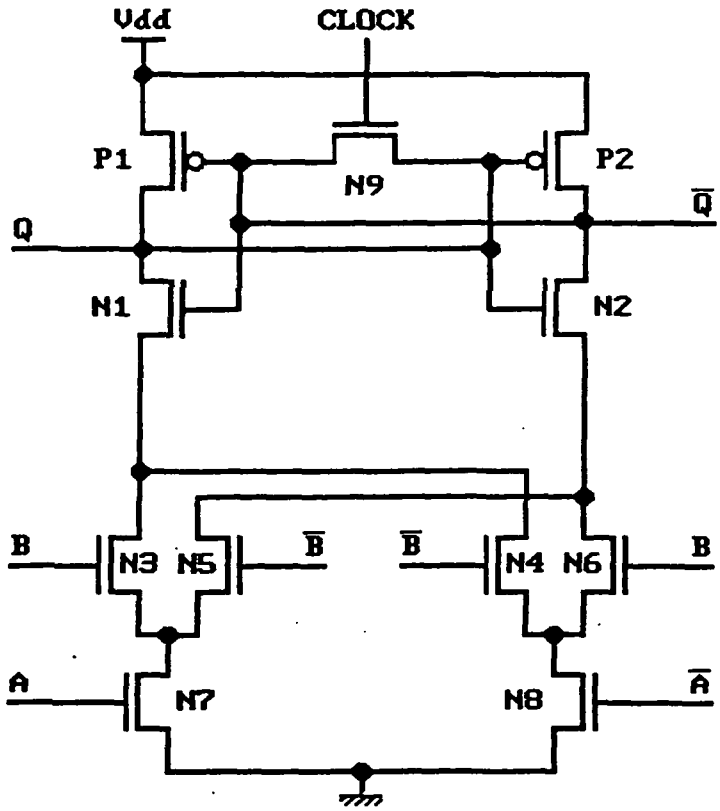


Figure 2: An example of a pipelined differential logic (PDL) circuit performing the exclusive-OR function

The PDL circuit operates in two phases. In one phase of the clock, the two output signals (Q,-Q) of the logic gate are forced by the equalization transistor to a voltage equal to one half of the power supply voltage. On a second phase of the clock, the outputs of the logic gate are partially released by the equalization transistor and the sense amplifier produces a differential output which is a function of the differential logic inputs applied to the cascade signal which has an amplitude substantially smaller than that of the power supply voltage.

The PDL circuits achieve extremely short switching time by:

1. reducing the voltage swing of the clock and logic signals
2. using a very fast current controlled sense amplifier
3. forcing the terminals of the sense amplifier to the most sensitive operating point, which is one half of the power supply voltage, before the circuit switches to valid logic levels

The performance of a very high speed PDL circuit is illustrated (Fig. 3) by an oscillogram showing the clock (top) and data out (bottom) of a 256 bit shift register fabricated with a 1.0 um process. The voltage swings inside the chip was reduced from 5V swing, which is commonly used in conventional CMOS circuits, to only 1V. A PDL off-chip receiver amplifies an input ECL signal, with voltage swings as low as 350 mV, to PDL signals of 1V. An off-chip driver has the capability to deliver ECL levels (up to 800 mv) on 50 ohm transmission lines at a clock frequency is 1 GHz while the data out is shown at half the clock frequency. However, it is worth noting that in a pipelined system, in general, and in the 256 bit shift register, in particular, the data rate is equal to the clock rate.

After the development of a cell library in which the transistors of each logic cell had been sized with estimated loads, we perform the layout of a 16-channel 1-bit quantization correlator. We started with the placement of the power distribution buses and the clock distribution network arranged such as to accommodate a standard-cell layout style. Next, the layout of each logic cell was performed so that all logic cell had the same height but different width. The requirement for equal height was necessary so that each cell fits into the layout of power and clock structure. The logic cells have been placed and wired as indicated by the gate-level logic diagram. After the completion of all interconnections, actual loading capacitances for each logic cell were extracted and transistor sizing, with the newly available loads, was repeated. The final layout was then verified to assure that the schematic diagram had been translated correctly into mask specifications.

The fabrication of the 16-channel correlator is being done at Hewlett Packard through the MOSIS Service. The chip will be packaged in high frequency leaded chip carries available from Triquint Corp.

3 Conclusions

The anticipated results of this project are expected to demonstrate experimentally that a 16-channel correlator will operate with a clock frequency of 1 GHz and that it will dissipate less than 1 W. The objective of a future project is to increase the number of channels per chip to 256 while maintaining the 1 GHz clock frequency and 40 mW power dissipation per channel. The 256-channel correlator will be fabricated in a 0.65 um (L_{eff}) CMOS process (i.e., Hewlett Packard CMOS34) through the MOSIS service.

References

[1] A. Bos, "A High Speed 2-Bit Correlator Chip for Radio Astronomy," IEEE Trans. on Instrumentation and Measurement, Vol. 40, No. 3, pp. 591-595, June 1991.

[2] A. Bos, "Functional Design of a Wideband Digital Spectrometer," Internal Technical Report nr. 179, Netherlands Foundation for Radio Astronomy, May 1986.

[3] J. Canaris and S. Whitaker, "A High Speed CMOS Correlator," Second NASA SERC Symposium on VLSI Design, 1990.

[4] K. M. Chandra and W. J. Wilson, "Digital Autocorrelator Spectrometer," Jet Propulsion Laboratory, D-8056, December 1990.

[5] S. Padin and S. L. Scott, "A Digital Cross-Correlator for the Owens Valley Millimeter Array," National Radio Science Meeting, Boulder CO, January 1992.

[6] C. Timoc, T. Tran and J. Wongso, "Development of a 1 GHz, 256-Channel, 1-Bit CMOS, Digital Correlator Chip," National Radio Science Meeting, 1992.

[7] B. Von Herzen, "VLSI Partitioning of a 2-Gs/s Digital Spectrometer," IEEE Journal of Solid-State Circuits, Vol. 26, No. 5, pp. 768-772, May 1991.

[8] S. Weinreb, "A Digital Spectral Analysis Technique and Its Application to Radio Astronomy," Tech. Rep. 412 Lab Electronics, MIT, Cambridge, 1963.

Low Power, CMOS Digital Autocorrelator Spectrometer for Spaceborne Applications

Kumar Chandra and William J. Wilson
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California 91109

Abstract - **A 128-channel digital autocorrelator spectrometer using four 32 channel low power CMOS correlator chips has been built and tested. The CMOS correlator chip uses a 2-bit multiplication algorithm and a full-custom CMOS VLSI design to achieve low DC power consumption. The digital autocorrelator spectrometer has a 20 MHz band width, and the total DC power requirement is 6 Watts.**

1 Introduction

In the future, multi-channel spectrometers using digital correlation techniques will replace analog filterbank spectrometers in airborne, balloon and space-borne millimeter wave radiometers. The digital technique for spectrum analysis has advantages over analog filterbanks due to the inherent stability of the digital circuits, flexibility in reconfiguring the bandwidth and resolution between observations, reliability, small size, low mass and low cost of digital correlator chips for spectrometers with more than 50 channels. The digital autocorrelation technique is a proven method and has been in use in many ground-based astronomy observatories. With continuing developments in both materials, processing technologies and using novel logic circuit design, it will be possible to custom-design high speed digital correlator integrated circuits for wideband (1 GHz) and high resolution (1 MHz) spectrometers with very low DC power consumption. The Microwave Limb Sounder for the Earth Observing System (Eos/MLS) and the astrophysics missions such as the Submillimeter Intermediate Mission (SMIM), and the submillimeter lunar array are some of the flight projects that will require low power and stable spectrometers.

The Jet Propulsion Laboratory has been involved in developing low power, wideband digital autocorrelator spectrometers for the flight projects for many years. In 1990, a prototype 52-channel spectrometer using specially designed ECL correlator chip was completed and tested as proof of concept project (Chandra et al, 1990). This spectrometer was used for analyzing signals with a 125 MHz bandwidth and consumed 35 Watts of DC power. Due to the large DC power requirement, it is not suitable for spaceborne spectrometer applications; however, the ECL correlator chips will be used in balloon-borne spectrometers where the DC power consumption is not as critical.

To reduce the DC power requirements, a narrowband VLSI CMOS correlator chip was designed at the Microelectronic Research Center (MRC) at the University of Idaho in collaboration with JPL. The chip was fabricated at the Hewlett-Packard foundry at Corvallis, Oregon with a 1.2 micron CMOS process. A prototype 128-channel spectrometer was built using four of the correlator chips and it consumed only 6 Watts with a 20 MHz bandwidth.

This low power CMOS spectrometer may be used by the Eos/MLS project for its narrow-band, high resolution applications. Two spectrometers using the CMOS correlator chips will also be flown in the JPL/UCSB (University of California, Santa Barbara) balloon for astrophysics observation in 1993. The chips are also under investigation for possible use in the NOAA's (National Oceanographic and Atmospheric Administration) Microwave Temperature Sounder (MTS).

In this paper, the background information on autocorrelator spectrometers is provided followed by the design of the CMOS correlator chip and the 128-channel CMOS autocorrelator spectrometer.

2 Digital Autocorrelator

2.1 Background

The autocorrelation function of a signal is expressed as follows:

$$R(\tau) = \lim_{T\to\infty} \frac{1}{T} \int_0^T f(t)f(t+\tau)dt \tag{1}$$

where $f(t)$ is the input signal,
τ is the delay time and
T is the integration time.

A theorem due to Wiener and Khintchine (Robinson, 1974) relates the autocorrelation function measured in the time domain to power spectrum in the frequency domain by the Fourier transform using the following equation:

$$S(f) = \sum_0^\infty R(\tau) cos\ 2\pi\ f\tau d\tau \tag{2}$$

where $S(f)$ is the power spectrum of the input signal.

Because the autocorrelation function is an even function, only the cosine transform is required. In the digital autocorrelator, shown in Figure 1, the input signal is band limited, sampled at the Nyquist rate and digitized to a few bits. The sampled signal is delayed using shift registers and multiplied with the undelayed sample using simple logic circuits. The multiplier output from each delay stage is accumulated in a binary counter. Each delay stage is called a channel or lag. The autocorrelation function for the sampled data can be expressed as:

$$R(n\delta t) = \frac{1}{K} \sum_{m=0}^{K-1} [X(t_(0)x(t_0 + (n+m)\delta t)] \tag{3}$$

where $n = 0, 1, ...N-1$ represent the delay in one of the signal paths,
K is the number of products in the integration time T,
$\pm$ is the delay, usually made equal to to the sampling interval.

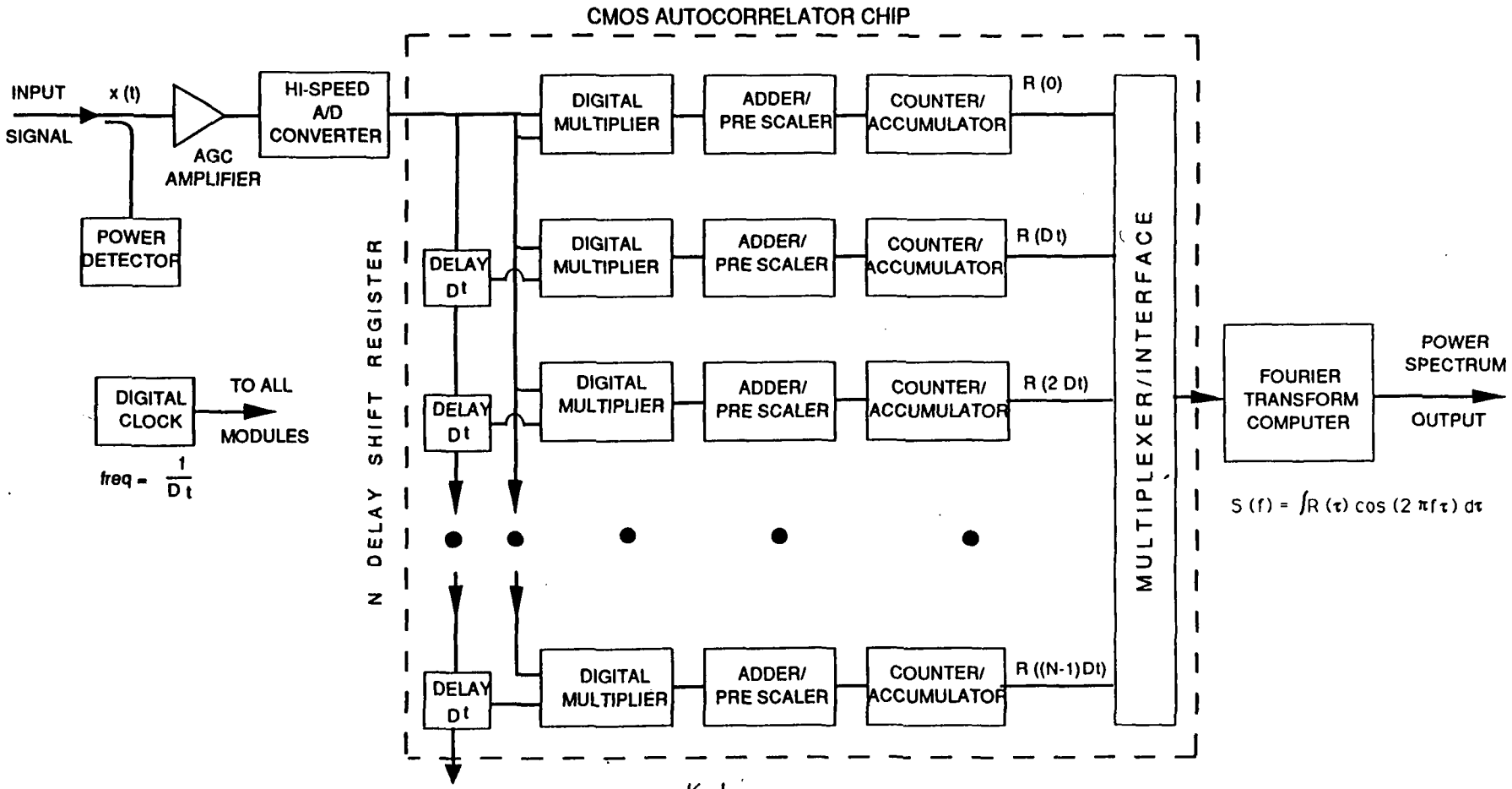


Fig. 1. DIGITAL AUTOCORRELATOR SPECTROMETER

The digital spectrometer uses the relationship between the autocorrelation function of the signal and its power spectrum by performing a Fourier transform on the autocorrelation data. The N channels or lags (corresponding to the N delay values) in the autocorrelation function, measured in the time domain, are transformed into N points on the frequency domain by using the Discrete Fourier transform (DFT) relationship,

$$P\frac{j}{2N\delta t} = \frac{1}{N}[R(n\delta t)cos(\pi nj/N)], j = 0, 1, ..., N-1 \quad (4)$$

where $P(j/2N \pm t)$ represents the power on the jth point on the output spectrum,
$R(0)$ is the correlation coefficient for the zero delay channel (=1 after normalization)
and $R(n \pm t)$ is the normalized autocorrelation coefficient for delay $n \pm t$.

Generally the input signal is digitized to only a few bits. Limiting the number of bits speeds up the multiplication and addition process because only a few digital operations are required. This permits the use of higher sampling frequency which increases the input bandwidth. However, the Signal to Noise Ratio (SNR) of the correlator is degraded by using only a few bits. The loss in SNR is 13% when two bit digitization is used with the correlator (Cooper, 1970). When more than two bits are used to digitize the input signal, the SNR increases rapidly to that of a continuous correlator. Quantizing schemes where the input signal is represented by more than two levels have also been considered by others (Cooper, 1976). However, the size and the complexity of the digital logic grows as the number of bits to represent the input signal increases. This will be of particular concern for space applications because the DC power requirement increases as the gate count is increased. The two-bit correlator scheme offers a reasonable trade-off between sensitivity and the size of the hardware which determines the DC power requirement for the logic circuit.

2.2 Digitizer

The first element in the digital correlator is the analog to digital converter is called the digitizer. The sensitivity and the stability of the spectrometer is determined by the digitizer characteristics such as threshold stability, gain variation, sampling aperture width and the uncertainties in the sample timings due to slow clock edges or jitter. The correlator sensitivity is decreased due to increase in the digitization noise by adverse effects in the above parameters.

The digitizer outputs a two-bit word for every sample of the input analog signal. One of the bits represents the sign (zero-crossing detector output) and the second bit represents the magnitude. The four states of the two-bit digitizer and the assigned weighting factors to these states are shown in Table 1. The sign bit is assigned a value "one" if the input voltage to the zero-crossing detector is negative and is assigned a value "zero" if the input voltage is positive. Similarly, the magnitude bit is assigned a value "one" if the input voltage to a window comparator is outside the pre-determined limits, $\pm Vref$, or assigned a value "zero"

SIGN	MAGNITUDE	WEIGHT
1	1	$-n$
1	0	-1
0	0	$+1$
0	1	$+n$

Table 1:

if the input voltage is between the limits.

For $n = 3$, setting the decision levels for the magnitude comparators at a level equal to the RMS voltage of the input signal gives an optimum SNR of 88% relative to the continuous correlator (Cooper, 1970).

2.3 Correlator Multiplier

Since the correlation is a multiplication and averaging process, the digitized signals represented as in Table 1 are multiplied after one of the signals is delayed in time using shift registers. A special multiplication algorithm (Cooper, 1970) is used to generate the products as shown in Table 2.

	Undelayed Signal				
Delayed Signal	SM	11	10	00	01
	11	+3	1	−1	−3
	10	1	0	0	−1
	00	−1	0	0	1
	01	−3	−1	1	3

Table 2:

The inner products are deleted which results in 1% loss to the correlator sensitivity (Cooper, 1976). A bias of +3 is added to the normalized multiplication Table 2 so that only positive numbers need to be added using full adders during each successive multiplication. Table 3 shows the final multiplication algorithm used in the hardware realization.

	Undelayed Signal				
Delayed Signal	SM	11	10	00	01
	11	6	4	2	0
	10	4	3	3	2
	00	2	3	3	4
	01	0	2	4	6

Table 3:

The binary coded output from the multiplier is added using a four bit adder and the carry output from the adder is accumulated using ripple counters.

2.4 Accumulators

The product from the 2-bit multiplier logic is counted using binary ripple counters. The length of the binary counters is determined by the rate at which a computer will be allowed to read the counter values. Typically the computer readout will occur a few times a second to once a second. The counter length is determined by the number of product terms that can be accumulated during each integration time. However not all the bits of the counter chain are readout to the computer and this reduces the size of the readout logic. In practice $(M/2)-3$ bits are discarded where M is the total number of bits in the binary counter chain (Cooper, 1976). A simple interface circuit is used to read the counter values from each delay channel in a short time compared to the integration period to minimize the "dead time" between each integration.

3 128-channel CMOS Autocorrelator Spectrometer

The prototype 128-channel autocorrelator spectrometer used four CMOS correlator chips. The chip was custom-designed at the Microelectronic Research Center (MRC) at the University of Idaho (Canaris, Whitaker, 1990) and fabricated at the Hewlett-Packard foundry at Corvallis, Oregon using 1.2 micron CMOS process. It measures 5.24 X 4.32 millimeters and is packaged in a 48-pin quad flat pack for surface mounting on a printed wiring board. The chip has 32 channels, consumes 400 milliWatts of DC power at 40 MHz clock speed. It also has 24-bit binary ripple counter in each channel and interface circuits for the computer to readout the correlated data. The block diagram of the 128-channel autocorrelator spectrometer is shown in Figure 2.

The CMOS autocorrelator spectrometer consists of two hardware modules. They are: amplifier and 2-bit analog to digital converter module and the 128-channel autocorrelator module. In addition, a personal computer (PC) and a digital I/O board are used for the autocorrelator data acquisition and spectrum calculations. The hardware modules are fabricated in multilayer printed wiring boards with controlled line impedance for handling the high frequency digital signals. Surface mount devices (SMD) are used for most of the logic gates. The power consumption for the two modules is about 6 Watts at a 40 MHz clock speed.

3.1 Low Power 2-bit Digitizer

The first circuit in the 128-channel autocorrelator spectrometer is a 2-bit digitizer. The amplifier is a part of the digitizer module and it consists of an AGC amplifier, and an op-amp which drives three ultrafast TTL comparators. The op-amp provides gain to the analog signal and impedance matching at the comparator inputs. The analog input to the comparators is set at 3 volts peak-to-peak across an equivalent resistance of 150 Ohms.

The three comparators are ultrafast TTL comparators, 9698KR and 9696KR, made by Analog Devices, Inc. The 9698KR is a dual comparator chip and it is used as the magnitude comparator. The 9696KR is a single comparator chip used as the sign or zero-crossing detector. The threshold voltages - two for the magnitude comparators, $+VTH$ and $-VTH$,

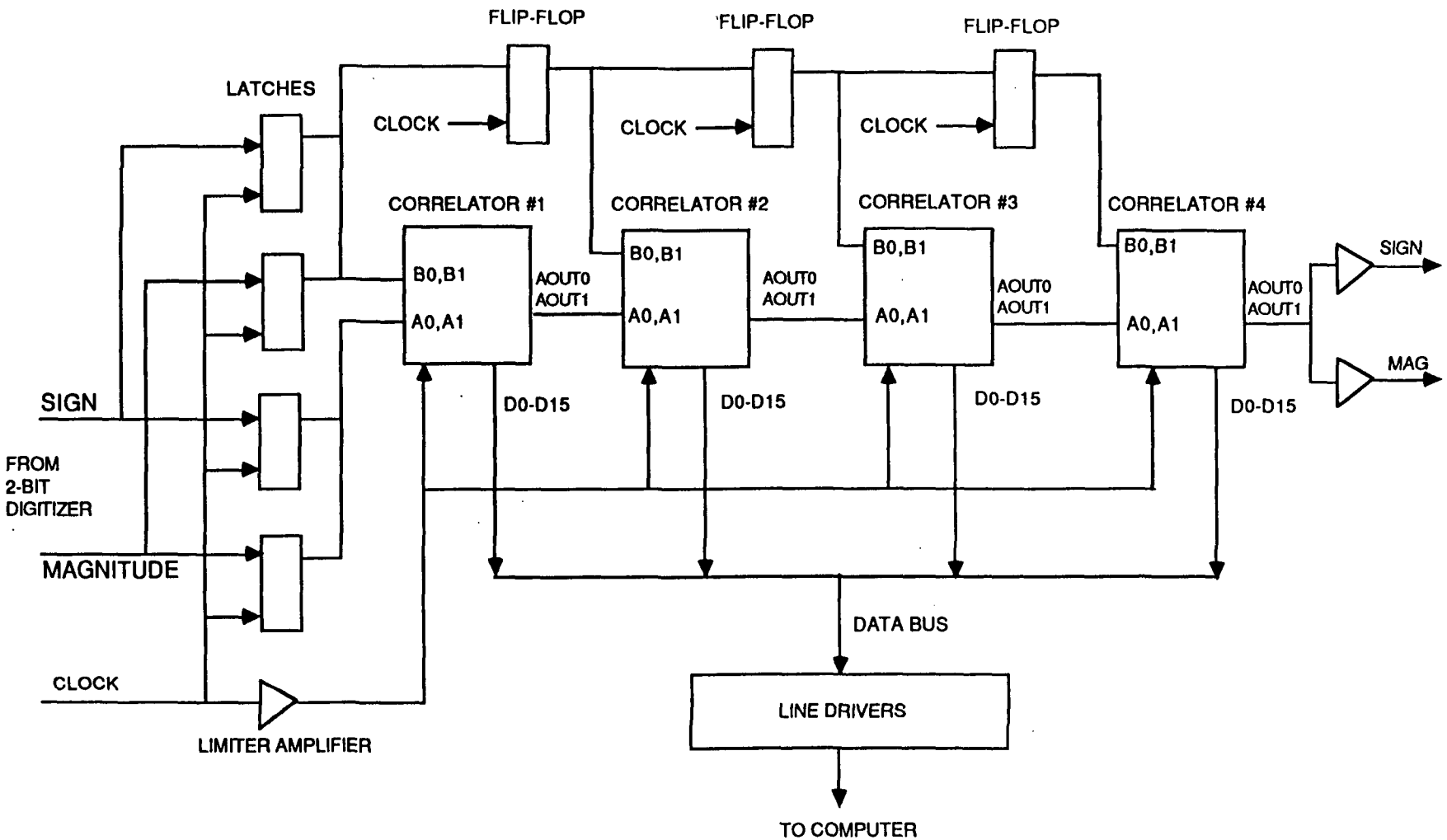


Fig. 2. 128-CHANNEL AUTOCORRELATOR BLOCK DIAGRAM

and one for the sign or the zero-crossing detector, VT0 - for the comparators are derived from a high stability voltage reference amplifier. The three threshold voltages can be set by low noise potentiometers. The three comparators digitize the input analog signal into a two-bit word shown in Table 1. The 2-bit data from the digitizer module is the input to the correlator module.

3.2 CMOS Correlator Integrated Chip

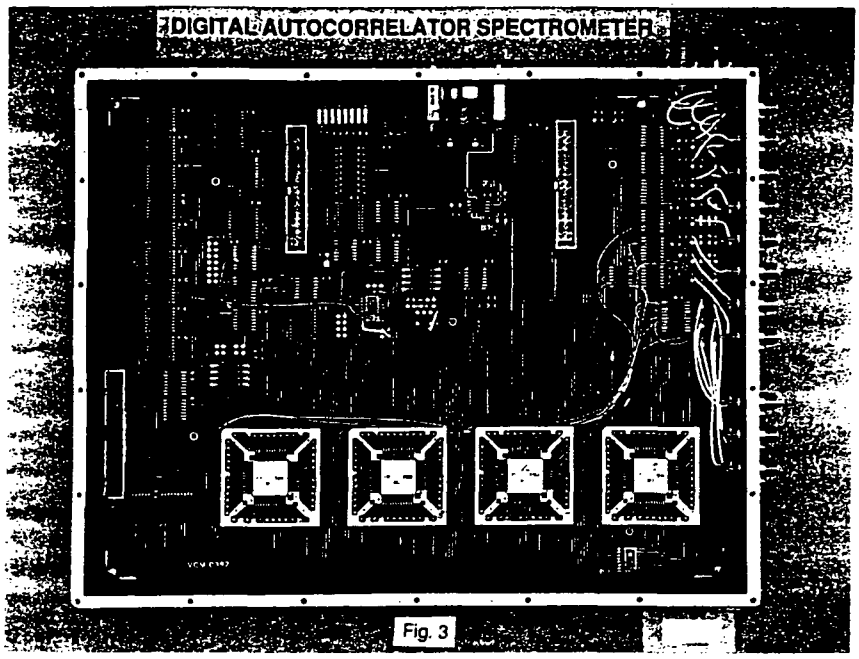
The CMOS correlator chip is an Application Specific Integrated Circuit (ASIC) implemented in VLSI technology. The chip performs 2-bit multiplication of digitized signal as given in Table 3. The main features of the correlator chip are listed below.

1. Autocorrelation or cross-correlation
2. 32 channels and integral binary ripple counter in each channel
3. Low power ($\leq$ 12 milliWatts per channel at 40 MHz clock rate)
4. Can be cascaded to increase the number of channels
5. Selectable auxiliary data input ports
6. Integration can continue while data is being sent to the computer
7. CMOS and TTL compatible input and outputs

3.3 128-channel Autocorrelator

The 128-channel autocorrelator spectrometer uses a four layer printed wiring board. The four correlator chips are mounted on a special carrier printed circuit board. The correlator board, shown in Figure 3, has circuits to drive the correlator clock input, to readout the channel data and a timer circuit to signal the end of an integration period. A programmable 32-bit timer IC counts the correlator clock and generates a pulse every 0.8 second. This pulse interrupts the correlation and the chip transfers the correlated counts in the 24-bit binary ripple counter into a shift register within the correlator chip. The chip also sets a data ready flag. The computer polls the data ready flag and resets the timer for a new integration cycle. The computer then reads the counter values in the shift register by generating a series of readout clocks until all the counter data is read. The correlator board has a hardware jumper connection to select the data output in either 8-bit byte or 16-bit word mode. The computer can also do post integration of the counter values.

The input clock and the output data lines of the four correlator chips are connected in a bus structure. The input clock to the correlator chips is also the sampling clock used in the digitizer. The prototype correlator chips require that the clock "high" input should not be more than 2.8 Volts. A limiter amplifier is used to limit the clock amplitude to the correlator chips. Also, to reduce the current transients due to simultaneous switching, the computer generated readout clock is skewed with respect to the correlator clock edges.



3.4 Computer Interface

The output data from the correlator module is CMOS/TTL compatible. A Burr-Brown, digital I/O board (PCI-20087W-1) is used with the computer to read the data from 128 channels in byte mode. The digital I/O board, under program control, reads each correlator chip's counter values from the output shift register by generating 96 readout clock pulses. Under program option, post integration and the averaging of the channel data can be performed in the computer before the Fourier transform is done on the autocorrelator data. Also, before the Fourier transformation, a continuous correlation function is obtained by applying a correction formula to the 2-bit correlation values.

$$R_{cont} = 1.146R_2 - 0.049R_2^2 \; for \; 0 \leq R_2 \leq 0.9 \tag{5}$$

$$R_{cont} = 1.34R_2 - 0.034R_2^2 \; for \; 0.9 \leq R_2 \leq 1.0 \tag{6}$$

where *Rcont* is the correlation coefficient for the continuous correlator and R_2 is the correlation coefficient using 2-bit digitization.

The above formulas are derived from the conversion table given by Cooper (1970). The computer uses separate programs to acquire the autocorrelation data, display the autocorrelation function, perform a 128-point cosine Fourier transform and display the power spectrum.

4 Power Spectrum Measurements

Tests on the correlator module were done with both static and dynamic inputs. The test setup is shown in Figure 4. A frequency synthesizer was used for generating the sample clock and a second synthesizer was used as the analog signal source for measuring the frequency and spacing of the channels. A stable, wideband noise source was also used as the input signal for the power spectrum measurements. The synthesizer frequency for the sampling clock was set at 25 MHz. The amplitude from the second synthesizer was set to give a 3 volts peak to peak signal at the three comparator inputs. The frequency of the second synthesizer was varied and the output spectrum from the autocorrelator spectrometer was displayed for each frequency setting. This test verified that the input frequencies appeared at the right channels after the autocorrelation and its Fourier transform. For the test with a noise source, a 19.8 MHz low pass filter was used at the digitizer input to eliminate alising of frequencies above 40 MHz which was used as the (Nyquist) sampling frequency. The results from the spectrum measurements made with a CW signal and a band limited noise signal are shown in Figure 5(a) and 5(b).

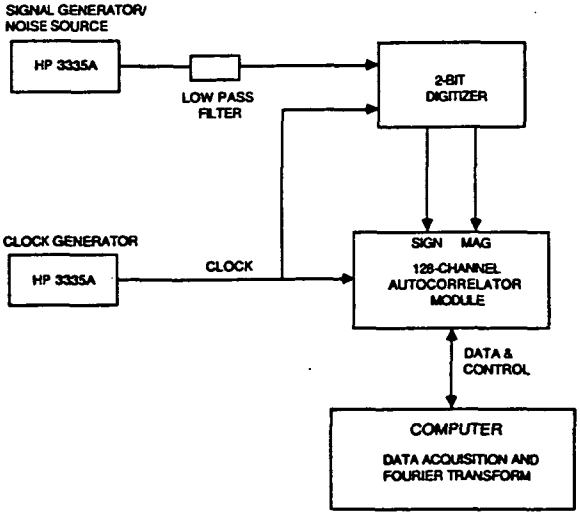


Fig. 4. 128-CHANNEL CMOS AUTOCORRELATOR TEST SETUP

One important test is to measure the stability of the spectrometer. According to the radiometer formula, the standard deviation of the power spectrum decreases with the increase in integration time as given by the equation:

$$\frac{\delta T}{T_{sys}} = \frac{2\beta}{\sqrt{B_N T}} \tag{7}$$

where $\pm T$ is the expected RMS value,
T_{sys} is the system noise temperature,
$B_N = 0.5/\tau_{max}$ is the noise bandwidth
β is the quantization loss factor (= 1.13 for two-bit correlator),
T is the integration time.

The factor 2 in the above equation is due to spending equal time on the signal and reference measurements and then subtracting two noisy signals for the RMS calculations. For the RMS calculations, two sets of spectra were measured, each set of two spectra for 0.8 and 80 seconds, with no difference in the input power level. One of the spectra is subtracted from the second and is gain normalized by the second spectrum. This is generally referred to as Y-factor measurement and expressed as where $\pm T$ is the expected RMS of input spectrum. Normally the Signal has the spectral feature of interest and the Reference is a comparison spectrum having uniform power over the bandwidth. The stability test showed that the spectrum RMS decreased with the increase in integration time and agreed closely with the calculated value.

5 Conclusion

Though the CMOS correlator chips were initially specified for 25 MHz clock speed, the tests revealed that the chips can be clocked at 40 MHz. Due to its low DC power consumption, and stability of at least 80 seconds the spectrometer is well suited for signals with bandwidth up to 20 MHz and high spectral resolution.

Acknowledgement

The digital autocorrelator spectrometer development was performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration. The CMOS chip development and the spectrometer development were supported by the NASA Civil Space Technology Initiative (CSTI).

References

[1] K. Chandra, J. Tarsala, H. Pickett and W. J. Wilson, "VLSI Correlator Chip for Space-Borne mm-Wave Radiometer Spectrometer", 2nd NASA SERC Symposium on VLSI design, 1990.

[2] F.N.H. Robinson, "Noise and Fluctuations", Clarendon Press, Oxford, 1974.

[3] B. F. C. Cooper, "Correlators with two-bit Quantization", Aust. J. Physics, Vol. 23, 1970.

[4] B. F. C. Cooper, "Autocorrelation Spectrometers", Methods of Experimental Physics, Vol. 12 - Part B, Chapter 3.5, Academic Press, 1976.

[5] J. Canaris and S. Whitaker, "A High Speed CMOS Correlator", 2nd NASA SERC Symposium on VLSI design, 1990.

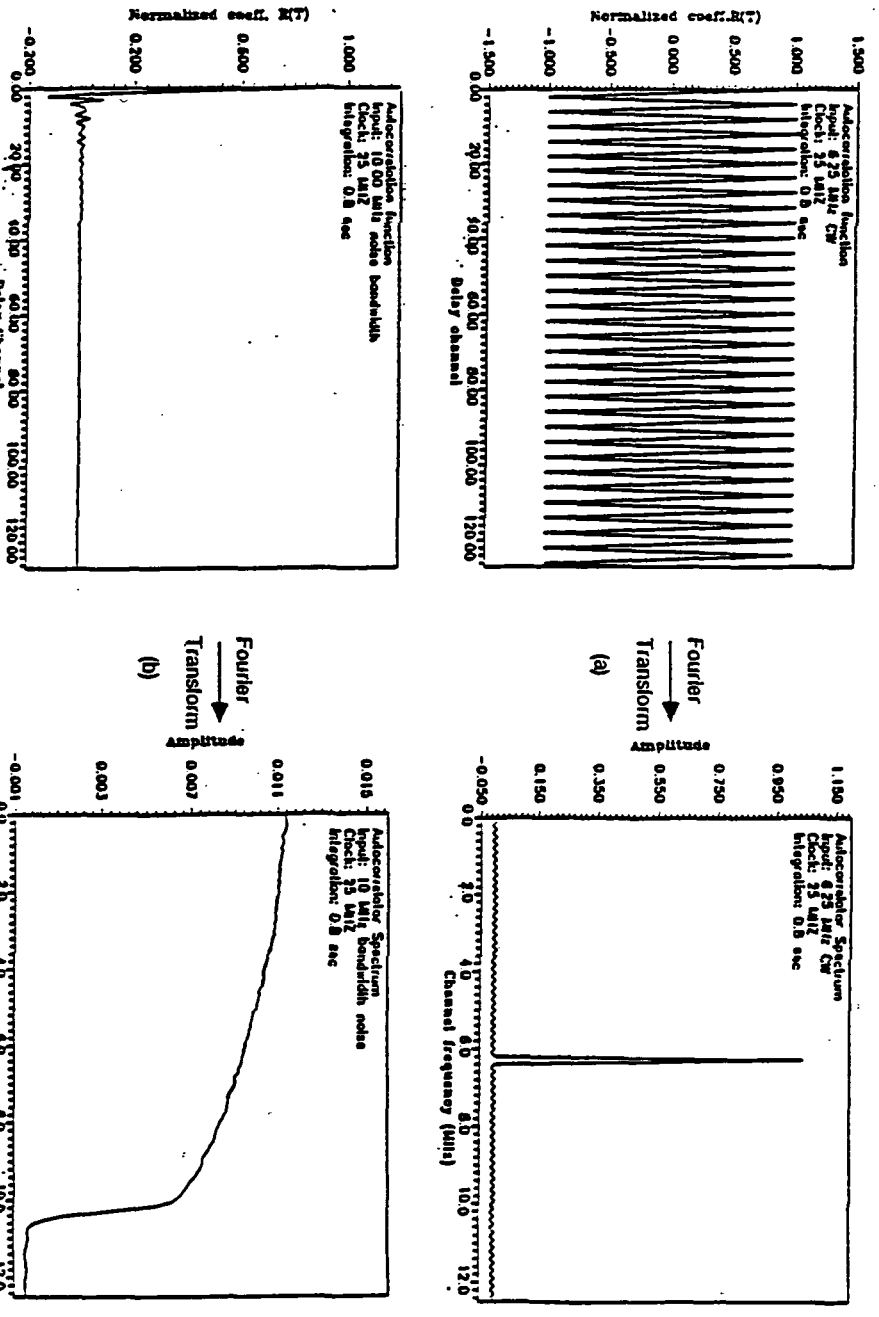


Fig. 5. AUTOCORRELATOR SPECTRUM MEASUREMENTS

A Real Time Correlator Architecture Using Distributed Arithmetic Principles

A. Benjamin Premkumar and T. Srikanthan
Nanyang Technological University
School of Applied Science
Singapore - 2263

Abstract- **A real time correlator design based on the principles of Distributed Arithmetic (DA) is described. This design is shown to be more efficient in terms of memory requirement than the direct DA implementation, specially when the number of coefficients is large. Since, the proposed architecture implements the sum of product evaluation, it can be easily extended to finite and infinite response filters. Methods to further reduce the memory requirements are also discussed. A brief comparison is made between the proposed method and different DA implementations**

1 Introduction

The increasing flexibility and decreasing cost of computing technology have made real time Digital Signal Processing (DSP) both possible and cost effective. In many areas such as radar and sonar detection, speech processing etc., real time signal processing is best performed by special purpose hardware. An operation that is common to many signal processing applications is the sum of products evaluation. Some of the most commonly used DSP implementations using the sum of products operation are: Finite Impulse Response filters (FIR), Infinite Impulse response filters (IIR) and Fast Fourier Transforms (FFT). This type of computation is executed most efficiently by DA principles. The advantage of DA is its efficiency of mechanization of the sum of products operation. For example, the DFT can be evaluated using DA, since the expression for the DFT is given by: $X(k) = \frac{1}{N}\sum_{n=0}^{N-1} x(n)e^{-j2\pi\frac{n}{N}k}$, and contains the sum of products. Generally, the FIR filter can be described by the equation $y_m = \sum_{k=0}^{K-1} a_k x_{m-k}$, where the infinite sequences $\{x_j\}$, $\{y_i\}$ are the input and the output sequences, respectively, and a_k is the set of K coefficients. This expression is very general, for a trivial modification to the above expression, converts it into a cross correlation operation, $y_m = \sum_{k=0}^{K-1} a_k xm + k$. Here, $\{x_i\}$ is an infinite sequence of samples of data which is cross correlated with K sample reference function to yield an infinite sequence of cross correlation samples, $\{y_i\}$. Since, both FIR and IIR filters can be described by similar equations, any architecture implementing one could well be used to implement the other.

In this paper, we propose an architecture which evaluates the sum of products efficiently using DA principles. In the implementation of the architecture, modifications have been made to the direct DA implementation resulting in reduced memory and faster throughput with minimum initial latency. Although, the implementation described is based on real numbers, the principle can be extended to complex numbers easily.

2 The Correlator

Several hardware implementations of the correlator architecture are available in literature [1, 2, 3]. In all these implementations, emphasis is laid on the multipliers to speed up product evaluation. However, in the implementation of correlator using the DA principle, emphasis is laid on the reduction in memory and hardware, since no multiplication is performed in the evaluation of the product of two numbers. The principle underlying the design of the proposed correlator is the somewhat different implementation of the DA concept.

It is well known that DA is a computational operation that forms the inner product of a pair of vectors [4]. DA is considered slow because of its bit serial nature of computation. However, this seemingly slow bit serial nature of DA is not real if the number of bits in each element of the vector is nearly equal to the number of elements in the vector. As an example of DA implementation of the sum of products, consider the expression:

$$y = \sum_{k=1}^{K} a_k x_k \tag{1}$$

where, a_k are the coefficients and the x_k are the input data. This expression for the sum of products can be rewritten as:

$$y = \sum_{k=1}^{K} a_k \sum_{n=0}^{N-1} b_{kn} 2^n \tag{2}$$

where each x_k has N bits. Interchanging the order of summation results in:

$$y_n = \sum_{n=0}^{N-1} \{ \sum_{k=1}^{K} a_k b_{kn} 2^n \} \tag{3}$$

Since b_{kn} can take values of either 0 or 1, it is easier to precompute the values for $\sum_{k=1}^{K} a_k b_{kn}$ and store them rather than compute them as and when b_{kn} arrive. This would mean that a memory of 2^K is needed for storing $\sum_{k=1}^{K} a_k b_{kn}$ for different combinations of b_{kn}. Stanley White, in his paper [4], has proposed a way by which the memory can be reduced by a factor of 2 by recasting the input data as offset binary code instead of straight binary code.

Modifications to the principles described above, have been made and implemented as architectures [5, 6, 7]. A correlator based on DA is described in [8] in which the sum of products expression is converted into three sums. By changing the order of summation of the three sums and using negabinary representation for the 2's complement input data, sum of products is evaluated. In the design to be discussed, the sum of products is evaluated using look up tables and high speed adders. In this design, the size of the look up tables used is smaller than that used in the direct DA implementation. The following sections describe the correlator implementation and also discuss the modifications to the proposed method which result in further reduction in memory. A comparison is made with the direct DA design of the correlator using 2 bits at a time (2BAAT) and 4 bits at a time (4BAAT).

3 Design of the Correlator

The proposed correlator is based on the design explained in reference [9]. In the evaluation of the expression $y = \sum_{k=0}^{K-1} a_k x_{m-k}$, it is seen that y at any instant contains the sum of products of all the input samples until that instant with all coefficients. Hence, it would be efficient to multiply all coefficients with each sample as it is input to the correlator and store them temporarily. When a new sample arrives at the input, it is also multiplied with all coefficients and then added to the delayed sums generated until the previous sample. However, in the proposed design, no multiplication of the input sample with the coefficients takes place. Look up tables in which the products of the coefficients and input data (for different combinations of the bits in the input data) are stored are used. A block diagram of the architecture is shown in Fig. 1.

The correlator is capable of accepting data serially and producing correlated output with a latency of one clock period. As can be seen from the figure, the number of tables required is equal to the number of coefficients used in the correlation. The input data sequence is fed serially into all coefficient tables. The bit pattern of each input data is used to access the memory location in which the product of the coefficient with that particular bit pattern is stored. The accessed data enters the summer where it is added with the delayed data corresponding to the previous input. Delay units in the circuit enable the previous sum to be added to the product of the current input and the coefficients. The output is available at the last adder unit with a delay corresponding to one clock cycle.

Storing the products of the coefficients with the input data requires a memory of size $M = n \times 2^b$, where, M is the total memory required, n is the number of coefficients and b is the number of bits in the input data. However, it is possible to partition the input data into two fields of $\frac{b}{2}$ bits each and store data with suitable weights assigned to the partitioned bits. This way, it is possible to reduce the total memory required. For any input, the products corresponding to the two partitioned groups of data bits are accessed and then added. This extra addition is not an overhead, since the existing adders can be clocked at twice the input data rate. Consider, for example, the following case: Supposing the number of coefficients is 8 and the input data width is 8 bits, direct implementation of the look up tables will require a total memory, $M = 8 \times 256$. However, partitioning the input data into two groups of 4 bits each results in a total memory requirement of $M = 2 \times 8 \times 15$, a considerable saving of memory. The memory can be further reduced by a factor of 2 by partitioning input data into smaller groups of 2 bits each. However, an additional set of adders will be required whenever data is accessed from the table. Fig.2 illustrates the architecture using this principle. With single partitioning, the total memory requirement is given by the expression: $2n(2^{\frac{b}{2}} - 1)$ for b even. For b odd, the memory requirement is given by the expression $M = n(2^{\frac{b+1}{2}} + 2^{\frac{b-1}{2}} - 2)$. The adders in the architecture can be speeded up using carry save techniques. In the correlation operation, the result, y, at the n^{th} sample instant is not expected at the output until n samples arrive at the correlator input. Hence, the summing operation can be performed in the carry save domain, where the propagation and the addition of the carry is deferred until the very last adder where the

Parameter	1BAAT		2BAAT		4BAAT		Proposed	
No. of Bits (input)	8	16	8	16	8	16	8	16
No. of Coeffts (K)	4	12	4	12	4	12	4	12
No. of Add. lines	3	11	7	23	15	47	7	13
Output Data/Clock	0.5	0.75	1	1.5	2	3	1	1
Memory Required	m	m	m^2	m^2	m^4	m^4	m_1	m_2

Table 1: Comparison between the DA and Proposed Method

output is available. This eliminates the delay due to carry propagation in the intermediate adders, thus speeding up the overall sum of product operation.

4 Comparison with Conventional DA Approach

In the direct DA implementation, with input data at the rate of 1 bit at a time (1BAAT) computation of the sum of products is slow. If b be the number of bits in the input data, b clock cycles are needed to form the sum of products. However, equivalent of K separate products are being formed during b clock cycles. Thus, it is seen, that if K is greater than b, then the DA processor is faster than the conventional multiplier in the evaluation of the sum of products. However, the number of inputs to the memory may restrict the number of coefficients. The computation can be speeded up at the expense of linearly increased or exponentially increased memory [4]. In the case of linearly increased memory, the input word is partitioned into L subwords. The memory requirement is determined by the number of bits in each subword. The data is input as subwords and this increases the speed by a factor of L. However, high speed and large capacity accumulators are needed. Because of the bit serial nature of the input data, shift operations become mandatory. By inputing data, p bits at a time, the memory increases exponentially and is given by the expression $M = \frac{1}{2}\{2^{Kp}\}$. Since the memory increases exponentially with the number of coefficients, this approach to DA implementation becomes impracticable in applications where K is large. Also, in this approach, the number of input lines to the memory is directly related to the number of coefficients. Thus, VLSI implementation of this method to applications where K is large, becomes exceedingly difficult. In the correlator described, the memory requirement is governed by the number of bits in the input data. Since, in most of the signal processing applications, either 16 or 8 bit data resolutions are sufficient, the memory requirements are not large. As stated earlier, by either increasing the number of adders or speeding up the adders, one might be able to reduce the memory considerably. Table 1 compares the direct DA implementation with p bits at a time with the proposed correlator with respect to the speed and memory requirements. In the table, m, m_1 and m_3 are defined as follows: $m = 0.5 \times 2^K$, $m_1 = 2K(2^4 - 1)$ and $m_2 = 2K(2^8 - 1)$. The expression for m assumes that the input data is recast into offset binary code. This table, however, does not take into account the additional time required in shifting the data for alignment in the case of 1BAAT, 2BAAT and 4BAAT approaches. This additional time may significantly reduce

throughput rates.

5 Conclusion

DA is a very efficient means of evaluating the sum of products. However, direct implementation of DA principles involves use of large memory. The memory requirements are even larger if the speed of the computation is to be increased or whenever the number of coefficients required is large. In this paper, a new correlator has been proposed in which the input data is partitioned into fields consisting of smaller number of bits. This way the total memory needed is reduced. The only additional requirement is that the existing adders be clocked at twice the data rate or high speed adders be used. In the design, carry save adders have been proposed, since in the evaluation of the sum of products, the output is not expected until all the samples arrive at the correlator. This design lends itself very well for VLSI fabrication of the correlation operation.

References

[1] J. Canaris and S. Whitaker, "A High Speed CMOS Correlator", *2nd NASA SERC Symposium on VLSI Design*, Univ. of Idaho, Moscow, Idaho, Nov. 1990.

[2] A. Peled and B. Liu, "A New Hardware Realization of Digital Filters", *IEEE Trans. on Acoustics, Speech and Signal Processing*, vol. ASSP - 22, Dec. 1974, pp. 456-462.

[3] S. Zohar, "New Hardware Realizations of Nonrecursive Digital Filters", *IEEE Trans. on Computers*, vol. C-22, Apr. 1973, pp. 328-338.

[4] S. A. White, "Applications of Distributed Arithmetic to Digital Signal Processing: A Tutorial Review", *IEEE ASSP Magazine*, vol. 6, no. 3, July 1989, pp. 4-19.

[5] S. G. Smith and P. B. Denyer, "Efficient Bit Serial Complex Multiplication and Sum of Product Computation Using Distributed Arithmetic", *Proc., International Conference on Acoustics, Speech and Signal Processing*, Tokyo, Apr. 1986, pp. 2203-2206.

[6] C. F. N. Cowan and J. Mavor, "New Digital Adaptive Filter Implementation Using Distributed Arithmetic Techniques", *IEE Proc*, vol. 128, Pt. F, no. 4, Aug. 1981, pp. 225-230.

[7] C. S. Burrus, "Digital Filter Realization in Distributed Arithmetic", *Proc. European Conf. on Circuit Theory and Design*, Genoa, Italy, Sept. 1976.

[8] S. Zohar, "A VLSI Implementation of a Correlator/Digital Filter Based on Distributed Arithmetic", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, no. 1, Jan. 1989, pp. 156-160.

[9] A. B. Premkumar, J. Purviance and M. Shamanna, "A Two Dimensional VLSI Correlator", *Northcon Conference*, Seattle, Wasington, Oct. 1990.

Low Energy CMOS for Space Applications[1]

Ramesh Panwar and Leon Alkalaj
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California 91109

1 Introduction

The current focus of NASA's space flight programs reflects a new thrust towards smaller, less costly and more frequent space missions, when compared to missions such as Galileo, Magellan, or Cassini. Recently, the concept of a *microspacecraft* has been proposed, whereby a small, compact spacecraft that weighs tens of kilograms performs focused scientific objectives such as imaging. Similarly, a Mars Lander *micro-rover* project is under study that will allow miniature robots weighing less than seven kilograms to explore the Martian surface. To bring the microspacecraft and microrover ideas into fruition one will have to leverage compact 3D MCM technologies. Low energy CMOS will become increasingly important because of the thermodynamic considerations in cooling compact 3D MCM implementations and also from considerations of the power budget for space applications. In this paper, we show how the operating voltage is related to the threshold voltage of the CMOS transistors for accomplishing a task in VLSI with minimal energy. We also derive expressions for the noise margins at the optimal operating point. We then look at a low voltage CMOS (LVCMOS) technology developed at Stanford University which improves the power consumption over conventional CMOS by a couple of orders of magnitude and consider the suitability of the technology for space applications by characterizing its SEU immunity.

2 MOS Transistor Model

A MOS transistor has three regions of operation where the behavior is described by the following equations [1]:

$$I_{ds} = \begin{cases} \beta n V_T^2 e^{(V_{gs}-V_t)/nV_T} & \text{if } V_{gs} < V_t \\ \beta[(V_{gs}-V_t)V_{ds} - V_{ds}^2/2] & \text{if } 0 < V_{ds} < V_{gs} - V_t \\ \frac{\beta}{2}(V_{gs}-V_t)^2 & \text{if } 0 < V_{gs} - V_t < V_{ds} \end{cases}$$

where

I_{ds}	=	drain current
V_{gs}	=	gate-source voltage
V_{ds}	=	drain-source voltage
V_t	=	threshold voltage
V_T	=	thermal voltage
n	=	a process parameter between 1.2 and 1.4
β	=	transistor gain factor

[1]Partial support for this work was provided by the Data Systems Program NASA Code-R CSTI

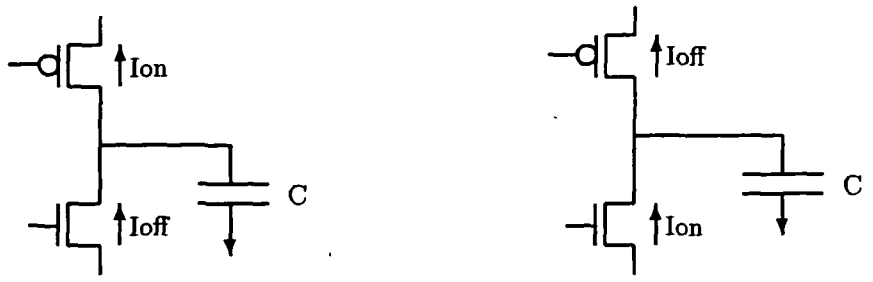


Figure 1: Gate capacitance charge/discharge

Note that the leakage current (I_{off}) through a transistor when the gate voltage is zero ($V_{gs} = 0$) is given by:

$$I_{off} = I_0 e^{-V_t/nV_T}$$

where $I_0 = \beta n V_T^2$. Thus the lower the threshold voltage (V_t), the higher the leakage current.

3 Gate Switching Time

Consider a complementary CMOS gate driving the gate of another transistor. The gate capacitance of the driven transistor will be charged or discharged by the driving gate. The charge needed to charge or discharge the gate capacitance is given by:

$$Q = CV$$

where C is the gate capacitance and V is the voltage swing at the gate (usually the operating voltage unless we are using differential mode logic). Let the time required to switch the gate capacitance be denoted by t_p. The charge Q is then given by:

$$Q = (I_{on} - I_{off})t_p$$

where I_{on} is the saturation current through the ON device and I_{off} is the leakage current through the OFF device. We thus have the following expression for the gate switching time:

$$t_p = \frac{CV}{\frac{\beta}{2}(V - V_t)^2 - I_0 e^{-V_t/nV_T}}$$

The charging/discharging of the gate capacitance by the ON/OFF currents is shown in Figure 1. We are now in a position to consider the energy required for VLSI computation.

4 VLSI Task Energy

The energy needed to accomplish a given task in VLSI is equal to the sum of the AC energy and the DC energy. The AC energy is consumed during computation due to transistors

switching state, while the DC energy is the energy that is dissipated due to leakage currents. Let us assume that we can attain a speedup $s(m)$ in a VLSI computation where m is the number of transistors, and let t_p denote the gate switching time of a transistor. The time required to complete a task is then expressed as $kt_p/s(m)$ where k is some constant for the task. The AC energy for the task is expressed as:

$$E_{ac} = \frac{1}{2} amCV^2 fkt_p/s(m)$$

where f is the frequency of operation and a is the average fraction of transistors that switch in a clock cycle. The DC energy for the task is given by:

$$E_{dc} = mI_{off}Vkt_p/s(m)$$

Thus the total energy for the task is given by:

$$E = E_{ac} + E_{dc}$$

By using the expressions for t_p and the leakage current I_{off}, it can be shown that:

$$E = \frac{km(afCV^2 + 2I_0Ve^{-V_t/nV_T})CV}{s(m)(\beta(V - V_t)^2 - 2I_0e^{-V_t/nV_T})}$$

This is minimized when

$$\frac{\partial E}{\partial V} = 0$$

which leads us to a cubic equation of the form:

$$c_3V^3 + c_2V^2 + c_1V + c_0 = 0$$

where the coefficients are given by:

$$\begin{aligned} c_3 &= af\beta C \\ c_2 &= -4af\beta V_tC \\ c_1 &= 3af\beta CV_t^2 - 4\beta I_0V_te^{-V_t/nV_T} - 6afI_0Ce^{-V_t/nV_T} \\ c_0 &= 4kI_0V_t^2e^{-V_t/nV_T} - 8I_0^2e^{-2V_t/nV_T} \end{aligned}$$

Note that this means that the optimal supply voltage is dependent not only on the threshold voltage but also on various other parameters like the frequency of operation, the activity of the logic, the capacitance of the nodes, and the transistor geometry. However, for sufficiently high threshold voltages where $V_t \gg nV_T$, the cubic equation reduces to the following quadratic:

$$V^2 - 4V_tV + 3V_t^2 = 0$$

whose solution is given by $V = 3V_t$. This result is independent of the activity and the frequency of operation and suggests that the threshold voltage should be set to a third of the supply voltage for optimal operation. We next show that the relative noise margins are unaffected by the operating voltage.

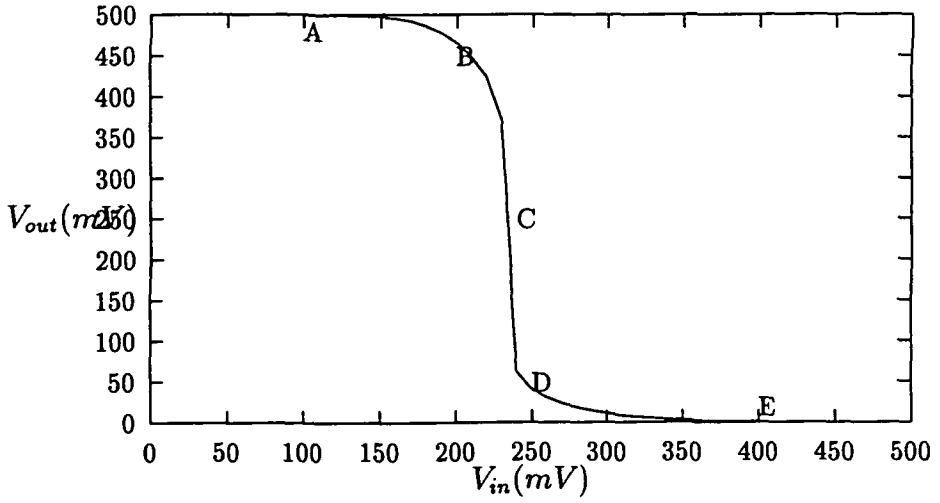


Figure 2: Regions of operation of an inverter

5 Noise Margins

The operation of a CMOS inverter can be divided into five regions [1]:

- Region A. In this region p-device is in the linear region while the the n-device is cut-off.
- Region B. In this region the p-device is in the linear region while the n-device is in saturation.
- Region C. In this region both the n- and p-devices are in saturation.
- Region D. In this region the p-device is in saturation while the n-device is in the linear region.
- Region E. In this region the p-device is cut-off and the n-device is in the linear mode.

The noise margins are defined to be:

$$NM_L = V_{IL} - V_{OL}$$

$$NM_H = V_{OH} - V_{IH}$$

where the voltages V_{IL} and V_{IH} are given by the unity gain points and the voltages V_{OL} and V_{OH} are 0 and V_{dd} for ideal inverters [1]. The relative noise margins are defined to be the ratio of the noise margins to the operating voltage and are given by:

$$RNM_L = \frac{NM_L}{V_{dd}}$$

$$RNM_H = \frac{NM_H}{V_{dd}}$$

The unity gain points occur in regions B and D and are given by setting the derivative dV_O/dV_I to -1. It can be shown that the output voltage in region B is given by [1]:

$$V_O = (V_I - V_{tp}) + [(V_I - V_{tp})^2 - 2(V_I - V_{dd}/2 - V_{tp})V_{dd} - \frac{\beta_n}{\beta_p}(V_I - V_{tn})^2]^{1/2}$$

Assuming $\beta_n = \beta_p$ and $V_{tn} = -V_{tp} = \alpha V_{dd}$, we have by taking the derivative and setting it to -1, the following expression for V_{IL}:

$$V_{IL} = \frac{1}{8}V_{dd}(3 + 2\alpha)$$

The noise margin NM_L is thus given by:

$$NM_L = \frac{1}{8}V_{dd}(3 + 2\alpha)$$

and the relative noise margin RNM_L is given by:

$$RNM_L = \frac{3}{8} + \frac{1}{4}\alpha$$

Similarly, in region D, the expression for the output voltage is given by [1]:

$$V_O = (V_I - V_{tn}) - [(V_I - V_{tn})^2 - \frac{\beta_p}{\beta_n}(V_I - V_{dd} - V_{tp})^2]^{1/2}$$

Again, by taking the derivative and setting it to -1, we have:

$$V_{IH} = \frac{1}{8}V_{dd}(5 - 2\alpha)$$

The noise margin NM_H is thus given by:

$$NM_H = \frac{1}{8}V_{dd}(3 + 2\alpha)$$

and the relative noise margin RNM_H is given by:

$$RNM_H = \frac{3}{8} + \frac{1}{4}\alpha$$

Thus at the optimal operating point for minimal VLSI energy, the relative noise margins would be given by:

$$RNM_L = RNM_H = 11/24$$

Note that this depends on several assumptions of which the most important ones are:

- The absolute values of the threshold voltages are equal.

- The β values of the n- and p-devices are equal.
- The supply and threshold voltages are such that all devices are capabale of operating in all 3 regions of operation (subthreshold, linear and saturation) and that the inverter characteristic shows 5 regions of operation as outlined above.

The reason for considering relative noise margins as opposed to noise margins is because the noise on the chip will scale down as the supply voltage is scaled down. Note that the relative noise margins for any computation at the minimal VLSI energy are independent of the supply voltage and this appears very encouraging since it allows us to scale the supply voltage as much as possible. We now show the operating characteristics of a low voltage CMOS technology that can be operated with a wide range of operating voltages.

6 Stanford LVCMOS

Low voltage CMOS devices built at Stanford University allow chips to be operated with supply voltages of several hundreds of millivolts to 5V. The devices are manufactured such that the zero-bias threshold voltage of the n- and p-devices is almost zero [2]. The bulk bias is then adjusted to set the threshold voltage to the appropriate value. The substrate of the p-devices is tied to a voltage $V_{pb} = V_{dd} + V_{bb}$ where V_{bb} is the extra bulk-bias voltage while the substrate of the n-devices is tied to a voltage $V_{nb} = -V_{bb}$. One disadvantage of this technology is that three voltages (V_{dd}, V_{pb}, V_{nb}) need to be routed on the chip besides the ground. The routing of four voltage levels may adversely affect the integration density of the devices. We are looking into the effect of the 4 rails on the integration density as opposed to the dual rail design in conventional CMOS. Figures 4-10 show some of the characterizations that have been done for this technology. One can see that the devices fail to saturate when operated at a supply voltage of 100mV with no extra bulk bias. However, the devices can be made to saturate at the same supply voltage by an extra bulk-bias of 0.5V. Increasing the extra bulk-bias decreases the saturation current through the devices since it increases the absolute value of the threshold voltage. This also makes the devices slower as shown by the inverter propagation delays in Tables 1 and 2. Tables 1 and 2 show the relative noise margins and inverter propagation delays for the LVCMOS technology at operating voltages of 100mV and 500mV respectively. The tables show that the inverter propagation delays for an operating voltage of 500mV degrade very gracefully with increasing bulk-bias but the propagation delays for an operating voltage of 100mV degrade very badly with increasing bulk-bias. Also, since the devices do not saturate at an operating voltage of 100mV without a significant amount of extra bulk-bias, it means that the devices cannot be operated at 100mV except for really slow applications. The inverter propagation delay degradation is less with increasing V_{bb} for higher operating voltages. For a 5V operating voltage, the inverter propagation delay remains virtually constant at 0.11ns. Figures 11-13 show the inverter characteristics for an inverter in this technology for three different supply voltages. For a low energy operation, one would choose an operating voltage of 500mV and set the threshold voltage to be a third of the operating voltage by adjusting the back-bias to achieve optimal operation. We next consider the SEU immunity of this technology.

$V_{bb}(V)$	RNM_L	RNM_H	$t_{inv}(ns)$
0	0.00	0.55	2
0.5	0.35	0.54	8
1.0	0.30	0.60	680
2.0	0.25	0.64	5840

Table 1: Relative noise margins and inverter propagation delays at 100mV

$V_{bb}(V)$	RNM_L	RNM_H	$t_{inv}(ns)$
0	0.20	0.60	0.34
0.5	0.26	0.58	0.41
1.0	0.38	0.48	0.53
2.0	0.44	0.48	1.00

Table 2: Relative noise margins and inverter propagation delays at 500mV

7 SEU Immunity of LVCMOS

We now characterize the SEU immunity of the LVCMOS technology by looking at the critical charge needed to cause an SRAM cell upset. Figure 3 shows the SPICE model used for the SEU immunity study. The particle strike is modeled by a pulsed current source with a triangular pulse having a 1:9 rise:fall shape. The pulse height is adjusted using a binary search to determine the critical charge at which the cell is just upset [3]. The critical charge for the upset (Q_c) is given by the area under the current pulse. Simulations were performed for an operating voltage of 500mV for different bias voltages. Simulations show that the critical charge is quite independent of the pulse width for pulse widths below 500ps. [The typical alpha particle pulse width is 200ps]. Table 3 shows the critical charge for various pulse widths and bias voltages for an operating voltage of 500mV.

The results show that this technology is much more prone to SEUs when operated at 500mV because the critical charge is much smaller than conventional CMOS. However, the advantge of this technology is that the devices can be operated at higher operating voltages

$V_{bb}(V)$	20ps	200ps	2ns
0	42fC	42fC	49fC
0.5	41fC	41fC	47fC
1.0	38fC	39fC	45fC
2.0	33fC	35fC	42fC

Table 3: Critical charge for various pulse widths at 500mV

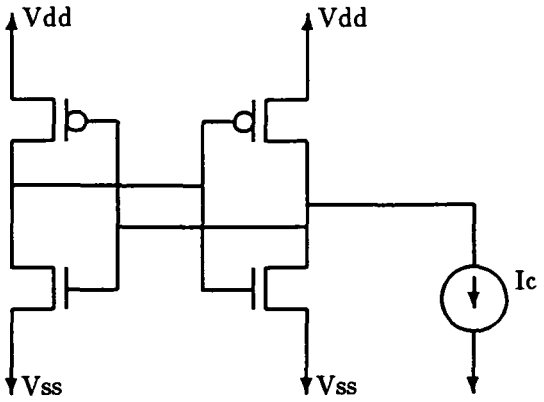


Figure 3: SRAM SEU model

$V_{dd}(V)$	Q_c
0.5	42fC
1.0	95fC
2.0	160fC
5.0	420fC

Table 4: Critical charge for various operating voltages

to ensure better SEU immunity. Table 4 shows the SEU immunity of the devices at $V_{bb} = 0V$ for various operating voltages when the SEU particle strike is modeled by a pulse of 200ps width. We see that the SEU immunity of the technology for an operating voltage of 5V is quite comparable to that of rad-hard CMOS. To achieve low energy operation, we can allow two different operating voltages for the chips. The chips will operate at a low voltage (say 500mV) in a clean environment and a higher voltage (say 5V) in a radiation intensive environment to ensure better SEU immunity. One can envision a system designed with a radiation detector that switches the operating voltage from 500mV to 5V when it encounters radiation and switches back to low voltage operation when the environment becomes clean again.

8 Conclusions

In this paper, we have presented the analysis for optimal supply voltage for achieving a task in VLSI with minimal energy. We also showed that the relative noise margins are independent of the operating voltage. We then showed some characteristics of the Stanford LVCMOS technology and showed how the logic speed depends on the operating voltage in this technology. The threshold voltage in this technology is adjusted by setting the back-bias appropriately. For optimal energy operation, the threshold voltage would have to be

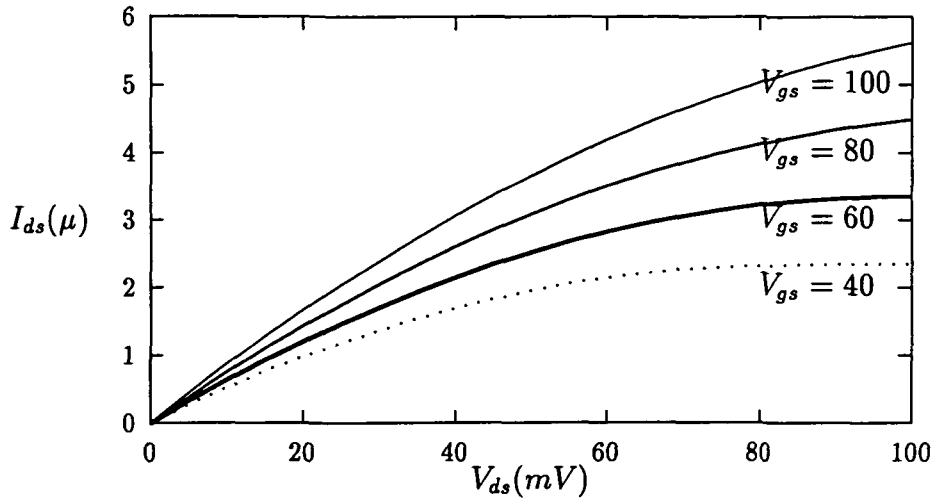


Figure 4: VI characteristic for $V_{bb} = 0V$ and 100mV operating voltage

set to a third of the operating voltage. We also characterized the critical charge for SEUs in this technology and showed it to be significantly less than conventional technology when the operating voltage was 500mV but comparable to rad-hard technology when the operating voltage is 5V. The benefit of the LVCMOS technology is that the supply voltage and the threshold voltage can both be changed and designed for more SEU tolerance. We propose the design of systems with two operating voltages - a lower operating voltage to achieve low energy in a clean environment and a higher operating voltage to achieve SEU tolerance in a radiation intensive environment. The results described in this paper represents work currently in progress. We need to explore further the tradeoffs between the energy efficiency and radiation tolerance by exploring not only the relationship between energy and SEU immunity but also the effect of fault tolerant design on the energy efficiency.

9 Acknowledgements

We acknowledge the discussions that we had with Martin Buehler, Bob Bunker and Bob Schober that helped us to understand the phenomena of SEUs in CMOS. We also acknowledge the discussions with Jim Burr who developed the LVCMOS technology and patiently answered the many questions that we had.

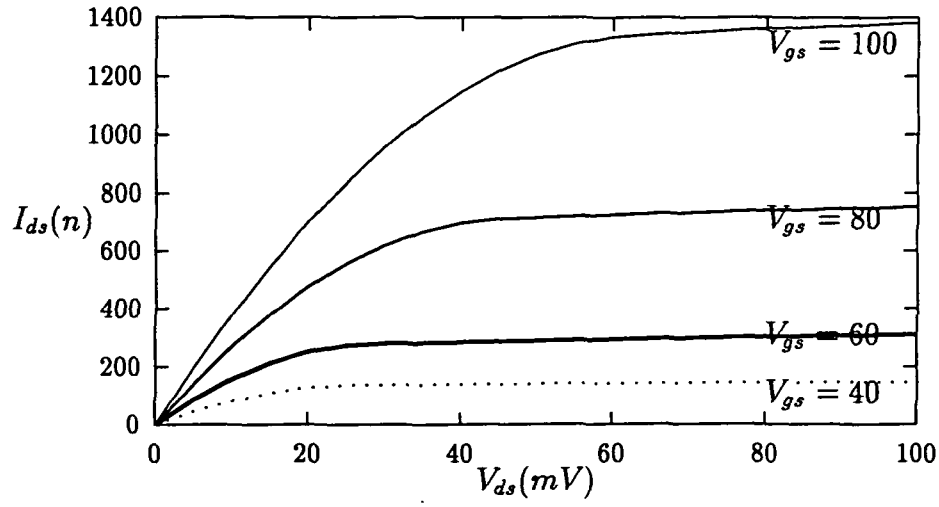


Figure 5: VI characteristic for $V_{bb} = 0.5V$ and 100mV operating voltage

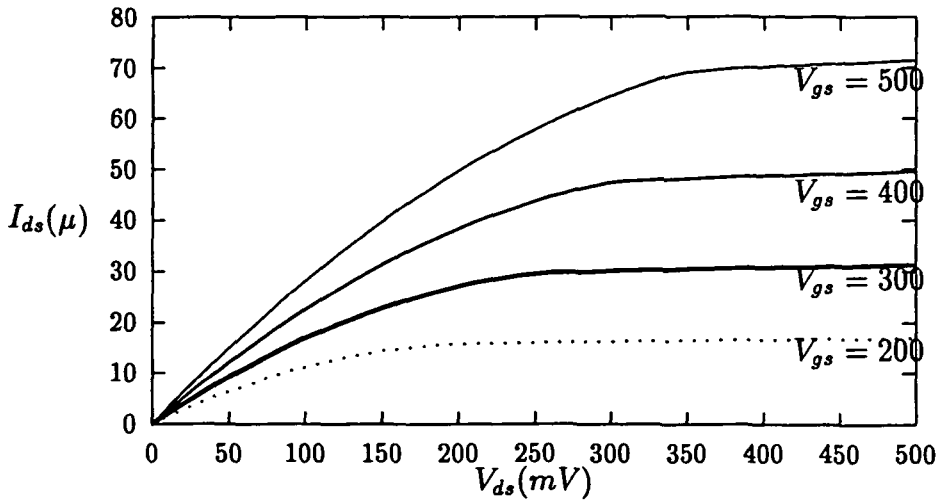


Figure 6: VI characteristic for $V_{bb} = 0V$ and 500mV operating voltage

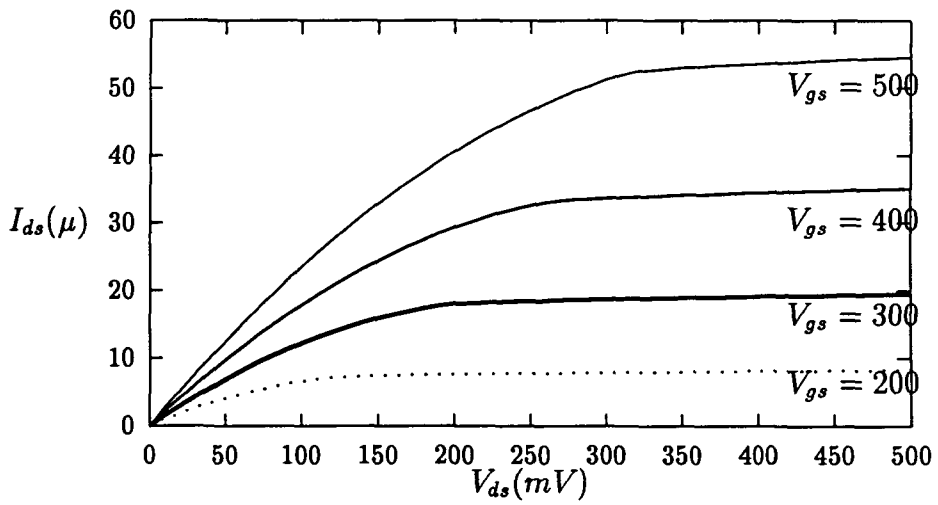


Figure 7: VI characteristic for $V_{bb} = 0.5V$ and 500mV operating voltage

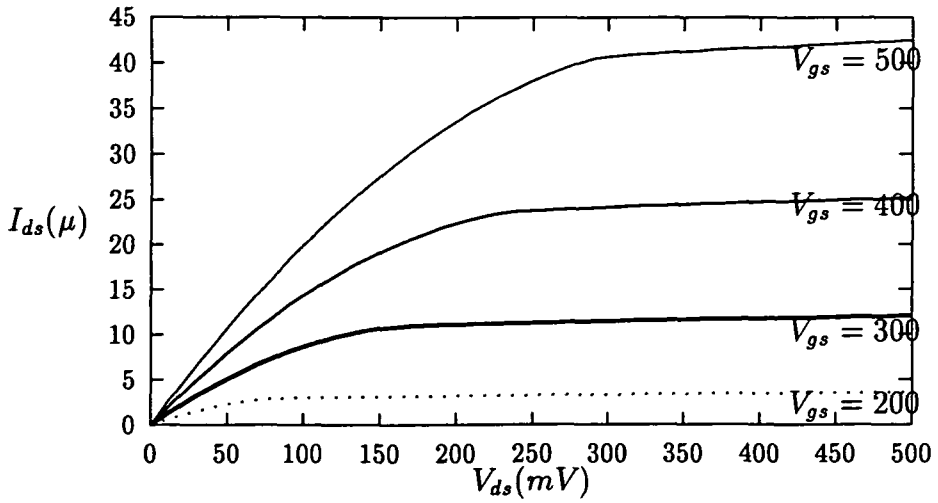


Figure 8: VI characteristic for $V_{bb} = 1V$ and 500mV operating voltage

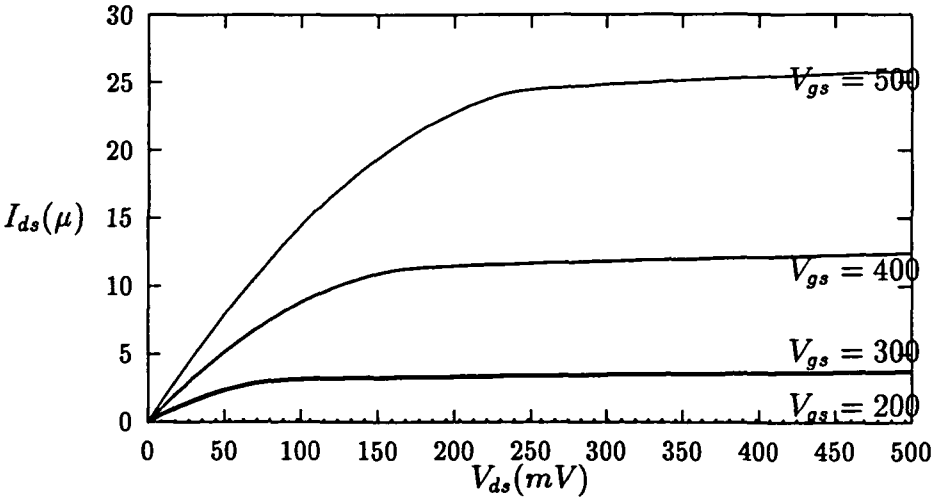


Figure 9: VI characteristic for $V_{bb} = 2V$ and 500mV operating voltage

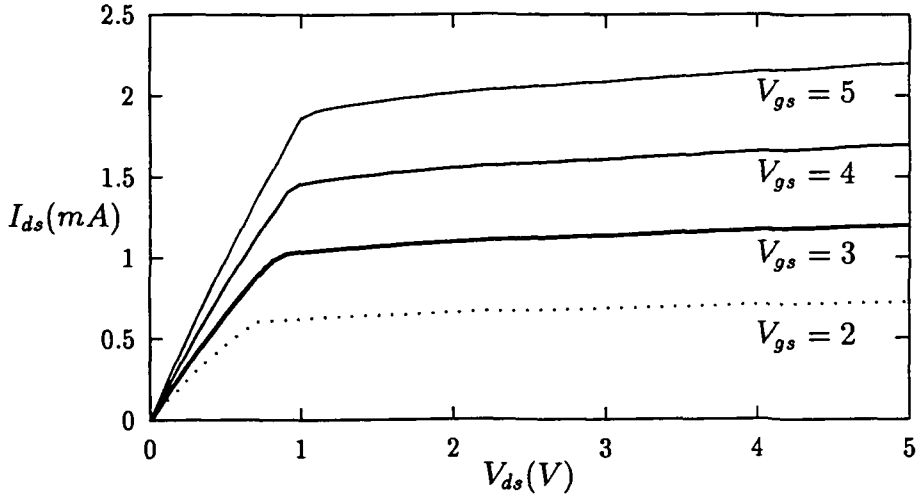


Figure 10: VI characteristic for $V_{bb} = 0V$ and 5V operating voltage

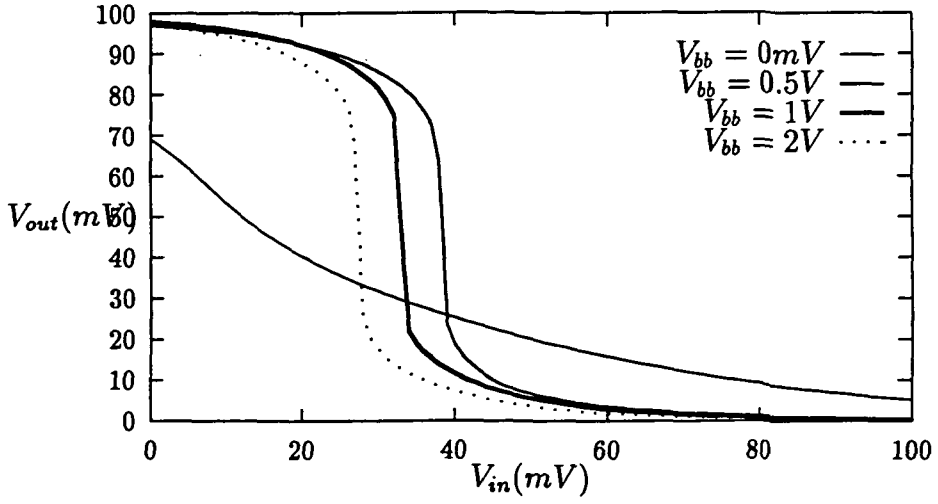


Figure 11: Inverter characteristic for 100mV operating voltage

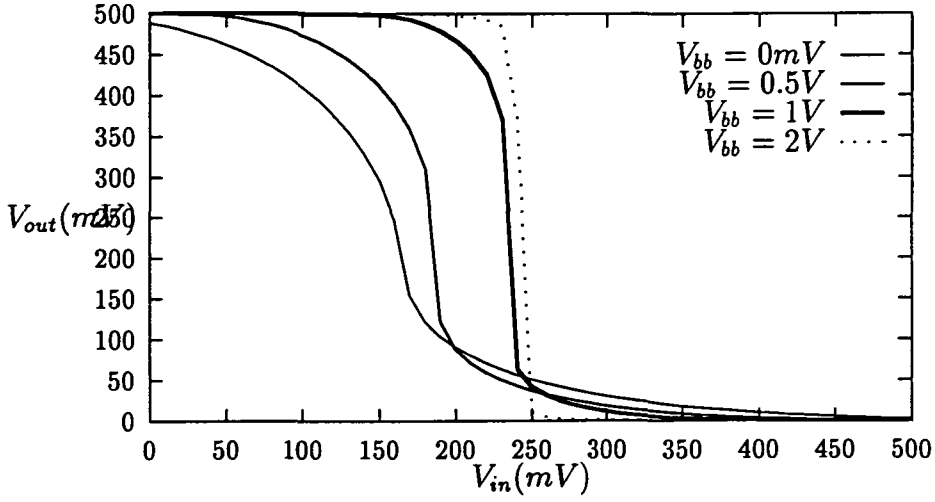


Figure 12: Inverter characteristic for 500mV operating voltage

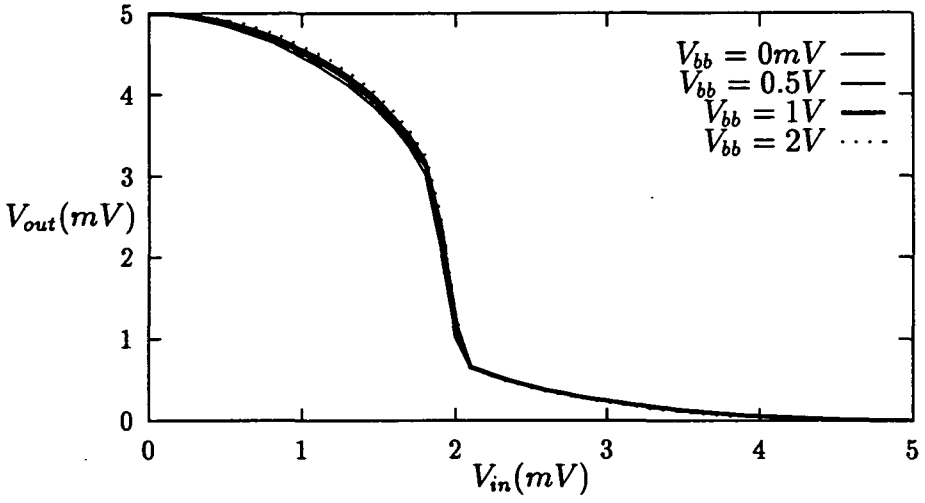


Figure 13: Inverter characteristic for 5V operating voltage

References

[1] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*, Addison-Wesley, Reading, MA. 1988.

[2] J. Burr and A. Peterson, "Energy considerations in multichip-module based multiprocessors," *IEEE International Conference on Computer Design*, 1991.

[3] M. Buehler and B. Blaes, "Alpha-particle Sensitive Test SRAMs," *IEEE Transactions on Nuclear Science*, Vol. 37, No. 6, Dec 1990.