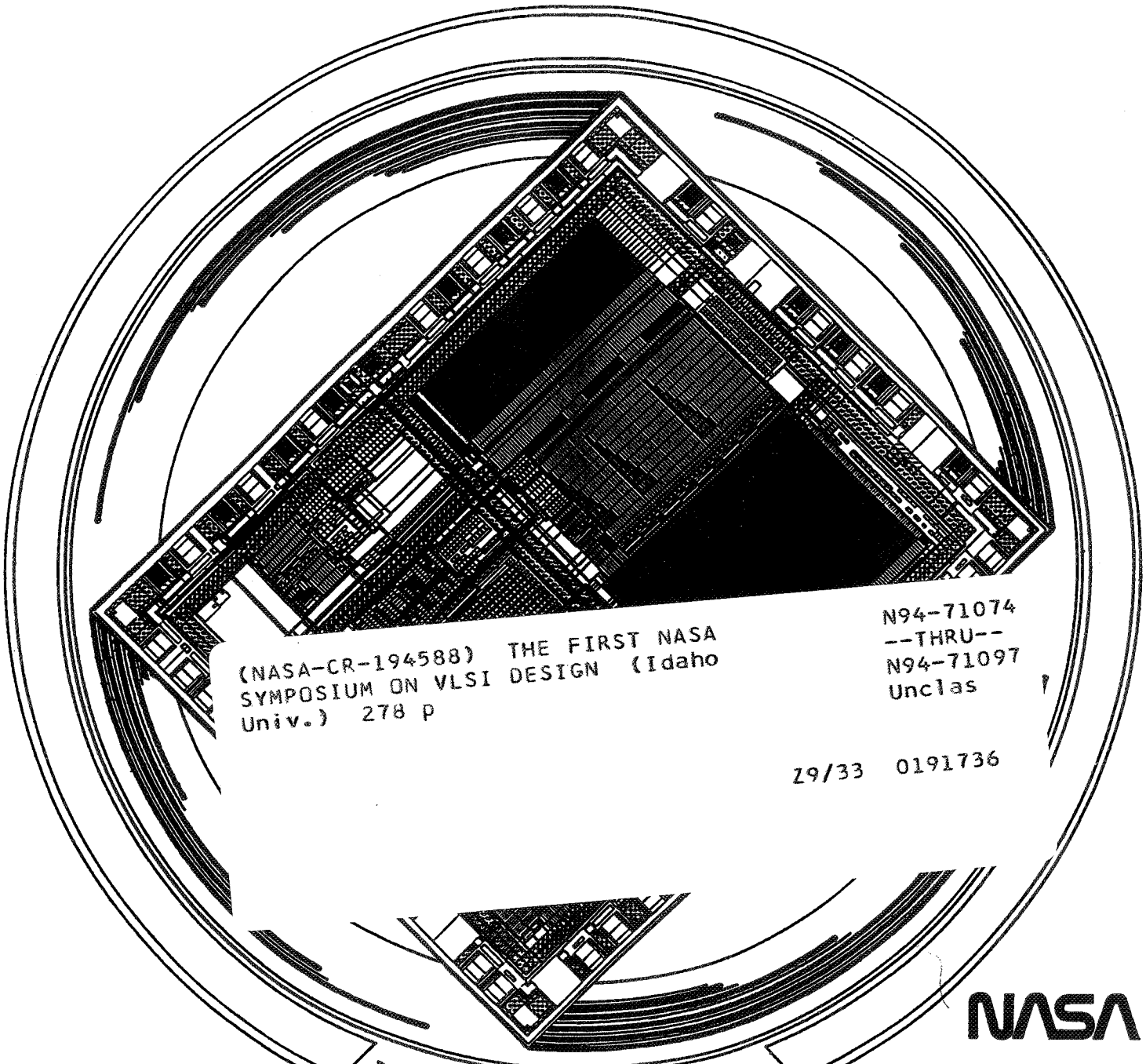
NAGW-3293

1990
First NASA Symposium on VLSI Design



NASA

National Aeronautics and Space Administration

Welcome to the first annual NASA Symposium on VLSI Design. NASA's involvement in this event demonstrates a need for research and development in the area of high performance computing. High performance computing addresses problems faced by the scientific as well as industrial communities. High performance computing is needed in:

- Manipulating large quantities of data in real time
- Sophisticated digital control of space craft systems
- Digital data transmission, error correction and image compression
- Expert system control of space craft

In addition to requiring high performance computing, NASA imposes the constraint of zero power, weight and space. Clearly, a valuable technology in meeting these needs is Very Large Scale Integration.

This conference addresses important issues of VLSI design.

- Digital System Architectures
- Electronics
- Algorithms
- CAD tools

It is clear that solutions to problems faced by NASA have commercial applications. One goal of this conference is to share technology advances with the industrial community and encourage interaction between industry and NASA.

This symposium is organized by the NASA Space Engineering Research Center 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 data system handling needs.

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

Clearly there are individuals whose assistance was critical to the success of this symposium. Barbara Martin worked long hours with every single manuscript to place into proper LaTeX form. Judy Wood did an excellent job at coordinating the many conference activities. The efforts of these professionals were vital and are greatly appreciated.

Our goal is to build upon this symposium in years to come and suggestions are encouraged that would allow a better symposium next year. I hope you enjoy your stay in Moscow, Idaho and I extend an invitation to visit the research laboratories during the symposium.

Gary K. Maki

Table of Contents

Automating Analog Design: Taming the Shrew

A. Barlow
Asahi Kasei Microsystems
Tokyo, Japan

1 Introduction

The march, or rather, the sprint of progress in integrated circuits continues to amaze observers both within and without the industry. Three decades ago, a 50 transistor chip was a technological wonder. Fifteen years later, a 5000 transistor device would "wow" the crowds. Today, 50,000 transistor chips will earn a "not too bad" assessment, but it takes 500,000 to really leave an impression.

In 1975 a typical ASIC device had 1000 transistors, took one year to first samples (and two years to production) and sold for about 5 cents per transistor. Today's 50,000 transistor gate array takes about 4 months from spec to silicon, works the first time, and sells for about 0.02 cents per transistor.

Fifteen years ago, the single most laborious and error prone step in IC design was the physical layout. Today, most IC's never see the hand of a layout designer: an automatic place and route tool converts the engineer's computer captured schematic to a complete physical design using a gate array or a library of standard cells also created by software rather than by designers. CAD has also been a generous benefactor to the digital design process. The architect of today's digital systems creates his design using an RTL or other high level simulator. Then he pushes a button to invoke his logic synthesizer-optimizer tool. A fault analyzer checks the result for testability and suggests where scan based cells will improve test coverage. One obstinate holdout amidst this parade of progress is the automation of analog design, and its reduction to semi-custom techniques. While the variety and power of architectural options available to the analog designer has mushroomed, his methods remain largely unchanged from two decades ago. Synthesis by repeated trial-and- error SPICE simulations is still the norm. The layout is still painstakingly hand-crafted, transistor by transistor. Unlike their digital counterparts, analog first silicon that does not perform to spec is yet the rule rather than the exception. Analog design has stubbornly refused to be tamed by the array and cell methodologies that have overwhelmed the digital world. While analog cell libraries are widely advertised, in practice they find very little use [1].

2 What's the Problem ?

The comparatively stunted growth of analog CAD has multiple causes. Some are natural consequences of macroeconomics and of the general state of the computing industry. Others

are intrinsic to the nature of the analog problem. Still others appear to be rooted in quirks of human nature. I will focus on three of the more significant barriers.

2.1 Help Wanted (Semicustom need not apply)

Gate array vendors quickly learned that like the memory product business, their's is basically a simple two dimensional problem: the trade off between speed and chip area. The definition of next year's new product line is ever so predictable: more speed, higher gate count, lower cost. And they can feel confident that their's is a reasonably broad and complete product line if it includes a half dozen arrays that span the range from 1k to 50k gates. Analog, by contrast is a multi- dimensional nightmare. If we try to offer a semi-custom, structured product line containing reconfigurable analog elements, in what ratio should we include amplifiers, capacitors, resistors, switches, free transistors, matched pairs, etc. How many different combinations constitute a complete family of such analog arrays? In our amplifier cell library, what combinations of DC gain, bandwidth, noise, PSRR+, PSRR-, common- mode input range, offset voltage, settling time etc. are required? In a phase-locked- loop, what combinations of center frequency, capture range, hold range, jitter immunity, no signal frequency drift, output phase angle, etc. will suffice? Do next year's improvements target lower power, noise, matching accuracy, higher speed or something else?

2.2 Where are the experts?

While digital design is as exciting and challenging, and even more economically rewarding, it differs from analog in a very fundamental way: it is not conceptually taxing. Digital systems may be mathematically sophisticated, but given a system design, the logic synthesis and layout is not mathematically challenging. We teach digital theory in its entirety to college freshmen. Digital's complexity and challenge is more akin to that of a large and involved cost accounting system than it is to analog design.

This essential difference in the nature of the problem has a very natural and interesting consequence: the digital world is readily comprehended by computer scientists and programmers who lack explicit training in electronic theory and its prerequisite mathematics. Thus those who best know how to create computerized automation can (and do) address themselves to the digital problem.

Analog automation on the other hand, demands a marriage of a circuit design expert and a design automation (computer) expert. As a species, analog designers still think of themselves, perhaps correctly, as artists, and are wary of the inevitable degradation and inelegance of an automated version of their craft. And as artists, they tend to enjoy the challenge of specific design situations, and the creation of entirely new architectures more than the broad, accountant-like thought process that must reduce a range of previously invented possibilities to fixed design procedures. The result of this incompatibility is that few really excellent analog designers, who are scarce breed to begin with, have found their way into the design automation field.

2.3 An expert is not enough

Even given the mathematical prowess of expert designers, the fact remains that many analog design problems are too involved to be reduced by manual methods to tractable solutions. Traditionally much of the designers' skill has been the paring of the problem to a manageable essence. And despite his best efforts, a large dose of trial and error remains. Can we automate trial and error? The answer is certainly "yes", but only at the price of enormous computing power. Computerized search algorithms rarely have the same degree of intelligence guiding the sequence of trials, and must make a far greater number of poor choices before arriving at a suitable solution.

To summarize then, pivotal barriers to progress in analog design automation include the lack a suitable methodology, and the lack of experts willing to take on the task of automation. Given capable people and plausible methodologies, a further problem remains: the algorithm maker needs tools to help him do previously unmanageably complex mathematics. Finally, even given that tool, the task demands access to fabulously large computing power. The good news: all of these barriers are beginning to crumble.

3 A Light at the End of the Tunnel

While arrays and standard cells have proven ineffectual in the analog domain, a slight variation on the concept, standard generators, holds excellent promise. This is not a novel concept: it has been applied to the physical design of standard cell libraries for 6-7 years. In this discussion, I expand the usual definition of a generator to include the design process as well as the layout, and propose that we think of them not only as tools for library generation, but also as custom design aids.

Like standard cells, generators are usually based on fixed circuit schematics, (though algorithmic arraying can also be included under this same label). But unlike standard cells, the component sizes are not predetermined and the layout is not fixed. Generators can have either of two distinct functions: component size determination based on performance specifications, and physical layout. By introducing the flexibility of variable device sizes, a vastly broader range of specifications can be addressed. The concept is hierarchically extensible: macro generators can call lower level generators to create their subcomponents.

Aiding the design automator's modeling efforts, a viable first generation of symbolic mathematics software has appeared in the last several years. These are as yet immature and require a good deal of training to use. And quite properly, they have not attempted to supplant the need for mathematical understanding by the user. But even in this infant stage they offer significant benefit in addressing very complex math problems. One aspect in which they prove particularly useful is in keeping track of the signs and coefficients of problems with many variables. For example, the algebraic (not numeric) solution of a ten by ten determinant would be a year's work by manual methods. These new tools compute the twenty-something page result in an hour on a personal computer.

Lastly, it should be apparent to the most skeptical observer that the age of boundless computing power is upon us. Five years ago, thirty engineers time-shared a 5 MIPs, 4

Meg RAM minicomputer. Today each has a 10 MIPs, 16 Meg machine sitting on his desk for his personal use. Within 5 years it will be a 100 MIP, 64 Meg superworkstation. In a single decade we will have transitioned from a world where CPU time was a dominant resource limitation to a time where it is a non-issue. Automated trial and error will be both wonderfully rapid and virtually free.

Exploiting this new computing muscle, a viable first generation of analog synthesis tools has begun to emerge. Carnegie-Mellon University researchers have created an op amp synthesizer capable of creating a very broad range of high performance custom amplifiers [2]. It hierarchically builds on generator submodules as small as matched transistor pairs and complementary drivers. Keying on various specification criteria, the system makes repeated educated guesses in determining both the device configuration and the device sizes. Based more, but not entirely on fixed amplifier schematics, CSEM (Switzerland) has created a tool that algorithmically sizes op amps, comparators, and even a few larger analog blocks such as sigma-delta converters [3]. It follows the design with a high quality automatic layout that is sensitive to analog design issues, and is interactively changeable by the user. Adding switched capacitor filter synthesis and layout to the CSEM tool, Silicon Compiler Systems is introducing the first fully featured commercial analog synthesis tools. The overall system behaves very similarly to a switched capacitor synthesis system previously reported by Asahi Kasei Microsystems (Japan) [4].

4 An Example: Asahi Kasei Microsystems' SCF Design System

Harnessing the power of recent workstations to implement the algorithmic guesswork of a non-linear programming numerical optimizer, and drawing on the modeling potential of symbolic mathematics tools, the Asahi Kasei system well illustrates the current state-of-the-art in analog design automation. The system integrates three new design modules into the design environment: a filter synthesizer (SCULPTOR), an op amp synthesizer (OPTIMIST), and a switched capacitor circuit layout synthesizer (SCARLET). A fourth required capability, op amp layout generation, has been implemented using a cell layout generation system similar to commercially available generator tools.

4.1 Automated Filter Design: SCULPTOR

The filter synthesizer is employed for both gain and delay designs. Filter order and coefficients may be determined either by classical approximations or by the numerical optimizer. This later choice allows optimization of particular specs. For example, the user may choose to minimize Q, and hence noise and sensitivity. Delay equalization filters are also designed using the numerical optimizer. The optimizer is particularly valuable in the design of non-standard filter functions. In telecommunication applications, gain equalization filters that compensate for frequency dependent line attenuation are often required. Since no formal mathematical solutions to these functions exist, designing them manually is a long and

tedious trial-and-error process. SCULPTOR's optimizer created a filter with the transfer characteristic of Figure 1 in five minutes on a SUN4-260.

SCULPTOR's analysis includes all key non-ideal effects: capacitor mismatch, both amp and switch noise and amplifier finite gain. Filters are implemented as composites of single stage, biquad, interpolator and cosine filter sections. Programmable gain functions can also be automatically included in the design, thanks to SCULPTOR's embedded mixed-mode switched capacitor / logic simulator. Filter sections may be analyzed separately or as cascaded composites. SCULPTOR outputs a captured schematic, op amp specifications to the amplifier generator and a netlist to the filter layout synthesizer.

4.2 Automated Amplifier Design: OPTIMIST

Also based on numerical optimization methods, OPTIMIST sizes the devices of amplifiers and switches to meet a specification received from SCULPTOR. Min/max limits on gain, bandwidth, noise, PSRR, etc. are inputs; device sizes and actual performance to spec are outputs. Using analytic models based on SPICE-like IV equations, and by including high order poles and zeros in the analysis, the result matches full conventional simulation very closely - within a fraction of a decibel for gain functions, and within 1 degree for phase. The creation of analytic models of this complexity is an entirely impractical task by manual methods. For OPTIMIST, the modeler employs a symbolic math modeling tool to create the transfer functions. Input is a set of node equations. Output is the transfer function polynomial coefficient expressions. Figure 2 summarizes the nature of OPTIMIST's modeling structure.

Each of OPTIMIST's design generators has a corresponding cell layout generator. The design and layout generators are correlated to have matching diffusion areas, etc., thus assuring correct modeling of parasitics.

While it has proven very helpful for amplifier design, and greatly accelerated a user defined trial-and-error process, OPTIMIST is a prime example of the need for still more computation speed in workstations. A full optimization search in OPTIMIST can take up to 30 minutes on a SUN4-260. This is acceptable in some instances, but detracts from the interactive feel that such tools should ideally have.

4.3 Automated Switched Capacitor Circuit Layout: SCARLET

SCARLET's layout capability is not limited to SCFs: it compiles any circuit comprised of op amps, switches and capacitors. It's ability to draw circuits, properly considering noise and crosstalk, stems from an intelligent preanalysis of the netlist to be drawn. Prior to layout, SCARLET decomposes the network into clusters of elements connected to charge sensitive nodes. Having thus analyzed the circuit, SCARLET can create the physical layout with the same attention to signal crossing of critical nodes as would a human layout expert. The resultant physical design is of comparable quality and density to hand drawn filters. Figure 3 is an example of a SCARLET layout for a 6th order bandpass filter. It took 7 minutes on a VAX 8650.

5 On the Horizon

The wealth of existing analyses of amplifiers and filters as well as the pervasiveness of their use made them ideal candidates for automation in this first generation. But many other common functional blocks also hold promise for reduction to automated techniques. ADCs, DACs and PLLs all seem amenable to automation via the principles described above. The computational problem will be more severe, but well within the capabilities of the coming generation of workstations. The mathematics to model these is also more difficult, but they can be attacked by an ever more potent arsenal of analysis aids. And the proliferation of CASE tools is beginning to relieve the tedium of many programming tasks, leaving algorithm creation as the dominant task of the CAD developer. This may incline more analog artists to take up the challenge of analog CAD.

The shrew now has a tamer, and the tamer has a whip.

References

[1] In a panel discussion about analog design methodologies at the 1989 IEEE CICC, representatives of three leading ASIC vendors, Sierra Semiconductor, IMP and AMI all admitted that while they have created analog cell libraries and actively market them, in actual practice, the cells are rarely reused without modification to fit each new application's requirements.

[2] R. Harjani et. al, "A Prototype Framework for Knowledge-Based Analog Circuit Synthesis", IEEE DAC, pp. 42-49, 1987

[3] M. Degrauwe et al, "IDAC: An Interactive Design Tool for Analog CMOS Circuits", IEEE JSSC, vol SC-22, no. 6, Dec. 1987.

[4] A. Barlow et al, "An Integrated Switched Capacitor Filter Design System", IEEE CICC, 4.5.1, 1989.

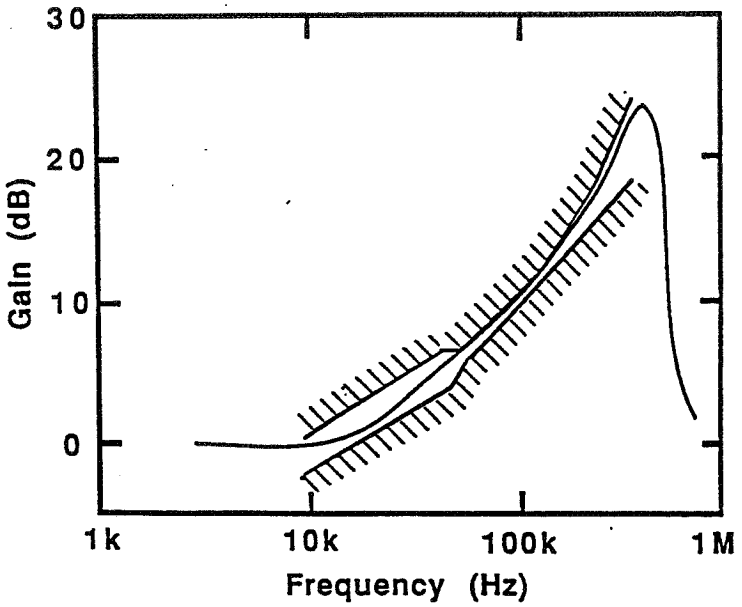


Figure 1: Gain Equalization

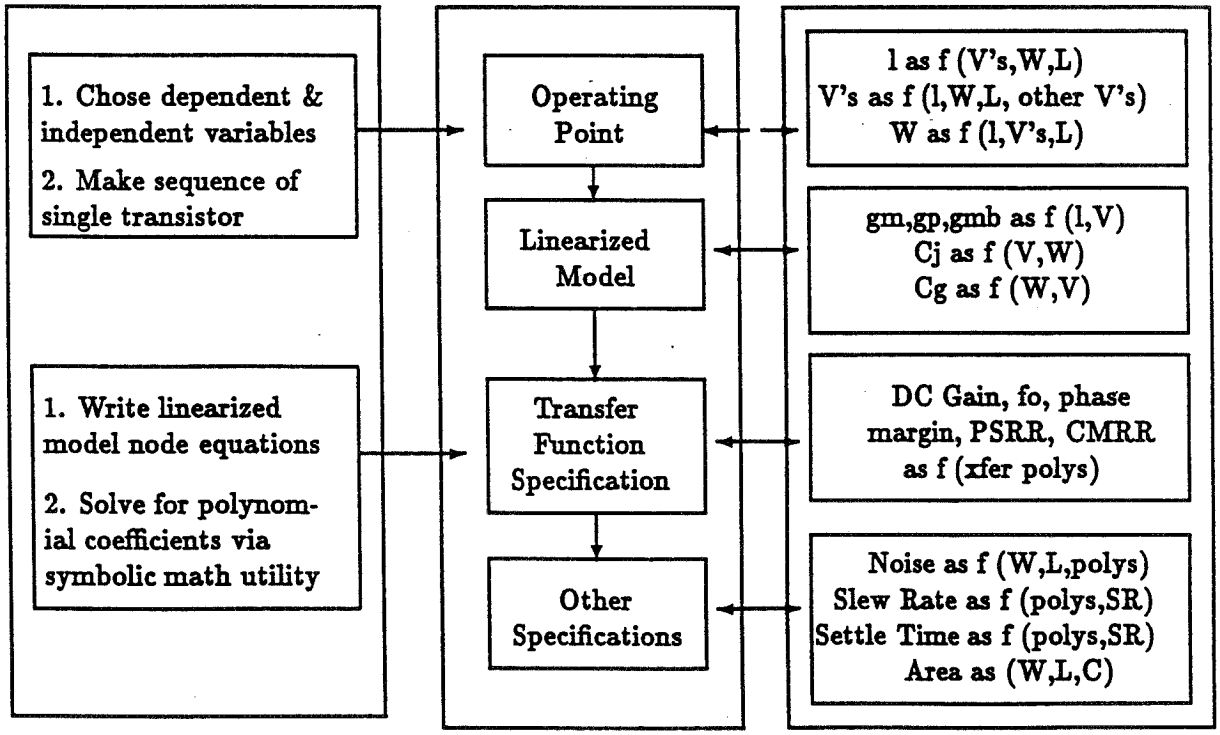


Figure 2: OPTIMIST model structure

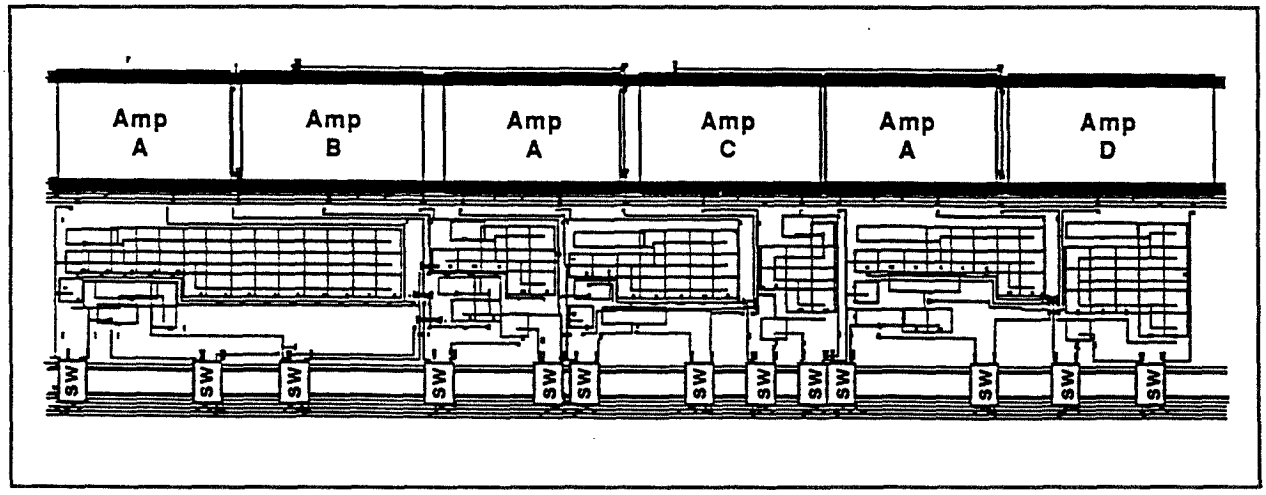


Figure 3: Generated layout for a 6th order SCF. Opamps and switches are represented by shaded bounding boxes

Next Generation VLSI Tools

J. Gibson Hewlett Packard Company
Disk Mechanism Division
Boise, Idaho

Abstract - **This paper focuses on what features would be useful in VLSI Computer Aided Design Tools and Systems to be used in the next five to ten years. Examples of current design tasks will be used to emphasize the areas where new or expanded VLSI CAD tools are needed.**

To provide a basis for projecting the future of VLSI tools, a brief history of the evolution of VLSI design software and hardware platforms is presented. The role of design methodology is considered, with respect to the anticipated scale of future VLSI design projects. Future requirements of design verification and manufacturing testing are projected, based on the challenge of surviving in a competitive market.

Examples of VLSI design and related issues are centered primarily on cell library based structured custom design. Structured custom design implies the use of a hierarchical block organization in the implementation of a complex IC. The perspectives of what capabilities are needed in future VLSI tools reflect the author's involvement on VLSI design teams developing integrated circuits for disk memory and other computer peripherals, in the last eight years.

1 Introduction.

The transition from nicely manageable, fully synchronous structured custom designs to mixed synchronous/asynchronous digital designs, coupled with analog and digital functions on the same die, will require considerable investment in tools and engineering skills. More engineers will become involved in high speed digital designs, requiring more analog circuit expertise.

The requirements of new products will determine the methodology necessary to produce cost effective and timely VLSI designs. The increasingly difficult demands on design verification will require simulation capability well beyond the limits of current tools. The manufacturability of complex integrated circuits becomes even more important as computer peripheral product volumes go from a few thousand units per month to tens of thousands or hundreds of thousands of units per month.

The capability of tools available to VLSI designers is increasing rapidly, but the management of large complex projects still requires considerable investment. Marketplace pressures are requiring shorter IC development times, with the need for perfect first pass parts growing dramatically. Cost issues are pushing chip architectures towards the most efficient and cost effective chip layouts, involving more custom design. Full custom design or library cell based structured custom design requires the best possible tool environments

to support a skilled design team. Future VLSI tools will need to effectively address the needs of full custom IC designers, to help provide a competitive edge in the marketplace.

The following discussion highlights VLSI CAD tool issues in the design capture, verification and testing processes of integrated circuit development, as encountered in several product development cycles at Hewlett Packard's Disk Mechanism Division. The requirements for next generation VLSI tools will be projected, at least from one organization's point of view.

2 History.

It is difficult to imagine a more dynamic area of technological development than computer aided design, with VLSI design being at the forefront of the CAD evolutionary process. Every VLSI chip project since the early 1980's has been accompanied by a new generation of computers, graphics tools and peripherals, usually a new operating system, and new VLSI software tools that often required a change in the design methodology used by the IC development teams. The rapidly changing tools made it difficult to anticipate technical issues and almost impossible to accurately predict project completion dates. VLSI manufacturing processes and design parameters were changing at an equally fast rate. The growth rate of most electronics companies has been very high in the last ten years. Consequently, many new engineers have been introduced into this complex, dynamic environment. The only thing different between the early 1980's and the late 1980's, is that the rate of change of IC technology, software tools and systems has further quickened. This rate of technological change appears to be permanently increasing, a bit more rapidly each year. It is the product of an amazingly competitive marketplace and a very diverse range of applications of VLSI technology.

Just ten years ago, many IC's, some of significant complexity, were still composed by hand, using tape and mylar film and requiring several years to be completed. The work was exhausting, tedious and very error prone, regardless of the methodology. The first CAD systems were minicomputer or mainframe systems with a rudimentary graphics capability. A typical collection of software included not much more than a layout rule checker, an analog simulator (SPICE) and a unit delay switch type digital simulator and possibly a graphics based schematics editor. No tools existed to compare a schematic netlist verified by simulation to the netlist extracted from a layout. Huge plots of chips were generated, spread on the floor, and sometimes four or five manual checking cycles, involving different people for each cycle, were necessary to find mismatches between schematics and layouts. Manual artwork verification of digital designs continued into the early 1980's, and is still important in analog designs.

The early minicomputer and mainframe based CAD systems generally cost $250,000 or much more, permitting only large corporations to participate in VLSI design. The systems would support only a few users, and the tasks had to be kept small, since the CPU's could easily be overloaded. Disk memory storage devices were expensive and not as reliable as today's products. One CPU handled all tasks and the only way to communicate

with other remote systems was with expensive modems on costly leased phone lines. Cost limited the number of graphics terminals, as most engineers used RS-232 terminals. Early computer aided design methodologies were developed primarily by experienced designers who learned integrated circuit design in the 1970's, using calculators and second generation minicomputers with limited software. The VLSI tool limitations caused the development of a rigorous and efficient design methodology which was, and still is, the best way to achieve success.

CAD really started to be a factor when machines that could be called workstations finally began to appear on the market. These machines combined a sharply reduced cost of computing power with a mix of features that fit naturally with engineering groups designing ICs. Workstations began to provide a hardware and software system of interactive desktop machines, of affordable cost, connected to a local area network to provide transparent access to data and programs in real time. This development gradually eliminated the need to go through time-consuming departmental minicomputers or to use mainframes for batch processing. Providing each engineer with a graphics terminal quickly became affordable, and greatly improved productivity. Additional processing power could be added incrementally. Although it could be debated in some respects, probably the first 32-bit graphics workstation to be shipped was the Apollo DN 100, the first of which was shipped in 1981. This was the earliest machine on the market that had most of the features of today's workstations. Some of the earlier proprietary machines changed so much between generations, that the older hardware was obsoleted at the end of a project, which was not unusual up to about 1987. UNIX became more common, providing a multiuser, multitasking environment ideal for engineering tasks. Software from several vendors slowly became available on most workstations. Networking allowed the sharing of computer and peripheral resources, which made it possible to apply every machine on a network to a time critical set of tasks.

VLSI CAD technology is hardly ten years old, but several revolutions have already taken place. The tremendous rate of hardware development, however, seems to have outpaced the development of software tools. The productivity of one engineer has probably increased several hundred to a thousand times the rate possible just one decade earlier. There is a good chance that the rate of productivity could increase nearly as much in the next ten years. Most of the VLSI CAD tool vendors are still enhancing their first generation products or are just beginning to introduce second generation products. Even large, diversified electronics companies are recognizing that the continuing investment in VLSI CAD tools is too large to justify only in-house use.

3 Platforms.

In 1980, a minicomputer with a quarter of a megabyte of random access memory was considered a powerful machine. This larger than average minicomputer may have had 50 to 100 megabytes of disk memory. The CPU was probably a sixteen bit machine, with no cache, some DMA, and moderate I/O data transfer rates, with a generalized performance

rating of about 500,000 instructions per second. The graphics screens may have had about 535 by 390 pixels of display capability, with a slow re-windowing rate. All hardware attached to the minicomputer probably was designed for that particular machine. IC's designed for disk memory devices in the early 1980's ranged in complexity from 5000 to 35,000 FETs.

The slightly above average workstation in 1990 will probably have 16 to 32 megabytes of RAM, one or more 700 megabyte disk drives, 64 to 256 thousand bytes of cache, a 32 bit CISC CPU with 20 MIPS performance or a RISC CPU of about 35 MIPS, and 5 megabyte per second I/O data rates. Most graphics terminals will have 1280 by 1024 pixel display capability, with the top end at 2048 by 2048 pixels. The graphics display will be handled by a separate processor system. In addition, industry standard interface busses permit the attachment of peripherals and specialized processors from a number of vendors. The integrated circuits designed in the late 1980's for disk memory devices ranged from 30,000 to 70,000 FETs for most designs, with one close to 380,000 FETs. Other organizations have developed designs of about 750,000 FETs.

What might the fairly well loaded workstation look like in the year 2000? Some of the possibilities include maybe a gigabyte of RAM, 10 gigabytes of solid state disk, 100 or more gigabytes of disk memory, two or more CPU's, each providing 300 to 400 MIPS, plus several processors facilitating communications with peripherals. Fiberoptic I/O should be much less expensive than it is today and should make possible I/O rates to peripherals of 50 to 100 megabytes per second, the limit being the rate that the peripherals can accept data. Inexpensive multiscreen graphics systems that allow the viewing of artwork, schematics, and textual simulation data simultaneously might be possible. Each screen may be able to display 4096 by 4096 pixels, and almost certainly will provide integrated video capability. Some of these projections may actually be on the conservative side, given the current rate of progress in the computer industry. Disk memory peripherals may not require IC's of greater than 200,000 to 300,000 FETs, but the shorter development times possible with more powerful VLSI tools and computers will be needed. Such workstations could easily support the development of IC's of greater than 1,000,000 FETs.

The future use of networks could contribute as much to productivity as will the increasing power of workstations. Network computing will require much more sophisticated software to effectively utilize multi-CPU workstations in a LAN environment. The network will likely contain several types of specialized processors that are very effective in processing some of the VLSI design tasks, especially compute intensive tasks such as artwork design rule verification. It will be difficult for software technology to keep up with hardware technology. Software technology will probably set the pace of productivity improvements in the next ten years.

The continual improvement of workstations will force most design groups to consider new equipment every three or four years. The issues of obsolescence and return on investment will probably not diminish in the next ten years. Competitive pressures will continue to shrink product lifetimes, making it mandatory to keep design productivity as high as possible.

4 Methodology.

Many large, complex designs have been entirely synchronous, since the cost and performance has been acceptable. Synchronous designs generally involve considerably less risk than asynchronous designs and are much easier to test. The methodology of synchronous structured custom IC design involves defining a set of specifications for a cell library that will allow the product performance and cost goals to be met. Then a cell library is constructed, so that each cell can be used in a design as a building block, with known limits for it's use. System designers generally work at the cell level and above, leaving cell circuit design to one or two experts. Individual cells are designed to specification by a FET circuit expert, using an analog simulation tool such as SPICE. Within a synchronous system, the cell design is fairly straightforward, since a fixed period of time is available for budgeting delays in the implementation of a cell. Cell loading is fairly predictable, since each type of cell is used in a regular structure. The types of cells are divided into four general groups: datapath (registers and arithmetic functions); programmable logic arrays (state machines); input/output (I/O pads); and testability circuits. For synchronous design, this basic methodology is not anticipated to change very much for future designs. Improvements are anticipated in the areas of test coverage, test time and overall cost per function.

The need for lower cost and large product volumes are making higher levels of integration more attractive each year. The smaller physical size of each generation of disk drive is requiring much smaller printed circuit assemblies. Where the 20 to 50 megabyte 14 inch disk drives of 1980 contained electronics on about 400 square inches on several circuit boards, 1989's 380 to 760 megabyte 5.25 inch disk drives are limited to 44 square inches on one board. The 3.5 inch and 2.5 inch drives of the future will require about the same amount of electronic functionality on printed circuit boards of well under 10 square inches. Rather than build disk drives with essentially perfect read/write heads and media, error correcting circuits are making possible lower cost means of achieving demanding error rate limits for disk products.

Nearly all of the digital functions in disk drive electronics, other than memory devices and microprocessors, have already been integrated in gate array, standard cell and structured custom circuits. Some performance and cost advantages can be obtained by merging some of these circuits together, but many of the benefits of digital integration have already been gained. The next step in the integration of disk electronics involves either more analog integration, or the conversion of currently analog circuit techniques into an acceptable digital form. Large complex analog circuits are still difficult and time consuming projects, which are very hard to fit into the short development times permitted for new disk products. The number of expert analog IC developers is very small. It is likely that few analog IC experts are also expert analog read/write or servo control designers. Servo control has been largely moved into the digital domain, by using commercial digital signal processor circuits. DSP architectures allow the flexibility needed to optimize servo performance, but are still somewhat expensive. Analog servo control IC's are available, but are limited to certain types of mechanisms and performance ranges.

In order to reduce the number of components on disk drive circuit boards, integrated circuits involving synchronous and asynchronous timing with some analog functions on chip, will be necessary in the next five years. Some examples of these complex designs are beginning to appear, mostly from analog circuit vendors. In order to continue to be successful in the development of complex mixed-methodology custom IC's, engineers will have to identify a methodology that partitions asynchronous and analog design into manageable sections. Managing the complexity of asynchronous designs will involve testing the boundaries of asynchronous and synchronous circuits, along with understanding the limits of the internal design of the asynchronous circuits. The main issues facing the mixing of analog and digital design on the same chip involve testability of the analog functions and overall yield or cost, due to the large number of processing steps for such IC's.

The implications for designers of mixed-methodology IC's include possibly larger design teams of people with a wider range of skills. The need for digital systems designers will continue, since pure digital design will continue to have the lowest cost per function and the widest range of application. Designers will have to know both synchronous and asynchronous (or timing) simulators to adequately verify future designs. Analog designers will be needed for the interfaces to mechanical devices that can't be handled digitally. Each of the above trends will require more investment in cell libraries used for structured design.

Analog effects will become more significant as overall system clock rates continue to increase. Where the 3 micron circuits of 1980 could be designed to operate at 10 megaHertz, today's 1.0 micron circuits are being designed for 30 to 40 megaHertz operation. In 1990, most circuits could be designed in .8 micron processes that will support 60 to 70 megaHertz system clocks. The .5 micron circuits of the mid-1990's will probably support 100 megaHertz system clock speeds. In the area above 30 megaHertz, secondary analog effects are already significant in most circuits. Inductance, parasitic capacitance, signal cross-coupling and other effects can have serious influence on circuits. More complicated analytical techniques, such as transmission line analysis, and more use of analog simulators could become common. Digital simulators that are aware of secondary analog effects may be needed for the higher speed designs.

The challenge for IC design teams in the next five to ten years will be to carefully identify the kind of IC's that will be needed in a particular kind of product, and then develop a strong methodology around the chosen technologies and design tools. It appears that the analog content of future designs will increase, unless good digital algorithms can be found to replace currently analog functions. Several vendors have tools that can provide the proper design environments, but choosing a tool set is only the start of the process of building an integrated circuit design methodology.

The future development of computers and peripherals is much easier to project than the future development of software tools and environments. The effort put into software in the next ten years will become an even larger portion of VLSI design system development costs. But, just as the hardware developers have drifted toward somewhat standardized architectures and interfaces, VLSI software tool developers are defining standards for interfacing some of the tools used for VLSI design. Frameworks or platforms upon which

tools of various capabilities can be attached, with a reasonable investment, appears to be the path to the future. As long as the interfaces between the platforms and the tools are fairly simple, and the connections fairly loose, then considerable flexibility could be gained. Some past attempts at large tightly coupled VLSI design tool sets either lost all performance advantages or became too large to manage. If a design tool platform is to be effective, most of the bandwidth of the workstation involved should be handed over to the specific tool currently being used, keeping platform overhead processing low. It seems inevitable that VLSI design software tools are going to grow in size as more services are provided, and overall productivity should gain accordingly.

5 Verification.

Efficient verification of integrated circuit design is made possible by having many strengths and few weaknesses in the design methodology used to implement an IC. The random logic design of the early microprocessors have given way to the highly structured RISC processors of today. The use of pin level high speed testers has been complemented with scan path circuitry on chip. The synchronous unit delay digital simulators used for much of the 1980's are gradually being replaced with event wheel timing simulators, using rise and fall time delays, and several levels of drive strengths. Fet level only simulators are being supplemented with simulators that can mix FET, gate, and functional or behavioral models in one simulation.

How will design verification be improved in the 1990's? It is likely that one answer will involve the efficient management of even more complex designs. The structure of future integrated circuits will need to be partitioned to whatever level is needed to keep each circuit blocks small enough for an engineer to work with efficiently. Too many blocks can become a file system nightmare, and too few blocks can cause tools to be slow processing blocks that are too large. It's a matter of knowing the design, hardware platform, and software tool practical limitations. Hierarchical design is fairly well developed now, and should improve with higher speed networks and file servers.

Just as the logical structure of an IC is partitioned into hierarchical blocks, the verification effort can be similarly structured. Behavioral modeling can be used in the top down analysis to help determine how the functions of an IC are to be implemented, before all of the lower level circuit detail is invented. During the bottom-up implementation phase, FET level modeling is generally used to make sure that the performance of the lowest level blocks is exactly as desired. As the major blocks of a chip come together, the opportunity exists for using a functional representation of a block rather than the FET representation, to increase simulation speed. In order to insure that a functional block exactly matches the operation of a FET level representation, it should be easy to switch representations. If extensive testing is necessary to verify the design, more higher level functional modeling may be feasible, depending on the project schedule, the simulation execution time, and engineer workloads. Each level of logical representation requires additional verification, so much in some cases, that functional modeling may not reduce overall design verification

time. High performance workstations on a network can greatly extend the utility of a low level simulation environment, if the simulation task can be shared between machines. Given personal preference, most design groups will find different paths to efficient design verification.

One of the growing needs for verification is the modeling of functions external to an IC. For example, if an IC has a Small Computer Systems Interface (SCSI), which has several asynchronous control lines, how is verification accomplished? One of the present techniques is to write vectors that cover all of the SCSI command and data operations. Such an approach can be made to work, but is fairly inflexible, since only one sequence of events is provided. Another approach is to interface the simulator to other workstations via the network and sockets, so that vectors can be computed based on previous simulation results. The use of sockets is flexible enough, but the performance can be slow. Another option that holds promise is to write functional models within the simulation environment that represent the SCSI operations. If a flexible language is used, such as "C", then almost any operation can be synthesized. This approach appears to have promise for representing the disk drive features necessary for the complete operation of an interface, formatting, and error correcting integrated circuit.

Going one more level up from the integrated circuit verification effort, the process of developing the drive electronics microprocessor firmware has become one of the more schedule critical tasks on recent projects. Quite often, firmware development is dependent on all of the drive electronics being functional. Firmware is usually developed using the real disk electronics, including any custom IC's and a functioning disk mechanism. Building breadboard prototypes is no longer practical, as breadboards have become too complex to build quickly, and the function of breadboards rarely matched that of the IC's. But, if the firmware development team could interact with a model that exactly represents the function of a custom interface IC, after the IC functionality has been frozen, but before IC silicon is available, firmware development may be able to start several months earlier than possible now. The IC model simulation performance has to be sufficient to allow the firmware developers to make reasonable progress on a daily basis. As high a level model as possible of a chip will be needed to have any possibility of being fast enough for firmware development. High level simulations may be able to take advantage of multiprocessor workstation architectures, where independent IC functions can be implemented in different processors, to emulate the parallel processing inherent in the IC itself. Such a verification environment would mean using much the same software techniques used now, but partitioning that software between several processors. Interprocess communication would have to be very efficient. Some VLSI tool environments that allow firmware developers to access IC models are beginning to be marketed at this time. Tools that allow firmware and IC design to interact will improve the ability of product development teams to reach optimal software/hardware partitioning in system designs.

More detailed extraction of layout capacitance will be needed to use timing simulators effectively, especially as system operating speeds increase. A delay calculator that considers the fan-in and fan-out for all cells in asynchronous circuits should be based on detailed layout and interconnect capacitances, and provide data that can be easily inserted into

a simulation model of an IC. Such a tool will be needed for the accurate verification of unclocked standard cell blocks, included in a largely synchronous chip, that provide fast asynchronous control response. Pieces of such delay calculation and simulation capability exist now in some tool environments, but the process is not completely automated at this time.

One of the major problem areas of current IC design involves still requiring the engineer to invent a set of vectors that prove that a given circuit performs exactly as intended. As circuits become more and more complex, the possibility of undiscovered functional flaws increases. The need exists for tools that will help the engineer understand the bounds of a design, by asking questions that the engineer may not have considered. The most vulnerable areas of a design are on block or system boundaries, where specifications may be imprecise or incomplete. Expert systems tied to automatic vector generators that can be directed to part or all of a design may be an effective way of improving circuit quality.

Only in the last year or two has the time required to define and verify a simulation model of an integrated circuit been shorter than the time to build and verify the circuit layout. For a completely new IC design of the type used in disk drives, the definition phase is about 8 months and the implementation phase about 5 months. It seems likely that in the future, the implementation phase will become shorter, especially for largely synchronous designs, as VLSI tools improve and workstations gain in performance. Very soon, the length of the definition phase will need to be reduced. Some improvement can be made through better use of existing tools, such as functional simulators. But the opportunity for entirely new tools, using the computational power that will be available in the 1990's, that manage most of the details of a model, so that the engineer can concentrate on the function of a design, should be welcome. Considerable software tool help will be needed to maintain short schedules if large amounts of asynchronous logic or analog circuits are added to designs.

6 Testing.

From 1980 to about 1987, integrated circuit test cost was a small fraction of the total manufactured cost of most custom IC's used in disk drives. For most of the $100 NMOS circuits, a test cost of $6 to $8 was not unusual. As the total IC cost was brought down, the test cost remained somewhat constant, and now is about 30% of the price of some custom IC's. The overhead expenses associated with IC testing are increasing gradually, making it difficult to achieve further IC cost reductions. At the same time, the much larger production volumes of recent years have added more emphasis on test coverage, tending to increase tests costs further. The issues of having synchronous, asynchronous, and analog circuits on the same chip could easily increase test costs even more.

Higher production volumes have focused attention on the rate of line scrap, which involves parts that fail to work at the final stage of assembly of a printed circuit board. Where a rate of 1% to 2% was once at least tolerable, when production volumes were about 1000 units per month, goals now are set at a few hundred parts per million, about

.03%. Production volumes greater than 100,000 units per month will require line scrap rates of less than .01%. The higher production volumes make it easier to purchase faster test equipment, but much more engineering effort is still required to improve test coverage.

One high production volume part has shown that just one test methodology may not be enough to achieve 100 ppm line scrap rates. This standard cell part was designed with scan circuitry access to almost all nodes, with automatic test vector generation used to achieve greater than 99.7% node coverage. When run on production lots, the scan path vectors produced about a 5% test escape rate. In a simultaneous effort, pin level parallel testing was implemented on the same chip, with somewhat better than 70% node coverage. When run at the normal operating speed of the chip, the parallel vectors also produced a test escape rate of nearly 5%. The conclusions derived from these tests indicated that, since the scan testing was based on stuck-at faults, and was not run at the full operating speed of the chip, several high impedance shorts and some open circuits were probably missed. In the parallel testing, incomplete node coverage allowed some faults to slip through. Running both tests has, in fact, nearly reached the design goal of 100 ppm line scrap rate, with a reasonable overall test cost. The overlap of the two test methodologies appears effective in catching more of the failure modes of CMOS IC's. It also shows that going from 99.7% node coverage to 99.999% node coverage for scan testing is probably not worth the effort. Similarly, going from 70% to 90% or higher for parallel test node coverage may not reduce the test escapes by very much.

Fault simulators are very effective in reducing line scrap rates of IC's but the use of only the stuck-at models misses shorts and opens, which can be some of the more significant fabrication problems in CMOS today. Fault simulators need to be provided, for example, with pairs of signals that are close together in routing channels for a 100 microns or more, so that tests can be generated for signals that may be shorted together by metal stringers or other flaws. The problem is that signals that are in routing channels may come from distant blocks that have no relation to each other, and have lengthy initialization sequences. Such lengthy tests can raise test costs quickly. One alternative is to avoid the use of layouts that require the use of long routing channels. Another choice may be to back further away from minimum design rules in routing channels, which could easily increase the silicon cost.

Board level scan testing, based on the IEEE 1149 standard, will no doubt see considerable development in the 1990's. Board testing that involves complex custom IC's can be difficult. If a test fixture cannot report which part has failed, the printed circuit manufacturing line tends to remove the most complex IC on a board first, in the hope that a test failure can be quickly fixed. The problem is that the removed part doesn't get a second chance, and goes into the line scrap bin, accompanied with very little or no diagnostic data. The proposals for an industry standard printed circuit board boundary scan methodology could help manufacturing lines, by providing more tools to identify when a complex IC has failed. With surface mount parts preventing as many board tester contact points as were possible with through-hole parts, boundary scan may be a big help towards improving printed circuit diagnostic capability. In theory, the boundary scan protocol of a printed circuit board could permit scan access to some blocks within the complex IC's, which could further improve the location of failure mechanisms. The use of built-in self

testing and firmware that is downloaded into RAM may add more flexibility to printed circuit board testing in the future.

Another diagnostic tool that could see considerable use in the future is the phase contrast scanning electron microscope. This equipment is capable of displaying a video interpretation of the surface of a portion of a chip on one screen, and a symbolic layout of the same portion of the chip in a second screen. A virtual probe can be used on the SEM display to see the voltage at any point on the surface of a passivated or non-passivated chip. A high speed parallel tester drives the pins of the chip with a repeating pattern. Even the voltage profile along interconnect can be observed. Delay times can be measured, so that new circuits could be characterized in considerable detail. The symbolic layout is used to locate noncontact probes on specific circuit nodes on the actual chip surface. A particular net of the circuit can be selected and highlighted in the layout, and the SEM processor will find and magnify the same area on the actual chip. Defects could be found by comparing voltage levels on the test chip with reference levels. This kind of capability suggests a common data base between the test and the design environments. Integrated design, test and manufacturing support tools will be increasingly valuable in the future.

7 Summary.

The VLSI tools developed in the next decade could easily increase one engineer's productivity by two or three orders of magnitude. The measurement of that productivity increase will probably not be solely in terms of FETs per day, but more likely in the ability to set and meet schedules, create highly manufacturable designs, and to produce manufacturable parts on the first pass through the fab. The VLSI design engineer may well be part of a small team, consisting of engineers that are specialists in using one or more VLSI software tools. More analog design skills will probably be needed.

The path to the most cost effective designs will very likely continue to be full custom or structured custom design, with the use of logic synthesis tools or function generators at the block level. Mixed asynchronous, synchronous and analog circuits will appear on more IC's. Higher speed designs will certainly require more attention to be placed on secondary analog effects in digital circuits.

In the next decade, the potential exists for single chip designs to replace almost all of the electronic components of present disk memory devices, at all levels of performance. The intensely competitive disk drive market suggests that shorter development schedules are most important, with cost and manufacturability following closely. Technology choices are going to be influenced heavily by potentially very large production volumes. These issues will no doubt be true for much of the electronics industry.

N94-71077

CCSDS Reed Solomon VLSI Chip Set

K. Cameron, S. Whitaker, N. Liu, K. Liu and J. Canaris

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

Abstract – **A highly efficient error correcting code has been selected by NASA as a CCSDS standard: the 16 symbol error correcting Reed Solomon code. A VLSI implementation of this decoder is described in this paper. A total of 4 full custom VLSI chips are needed that correct data in real time at an sustained rate up to 80 Mbits/second.**

1 Introduction

A Reed Solomon (RS) code has been selected by both the European Space Agency (ESA) and NASA [1] as the outer code in a concatenated coding scheme for CCSDS space communications. This Reed Solomon code is a (n, n-32) block code of 8-bit symbols capable of correcting up to 16 symbol errors; n assumes values less than or equal to 255. When $n < 255$, a shortened code is generated which is desirable for certain applications.

Several VLSI implementations of the decoder have been presented in the literature. The first by Liu [2] required 40 VLSI chips with 100 support chips and operated at a 2.5 Mbit/second rate. Another VLSI implementation was suggested by Shao et. al. [3] with no performance data given. Both of these designs utilized systolic arrays. The design presented in this paper does not utilize systolic arrays but rather is a set of custom VLSI chips. Moreover, the architecture is invariant for any RS code defined over GF(2^8).

The VLSI architecture requires a small chip count and guarantees real time decoding for data rates up to 80 Mbits/second. The design presented in this paper achieves the data rate with 4 VLSI chips.

2 Code Specification

The RS code used can be described with the following parameters and notation:

Symbol	Definition
q	the number of bits in each symbol
$n \leq 2^{q-1}$	the number of symbols per RS codeword
t	the number of correctable symbol errors
$2t$	the number of check symbols
$k = n - 2t$	the number of information symbols
$c(x)$	the code block represented as an order $n-1$ polynomial
$m(x)$	the k information symbols represented as an order $k-1$ polynomial
$g(x)$	the order $n-k$ generator polynomial

For the code under consideration, $q = 8$ and $t = 16$.

2.1 Code Description

The RS code word is defined as:

$$c(x) = x^{2t}m(x) + m(x) \bmod g(x). \tag{1}$$

Simply stated, every valid code word is a multiple of the generator polynomial $g(x)$. In its simplest form, the generator polynomial is defined as

$$g(x) = \prod_{i=0}^{2t-1}(x - \alpha^i) = \sum_{j=0}^{2t} g_i x^i \tag{2}$$

where α is a primitive element of the field.

A more general form of the generator polynomial is defined as:

$$g(x) = \prod_{i=s+1}^{s+2t-1}(x - \beta^i) = \sum_{j=0}^{2t} g_i x^i \tag{3}$$

where s is an offset and β is a primitive field element equal to α^h. This form is the one used by NASA and ESA, where $\beta = \alpha^{11}$ and $s = 112$. Symmetrical coefficients of $g(x)$ result in an offset of 112 [1].

2.2 Decoding Algorithm

During transmission, errors can occur due to noise in the channel which is equivalent to an error polynomial being added to the code polynomial $c(x)$. Let the received polynomial be:

$$R(x) = c(x) + E(x) = R_{n-1}x^{n-1} + \cdots + R_1 x + R_0 \tag{4}$$

where E(x) is the error polynomial, $n < 255$ and each R_i is a field element. Symbols $R_i, i < 32$, are the check symbols. The first step in the decoding algorithm is to calculate the syndromes. The syndrome polynomial is defined as:

$$S(x) = R(x) \bmod g(x) \tag{5}$$

and contains the information needed to correct errors and/or detect the presence of an uncorrectable error. Each byte S_k of the syndrome polynomial is defined as:

$$S_k = \sum_{i=0}^{n-1} R_i \beta^{i(k+1)}, \tag{6}$$

where $0 \leq k \leq 2t-1$. The syndrome polynomial can be expressed as:

$$S(x) = \sum_{k=0}^{2t-1} S_k x^k. \tag{7}$$

The next step is to obtain the error location $\lambda(x)$ and error magnitude $\Omega(x)$ polynomials. These polynomials have the following relationship with the syndrome polynomial:

$$S(x)\lambda(x) = \Omega(x) \bmod x^{2t} \tag{8}$$

The error location and error magnitude polynomials can be obtained by using Euclid's greatest common divisor algorithm [4], which is a recursive operation. The algorithm is described later.

Once the two polynomials are known, the location and magnitude of a given error is found as follows:

Let β^i be a zero of $\lambda(x)$ (i.e. $\lambda(\beta^i) = 0$), then the error magnitude at location $n-i-1$ is:

$$\frac{\Omega(\beta^i)}{\lambda'(\beta^i)} \beta^{112i} \tag{9}$$

where $\lambda'(x)$ is the first derivative of $\lambda(x)$ with respect to x.

For more details and examples, the reader is referred to Clark and Cain [4].

2.3 Mathematical Considerations

Each of the 255 8 bit symbols of the code polynomial are members of the finite Galois Field GF(2^8). A Galois Field can be defined by an irreducible polynomial $p(x)$ [4]. For the field under consideration, $p(x) = x^8 + x^7 + x^2 + x + 1$. Addition of field elements is accomplished by bit-wise modulo 2 addition (exclusive-or).

Multiplication of field elements is a bit more complicated. If each of the two field elements is represented as a polynomial of order 7, then the product is accomplished by multiplying each of the polynomials modulo p(x). The result is an order 7 polynomial, which represents a field element.

Multiplication by a constant is a special case which is used frequently in the implementation of the encoder/decoder. Multiplication by a constant is a unary operator that operates on a polynomial representation of the field element. The operator can be represented by an 8 by 8 matrix that maps the polynomial onto its final representation [5,6].

Moreover, it is possible to allow the code to be described in a dual basis [7]. A dual basis is actually just another representation of the original field. If v is a q bit symbol in

the original representation of the field, it can be represented by the vector v' in the dual basis. The relationship between v and v' is

$$v' = Tv \text{ and } v = T^{-1}v'$$

where T is a linear operator in the field. Any operator L in the original representation of the field can be used in the normal representation by transforming it as follows:

$$L_{dual} = TL_{original}T^{-1}. \tag{10}$$

A single chip implementation of the encoder that produces RS block codes in the dual basis has been implemented [5]. The decoder described here operates in either the dual basis or regular representation.

3 Architecture

The architecture and cell design are crucial factors in efficient use of silicon area. Cell interconnect is the most important issue in efficient chip design. Interconnect can consume major portions of a chip and greatly limit the amount of circuitry that can be placed on a chip. The objective in the design here was to minimize the amount of cell interconnect.

One of the major problems to overcome in using a Reed Solomon code is the large number of operations that must be executed to perform error correction. The operations that must be performed for each message are:

Syndrome	evaluation of 32 equations of order 254
Euclid	recursive evaluation between polynomials of degree 32 and 31
Polynomial	256 evaluations of polynomial of degree 16
	256 evaluations of polynomial of degree 15
	256 evaluations of polynomial of degree 15
Correct	256 divisions and additions of field elements

The number of operations for each of the above modules is:

Module	*Number of Operations*
Syndrome	16,320
Euclid	693
Polynomial	20,736
Correct	1023
Total	38,772

Table 1: Number of Operations per Module

The number of calculations per message in the CCSDS code is 38,772. Operating at 80 Mbits/second, the number of operations per second is 1.5 billion. Clearly, this operation rate cannot be realized with a stored program computer.

Module	*Gate Equivalents*	*Transistors*
Syndrome	35,860	143,442
Euclid	29,400	117,600
Polynomial	61,040	244,160
Correction	5,236	20,944

Table 2: Standard Cell gate Equivalents

Module	*Transistors*
Syndrome	26,100
Euclid	61,900
Polynomial	27,600
Correct	23,200

Table 3: Custom Transistor Count

VLSI is one approach to implementing high performance Reed Solomon decoders. There are three technologies for realizing VLSI: Gate arrays, standard cells and full custom. The first two approaches are relatively easy to implement but are limited in both performance and complexity. The CCSDS decoder would require approximately 131,000 gate equivalents, not counting necessary ROM and RAM. Clearly, it is impossible to realize the entire decoder on a single chip using standard cells or gate arrays. The next step would be to try to partition the decoder into separate modules. Shown below are the gate equivalents and number of transistors using standard cell logic needed to realize the CCSDS decoder:

Full custom VLSI yields higher performance and greater density. Shown below are the number of transistors required to realized the CCSDS decoder with full custom VLSI.

Notice that the full custom approach requires only 138,800 transistors compared to 526,146 using standard cells.

The above system resides on 4 full custom CMOS VLSI circuits. The critical element in achieving high level integration is to implement a custom architecture that produces highly dense circuits. Approaches that are effective using discrete MSI or SSI logic do not result in similar saving in full custom VLSI. One such example involves selecting the generator polynomial. In discrete logic, selecting a symmetric generator polynomial results in major savings [7]. However, in VLSI, this savings does not materialize [5,6,8]. Reducing interconnect is a major concern and therefore it is often more efficient to replicate a functional unit like a multiplier than it is to attempt to share it. Sharing a multiplier will greatly increase interconnect which consumes more area and also increases the capacitance values thereby reducing speed.

The VLSI cells used throughout the decoder consists of the following Galois Field

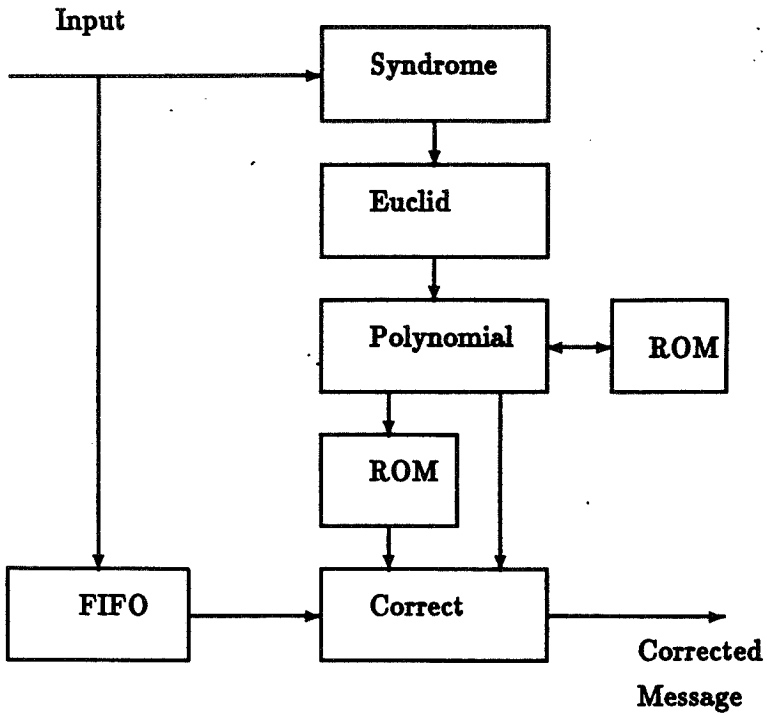


Figure 1: Reed Solomon VLSI System

processing elements: adder, constant multiplier, general multipler, and field inverse. The constant multipler performs the operation $c\ x$, where c is a constant and x is a variable; the general multiplier performs the operation $x_1 x_2$ on variables x_1 and x_2.

4 System Architecture

The decoder consists of 4 VLSI chips as depicted in Figure 1. The system is configured to perform in a pipelined manner where several messages are being processed simultaneously as depicted next:

Message i:	Syndrome Generator
Message i-1:	Euclid Multiply and Euclid Divide
Message i-2:	Polynomial Solver and Correction
Message i-3:	Data Output

Therefore, the latency of this system is 4.

The general operation can described as follows: A serial data stream is input into the serial-to-parallel converter from which the received message polynomial $R(x)$ is generated. $R(x)$ is stored in a buffer RAM for temporary storage. The syndrome generator produces the 32 symbol syndrome polynomial that is received by Euclid. The Euclid chip perform the division and multiply portions of Euclid's algorithm. A ROM is attached to Euclid

to calculate the inverse of a given field element. The Euclid produces the error location polynomial and the error magnitude polynomial. Polynomial Solver receives these polynomials from Euclid and performs the following simultaneous operations. The error location polynomial is evaluated for each element in the field generated by the primitive element β. If β^i is a root of $\lambda(x)$, then a signal Zero-Found is passed to the Error Correction Module. Both $\lambda'(x)$ and $\Omega(x)$ are evaluated for $x = \beta^i$ and these results are also presented to Error Correction. Error Correction determines the error magnitudes; if Zero Found is true for $x = \beta^i$, then the magnitude for location $n - i - 1$ is given by Equation 9; otherwise the magnitude for location $n - i$ is 0 (no error). Since the Polynomial Solver calculates both $\Omega(x)$ and $\lambda'(x)$, Error Correction only has to divide these two values. Finally, the error magnitudes are exclusive-ored with the original information.

Real time decoding is achieved. The system clock being the symbol clock is a very important feature. Therefore, this decoder can decode symbols at the same rate message symbols are presented. Decoders that cannot use the symbol clock as the system clock must utilize a more complex clock system where the decoder operates at a higher clock rate than the symbol clock. Therefore, for a given technology, this decoder can operate faster than other designs which require a system clock that operates at a higher rate than the symbol clock. Moreover, operating at the symbol clock rate reduces the amount of message buffering.

4.1 Syndrome Generator

The calculation of the syndromes is given in Equation 6. The calculation $R_i\beta^{i(k+s)}$ for syndrome byte is evaluated for all R_i and each k in {0,1,2,. . .,2t-1} and i in {0,1,2,. . .,n-1} (the number of input symbols in the message). A well known logic circuit for calculating syndrome S_j is shown in Fig. 2 [4]. The multiplier is a constant multiplier with the constant β^{1+j}. A CMOS version of this circuit is implemented here with a constant multiplier. With n input R_i symbols, a total of n clock pulses are needed to calculate a syndrome. All 32 syndromes are calculated simultaneously with 32 circuits operating in parallel.

Since one of the design constraints placed on the syndrome generator is that the system clock be equal to the symbol clock, it is necessary to calculate 32 syndromes in n clock pulses. A common means to configure 32 circuits depicted in Figure 2 is to first calculate 32 syndromes and then reconfigure the registers into a shift register and shift the syndromes out. However, this would require $n + 32$ clock pulses to calculate and shift out the syndromes, which is unacceptable.

Let the registers depicted in Figure 2 be called Syndrome registers. Let another register be defined as part of the register stack be called Shift and serve as a shift register. With the system clock being the symbol clock, if the contents of Syndrome are transferred to Shift after n clock pulses (n input symbols), the contents of Shift can be shifted out while the next set of syndromes are being calculated.

The NASA specification requires that decoder be capable of decoding dual basis RS code words. It is necessary to transform the dual basis code words into regular field code words; this is accomplished by operating on each received word by T' as defined

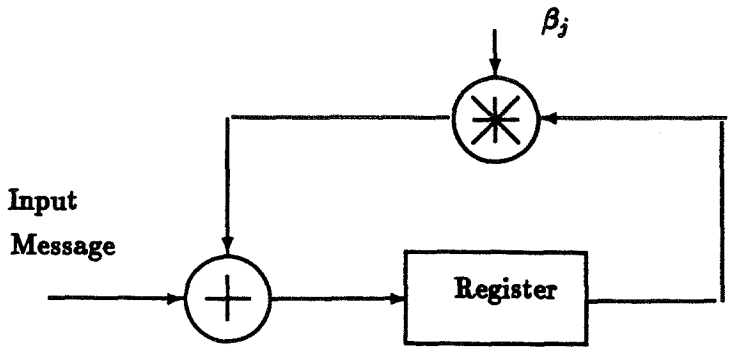


Figure 2: Syndrome generator

above. Operating on T' is equivalent to multiplying by a constant and therefore can be implemented in a similar manner as a constant multiplier. An extra feature is added to the syndrome generator to operate in either the regular field or the dual basis. An input signal DUAL is provided such that if DUAL is 1, then each input symbol is multiplied by T' (translation into regular field); if DUAL = 0, then the input symbols are not affected.

The Syndrome engine is implemented on a single, 3 micron CMOS chip 4800 x 5140 microns. There are approximately 26,000 transistors with only 5% of the area devoted to interconnect. With 32 additions and 32 multiplications occurring every 100 nano seconds, the equivalent instruction rate is 640 MOPS for a classical processor with a Galois Field ALU.

4.2 Euclid Divide and Multiply

The syndrome polynomial is shifted serially into the Euclid chip from the syndrome chip. The Euclid multiply and divide circuits recursively apply Euclid's Algorithm to find the error location and magnitude polynomials. The Euclid module uses the following algorithm to recursively obtain $\lambda(x)$ and $\Omega(x)$.

$$\Omega_i(x) = \Omega_{i-2}(x) \bmod \Omega_{i-1}(x) \tag{11}$$

$$\lambda_i(x) = -q_i(x)\lambda_{i-1}(x) + \lambda_{i-2}(x) \tag{12}$$

where $q_i(x)$ are the non-negative powers of the division of $\Omega_{i-2}(x)$ and $\Omega_{i-1}(x)$. The initial conditions are:

$$1.\Omega_{i-1}(x) = x^{2t}\ 2.\ \lambda_{-1} = 0\ 3.\ \Omega_0(x) = S(x)\ 4.\ \lambda_0(x) = 1$$

The algorithm continues until the order of $\Omega_i(x)$ is less than t.

The organization of Euclid minimizes interconnect, and when implemented with the general multiplier, can calculate the error magnitude polynomial very rapidly. The version

implemented for NASA finds the location and magnitude polynomials in less than 237 clock cycles.

The Euclid chip is implemented on a single, 3 micron CMOS chip. Even though there are approximately 61,900 transistors in a 7600 x 6800 area. The extraordinary density is achievable because : 1) The general multiplier can be drawn exceedingly dense, and 2) The given architecture is highly regular and requires virtually no interconnect. These two characteristics make it ideal for VLSI implementation.

4.3 Polynomial Solver

Polynomial Solver evaluates three polynomials simultaneously: the error location polynomial $\lambda(x)$, error magnitude polynomial $\Omega(x)$, and the first derivative of the error location polynomial $\lambda'(x)$. These polynomials are evaluated in three register stacks. One stack, the $\lambda(x)$ stack, searches for the zeros of the error location polynomial. An adjacent register stack evaluates the derivative of the error location polynomial. The $\lambda'(x)$ register stack shares the same input bus as error location, but only loads the odd coefficients of the location polynomial. The third register stack receives the error magnitude polynomial from the Euclid module. The $\Omega(x)$ register stack has a data path totally separate from the other two register stacks.

With n symbols in the received polynomial, there are n possible symbol errors. The zero's of the error location polynomial $\lambda(x)$ specify the location of the symbol errors as defined in Equation 9 and restated here: If β^i is a zero to $\lambda(x)(\lambda(\beta^i) = 0)$, then the location of the error is in location $n-i-1, i = 0, 1, \cdots, n-1$. Finding the zero's of $\lambda(x)$ involves a search of the elements in the field. If the number of zero's of $\lambda(x)$ is equal to the degree of $\lambda(x)$, then the message is said to be correctable, otherwise an uncorrectable error condition exists [4].

For a full code length where $n = 254$, all 255 field elements must be searched. If $\lambda(x)$ can be evaluated for each field element in one clock pulse, then a total of 255 clock pulses are required to search through the elements of the field. Moreover since the complete set of field elements are being examined, it does not matter which order the field elements are searched relative to execution speed.

For shortened codes and with the constraint that the system clock is equal to the symbol clock, searching through 255 elements cannot be permitted. However, for $n < 254$, the possible error message locations are $n-1, n-2, \cdots 1, 0$. To determine if an error occurred in one of the locations $n-1, n-2, \cdots 1, 0$, field elements $\beta^0, \beta^1, \cdots, \beta_{n-1}$ respectively must be evaluated in $\lambda(x)$. Any zero of $\lambda(x)$ for $x = \beta^j, j > n-1$, would correspond to an nonexistent message symbol. To evaluate $\lambda(x)$ for only n field elements and hence require only n clock pulses, it is necessary to search the field in the order defined above, which is done in the Polynomial Solver module.

Since only n clock pulses are allowed to determine the zeros of $\lambda(x)$, there must be a separate control section to input the $\lambda(x)$ coefficients from the Euclid module. As in the case of the Syndrome Generator, the Polynomial Solver has a serial shift register that accepts $\lambda(x)$ asynchronously. In this mode, it is possible to receive the coefficients of $\lambda(x)$

for one message while at the same time searching for the zero's of the previous message. When the field elements of one error location polynomial have been completely searched, the coefficients from the $\lambda(x)$ previously loaded into the shift register can begin immediate evaluation and hence completing the required search in n clock pulses.

The error magnitude and first derivative of the error location polynomials are evaluated at the same time and for the same field elements as the error location polynomial. The correction module is interested in the evaluation of $\Omega(x)$ and $\lambda'(x)$ only for those field elements where $\lambda(x) = 0$. Even though it is not necessary to evaluate $\Omega(x)$ and $\lambda'(x)$ at every field element, it does no harm to perform these calculations. However, from a speed of operation point of view, there is a great advantage in parallel evaluation of all polynomials for each field element and to calculate all polynomials in synchronism. When $\lambda(x) = 0$, the value of $\Omega(x)$ and $\lambda'(x)$ at the field element which forced $\lambda(x)$ to 0 has already been calculated. The error correction module simply divides these two values as defined in Equation 9.

Since the evaluation of $\Omega(x)$ and $\lambda'(x)$ must operate in synchronism with the evaluation of $\lambda(x)$, it is necessary to have the same shift register storage system that $\lambda(x)$ has to accept the data from the Euclid modules.

4.4 Error Correction

The inputs from Polynomial Solver are signals:

Zero-found	1 bit
Error magnitude evaluation	8 bits
Derivative of error location	8 bits

The essential calculation of this module is given in Equation 9. Since all the data values to make this calculation are input from the polynomial solver, determining the error magnitude is straight forward. A ROM is inserted in the data path between the chips to provide the inverse of the derivative of the error location polynomial. The division specified by Equation 9 becomes a multiplication and a general multiplier can by utilized to determine the error magnitude.

When Zero-found is true, the error magnitude is stored in a RAM; when Zero-found is false, a zero error magnitude is stored in the RAM. The number of errors is counted and uncorrectable error condition is noted. In outputting corrected data, the error magnitudes are fetched from the RAM and added to the input message symbols to present them to the output system.

5 Summary

A decoder has been presented that corrects up to 16 symbol errors for a Reed Solomon code at a 80 Mbit/second data rate. The output consists of the corrected information symbols and a status word. The status word, which is inserted in symbol location 31 (location of the

first check symbol), contains the number of errors found and an uncorrectable error flag. If the message is uncorrectable, the information symbols are unchanged. The equivalent instruction rate of the decoder chip set as a whole is 1.5 BOPS. This figure ignores all loading of registers, reading ROMs and writing RAMs, and interchip communication.

The size and transistor count for each chip is summarized next:

Module	Number of Transistors	Chip size
Syndrome	26,100	4800 x 5140
Euclid	61,900	7600 x 6800
Polynomial	27,600	5540 x 5750
Correct	23,200	7600 x 6800

The final chip set was fabricated at Hewlett Packard in a 3.0 micron CMOS process. The VLSI chips with support FIFO and ROM chips have been incorporated onto a single board and delivered to GSFC NASA in September 1989.

Acknowledgement The authors wish to acknowledge the efforts and support from Warner Miller and Jim Morakis of Goddard Space Flight Center in guiding this project. Jack Venbrux, T. J. Berge, Jay McDougal and Carrie Claflin, former graduate students, are recognized for their efforts in designing chips that now comprise this system. Patrick Owlsey was a member of the original team that designed this chip set; he is now with Advanced Hardware Architectures which has commercialized this chip set under the product name AHA 4600.

References

[1] H. F. Reefs and A. R. Best, "Concatenated Coding on a Spacecraft-to-ground Telemetry Channel Performance," Proc. ICC-81, 1981

[2] K. Y. Liu, "Architecture for VLSI Design of Reed-Solomon Decoders," IEEETC vol C-33, pp. 178-189, Feb 1984

[3] H. M. Shao et. al. "A VLSI Design of a Pipelined Reed-Solomon Decoder," IEEETC vol C-34, pp. 393-403, May 1985

[4] G. C. Clark and J. B. Cain *Error Correcting Coding For Digital Communications*, New York NY, Plenum Press, 1981

[5] G. Maki, P. Owsley, K. Cameron, and J. Shovic, "A VLSI Reed Solomon Encoder: An Engineering Approach," IEEE Custom Integrated Circuit Conference, pp. 177-181, May 1986

[6] G. Maki, P. Owsley, K. Cameron and J. Venbrux, "VLSI Reed Solomon Decoder Design", IEEE Military Communications Conference, pp. 46.5.1 - 46.5.6, Oct 1986

[7] M. Perlman and J. Lee, "Reed-Solomon Encoders - Conventional vs Berlekamp's Architecture," Jet Propulsion Laboratory, 82-71, Dec 1982

[8] G. Maki, and P. Owsley, "Parallel Berlekamp vs Conventional VLSI Architectures", Government Microcircuit Applications Conference, pp 5-9, November 1986

Reed Solomon Error Correction for the Space Telescope

S. Whitaker, K. Cameron, J. Canaris, P. Vincent, N. Liu and P. Owsley [1]
NASA Engineering Research Center
for VLSI System Design
University of Idaho
Moscow, Idaho 83843

Abstract - **This paper reports a single 8.2*mm* by 8.4*mm*, 200,000 transistor CMOS chip implementation of the Reed Solomon code required by the Space Telescope. The chip features a 10 MHz sustained byte rate independent of error pattern. The 1.6μm CMOS integrated circuit has complete decoder and encoder functions and uses a single data/system clock. Block lengths up to 255 bytes as well as shortened codes are supported with no external buffering. Erasure corrections as well as random error corrections are supported with programmable correction of up to 10 symbol errors. Correction time is independent of error pattern and the number of errors.**

1 Introduction

Reed Solomon (RS) codes are highly-efficient and powerful error correcting codes used by NASA for space communication. The efficiency and power has led to the selection of the (255, 239) RS code for the Space Telescope (ST). One of the major problems to overcome in using a Reed Solomon code is the large number of operations that must be executed to perform error correction. For example, the number of calculations per message in the NASA CCSDS (255,223) code is 38,772 [1,2]. Operating at 80 Mbits/second, the number of operations per second is 1.5 billion. Clearly, this operation rate cannot be realized with a stored program computer.

A VLSI RS coder chip that supports real time decoding for the ST code has the following features:

- Functions either as an encoder or decoder
- Programmed error correction capability up to 10 symbol errors
- 10 Mbytes/sec sustained data rate
- User selectable symbol clock rate, block length, number of check symbols and resulting error correction capability
- Erasure capability

[1] P. Owsley is now with Advanced Hardware Architectures.

- Shorten block length capability
- Single VLSI chip that contains the RS coder and all ROM and FIFO circuitry

2 Reed Solomon Codes

The RS code used can be described with the following parameters and notation:

Symbol	Definition
q	the number of bits in each symbol
$N \leq 2^{q-1}$	the number of symbols per RS codeword
t	the number of correctable symbol errors
$2t$	the number of check symbols
$k = N - 2t$	the number of information symbols
$c(x)$	the code block represented as an order $n-1$ polynomial
$m(x)$	the k information symbols represented as an order $k-1$ polynomial
$g(x)$	the order N-k generator polynomial

For the Space Telescope code, $q = 8$ and $t = 8$.

2.1 Code Description

The RS code word is defined as:

$$c(x) = x^{2t}m(x) + m(x) \bmod g(x). \tag{1}$$

Simply stated, every valid code word is a multiple of the generator polynomial $g(x)$. In its simplest form, the generator polynomial is defined as:

$$g(x) = \prod_{i=0}^{2t-1}(x - \alpha^i) = \sum_{j=0}^{2t} g_i x^i \tag{2}$$

where α is a primitive element of the field.

A more general form of the generator polynomial is defined as

$$g(x) = \prod_{i=s+1}^{s+2t-1}(x - \beta^i) = \sum_{j=0}^{2t} g_i x^i \tag{3}$$

where s is an offset and β is a primitive field element equal to α^h. The Space Telescope RS code specifies $\beta = \alpha$ and s = 120. The generator polynomial is

$$g(x) = \prod_{i=120}^{119+2t}(x - \alpha^i) \tag{4}$$

where $t = 8$.

2.2 Decoding Algorithm

During transmission, errors can occur due to noise in the channel which is equivalent to an error polynomial being added to the code polynomial $c(x)$. Let the received polynomial be

$$R(x) = c(x) + E(x) = R_{n-1}x^{n-1} + \cdots + R_1 x + R_0 \tag{5}$$

where $E(x)$ is the error polynomial, $N < 255$ and each R_i is a field element. Symbols $R_i, i < 2t$, are considered to be the check symbols. The first step in the decoding algorithm is to calculate the syndrome polynomial $S(x)$ which contains necessary information to correct correctable errors or detect uncorrectable errors. Each byte S_j of the syndrome polynomial is defined as:

$$S_j = \sum_{i=0}^{n-1} R_i \alpha^{i(j+1)}, \tag{6}$$

where $0 \leq j \leq 2t-1$. The syndrome polynomial can be expressed as:

$$S(x) = \sum_{j=0}^{2t-1} S_j x^j. \tag{7}$$

The next step is to obtain the error location $\lambda(x)$ and error magnitude $\Omega(x)$ polynomials. These polynomials have the following relationship with the syndrome polynomial:

$$S(x)\lambda(x) = \Omega(x) \bmod x^{2t} \tag{8}$$

The error location and error magnitude polynomials can be obtained by using Euclid's greatest common divisor algorithm [5], which is a recursive operation.

Once the two polynomials are known, the location and magnitude of a given error is found as follows: Let α^i be a zero of $\lambda(x)$ (i.e. $\lambda(\alpha^i) = 0$), then the error magnitude at location $n-i-1$ is

$$\frac{\Omega(\alpha^i)}{\lambda'(\alpha^i)}\alpha^{120} \tag{9}$$

where $\lambda'(x)$ is the first derivative of $\lambda(x)$ with respect to x. For more details and examples, the reader is referred to Clark and Cain [5].

3 Chip Overview

The circuit implements both the encoder and the decoder functions for a set of RS codes. The code is defined over the finite field GF(2^8) specified by the primitive polynomial is $p(x) = x^8 + x^7 + x^2 + x^1 + x^0$ and the generator polynomial, dependent on the variable t, is given by:

$$g(x) = \prod_{i=120}^{119+2t} (x - \alpha^i) \tag{10}$$

Number of *Actual* Errors	Correct	Remarks
$E/2+e \leq P/2$	True	Attempt Correction
$P/2 < E/2+e \leq t$	False	Attempt Correction
$t < E/2+e$	False	No Correction
$2t < P$	False	No Correction
$2t < E$	False	Ignore Erasures Attempt Correction

Table 1: Error correction parameters.

where $t \in \{1, 1.5, 2, 2.5, \ldots, 10\}$. The coder circuit has data in and out ports. Data is input at a constant rate, and output with a fixed latency. The coder operates in either an encoder or a decoder mode. All buffering is internal to the chip.

The block length of the code is variable, as large as 255 bytes and as small as $23 + 10t$ bytes. The code block consists of the message and $2t$ parity bytes, where $2t$ ranges from 2 to 20.

The correction/detection ability of the code is quite flexible, with the limits given by

$$t = E/2 + e + d/2 \tag{11}$$

where $2t$ is the number of parity symbols, E is the number of erasures, e is the number of random errors, $e + E/2$ is the correction ability of the code and d is the additional detection ability of the code. Also let P be the number of parity symbols that will be used for correction.

An erasure is any symbol that is identified to be in error prior to the actual decoding process. Any byte flagged as an erasure will count against the correcting ability of the code whether that byte is in error or not. The detection ability of the code is the ability to detect errors beyond the correction ability of the code.

The parameters P and $2t$ are fixed. The first is read during reset on the P0:4 inputs and the second is the number of parity input with the first code block. The relationship between the number of errors and erasures and the fixed parameters P and t are given in the following table.

Full correction ability of the coder is achieved when $P = 2t$. Making P smaller does not change how the data path will perform correction, but it does change how the coder reports the integrity of the output data. When $P < 2t$, three regions of error magnitude are determined. In the first, where $E/2+e \leq P/2$, the error pattern is guaranteed correctable. In the second region, where $P/2 < E/2 + e \leq 2t - P/2$, the error pattern is guaranteed detectable. The coder will make a best guess to the right code word. If $P < E/2 + e \leq t$, the coder will perform correction, but report that the block was uncorrectable. In the third, where $2t - P/2 < E/2 + e$, the error pattern is neither correctable nor detectable.

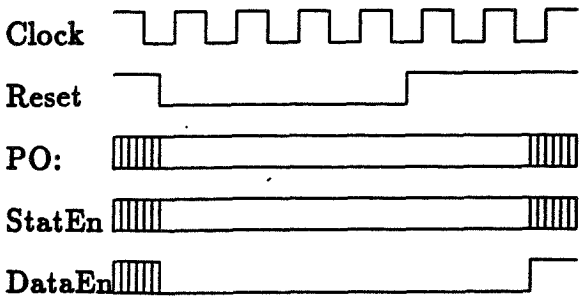


Figure 1: Reset Timing.

4 Operation

4.1 Initialization

The initialization sequence consists of the reset timing and the first code block through the coder. The reset timing is shown in Figure 1. Reset must be low for 4 clock cycles and high for 2 clock cycles. The number indicated on P0:4 sets P. The value of StatEn is latched at this time. DataEn must be low during reset to ensure that no spurious messages are processed.

If StatEn is low during reset, then the chip will output the corrected parity symbols on Dout0:7 after the message symbols of the code block. If StatEn is high, then status will be output instead of the parity. The first status byte will indicate the number of erasures and the second byte will indicate the total number of errors processed. The first status byte will include a flag that indicates whether correction was attempted. If correction is attempted then the most significant bit, Dout7, will be low, and it will be high if no correction was attempted.

The coder will not function properly if P is set to an invalid value during initialization. Invalid values are 0 and 21 through 31. If $P > 2t$, but still a valid value, then the circuit will work as if $P = 2t$.

During the first code block, the number of actual parity bytes, $2t$, will be set. In subsequent blocks, if DataEn is low for more than $2t$ clocks, then data input on Din0:7 will be passed through the decoder with no correction applied to those bytes, i.e. they will be treated as data inserted between code blocks and not as an element of a code block.

4.2 Normal Operation

After initialization, the coder receives data as indicated in Figure 2. DataEn is the timing signal. It must be high when the message symbols are input and it must be low when the parity symbols are input. The coder takes in code blocks consecutively, performs the appropriate coding operation and outputs the data with a fixed latency of $2N + 10t + 34$ clock pulses. The $2t$ parity symbols are indicated by the number of clock pulses that DataEn is low during the first block.

The coder will allow two different block lengths to be intermixed. In that case, the

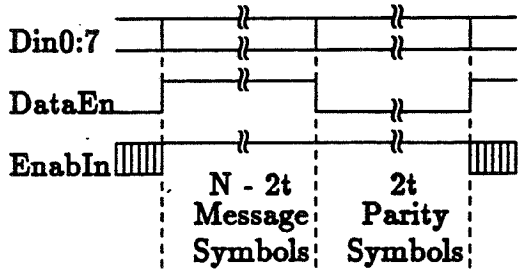


Figure 2: Decoder input timing.

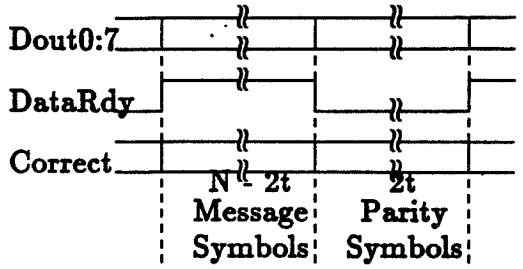


Figure 3: Decoder output timing.

latency is a function of the longest block length, N_l. If the shorter block of length N_s is processed first, the latency will be $2N_s + 10t + 34$ until the first large block is ready to be output, at which time the latency will become $2N_l + 10t + 34$.

EnabIn must be high for each block of data. If EnabIn is low for a block of data, that block will not be corrected as it passes through the decoder. However, if EnabIn is low for the entire block, the error statistics for that block will be output when the block is output. This allows a block of data to be passed through the coder without correction, but error statistics can also be presented if that option is selected.

Data can also be passed through the decoder transparently between code blocks. As indicated in the previous section, if the DataEn is low for more than $2t$ clock pulses, the data that is on the bus at that time will pass through the decoder without being operated upon. It should be noted that this is not possible during the first code block after a reset.

The coder outputs data as shown in Figure 3. The timing signal for the output is DataRdy. It is high while the message bytes are output and it is low for the parity symbols. If the Correct line is high, then the message was correctable, if it was low, then the code was determined to be uncorrectable. If StatEn is low during initialization, then Dout will have the corrected message and parity during the times shown. If StatEn is high during initialization, then Dout will have status bytes for the first two clock pulses during the time labeled parity. The status words will report the number of erasures, the total number of errors, and whether or not a correction was attempted.

4.3 Encoder Operation

The encoder function is a special case of the decoder function. A code block is input to the coder with the message to be encoded input while both Erase and DataEn are high.

Module	Gate Equivalents	Transistors
Syndrome	22,412	89,650
Euclid	18,375	73,125
Polynomial	38,150	152,600
Correction	3,272	13,090

Table 2: Standard cell hardware requirements

After the message is input, both Erase and DataEn are brought low for the $2t$ clock pulses which correspond to the parity symbols. Erase low during the parity indicates that these locations are in error. The coder will correct these locations to the proper parity. Of course, the StatEn line must be low during initialization to allow the parity to be output correctly.

5 VLSI Implementation

VLSI is one approach to implementing high performance Reed Solomon decoders. There are three VLSI technologies that could be used: Gate arrays, standard cells and full custom. The first two approaches are relatively easy to implement but are limited in both performance and density. Using standard cells, a 10 symbol error correcting decoder would require approximately 82,200 gate equivalents, not counting necessary ROM and RAM. Shown in Table 2 are the number of gate equivalents and the associated number of transistors that would be required to realize a standard cell design for each module that comprises an RS decoder, except ROM and RAM. The total number of transistors needed for a standard cell design is 328,465. The full custom chip presented here requires only 200,000 transistors which includes the RAM and the ROM.

Full custom VLSI was used to achieve both circuit density and speed. Full custom allows control on the amount of interconnect. Speed with is a function of capacitance which is a function of interconnect is an important parameter in high performance VLSI. Interconnect was minimized in this design. The VLSI architectures implemented here are similar to previous full custom designs presented in the literature [2,3,4].

The functional modules within the coder are identified by their function. The syndrome module produces the syndrome values according to Equation 6. Circuitry exists to calculate 2t syndromes in parallel; since $t_{max} = 10$, there are 20 parallel syndrome generator circuits. For the ST code, 18 syndrome values are determined in parallel. The recursive Euclid module implements Equation 8 and determines the error magnitude $\Omega(x)$ and error location $\lambda(x)$ polynomials. The Euclid module uses an internal ROM to calculate the field inverse. The Polynomial Solver module evaluates polynomials $\lambda(x), \Omega(x)$ and $\lambda'(x)$ in parallel. This evaluation identifies the location of an error and produces the values to calculate the error magnitude according to Equation 9. The Correction module performs the field division and multiplication as specified by Equation 9 to determine the error magnitude and corrects the raw data which has been stored in the FIFO.

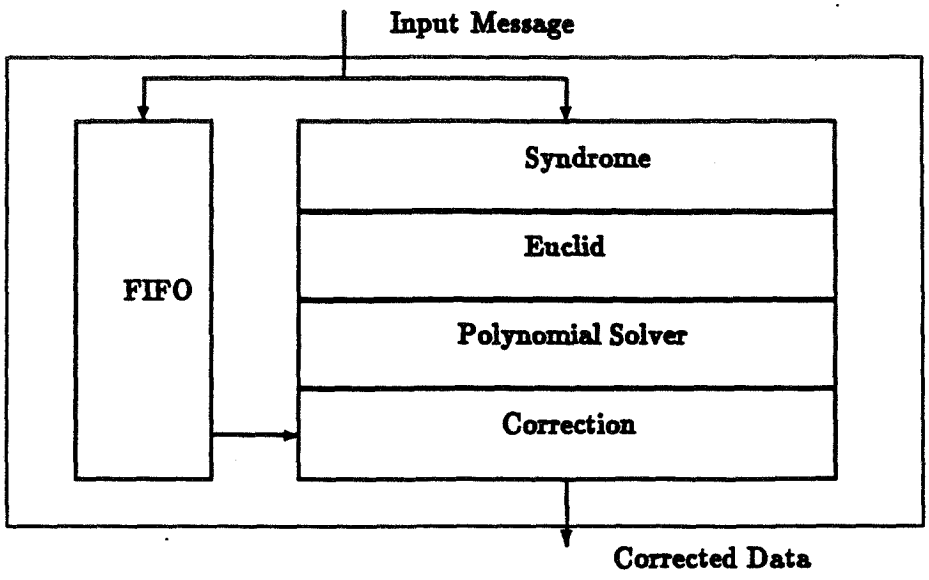


Figure 4: Chip Organization

Message data enters the chip and is stored in the FIFO and input to the Syndrome module. Messages are processed in a pipeline fashion through each of the modules. The architecture of the chip is depicted in Figure 4. Each module was configured to minimize interconnect. This was accomplished through careful data path placement such that functional modules were adjacent.

6 Summary

A VLSI coder was presented that can function either as an encoder or decoder for Reed Solomon codes. The error correction/detection capability of the coder can be programmed by the user. The maximum error correction is 10 symbol errors and the maximum data rate is 10 Mbtyes/second. The correction time is independent of the number of errors in the incoming message.

The chip was designed in a 1.6 micron CMOS process and fabricated at Hewlett Packard. This chip was delivered to Goddard Space Flight Center in April, 1989, and has been installed in the ground communication link for service in the Space Telescope

system.

Acknowledgement The authors wish to acknowledge the support from Warner Miller and Jim Morakis at Goddard Space Flight Center in guiding this project. Support is also appreciated from Dr. Paul Smith of NASA Headquarters and the NASA Space Engineering Research Center program. This chip is commercially available from Advanced Hardware Architectures.

References

[1] K. Cameron, et. al. "CCSDS Reed Solomon VLSI Chip Set," NASA Symposium for VLSI Design, January 1990

[2] G. Maki, P. Owsley, K. Cameron and J. Venbrux, "VLSI Reed Solomon Decoder Design", IEEE Military Communications Conference, pp. 46.5.1 - 46.5.6, Oct 1986

[3] G. Maki, and P. Owsley, "Parallel Berlekamp vs Conventional VLSI Architectures", Government Microcircuit Applications Conference, pp 5-9, Nov 1986

[4] G. Maki, P. Owsley, K. Cameron, and J. Shovic, "A VLSI Reed Solomon Encoder: An Engineering Approach," IEEE Custom Integrated Circuit Conference, pp. 177-181, May 1986

[5] G. C. Clark and J. B. Cain *Error Correcting Coding For Digital Communications*, New York NY, Plenum Press, 1981

VLSI Chip-set for Data Compression Using the Rice Algorithm

J. Venbrux
N. Liu

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

Abstract – **A full custom VLSI implementation of a data compression encoder and decoder which implements the lossless Rice data compression algorithm is discussed in this paper. The encoder and decoder reside on single chips. The data rates are projected to be 5 and 10 Mega-samples-per-second for the decoder and encoder respectively.**

1 Introduction

An encoder/decoder VLSI chip-set is designed for lossless image compression applicable to NASA space requirements. With the ever increasing precision of flight instruments that produce greater amounts of data comes a need for data compression. This data-compression chipset uses the Rice Algorithm [1,2], and is able to perform lossless compression of pixel data at 5 Mega-samples-per-second. The pixel quantization levels may range from 4 bits through 14 bits. Designed using full custom VLSI for a 1.6 micron CMOS technology, the chips are low power and not larger than 8mm on a side. Rather than design the chip-set for a single project or single application, flexibility was designed into the chip-set to accommodate future imaging needs.

2 Algorithm Overview

This presentation assumes that the reader is familiar with the Rice algorithm [1,2]. The Rice Algorithm codes differences between the present pixel value and a predictor value. As a default, the previous pixel is used to predict the value of the present pixel. An externally supplied predictor may also be supplied by the user to change prediction from the X direction to either Y or Z directions.

Taking the difference between two N bit pixels results in a difference that is N+1 bits in precision. A mapper adjusts this difference back to N bits without any loss in information. To simplify the VLSI implementation, the mapper function used by the encoder and decoder, is slightly different than the function specified in the literature. The mapped difference is called a "sigma" value. Because the compression method operates on differences, an original reference pixel must precede the coded data. Without the original

reference, the decoder is not able to reconstruct the original pixels from all the coded differences.

A block of sigma values are encoded by multiple parallel encoders. The winning coder is the coder that achieves the highest compression ratio. Before the compressed data is transmitted, an ID is sent that specifies the winning coder for that particular block. Compressed data follows the ID bits.

The next section describes some of the theory behind the Rice Algorithm. For a more detailed discussion of the Rice Algorithm, the reader is asked to refer to Rice's work [1,2].

3 Algorithm Theory

The Rice Algorithm uses two techniques that allow it to efficiently compress image data that varies over a wide range of entropy conditions. The first technique, that of using multiple coders, was briefly mentioned in the Algorithm Overview. The second technique, is to use small block sizes. Both are discussed in more detail in the following paragraphs.

3.1 Multiple Coders

Many compression schemes adapt to varying entropy conditions by some form of estimation. Based on past history, a certain codebook is either generated or chosen to compress the present data. The Rice Algorithm uses a brute force approach to coding by performing the equivalent of using multiple coders, each targeted for a different entropy level, and then choosing the winner. Instead of estimating the winner this implementation of the Rice Algorithm chooses the actual winner. Even if the Entropy radically changes within an image, the method proposed by Rice will track the changes and result in efficient compression.

The encoder uses 8 different coders, each targeted for a particular entropy range. The 8 coders are formed from 2 coding techniques: default coding and fundamental sequence coding. The default coding option is selected when all the other options fail. A default block of data includes a 3 bit ID followed by sigma data. Because sigma words are the same size as input pixel data, the default blocksize is equal to the number of input bits plus 3 ID bits. Instead of expanding during high entropy conditions, as happens in most compression schemes, the default condition limits expansion for any block to just a few ID bits.

The next option type is the Fundamental Sequence (FS) which is Huffman code with a few special properties. The length of every codeword is equal to the magnitude of the number to encode plus one. The unique prefix property of Huffman codes is attained in the FS by having all zeros precede a single one in a given codeword. The decoder simply counts the number of zeros until it finds a one. The decoded value (before any un-mapping) is equal to the number of zeros. If the decoded value has no zeros preceding a one, then the magnitude is equal to zero.

The remaining options are variations of the Fundamental Sequence. Before the codeword is encoded with the Fundamental Sequence, the least significant bit is removed ("pre-split") from the word to encode. Removing the least significant bit is equivalent to consid-

ering that bit as being random, and hence, not able to be compressed. The remaining bits are then coded using the FS. The stripped off least significant bits are sent to the decoder before the coded FS bits.

The encoder option set consists of the default condition, FS with no splitting, and six pre-split bit options. The number of split bits ranges from 1 to 5, and then jumps to 7. The jump was added to extend the efficient coding range up to approximately 10.5 bits while maintaining an 8 option code set. An 8 option code set only requires a 3 bit ID, larger than 8 options requires using a 4 bit ID.

The decoder is capable of decoding an encoded data stream that may have 12 different options. It will handle both the 8 option set, with a 3 bit ID, and a 12 option set with a 4 bit ID.

3.2 Small Block Size

All the encoding and decoding is done in blocks of pixel data. Small block sizes have the advantage of allowing the encoding to quickly adapt to changes within an image. Encoding small blocks of data also reduces the storage requirements within the encoder and decoder. The disadvantage of encoding a small block of pixels at a time is that there is a slight increase in overhead due to the ID bits that must precede each block.

The present encoder and decoder require that the block size be 16 pixels. If the scanline does not end on a multiple of 16, the encoder will fill in the missing data with the last valid pixel.

4 Encoder/Decoder Chip Set

4.1 System Overview

Figure 1 shows the system diagram for the encoder/decoder chipset. In addition to the encoder and decoder, additional system blocks must include: a packetizer, error correction, circuitry to unpack the data, and an input FIFO for the decoder.

The packetizer is needed to concatenate the variable length data blocks that have been encoded. In the concatenating process, the "fill bits" that were added to make each block end on a word boundary, must be stripped off. After the packet is constructed it should be protected using error correction. As with most data compression schemes, an error in the compressed data will generally result in incorrectly decoded packet data. However, the decoder is designed to not allow errors to propagate between packets.

The decoder is able to handle packets of various types and sizes. If a packet is fixed in length, the two typical scenarios for decoding are processing a truncated packet or processing a packet that has fill bits at the end. A truncated packet occurs when the expected compression ratio was not achieved and the packetizer had to truncate the encoder output data. Most compressed image data, however, will fit within the packet size and the packetizer will add fill bits to make the packet the required bit length. The decoder is

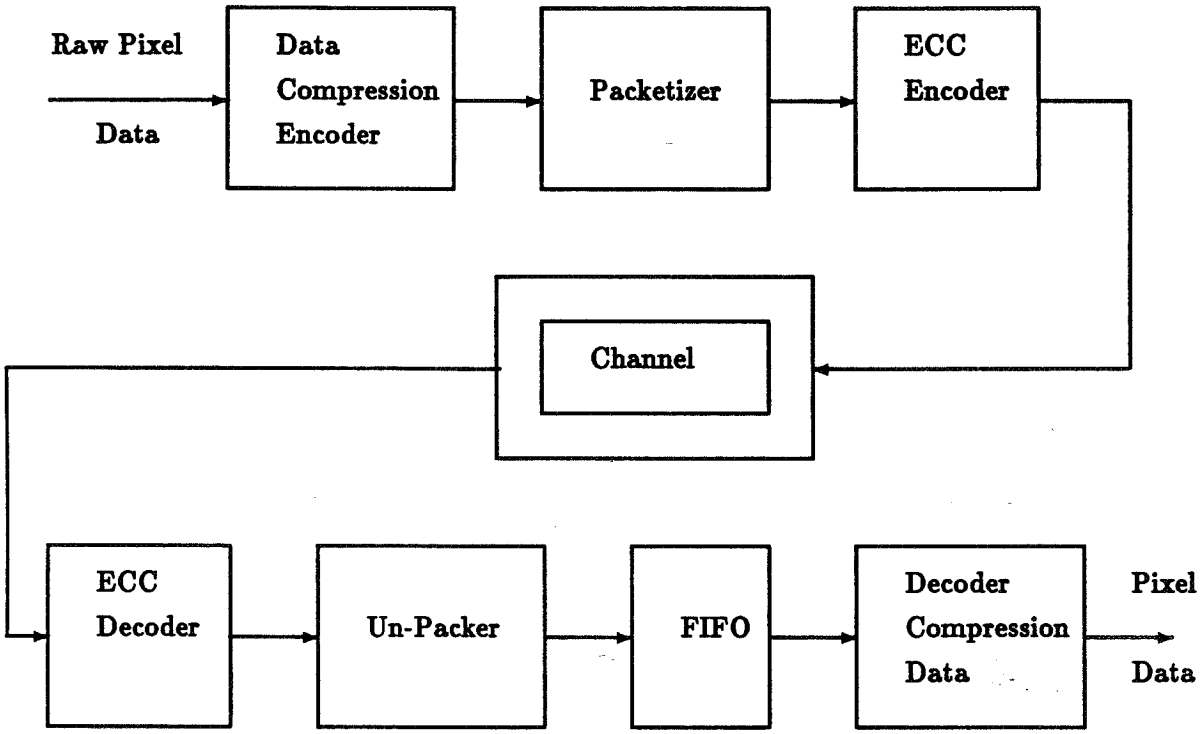


Figure 1: System Diagram

even able to handle compressed data that is not in any packet, but is a continuous stream of data.

At the receiver, error correction is followed by an operation which removes packet header bits from the packet. The decoder is expecting all packets to begin with an ID followed by a reference pixel. Packet data must be continuous with no fill bits between blocks. An external input FIFO stores data while the decoder de-compresses blocks of data into pixels.

4.2 Chip Set Features

1. Variable Quantization Levels

   - The chipset will encode and decode N bit wide pixel data with N in the range: $4 <= N <= 14$.

2. External Prediction

   - Nearest neighbor prediction is the default condition, where the previous pixel acts as the predictor for the present pixel. The chip-set supports an externally supplied predictor. An external memory chip could be used to store scanlines or frames to allow for prediction in the Y and Z dimensions.

3. Inserting References

   - By setting a few control lines that specify the number of blocks per-reference, the encoder will automatically insert a reference pixel. References may be inserted every block or as infrequent as once every 128 blocks (2K pixels). The decoder will correctly interpret reference data by setting the control lines to the same number as was used on the encoder.

4. Entropy Range

   - As discussed in the algorithm overview, the chip-set can be used to efficiently encode and decode a wide range of entropy levels. The encoder's 8 option coder set will efficiently code from 2 bits-per-pixel of entropy to approximately 10.5 bits-per-pixel of entropy. Conditions higher than 10.5 bits, will require the default condition. The decoder will decode up to 12 options and efficiently decode compressed data from 2 bits of entropy up to 14 bits-per-pixel.

     The option set used by the encoder and decoder is not designed to handle entropy conditions less than 2 bits-per-pixel very efficiently. The reason for this is that the lowest entropy option is simply a Huffman code, which requires a minimum of one bit-per-symbol even if the entropy is close to zero.

5. Performance

- The encoder will encode a maximum of 10 Mega-samples-per-second. The decoder, which requires more complex state machines, will decode at a maximum of 5 Mega-samples-per-second. By using one encoder with two decoders and some external logic, the chipset could operate at 10 Mega-samples-per-second. Using two decoders with external logic, the maximum encoding and decoding rate would be 140 Mega-bits-per-second for 14 bit pixels.

  The architectures for both the encoder and decoder involve distributed processes that operate on one block of data at a time. The processes, described in more detail in encoder and decoder sections of this paper, are pipelined.

6. Full Custom VLSI CMOS

- The full custom VLSI chipset is designed in a 1.6uM CMOS process. Full custom design, as opposed to standard cell design, allows greater flexibility in algorithm implementation, the potential of decreased die size and increased speed due to reduced interconnect. By using a custom RAM design in the encoder, the logic to generate the fundamental sequence was simplified. By designing a custom shifter for the decoder, words were able to be pulled from the bit stream in one clock pulse. Without the flexibility of custom design, complexity would have increased and performance would have been impacted.

  The encoder, with 66 pins on the IC, will have a die size of approximately 7mm on a side. The decoder, with 74 pins, is pad limited and will require a die size of approximately 8 mm on a side.

5 Encoder

The encoder is implemented on a single VLSI chip approximately 7mm on a side. It uses a single clock which may run up 10MHz. It will process up to 14 bit pixel data at a continuous rate up to 10 Mega-samples-per-second. The encoder logic assumes that all data is continuous within blocks, but the chip may be placed in a wait state when there are no blocks of data. The output is 16 bits parallel clocked on the 10MHz clock. The compressed output data must be sent to a packetizer to concatenate the variable length data blocks.

Figure 2 depicts the major blocks of the encoder. The eight major blocks include:

1. Mapper
2. Sigma FIFO
3. Count Section
4. Evaluate Section
5. FS generate
6. k generate
7. Default generate

8. Output FIFO

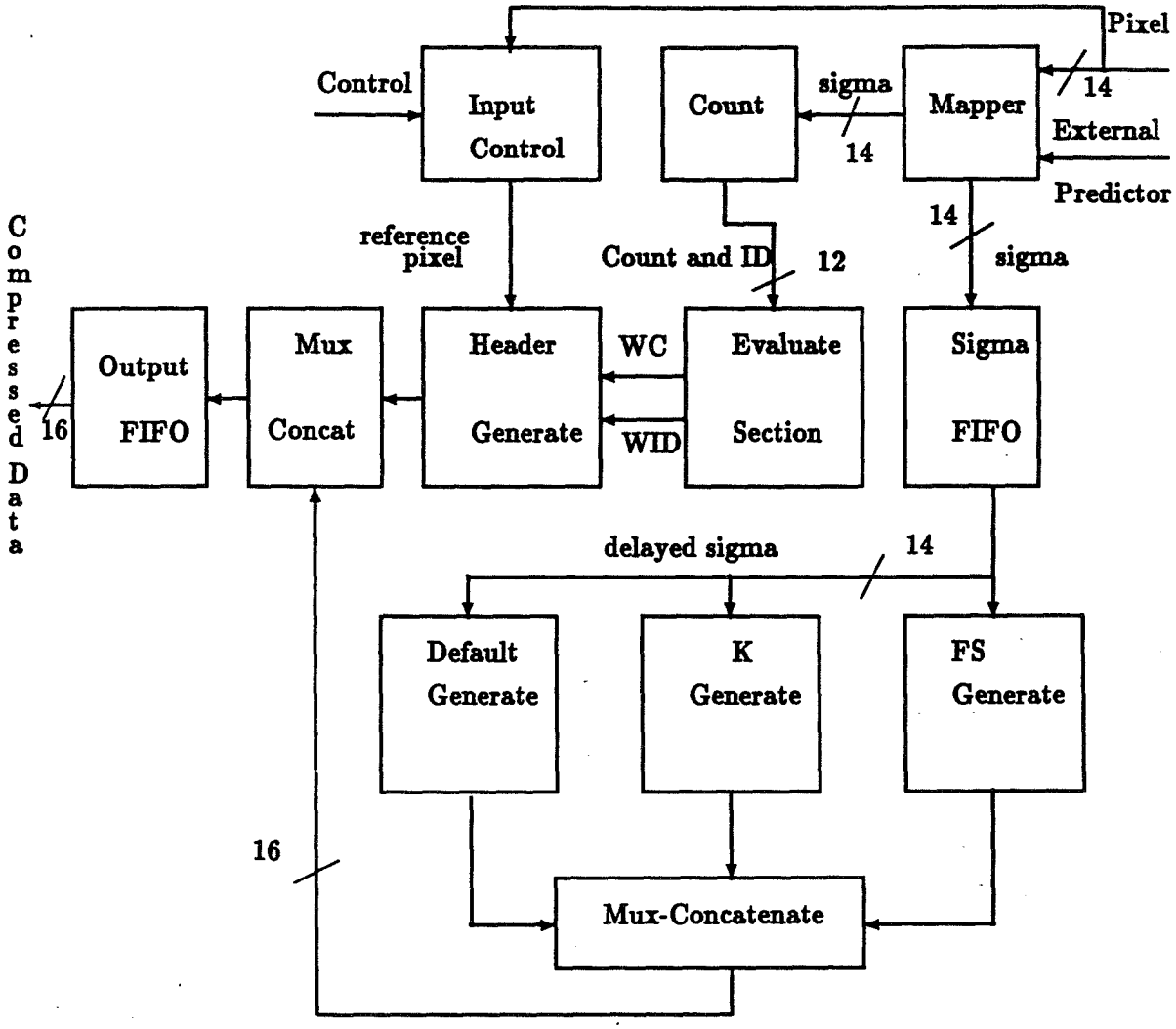


Figure 2: Block Diagram for Rice Algorithm Encoder

Control is distributed around the chip to minimize interconnect between control centers and the data paths or memories. Most of the state machines are being implemented using a modified ring counter. There are five main control sections that control the input section, the data output, and the three generate blocks.

The Mapper block takes the difference between the pixel value X, and the predictor Xp. The difference creates N+1 bits, so the mapper adjusts the value back into the range covered by the N bits, creating a "sigma" value. It is the sigma value that is encoded. The sigma value is stored in a 32 word X 14 bit FIFO to be used when the block is encoded in the generate sections.

The Count section calculates an exact count of the number of bits that will be required to encode a block for each of the eight options. The evaluate section does a compare between the eight counts and chooses the winning option. It is much more area efficient to first find the winning option and then encode the option then it is to generate all 8 options and then choose the winner. The generate sections are larger and more complicated than the count or evaluate sections.

The FS block, generates a fundamental sequence from a block of sigma values. The coded sigma values are variable in length, with the codeword length in bits equal to the magnitude of the sigma value plus one. For example, if the sigma value had a magnitude of 12, the coded word would have a length of 13 bits. The codeword has 12 zeros followed by a single one, 0000000000001. Because the FS codewords can radically vary in size, from a single bit to over two-hundred bits in length, a unique RAM was designed that allows single bit writes to the RAM with full 16 bit reads. The RAM avoided the need to generate serial FS codewords. Generating the codewords in a serial manner would require more control or memory than generating a complete word every clock pulse. Two such RAMS are ping-ponged, one is being written to while the other is being read from. Each RAM is cleared in single clock cycle just before a write operation.

The k generate data-path and control sections split off the k least-significant bits from each sigma value and pack them into words that are stored in a FIFO. While the k generate section is operating, the FS section is encoding the Fundamental Sequence on the remaining N-k bits. After the block is encoded, the k bits are read from the FIFO, followed by the FS bits that are read from one of the FS RAMS.

The default generate section packs sigma values into 16 bit words.

The output FIFO and control follows the generate sections. Sixteen bit words are read from the generate sections' FIFOs or RAMs and are stored in the Output FIFO. When an entire block is present in the output FIFO, the data words are sent from the chip, with no handshaking.

A block of output data consists of a header word that contains the number of bits in the block. With variable length data it is unlikely that the coded blocks will end on a word boundary. The packetizer can read the count contained in the header, and strip off any fill bits that trail actual data.

Although the chip could be designed to have an on board packetizer, it would add more complexity and area to the design. Since packets can vary widely in size and structure from one mission to the next, requiring the use of a separate packetizer results in a clean encoder design with maximum flexibility as to packet type.

6 Decoder

The decoder is on a single VLSI chip with approximately 14,000 gates. The chip has one clock which runs at 10MHz. The effective decoding rate is slightly greater than 5 Mega-samples-per-second.

The decoder consists of the following sections, as shown in Figure 3 :

1. Packet interface
2. Databus interface
3. Time-Frame controller
4. Moving-Window-Shifter
5. Decode Data-path
6. Decode controller
7. Unsplitter
8. Unmapper
9. Reference/prediction controller
10. Output controller

Global control is provided by the Time-Frame controller. Because the data blocks are variable in length, the control would be complex if the sequencing depended on the widely varying block size. To reduce state machine logic and complexity, each of the decoding tasks was assigned a fixed number of clock cycles, independent of block length.

Local control sections control major data paths and output logic within the chip. The control sections include state machines that were designed using a special binary tree structure (BTS) developed by Whitaker and Maki [3,4]. The BTS structure has many advantages over random logic designs. First, each bit of the state machine is identical in structure, greatly simplifying layout tasks. The individual bits of the state machine are programmed with supply connections, allowing design changes to be implemented with minimal effort. Secondly, with the identical cell structure, only one cell must be characterized to determine performance characterists. Finally, the BTS network, being a very structured form of pass logic design, can be laid out in a very compact manner with high speed performance.

The databus interface acquires 8 bits of compressed data from an external FIFO using handshaking, and concatenates two bytes to form a 16 bit word. The input section of the decoder operates on 16 bit words instead of 8 bit bytes. Even though data is input a byte at a time, the decoder must parse and decode the data stream in a serial manner. The beginning of data words may occur anywhere within the 16 bit input word. Rather than using a serial shift register with some control logic, which would slow down decoding, a special moving-window-shifter was designed. It performs a special purpose serial to parallel conversion. Given the start bit position of a new word, the next 16 bits of the word are provided within one clock cycle. The moving-window-shifter was implemented as a special full custom VLSI module.

A data word from the moving-window-shifter, may be decoded as default data, FS data, or k split bits. If the data is default data (sigma data), the data is stored in the default

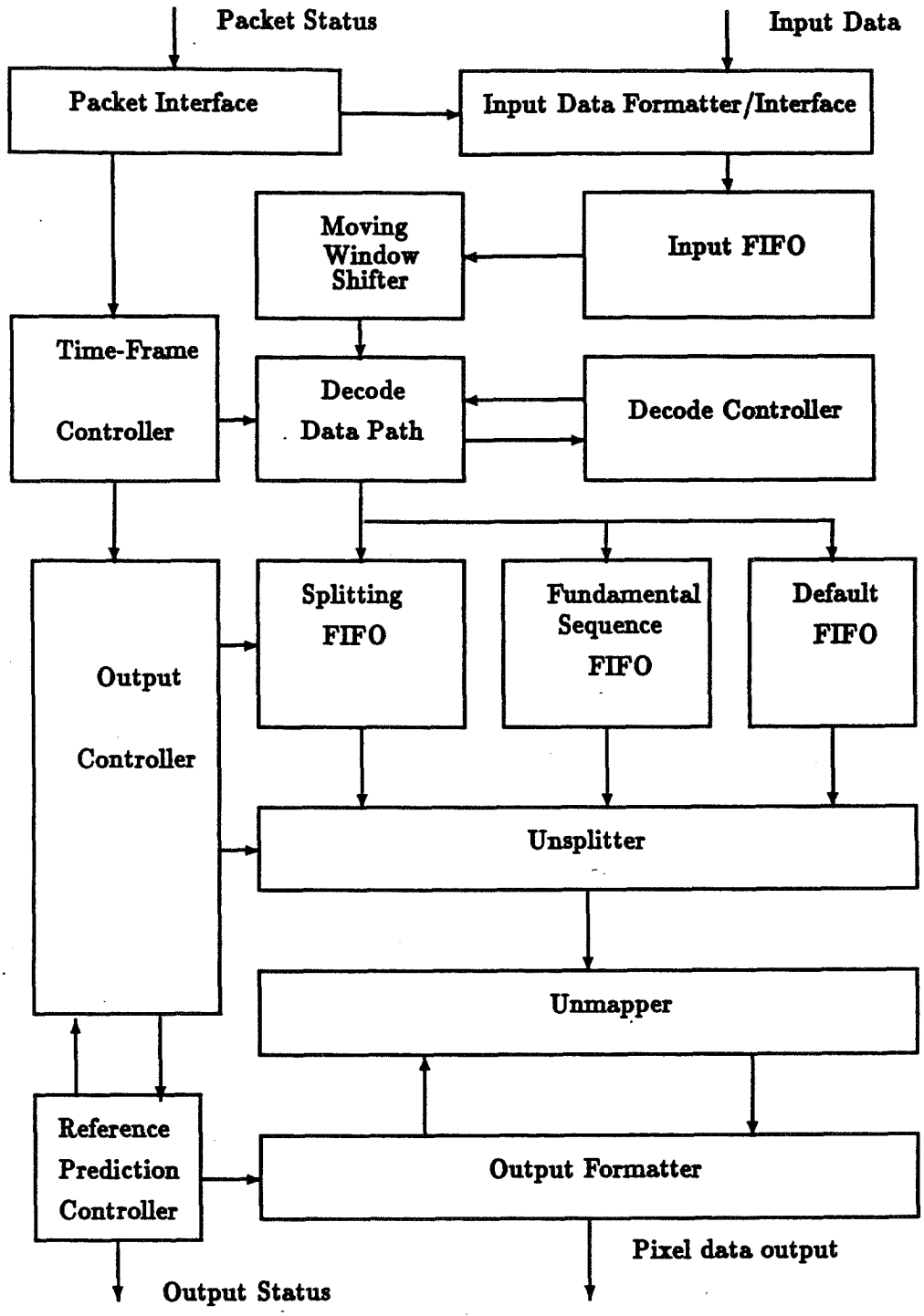


Figure 3: Data Compression Decoder Architecture

FIFO with no decoding. If the data contains split bits, the decode data-path simply packs them into the k split FIFO. If the data is an FS sequence, it is first decoded, then stored in the FS FIFO.

The unsplitter section takes data from say, the FS FIFO and concatenates the corresponding k split bits to form a sigma word. The k split bits are packed into 16 bit words with k ranging from 0 to 10 bits. To pull out the desired k split bits from a word, something very similar to the moving-window-shifter was used.

The unmapper converts sigma data into a pixel value. The unmapper performs the reverse of the encoder's mapper function. Using 2's complement arithmetic and performing some algebraic manipulations produced a form for the unmapper that simplified implementation. The output of the unmapper is pixel data. Since the algorithm is lossless, the decoded pixels are identical to the pixel values encoded at the source.

7 Summary

An data compression encoder/decoder full custom VLSI chip-set that implements the RICE algorithm has been designed. Image data is compressed at a rate of 10 Mega-samples-per-second. The decoder is designed to operate at 5 Mega-samples-per-second. By paralleling two decoders and adding extra logic, one encoder could compress data for two decoders for a maximum rate of 10 Mega-samples-per-second. Pixel size can range from 4 bits through 14 bits and external prediction is supported. Both designs employ pipelined architectures, have a single clock, and take advantage of full custom VLSI to reduce die size and maximize processing speed. The chips are in the layout phase of the design cycle, with parts expected in the Fall of 1990. Using a 1.6uM CMOS process, the die size for the encoder should be approximately 7mm on a side, and the decoder die should be approximately 8mm on a side.

References

[1] R. F. Rice, "Some Practical Universal Noiseless Coding Techniques", JPL Publication 79-22, March, 1979.

[2] R. F. Rice and Jun-Ji Lee, "Some Practical Universal Noiseless Coding Techniques, Part II", JPL Publication 83-17, March 1983.

[3] S. Whitaker and G. Maki "A Programmable Architecture for CMOS Sequential Circuits", NASA Symposium on VLSI Design, January 1990.

[4] 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, pp. 139-144, Oct. 1984.

Optimal Digital Control Of A Stirling Cycle Cooler

J. Feeley, P. Feeley and G. Langford
Department of Electrical Engineering
University of Idaho
Moscow, Idaho 88343

1 Introduction

This short paper describes work in progress on the conceptual design of a control system for a cryogenic cooler intended for use aboard spacecraft. The cooler will produce 5 watts of cooling at 65°K and will be used to support experiments associated with earth observation, atmospheric measurements, infrared, x-ray, and gamma-ray astronomy, and magnetic field characterization. The cooler has been designed and constructed for the National Aeronautics and Space Agency (NASA) Goddard Space Flight Center by Philips Laboratories and is described in detail in Reference 1. The cooler has a number of unique design features intended to enhance long life and maintenance free operation in space, including use of the high efficiency Stirling thermodynamic refrigeration cycle, linear magnetic motors, clearance-seals, and magnetic bearings. The proposed control system design is based on optimal control theory and is targeted for custom integrated circuit implementation. The resulting control system will meet mission requirements of efficiency, reliability, optimal thermodynamic, electrical, and mechanical performance, freedom from operator intervention, light weight, and small size.

2 System Description

The Philips cryogenic refrigerator consists of three sections: the expander, the compressor, and the counter-balance. The moving part in the expander section is the displacer. It is supported by magnetic bearings at each of its ends and is driven by a linear motor to produce axial rectilinear motion with little, or no, rotation. The moving part in the compressor section, the piston, is also supported by magnetic bearings at its ends, and is similarly driven by its own linear motor. The moving part in the counter balance section is the counter balance. Like the displacer and the piston it is also supported by magnetic bearings and driven by a linear motor. The axial positions of the displacer, piston, and counter balance are measured by linear variable differential transformers (LVDT's). The radial positions of each moving part are measured by optical sensors located in each magnetic bearing. In addition, the pressures in the compression volume and the buffer volume behind the piston are also measured, as is the temperature at the "cold finger" end of the expansion volume. The acceleration of the refrigerator case is also measured to aid in controlling the motion of the counter balance. Cooling is produced in the cold finger end

of the expansion section of the refrigerator by carefully controlled motion of the displacer and the piston. Motion of the counter balance is controlled to produce a force equal in magnitude and opposite in direction to the combined force produced by the motion of the displacer and the piston so that there is no net motion of the refrigerator frame.

The linear electromagnetic motors that drive the displacer, the piston, and the counter balance are similar in design. Permanent magnets are mounted on the moving parts to create a unidirectional constant amplitude magnetic field. A coil wound circumferentially on the fixed part is energized by an alternating voltage source that produces a bidirectional magnetic field of variable amplitude. The interaction of the two magnetic fields produces the force used to accelerate the moving part. The motors are carefully designed so that the force exerted on the moving part is almost entirely axial and is very nearly proportional to the current in the coil.

3 Mathematical Models

A mathematical model of the linear motor subsystems of the refrigerator is developed in this section. Interaction between the linear motors and the magnetic bearings is negligible, therefore magnetic bearing modeling and control will be considered separately later. The motor model will be used for two purposes. First, a simplified and linearized version of the model will be used in a state variable control system design procedure to design the controller. Second, the complete nonlinear model equations will be used to develop a computer simulation of the refrigerator. This computer simulation will then be used to assess the integrated performance of the refrigerator and its controller under various transient and steady-state operating conditions. The intended uses of the model help determine the level of detail it should contain.

A useful mathematical model of a linear motor can be obtained by applying Kirchoff's voltage law to a series circuit containing the controlling voltage source, the winding resistance, the winding inductance, and the motor back emf generator. The electromagnetic force exerted on the moving part is assumed proportional to the motor current. Because of the similarity of the displacer, piston, and counter balance motors the same mathematical model is used in each case. Appropriate parameter values for each motor are available from design calculations and test data.

A useful set of equations describing the motion of the displacer, piston, and counter balance may be obtained by applying Newton's second law to each moving mass. The forces acting on the displacer and piston include the applied electromagnetic forces imparted by the linear motors, the forces due to the differential pressures between the expansion, compression, and buffer volumes, drag forces exerted by gas flow, and friction forces due to displacer and piston motion. Expressions for the pressures in each volume are obtained by applying the principle of conservation of mass and the ideal gas law. Counter balance forces are similar but do not involve differential pressures or gas flow effects.

4 Control System Design Approach

A multi-input multi-output digital control system is being designed using optimal linear-quadratic-gaussian theory. The mathematical model described above consists of fourteen nonlinear differential equations and involves fourteen state variables, three control variables, and seven output variables. Two alternative control system designs are currently being considered, based on minimizing two different performance indices. The first is a tracking controller where the position error between actual and desired displacer and piston positions is minimized. The second controller will maximize overall efficiency by maximizing thermodynamic output power and minimizing control input power. A Kalman filter will also be designed to estimate unmeasured states both for use in the controller and for performance monitoring. Controller and estimator designs are being carried out using the PC-MATLAB computer aided design package.

5 Computer Simulation

The refrigerator and control system models are being simulated using the Advanced Continuous Simulation Language (ACSL). ACSL permits simulation of the complete system consisting of the discrete time digital controller interfaced with the continuous time nonlinear refrigerator model. The simulation will facilitate controller comparisons, sampling rate and word length investigations, and transient and steady state performance studies.

6 Results to Date

Efforts to date have focused on understanding refrigerator operation and developing a dynamic mathematical model appropriate for control system design. The model described above is new in that it introduces pressures as state variables through conservation of mass and ideal gas law considerations. This clarifies the relationship between the thermodynamic and dynamic aspects of refrigerator operation and should lead to an improved control system design. Efforts are underway to validate the model using test data given in Reference 1. Efforts have also been initiated on the optimal multi-variable control and estimation system design. This integrated design approach is expected to lead to a control system that is superior in performance and simpler in implementation than the control systems described in Reference 1 or a more conventional microprocessor based digital control system.

References

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

Semiautomated Switched Capacitor Filter Design System

D. Thelen
Gould Inc. Semiconductor Division
2300 Buckskin Rd., Pocatello, ID 83201

Abstract – **A software system is described which reduces the time required to design monolithic switched capacitor filters. The system combines several software tools into an integrated flow. Switched capacitor technology and alternative technologies are discussed. Design time using the software system is compared to typical design time without the system.**

1 Introduction

CMOS switched capacitor filters are a wise choice for precisely filtering electronic signals while keeping power consumption and board space to a minimum. Monolithic switched capacitor filters with custom transfer functions are especially attractive because they can be integrated with other analog and digital functions on the same chip. Switched capacitor filters have historically been full custom designs, and took a long time to design, and often did not work properly on first silicon.

A software package called Filgen has been designed, which shortens the time to market, and increases the chances of first time functionality of CMOS chips with switched capacitor filters. Filgen achieves shorter design spans, and higher first time functionality by integrating filter design software and utilizing analog circuit synthesis software. Each software tool in the filter design flow creates a file which is the input for the next tool in the flow. This approach to design automation speeds the design process for most switched capacitor filters, while still allowing each tool to be used individually for special cells which do not fit the standard filter design flow.

2 Monolithic Filter Technology

Building an electronic filter on a silicon die presents some unique challenges. First, passive RLC filters are not practical, because high quality inductors can not be built on silicon. Active RC filters can be built, but tolerances on resistors and capacitors cause time constants to vary over a five to one range when temperature and process variations are taken into account. Loop tuning schemes can be used to control the resistance or the transconductance of active devices to maintain well controlled time constants in active RC filters. Examples of tuned active RC filters are MOSFET-C [3], and transconductor-C filters[4,5].

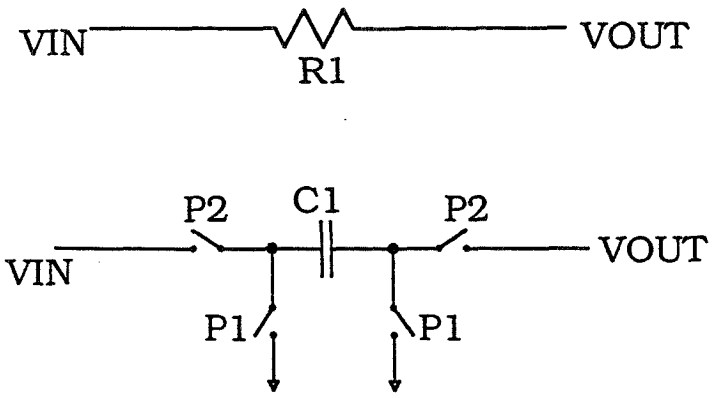


Figure 1: Switched Capacitor Equivalent Resistor

These filters hold much promise for high frequency applications, and do not require antialiasing, and smoothing filters. Digital filters [6] can also be realized on a silicon die, and have the advantage of realizing several transfer functions without changing the layout. As with any digital circuit, digital filters have excellent rejection of process, temperature, and power supply affects. Digital filters do however require A to D, and D to A converters, and analog antialiasing and smoothing filters. In most cases digital filters are larger than their analog counterparts when realized in silicon.

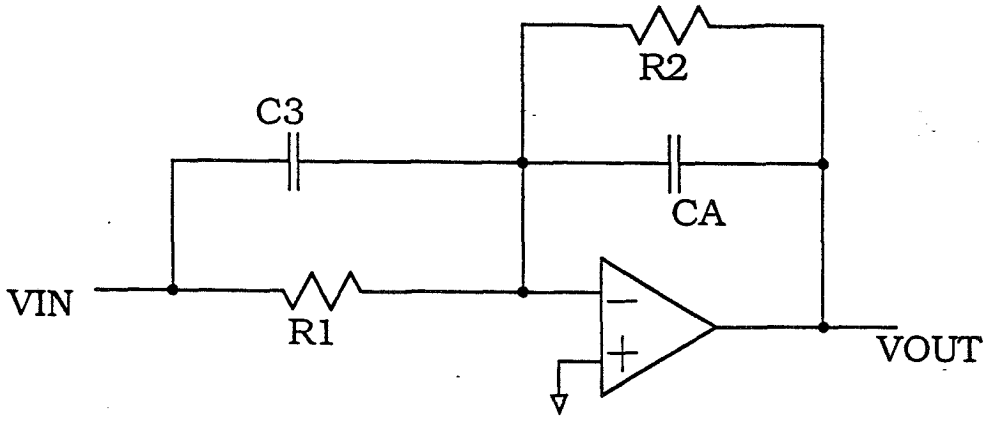


Figure 2: Single Pole Active RC Filter

Switched capacitor filters replace the resistors in an active RC filter with switches and capacitors. An approximation of a resistor can be built with four switches and a capacitor as shown in Figure 1. The switches are run from non-overlapping clocks which do not allow phase 1 switches to be closed when phase 2 switches are closed. When the clock is running, a packet of charge proportional to the voltage across the capacitor moves through

the capacitor every clock cycle. These packets of charge approximate a current through a resistor when the clock rate is constant. Such capacitors and switches can replace all the resistors in a biquad as shown in Figures 2 and 3. A closer examination of circuit

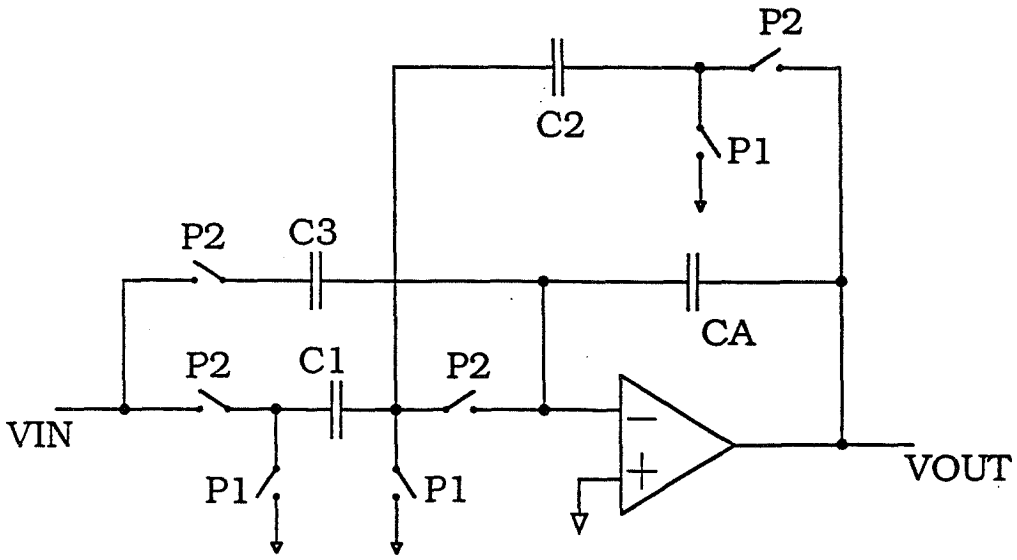


Figure 3: Single Pole Switched Capacitor Filter

operation reveals that moving charges every clock cycle is a sampled data system, which can be modeled by a z transfer function [1,6]. Carefully laid out silicon capacitors, can be matched to each other to about 0.1%. Since the equivalent resistance, is determined by capacitance, very accurate time constants are formed, from which very accurate pole and zero frequencies and Q's can be realized. Switched capacitor filters do not require tuning schemes to maintain accurate transfer functions, but do require antialiasing and smoothing filters. For a more complete tutorial on the operation of switched capacitor filters see [1]. The frequency of operation of switched capacitor filters is limited by opamp gain bandwidth product, and slew rate. Some papers [2] have reported switched capacitor filters operating well into the megahertz. The techniques used by Filgen limit the filters to signals of a few hundred kilohertz. Dynamic range in switched capacitor filters is limited by the signal swing of the opamps, opamp noise, and switch noise.

3 Filter Design System

The design flow (Figure 4) starts with Scholar which is a filter synthesis program. Scholar calculates capacitor sizes, opamp specifications, and switch sizes from filter specifications entered by the user. Scholar writes two files when all the information has been entered. The first file is a command file for a Mentor graphics workstation which draws a schematic, and

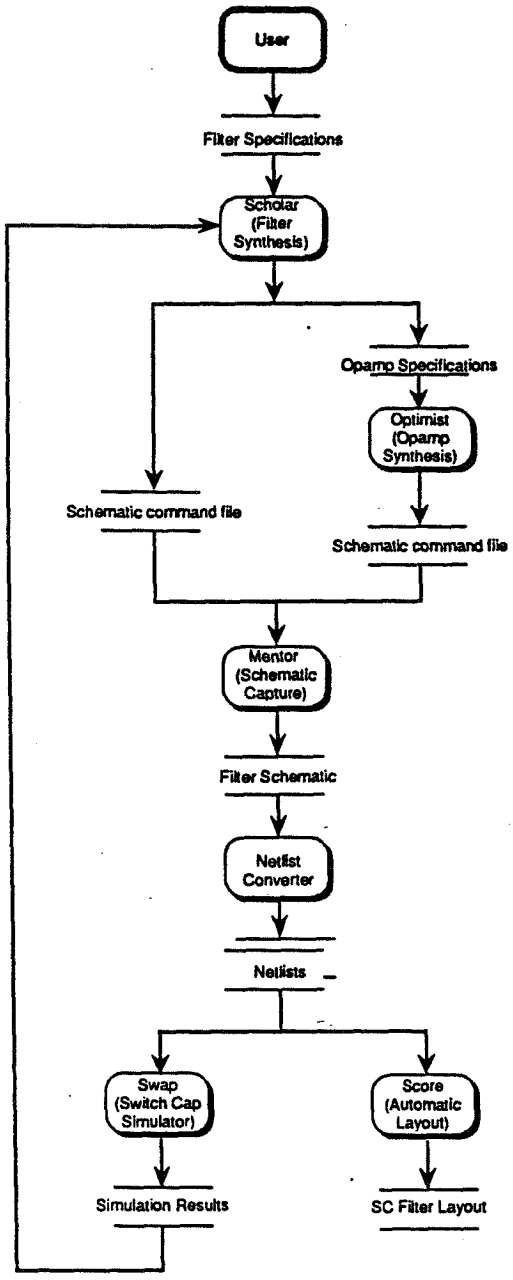


Figure 4: Filgen Design Flow

the second file defines the opamp specification for Optimist. Optimist is an opamp synthesis program which calculates transistor sizes from specifications using an optimization routine. The output of Optimist is a Mentor command file which modifies the transistor sizes on the schematic created by Scholar. Netlists can be created from the schematic to run simulations using Switcap, Swap, or Scar. A Swap noise simulation is required by Scholar to size the unit capacitor. A netlist is also created for Score, our procedural layout program. Once the layout is completed, a layout verses schematic program is run.

4 Scholar (Filter Synthesis)

Scholar first prompts the user for filter specifications such as pass band and stop band edges, pass band ripple, stop band attenuation, and clock frequency. Scholar then creates a low pass s domain transfer function using standard filter approximations (Butterworth, Bessel, Chebyshev, inverse Chebyshev, and elliptic) which meets the band edge specifications. Scholar then transforms the transfer function to a frequency scaled low pass, high pass, band pass, or band reject filter. Once the transfer function has been determined, frequencies are pre-warped to accommodate the bilinear z transform [6], and capacitor ratios are calculated for a cascaded biquad filter. The low Q poles are placed first in the cascade with the high Q's last. High Q poles are paired with the closest zeros as shown in Figure 5. These methods tend to maximize dynamic range. A more thorough approach is discussed in [7]. Scholar uses six circuit types which may be cascaded to realize any IIR transfer function. The circuits include a single pole single zero stage, a high Q biquad stage, and low Q biquad stage. Three similar stages are used to realize IIR transfer functions with zeros in the right half s plane to make all pass group delay equalizers. The topologies used are discussed in [8], and are shown to have low sensitivities. The second order biquad sections have eight capacitors, but only five coefficients in the transfer function. Two of the extra degrees of freedom are used to scale the gain to the outputs of the opamps to maximize dynamic range. The third degree of freedom is chosen to keep capacitor ratios as small as possible.

Swap is used to simulate the noise of the switched capacitor filter with a nominal unit capacitor size. Scholar assumes that folded cascode opamps will be used, and therefore most of the noise comes from the switches [9]. Scholar resizes the unit capacitor based on the simulation results. Once the unit capacitor size is determined, coupler sizes are calculated to insure adequate settling time, without excessive charge injection. Opamp gain bandwidth [10], slew rate, and capacitive load are calculated, and passed to Optimist, the opamp synthesizer.

All capacitor ratios, unit capacitor size, coupler sizes, and biquad types are written to a file which creates a schematic on a Mentor Graphics workstation. The schematic is used to create netlists for additional simulations, and is also used to create a command file for the automatic layout. The schematic can be modified if the user does not like the assumptions made by Scholar.

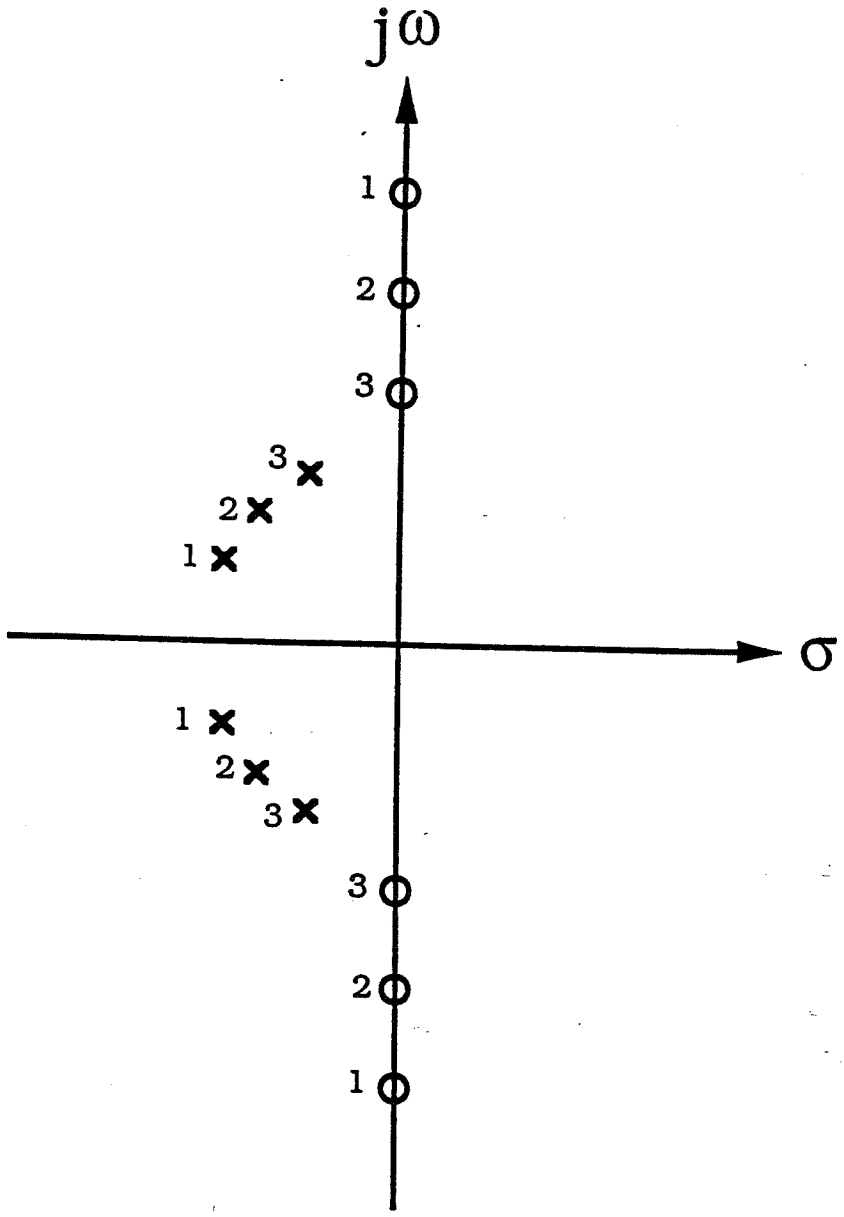


Figure 5: Pole Zero Parting, and Biquad Order

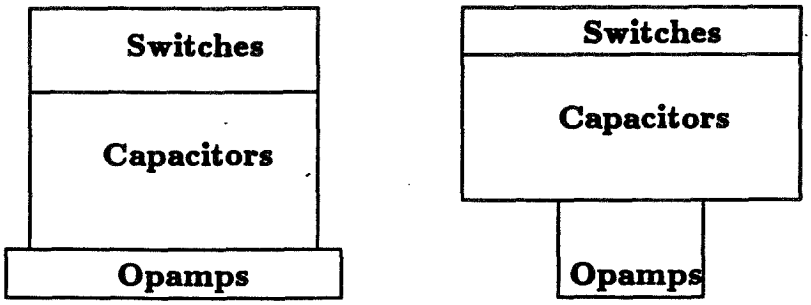


Figure 6: Biquad Layout with Different Aspec Ratios

5 Optimist (Opamp Synthesis)

Optimist is an optimization program with a special shell around it for opamp design. Optimist contains Gould AMI's proprietary Spice transistor model which it uses to find the DC operating point of the opamp. Once the DC operating point is found, a small signal model is used to calculate AC performance. Optimist can be used in several modes of operation. First, Optimist can search for a set of transistor and capacitor sizes (parameters) which meet all specifications (constraints) at one set of process temperature, load, and power supply conditions. Once device sizes are chosen, Optimist can show what the performance is at four sets of process temperature, load, and power supply conditions. When performance is not sufficient at all operating conditions, devices sizes may be changed manually, or Optimist can be asked to search for a new set of device sizes given a new set of operating conditions, and/or specifications. Finally, Optimist can choose device sizes to maximize desireable parameters such as gain bandwidth product, and common mode rejection ratio while minimizing objectionable parameters such as silicon area, and power supply current. Optimist writes a file which modifies the opamp transistor sizes in the schematic created by Scholar.

6 Score (Procedural Layout)

When the engineer is satisfied with the filter simulations, device sizes are passed from the schematic to Score, our automatic layout tool. The filter is laid out procedurally one biquad at a time to control parasitic capacitors. Each biquad is partitioned into switches, capacitors, and opamps. The three sections are assembled along a common centerline as shown in Figure 6. In some cases, the opamp will be wider than the switches, and capacitors, while in other cases the capacitors will be wider. Since the layout is automatic, the dividing line between capacitors, and opamps and the total cell height can be tweaked until sometimes the opamps are wider, and sometimes the capacitors are wider. The biquads are then placed in the order which minimizes total width by allowing cells to abut on a zig zagged line as shown in Figure 7. The interconnect maintains the correct electrical order of the biquads. Using this type of layout scheme, the capacitors are never very far

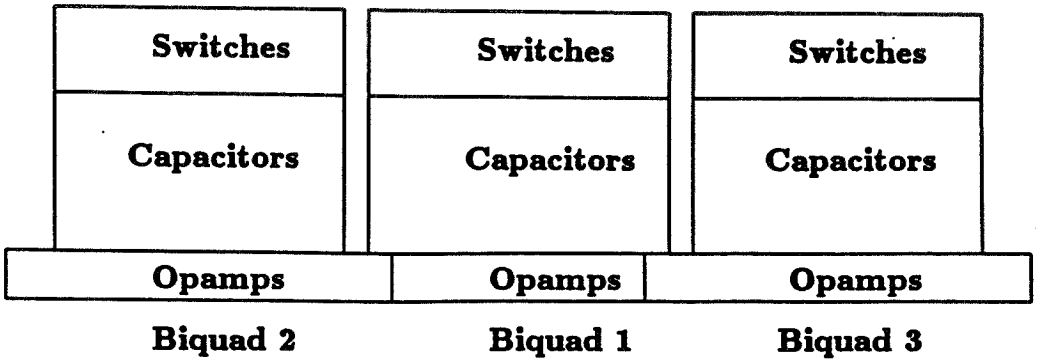


Figure 7: Filter Layout

TASK	NEW TOTAL	OLD TOTAL
Design Cap Ratios	1 day	1 day
Create Netlists	2.5 hours	1 week
Noise Analysis	12 hours	ECO
Opamp Design	8 weeks	20 weeks
Layout	2 hours	2 weeks
Layout Checking	2 days	5 days
TOTAL	9 weeks	24 weeks

Table 1: Estimated Time Savings (6 biquads)

from their switches and opamps, which keeps parasitic capacitors small.

7 Conclusion and Results

Typical runtimes for filter designs are reduced as shown in Table 1 This system is functional, and significantly reduces design time, while increasing the probability of first time success for switched capacitor filters.

References

[1] R. Gregorian, K. W. Martin, G. C. Temes, "Switched-Capacitor Circuit Design," Proc. IEEE Vol. 71, no. 8, pp. 941-964?. Aug. 1983

[2] Bang-Sup Song, "A 10.7-MHz Switched-Capacitor Bandpass Filter," Proc. CICC 1988, pp. 12.3.1-12.3.4

[3] Y. Tsividis, M. Banu, and J. Khoury, "Continuous-time MOSFET-C filters in VLSI," IEEE J. Solid-State Circuits, vol. 21, pp. 15-30, Feb. 1986.

[4] F. Krummenacher, N. Joehl, "A 4-MHz CMOS Continuous-Time Filter with On-Chip Automatic Tuning," IEEE J. Solid-State Circuits, vol. 23, no. 3, pp. 750-758, June 1988.

[5] C. S. Park, R. Schaumann, "Design of a 4-MHz Analog Integrated CMOS Transconductance C Bandpass Filter," IEEE J. Solid-State Circuits, vol. 23, no. 4, pp. 987-996, Aug. 1988.

[6] A. V. Oppenheim, and R. W. Schafer, Digital Signal Processing. Englewood Cliffs, New Jersey: Prentice-Hall, 1975.

[7] A. S. Sedra, and P. O. Brackett, Filter Theory and Design: Active and Passive. Champaign, Illinois: Matrix Publishers, 1978.

[8] K. R. Laker, A. Ganesan, and P. E. Fleischer, "Design and Implementation of Cascaded Switched-Capacitor Delay Equalizers," IEEE Trans. Circuits and Systems, vol. 32, no. 7, pp. 700-711, July 1985.

[9] R. Castello, and P. R. Grey, "Performance Limitations in Switched-Capacitor Filters," IEEE Trans. Circuits and Systems, vol. 32, no. 9, pp. 865-876, Sept. 1985.

[10] K. Martin, and A. S. Sedra, "Effects of the Op Amp Finite Gain and Bandwidth on the Performance of Switched-Capacitor Filters," IEEE Trans. Circuits and Systems, vol. 28, no. 8, pp. 822-829, Aug. 1981.

N94-71082

Integrated CMOS RF Amplifier

C. Charity
HP Disc Mechanism Division
P.O. Box 39
Boise, Idaho 83707

S. Whitaker, J. Purviance and M. Canaris
NASA Engineering Research Center
for VLSI System Design
University of Idaho
Moscow, Idaho 83843

Abstract - **This paper reports an integrated 2.0μm CMOS RF amplifier designed for amplification in the 420-450 MHz frequency band. Design techniques are shown for the test amplifier configuration. Problems of decreased amplifier bandwidth, gain element instability and low Q values for the inductors were encountered. Techniques used to overcome these problems are discussed. Layouts of the various elements are described and a summary of the simulation results are included. Test circuits have been submitted to MOSIS for fabrication.**

1 Introduction

Signals carrying information are attenuated as they propagate through communication channels. Amplification is needed to restore the attenuated signals. Radio Frequency (RF) amplifiers are usually manufactured with discrete components. Integration of the amplifier can improve manufacturability [1]. Many amplifier designs are now implemented in integrated circuits (ICs) [2,3,4].

Bipolar has been the dominant technology for integrating amplifier circuits. In the last decade, great progress has been made in MOS fabrication techniques. As a result, MOS circuits have expanded from just memory and digital logic applications to include many analog circuits. New developments in communication technology require functional blocks to consist of both analog and digital sections. Since many digital circuits are integrated in MOS, there is a strong motivation to develop analog circuits in MOS [5].

The viability of developing an integrated CMOS RF amplifier is explored with this test chip project. A single gate MOSFET RF amplifier was chosen as the test circuit [6]. Section 2 discusses and analyzes the circuit. Design techniques are shown for this particular amplifier configuration. In the process of the design, a few problems were encountered. Techniques were modified to overcome these problems. Section 3 presents the layout strategies for the components. Section 4 gives a summary of the simulation results.

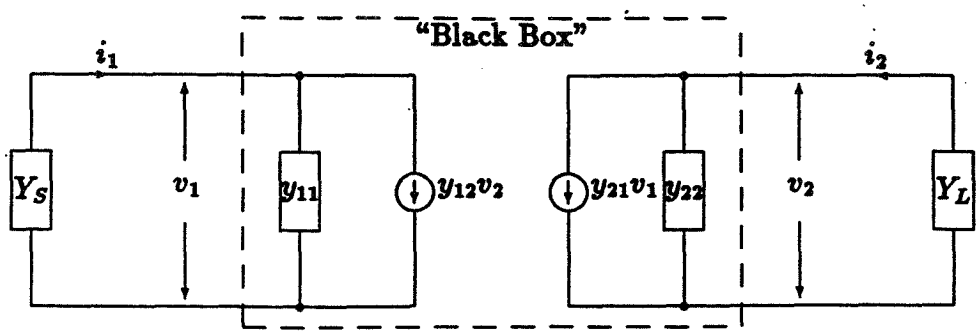


Figure 1: Y-equivalent circuit with source and load.

2 Circuit Design

CMOS RF amplifiers can be designed by using a four-terminal network model for the amplifier. Short circuit admittance parameters are used to describe the network. An admittance parameter is a complex number with the form $y = g + jb$ where g is the real (conductive) part and b is the imaginary (susceptive) part. The y equivalent circuit is shown in Fig. 1 and is described by the following equations.

$$i_1 = y_{11}v_1 + y_{12}v_2 = -v_1Y_S \tag{1}$$

$$i_2 = y_{21}v_1 + y_{22}v_2 = -v_2Y_L \tag{2}$$

The short circuit admittance parameters are y_{11} (input admittance), y_{21} (forward transfer admittance), y_{12} (reverse transfer admittance), and y_{22} (output admittance). Y_S and Y_L are the source and load admittances. From Eqs 1 and 2 with the output shorted ($v_2 = 0$), $y_{11} = i_1/v_1$ and $y_{21} = i_2/v_1$. When the input is shorted ($v_1 = 0$), $y_{12} = i_1/v_2$ and $y_{22} = i_2/v_2$.

The admittance of a circuit varies with frequency. In tuned circuits, the imaginary component disappears at the resonance frequency [7]. The width of the resonant peak where the magnitude differs from the resonant magnitude by less than 3 dB is called the bandwidth. A quantity Q (quality factor) is the ratio of the resonant frequency to the bandwidth, which gives a measure of sharpness of the resonance.

2.1 Circuit Analysis

The RF amplifier chosen for this project is shown in Fig. 2. Tuning the amplifier to the desired resonant frequency is accomplished by the input and output passive elements: C_1 through C_4, L_1, and L_2. Amplification is provided by the active device, $Q1$. Biasing of the amplifier is accomplished by a voltage divider consisting of R_1 and R_2. The two resistors are isolated from the a.c. input by L_1 and the bypass capacitor, BPC. An RF choke inductor and a bypass capacitor (1 μF) are used to properly isolate the power supplies. The a.c. equivalent input and output circuits are shown in Fig. 3.

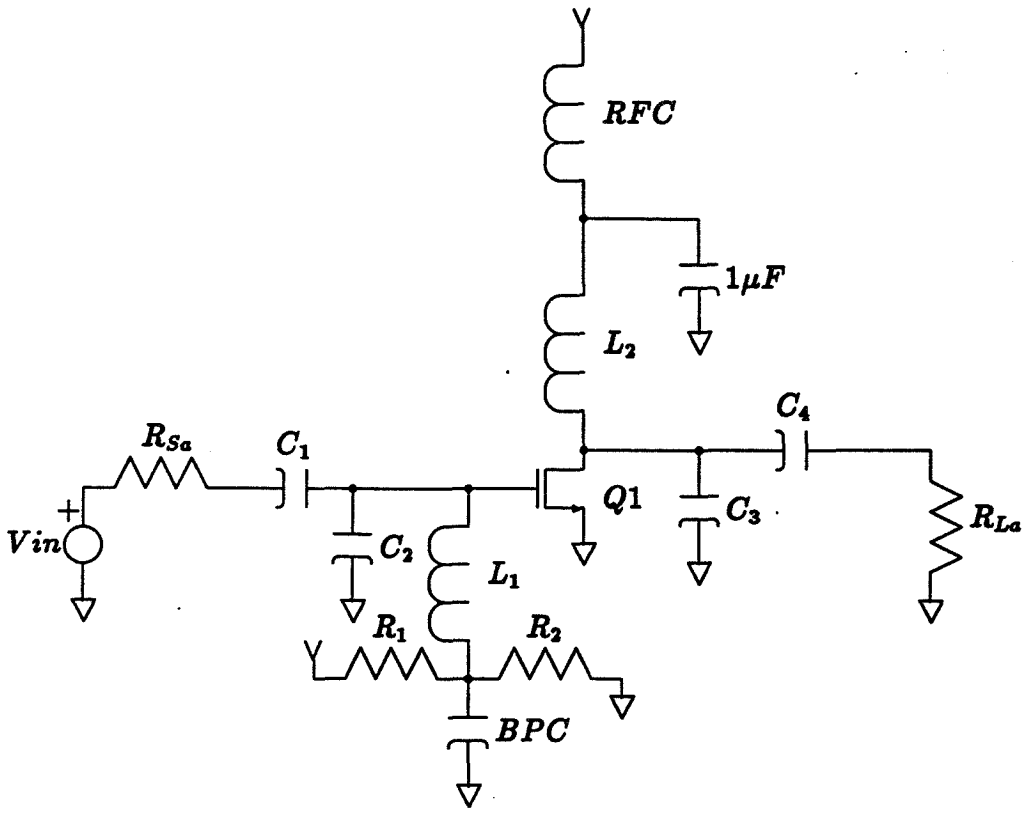
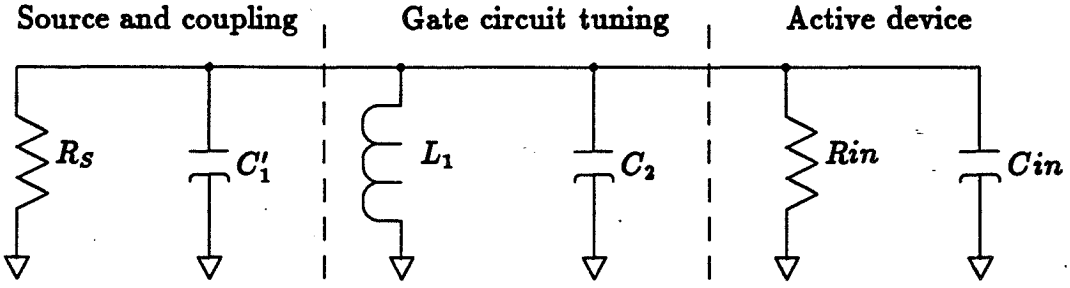
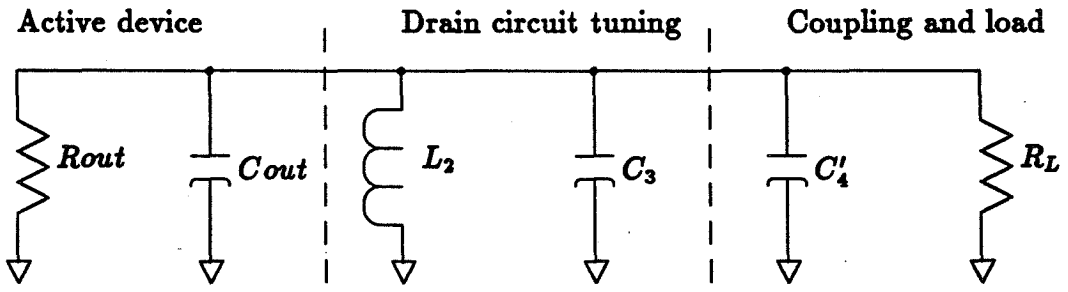


Figure 2: RF Amplifier Schematic Diagram



(a) Equivalent input circuit



(b) Equivalent output circuit

Figure 3: Equivalent input and output circuit of CMOS RF amplifier.

The inside of the "black box" in Fig. 1 shows up as R_{in}, C_{in}, R_{out}, and C_{out} in Fig. 3. The output equivalent circuit is essentially the same as the input equivalent circuit. The equations developed next will apply to both circuits.

The actual source impedance is transformed to the optimum source impedance for the MOSFET by load matching. Load matching produces the desired signal transmission without signal loss. $Z = R + jX$ where R is the resistance and X is the reactance. The optimum source resistance for the MOSFET is

$$R_S = \frac{1}{G_S} \tag{3}$$

The capacitor C_1 is in series with the actual source resistance R_{Sa}. To find the series capacitive reactance for the optimum source impedance, the series RC network must be converted to the parallel RC network. Table 3-5.1 in [7] defines the Q's for parallel and series RC networks.

$$Q = X_s/R_s \quad X_s = 1/\omega C_s$$
$$Q = R_p/X_p \quad X_p = 1/\omega C_p$$

where s designates series and p designates parallel. The Thévenin equivalent for the series RC network is

$$Z_{th} = R_s + \frac{1}{j\omega C_s} \tag{4}$$

The Thévenin equivalent for the parallel RC network is:

$$Z_{th} = \frac{R_p}{j\omega C_p R_p + 1} \tag{5}$$

By combining the definitions with Eqs. 4 and 5 and then equating the two Thévenin equivalent equations, the following formula results:

$$R_p = R_s\left(1 + \left(\frac{X_s}{R_s}\right)^2\right) \tag{6}$$

By rearranging 6, the next equation gives the series capacitive reactance for the actual series circuit.

$$X_{C1} = R_{Sa}\sqrt{\frac{R_S}{R_{Sa}} - 1} \tag{7}$$

The optimum source resistance, R_S is the parallel resistance in Eq. 6 and the actual source resistance, R_{Sa}, is the series resistance. The numerical value of C_1 can be obtained from the resonant frequency and X_{C1}. From Fig. 3a, the total resistance of the input resonance circuit is R_{in} in parallel with R_S or

$$R_T = \frac{1}{G_{in} + G_S} \tag{8}$$

As indicated above, C_1 is the passive element involved in matching the actual source to the MOSFET resistance.

To determine the bandwidth of the circuit, the Q and the resonant frequency needs to be known. Since Q is dependent on the ratio of resistance to reactance in the input circuit, the following equation results:

$$Q = \frac{R_T}{X_T} = \frac{f}{BW} \tag{9}$$

where

$$X_T = \frac{1}{\omega C_T}$$

The resistance R_T has already been determined for matching purposes. Using Eq. 9, C_T is related to the bandwidth and total resistance by

$$C_T = \frac{1}{2\pi R_T BW} \tag{10}$$

From Fig. 3a, the total capacitance in the input resonant circuit is

$$C_T = C_{in} + C_2 + C_1' \tag{11}$$

where C_1' is the parallel equivalent of C_1. Thus, the reactance of C_2 fixes the bandwidth. Finally, to make the input circuit resonant at the desired frequency, the reactance terms of the circuit need to cancel each other. The two reactance terms are ωL_1 and $1/\omega C_T$. By setting the two terms equal to each other and rearranging the equation, the value for the inductor L_1 can be found by

$$L_1 = \frac{1}{(2\pi f)^2 C_T} \tag{12}$$

One potential problem of an amplifier circuit is instability. Sustained oscillation occurs when an amplifier is unstable. An amplifier can become unstable if a feedback path occurs that adds rather than subtracts the output signal from the input signal. Coupling between the input and the output can occur through capacitance within the active device as well as through the passive circuit elements. At higher frequencies the reactance of the capacitor decreases, thus decreasing the phase margin of the system.

Two criteria are used for determining the stability of the amplifier: the Linvill C factor and the Stern K factor. The Linvill factor measures stability under worst-case conditions; when the input and the output terminals are unloaded. The following equation determines the Linvill C factor.

$$C = \frac{y_{12}y_{21}}{2g_{11}g_{22} - Re(y_{12}y_{21})} \tag{13}$$

If C is less than 1, the device is unconditionally stable. If C is greater than 1, certain combinations of load and source admittances can be found to produce oscillations.

The Stern factor includes the effect of source and load admittances. The Stern K factor is calculated from

$$K = \frac{2(g_{11} + G_S)(g_{22} + G_L)}{|y_{12}y_{21}| + Re(y_{12}y_{21})} \tag{14}$$

If K is greater than 1, the amplifier is stable. If K is less than 1, the circuit is potentially unstable. It is recommended to obtain a K value of 3 or 4 rather than 1 as a safety margin in an amplifier design [6].

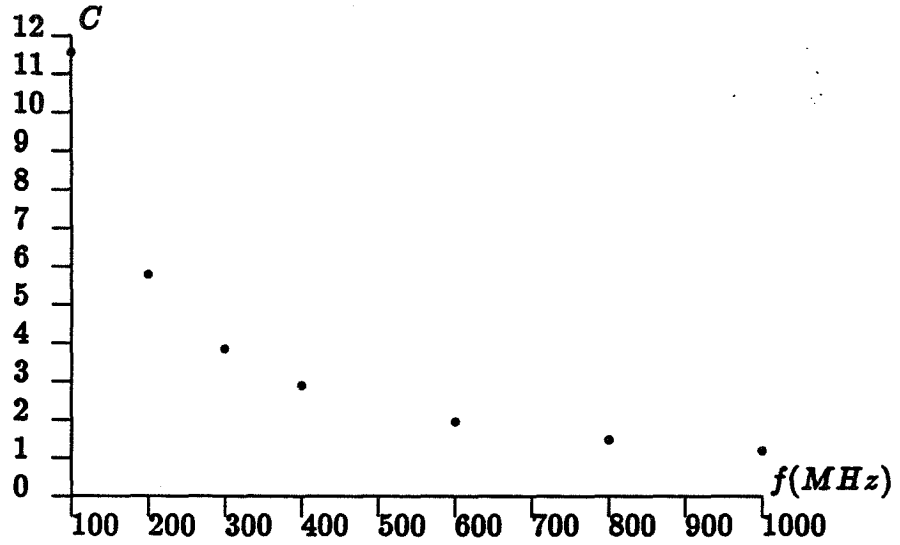


Figure 4: Linvill stability plot

The Stern solution for creating a stable amplifier, is to deliberately add some mismatch into the source or load tuning circuits. From Eq. 14, the Stern K factor can become greater than 1 (ensuring a stable circuit) by choosing G_S and G_L large enough. Since the source and load are not actually modified, but designed as if they were, a mismatch results. There is however a reduction in gain.

3 Design Procedure

The RF amplifier design is for a 420 MHz to 450 MHz operating range. Thus, the resonant frequency will be at 435 MHz with a bandwidth of 30 MHz. The first step in this RF amplifier design is to characterize the MOSFET by the y parameters. SPICE simulations were used to determine the y parameters at various frequencies. At 435 MHz, the typical y parameter values are $y_{11} = 6.6 \cdot 10^{-4} + j5.2 \cdot 10^{-3}$, $y_{21} = 8.6 \cdot 10^{-3} - j1.8 \cdot 10^{-3}$, $y_{12} = -1.8 \cdot 10^{-6} - j7.5 \cdot 10^{-5}$ and $y_{22} = 6.9 \cdot 10^{-5} + j6.6 \cdot 10^{-5}$.

The potential instability of the amplifier needs to be determined by using the Linvill C factor equation (13). The results of the Linvill calculations at different frequencies are shown in Fig. 4. As Fig. 4 shows, the device is potentially unstable up to 1 GHz.

An optimal Stern solution to stabilize the amplifier is difficult, but a good solution as defined in [6], finds the appropriate source and load admittances (Y_S and Y_L) such that

$$B_L = -b_{22} = -6.6 \cdot 10^{-5}$$

$$B_S = -b_{11} = -5.2 \cdot 10^{-3}$$

To calculate G_S and G_L, a mismatch ratio R is defined as

$$R = \frac{G_S}{g_{11}} = \frac{G_L}{g_{22}} \tag{15}$$

It is necessary to find a ratio that gives the desired Stern K factor. By using Eq. 14 and Eq. 15, the following equation is derived, which relates R to the K factor.

$$R = \left(\sqrt{K \left[\frac{|y_{21}y_{12}| + Re(y_{21}y_{12})}{2g_{11}g_{22}} \right]} \right) - 1$$

With $K = 3$, $R = 3.07$ and the appropriate conductances are found from Eq 15. The mismatched source and load admittances are $Y_S = 2.0 \cdot 10^{-3} - j5.2 \cdot 10^{-3}$ and $Y_L = 2.1 \cdot 10^{-4} - j6.6 \cdot 10^{-5}$.

The device input and output admittances need to be calculated. Using Eq. 1 and Eq. 2, the following equation results:

$$Y_{in} = \frac{i_1}{v_1} = y_{11} - \frac{y_{12}y_{21}}{y_{22} + Y_L} = 1.2 \cdot 10^{-3} + j7.5 \cdot 10^{-3}$$

Rearranging Eq. 2 and Eq. 1, the output admittance for the MOSFET is

$$Y_{out} = \frac{i_2}{v_2} = y_{22} - \frac{y_{21}y_{12}}{y_{11} + Y_S} = 126.3 \cdot 10^{-6} + j304.6 \cdot 10^{-6}$$

Now, the actual values of the passive elements can be determined. The optimum source or load resistance and then the series capacitive reactance are calculated by Eqs. 3 and 7 respectively. To convert a capacitive reactance to capacitance, the following equation is used.

$$C = \frac{1}{(2\pi f)X_C} \tag{16}$$

The capacitance that provides the series reactance at 435 MHz is $C_1 = 2.47pF$ for the input circuit and $C_4 = 0.759pF$ for the output circuit. The resonant circuit capacitors can be obtained by

$$C_2 = C_T - C_{in} - C_1' \tag{17}$$

and

$$C_3 = C_T - C_{out} - C_4' \tag{18}$$

To determine the required total capacitance (C_T) of each circuit, Eqs. 8 and 10 are used. The bandwidths for the input and output circuits need to be different as these circuits are cascaded. The bandwidths should be chosen so that the output circuit determines the desired frequency response. In this design, the input circuit bandwidth will be 60 MHz and the output circuit bandwidth will be 30 MHz.

The equivalent device input capacitance (C_{in}) and output capacitance (C_{out}) are found from $C = B/2\pi f$, where B is the susceptance. The parallel capacitive reactance of each circuit is found by an equation derived from 6 and the RC network definitions for Q,

$$X_p = X_s \left(\left(\frac{R_s}{X_s} \right)^2 + 1 \right)$$

Using Eq. 16, the parallel equivalents of C_1 and C_4 are found. The values for the capacitors at the resonant frequency of 435 MHz are $C_2 = 3.66pF$ and $C_3 = 0.937pF$.

To finish the design for the tuning circuits, the input and output inductances need to be calculated. Using Eq. 12, $L_1 = 15.5nH$ and $L_2 = 74.4nH$ for the resonant frequency of 435 MHz.

The RF choke, the BPC capacitor, and the 1 μF capacitor are external discrete components. The discrete inductor needs to have a self-resonant frequency above 450 MHz to operate as an RF choke. The bias circuit needs to be well bypassed otherwise it will become part of the a.c. input circuit.

Finally R_1 and R_2 have to be determined to bias the MOSFET at $V_g = 2.5V$. Since the supply voltage is +5V, the two resistor values are equal. The value of the resistors is somewhat arbitrary. When $R_1 = R_2 = 2K\Omega$, the current through the bias stick is 1.25 mA.

The SPICE simulator reported that although the resonant frequency was within 0.5% of the designed resonant frequency, the circuit element values did not give the desired bandwidth for the amplifier. The bandwidth was 43% of the expected bandwidth. Using the SPICE simulator again, a simulation was made of the equivalent circuits in Fig. 3 with the designed values. This resulted in two circuits with the desired bandwidths and resonant frequency. At this point, the active device model or the Stern solution for stability which gives the input and output admittances were suspect in the bandwidth reduction. The active device model seemed unlikely, so it was decided to re-visit the Stern solution. The calculation for the susceptance term is the only difference between the optimal Stern solution and the good solution. The following equations from [8] are used for the optimal Stern solution.

$$B_1 = \frac{z_0\sqrt{K\left(|y_{21}y_{12}| + Re(y_{21}y_{12})\right)}}{2G_2}$$

where z_0 is a real root of a third order equation and $G_2 = G_L - g_{22}$. To find the source susceptance term, $B_S = B_1 - b_{11}$ is used. The third order equation is

$$z^3 + [K(|X| + Re(X)) + 2Re(X)]z - 2Im(X)\sqrt{K(|X| + Re(X))} = 0$$

where $X = y_{21}y_{12}$. The load susceptance term is found by $B_L = B_2 - b_{22}$ where

$$B_2 = \frac{G_2 z_0}{\sqrt{K\left(|y_{21}y_{12}| + Re(y_{21}y_{12})\right)}}$$

The recalculated mismatched source and load admittances are $Y_S = 2.0 \cdot 10^{-3} - j8.8 \cdot 10^{-3}$ and $Y_L = 2.1 \cdot 10^{-4} - j1.6 \cdot 10^{-4}$. Re-calculating all the passive tuning elements and simulating the new design shows an increase in the bandwidth to 75% of the desired result. The optimum Stern solution maximizes both the susceptances and the conductances for maximum power gain realizable for a given stability factor, K. As Stern points out in [8], maximizing the power gain imposes certain restrictions on the bandwidth. The restrictions could be due to the active device or the circuit elements. There is no simple relationship that ties power gain with bandwidth. This seems like a reasonable explanation for the reduced bandwidth.

Passive Element	Value
C_1	$2.02pF$
C_2	$0.00pF$
C_3	$0.188pF$
C_4	$1.20pF$
L_1	$36.8nH$
L_2	$89.0nH$

Table 1: Table of the Passive Element Values

The amplifier design was then analyzed using a microwave optimization program called Touchstone by EEsof [9]. The Touchstone results showed a frequency response similar to the SPICE results. Unfortunately, the input and output reflection coefficients indicated a potential instability within the desired operating range of the amplifier. A SPICE run indicated the amplifier was stable with a source and load of 50 Ω. A second run with the source and load open-circuited displayed an oscillator. Stern's paper [8] implies the amplifier must be used with the designed terminations. Touchstone was then used to optimize the amplifier design. The resulting values for the passive elements in Fig. 2 are shown in Table 1.

4 Layout

The RF amplifier is designed to be fabricated in an n-well 2 μm CMOS process. The layout of each type of circuit element is now addressed.

A capacitor is formed by placing an insulator (dielectric) between two conductive plates. In a CMOS process, the dielectric is generally silicon dioxide SiO_2. The value is determined by

$$C = \frac{\varepsilon_{SiO_2}\varepsilon_o A}{t_{ox}} \tag{19}$$

where t_{ox} is the thickness of the oxide and A is the area of the capacitor. Any variations in t_{ox} or the area will change the result of the capacitance. The tolerance is usually within $\pm 15\%$ and is mainly determined by the oxide thickness variation. [5]

There is a parasitic capacitance from the bottom plate to the substrate. A parasitic capacitance also exists on the top plate due to the connecting leads. To minimize the parasitics in the layout of the RF amplifier, the bottom plate of C_1 is the top plate of C_2 as shown in Fig. 5, where $M1$ designates the Metal 1 layer, $M2$ designates the Metal 2 layer, and POLY designates the polysilicon layer. Likewise, the bottom plate of C_4 is the top plate of C_3. This layout configuration removes the parasitic capacitance that would be between the bottom plate of one capacitor and the top plate of the other capacitor. The bottom layer of the structure is grounded and the substrate is also at an a.c. ground, eliminating the effects of that bottom plate parasitic.

The RF amplifier layout uses a polysilicon type resistor. A polysilicon resistor is a

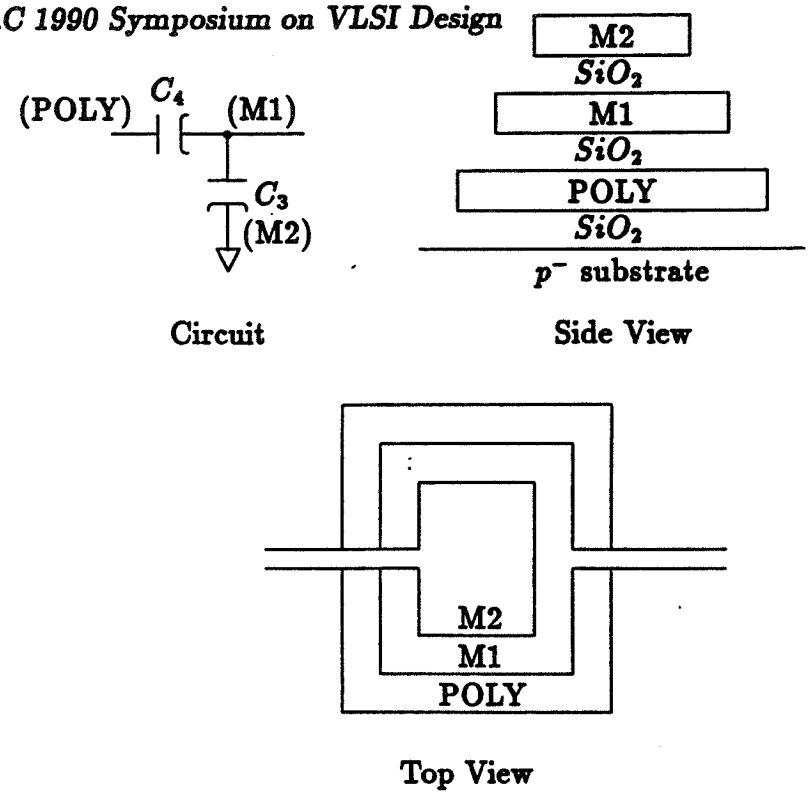


Figure 5: Capacitor Layout

uniform slab of polysilicon surrounded by silicon dioxide. The resistance is determined by

$$R = \frac{R_s l}{w} \tag{20}$$

where R_s is the sheet resistance, l is the length, and w is the width. The sheet resistance is defined as

$$R_s = \frac{\rho}{t} \tag{21}$$

where ρ is the resistivity of the polysilicon and t is the thickness. In this particular process the sheet resistance is 40 Ω/ square.

Inductors are difficult to implement in an IC due to several problems. One problem is deriving an accurate model. Several shapes may be used to make a thin film inductor. A straight line can be used for low inductances on the order of two to three nH. Circular spiral or square spiral shapes provide a much higher inductance and Q value. The circular spiral has a higher Q value of about 10 % than a square spiral for the same diameter [10]. There have been many papers written on how to model the various structures [11,12,13]. Remke [14] has a good summary of the traditional equations used in circular spiral inductor design.

Another problem with thin film inductors is the large area needed to implement even a small sized inductor (mH). Since the RF amplifier is designed for UHF frequencies, the inductor sizes are reduced. A third problem for this amplifier was how to implement the shape given the tools that are available. A circular spiral was implemented by placing the center of small width rectangles around a spiral line so that the rectangles overlapped

equally. The rectangles were rotated around the spiral. The spiral ribbon is a metal 2 layer.

Using Wheeler's original formula [14] with an adjustment for ground plane effects, the resulting inductance formula [10] is

$$L(nH) = \left(\frac{1}{25.4}\right)\left(\frac{a^2n^2}{8a+11c}\right)K_g$$

where n is the number of turns, K_g is the ground plane adjustment and a and c are described in the following equations $a = (d_0 + d_i)/4$ and $c = (d_0 - d_i)/2$ where d_0 is the outer diameter and d_i is the inner diameter. The ground plane adjustment equation [15,16] is

$$K_g = 0.57 - 0.145ln\frac{W}{h} \tag{22}$$

where W is the width of the spiral track and h is the separation height of the inductor to the ground plane.

In order to find the Q of the inductor, the resistance must be found. The resistance of a circular spiral inductor is [10]

$$R = \frac{K\pi anR_s}{W}$$

where R_s is the sheet resistance of the metal and K is a correction factor that takes into account the crowding of the current at the corners. To find K, the following equation is given

$$K = 1 + 0.333\left(1 + \frac{S}{W}\right)^{-1.7}$$

where S is the spacing in the spiral. The Q value for each of the required inductors is low: $L_1 = 7.6$ and $L_2 = 10.0$.

The gate of the active device, $Q1$ of Fig. 2, is made of a polysilicon layer. Because of polysilicon's high resistance with long lines and the parasitic capacitances associated with the active device, the propagation of signals is delayed. A way of reducing the propagation delay is to segment the gates.

The signal delay of a distributed n-section network as n becomes very large is $t_d = (rcl^2)/2$ where r denotes the resistance per unit length, c denotes the capacitance per unit length, and l denotes the length of the wire. The active device was designed for a gate length of 5 μm and a gate width of 500 μm. If the poly is a distributed network of length 500 μm, an unacceptable propagation delay of 7 ns would result. If the gate is divided into 20 segments of 25 μm lengths, the resulting propagation delay is only 17.5 ps.

The I/O pad cells contain electrostatic discharge (ESD) protection. Only parallel protection was used to eliminate series resistance delays. To help prevent latch-up in the I/O pad cells, a p^+ diffusion ring surrounds the nMOS transistors and an n^+ ring is diffused into the perimeter of each of the n-wells. The active device, $Q1$, is surrounded by a p^+ guard ring as well as the entire amplifier.

Since MOSIS requires 30 pads in the tiny chip frame, some of the extra pads are used as taps for the inductors and capacitors. The tap and power supply lines to the inductors

Parameter Changed	Gain (dB)	Center Frequency (MHz)	Bandwidth (MHz)
Standard Amplifier	15.2	420	48
+15%	15.3	402	46
-15%	13.9	440	58
+30%	17.7	365	31
-30%	12.3	511	75
Parasitic Capacitors	18.6	425	17
Lossy Inductors	6.5	436	96
1.5V Bias	6.9	433	71
3.5V Bias	19.0	419	33

Table 2: Table of SPICE Results

are made as wide as possible to reduce their contribution to the total inductance. The inductance of a straight strip of ribbon is [10]

$$L(nH) = 2 \cdot 10^{-4} l \left[ln\left(\frac{l}{w+t}\right) + 1.193 + 0.2235\frac{w+t}{l} \right] K_g$$

where l is the length of the strip, w is the width, t is the thickness and K_g is given in Eq. 22. In addition to the metal line inductance, there is inductance associated with the bond wire from the pad to the pin of the packaged IC. The bond wire inductance is reduced by using a PLCC (plastic leaded chip carrier). The tap and signal lines to the capacitors are made as wide as possible to reduce the resistance of each of the lines.

5 Conclusions

Numerous SPICE simulations were run to show the effects of components value variations, inductance Q's, parasitics and voltage variations. The results are summarized in Table 2. The amplifier continues to have gain and a usable bandwidth under the changes simulated. A test circuit has been submitted to MOSIS to verify the theory presented in this paper.

Although the amplifier continues to function, there are dramatic changes in center frequency and bandwidth. This is to be expected for L and C component value changes since the center frequency is a function of these values. ±15% variations in capacitance is expected for MOS capacitors. For an amplifier to meet a manufacturing specification, a digitally trimmed capacitor array might be necessary. The inductor variation should be much smaller than the capacitance variation but no data can be found in the open literature on MOS inductors. A study of MOS inductors needs to be conducted to establish this data. ±30% variations were simulated since the accuracy and applicability of the models established in the literature [15,11,12] for inductors in GaAs was in question. Since both inductors are connected to supplies through pins, external trimming is possible. Bias voltages can be controlled accurately and present no real problem. Parasitics can be

accurately modeled and can be incorporated into the design optimization procedure. The real concern is the low Q values for the inductances. Additional amplifier configurations need to be studied to find configurations requiring smaller values of inductances which can be designed for higher Q values.

This work indicates that RF amplifiers can be designed in MOS, but that further work needs to be conducted to establish techniques and configurations that will result in manufacturable RF amplifiers.

References

[1] P. Allen and D. Holberg, *CMOS Analog Circuit Design*, New York, N.Y., Holt, Rinehart and Winston, 1987, Chap 1

[2] M. Milkovic "VLSI High Frequency CMOS Operational Amplifiers for Communications Applications", Proceedings of the 27th Midwest Symposium on Circuits and Systems, vol. 2, June 1984, pp.784-787

[3] K. Niclas, W. Wilser, R. Gold, and W. Hitchens, "The Matched Feedback Amplifier: Ultrawide-Band Microwave Amplification with GaAs MESFET's", IEEE Transactions on Microwave Theory and Techniques, vol. MTT-28, April 1980, pp.285-294

[4] D. Ribner "Some variations in CMOS Operational Amplifier Design", Proceedings of the 27th Midwest Symposium on Circuits and Systems, vol. 2, June 1984, pp. 788-791

[5] R. Gregorian and G. Temes, *Analog MOS Integrated Circuits for Signal Processing*, New York, John Wiley & Sons, 1986

[6] J. Lenk, *Manual for MOS Users*, Reston, Virginia, Prentice-Hall, 1975

[7] H. Krauss, C. Bostian, and F. Raab, *Solid State Radio Engineering*, New York, John Wiley & Sons, 1980

[8] A. Stern, "Stability and Power Gain of Tuned Transistor Amplifiers", Proceedings of the IRE, vol. 45, pp. 335-343, March 1957

[9] EEsof, Inc., *User Manual for Touchstone*, Westlake Village, California, 1985

[10] I. Bahl and P. Bhartia, *Microwave Solid State Circuit Design*, New York, John Wiley & Sons, 1988

[11] H. Dill, "Designing Inductors for Thin-Film Applications", Electronic Design, February 17, 1964, pp. 52-60

[12] P. Shepherd, "Analysis of Square-Spiral Inductors for Use in MMIC's", vol. 34, April 1986, pp. 467-472

[13] E. Pettenpaul, Hartmut Kapusta, Andreas Weisgerber, Heinrich Mampe, Jurgen Luginsland, and Ingo Wolff, "CAD Models of Lumped Elements on GaAs up to 18 GHz", vol. 36, February 1988, pp. 294-304

[14] R. Remke and G. Burdick, "Spiral Inductors for Hybrid and Microwave Applications", Proceedings of the 1974 Electronic Components Conference, May 1974, pp. 152-161

[15] R. Chaddock, "The Application of Lumped Element Techniques to High Frequency Hybrid Integrated Circuits", The Radio and Electronic Engineer, vol. 44, 1974, pp. 414-420

[16] K. Gupta, R. Garg, R. Chadha, *Computer-Aided Design of Microwave Circuits*, Dedham, Massachusetts, Artech House, Inc., 1981

N94-71083

A Comparison of Two Fast Binary Adder Configurations

J. Canaris and K. Cameron

NASA Engineering Research Center for VLSI System Design
College of Engineering
University of Idaho
Moscow, Idaho 83843
Phone (208) 885-6500

Abstract - **Conditional sum and binary lookahead carry are two methods for performing fast binary addition. These methods are quite different, but the adders have a common feature that makes them interesting to compare. Both adders have the carry generating logic implemented as a binary tree, which grows in depth as $\log_2 n$, n equals number of bits in the adder. The delay in the carry paths also grows in proportion to $\log_2 n$. This paper shows that the Transmission-Gate Conditional-Sum adder and the binary lookahead carry adder have the same speed of addition, but that the conditional sum adder requires only 46% of the area.**

1 Introduction

There are many high performance binary adders described in the literature [1,2,3,4,5,6]. These adders use a variety of techniques to speed up the generation of the carry signal. Carry lookahead, carry select and carry completion are among the techniques used. Some methods [4] use specialized encodings to perform addition without a carry being generated. This paper covers two techniques which, although quite different in the method they use, lend themselves to a regular layout as described in [7]. The two high performance adders discussed in this paper are the Binary Lookahead Carry (BLC) adder [3] and the Transmission-Gate Conditional-Sum (TGCS) Adder [6]. These adders are interesting because their area grows in proportion to $n \log_2 n$ and their propagation delay grows with $\log_2 n$, where n is the number of bits in the adder.

2 Binary Addition

Binary addition can of course be implemented by a one-dimensional array of full adder cells which implements the truth table shown in Table 1.

A	*B*	*Ci*	*So*	*Co*
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 1: Full Adder cell truth table.

This function can also be represented by:

$$Si = A_i \oplus B_i \oplus Ci_i \quad (1)$$

$$Co = (A_i \cdot B_i) + (A_i \cdot Ci_i) + (B_i \cdot Ci_i) \quad (2)$$

It is Equation 2 which is the target of addition optimization techniques, as it forms an n bit ripple path. The methods which the BLC and TGCS adders use to speed up this carry path will be discussed next.

2.1 BLC carry generation

This discussion parallels that of [3]. First we look at carry lookahead adders in general. The linear growth in the delay of the carry chain can be improved by calculating the carries to each bit in parallel.

$$C_i = G_i + P_i \cdot C_{i-1} \quad (3)$$

$$G_i = A_i \cdot B_i \quad (4)$$

$$P_i = A_i \oplus Bi \quad (5)$$

The carry equation, Equation 3, can be expanded as follows:

$$C_i = G_i + P_i \cdot G_{i-1} + P_i \cdot P_{i-1} \cdot G_{i-2} + \cdots + P_i \cdots P_1 \cdot C_0 \quad (6)$$

The sum, S_i is generated by:

$$S_i = C_{i-1} \oplus P_i \quad (7)$$

Writing down a small number of terms of Equation 6 will show that the layout of such a network will be quite irregular and that the number of gates needed to implement such a scheme increases rapidly. Four stages of lookahead, for example, will have the follow will have the following terms.

$$\begin{aligned}
C_1 &= G_1 + (P_1 \cdot C_0) \\
C_2 &= G_2 + (P_2 \cdot G_1) + (P_2 \cdot P_1 \cdot C_0) \\
C_3 &= G_3 + (P_3 \cdot G_2) + (P_3 \cdot P_2 \cdot G_0) + (P_3 \cdot P_2 \cdot P_1 \cdot C_0) \\
C_4 &= G_4 + (P_4 \cdot G_3) + (P_4 \cdot P_3 \cdot G_2) + (P_4 \cdot P_3 \cdot P_2 \cdot G_1) + (P_4 \cdot P_3 \cdot P_2 \cdot P_1 \cdot C_0)
\end{aligned}$$

For these reasons a pure carry lookahead adder is usually implemented with small sections of lookahead combined with some other addition scheme, such as carry select. The BLC adder however takes a different approach, which lends itself to a regular layout.

A new operator, o is introduced, such that:

$$(g,p)o(g',p') = (g + (p \cdot g', p \cdot p') \tag{8}$$

where g, p, g' and p' are boolean variables. This operator is associative and the carry signals can be found by:

$$C_i = G_i \tag{9}$$

with

$$(G_i, P_i) = \begin{cases} (g_1, p_1) & \text{if } i = 1 \\ (g_i, p_i) \cdots o \cdots (G_{i-1}, P_{i-1}) & \text{if } 2 \leq i \leq n \end{cases} \tag{10}$$

that is:

$$(G_i, P_i) \;=\; (g_i, p_i)o(g_{i-1}, p_{i-1}) \cdots o \cdots (g_1, p_1) \tag{11}$$

The associative property of the o operator allows the carry lookahead circuitry to be organized as a binary tree structure whose depth is proportional to $\log_2 n$, hence the name Binary Lookahead Carry adder. The propagation delay through this carry section is also proportional to $\log_2 n$. This decrease in the carry propagation delay is quite significant when compared to the delay through a ripple carry adder. A 16 bit adder, for example, with a delay of 1 nanosecond/stage, will drop from 16 nanoseconds to 4 nanoseconds of total delay.

Figure 1 shows the organization of a Binary Lookahead Carry adder. It should be noted that the BLC adder has no carry signal into the least significant bit.

2.2 TGCS carry generation

The BLC adder introduces parallelism in the carry propagation path through the clever use of a new operator, the o operator. The TGCS adder, as described in [2,6], uses a more brute force approach to increase performance. In the TGCS adder, parallelism is introduced at the beginning of the addition process by calculating two sums and two carries at each bit. These four outputs are calculated from the two input signals (the addend and the augend) as if a carry was/was not propagating into that bit position. Multiplexers are then used to select the proper sum and carry from that bit position based on whether C_i does/does not propagate. This calculation is known as the conditional sum, and can be described as follows.

$$S_i^0 = A_i \oplus B_i = (A_i \cdot \bar{B}_i) + (\bar{A}_i \cdot B_i) \tag{12}$$

$$C_{i+1}^0 = A_i \cdot B_i \tag{13}$$

$$S_i^1 = A_i \odot B_i = (A_i \cdot B_i) + (\bar{A}_i \cdot \bar{B}_i) \quad (14)$$

$$C_{i+1}^1 = A_i + B_i \quad (15)$$

As in the BLC adder the multiplexers required to chose the proper sum and carry from each bit can be organized into a binary tree of depth $\log_2 n$. Each subsection of the tree can calculate the provisional sum and carry in parallel with other subsections, so a propagation delay proportional to $\log_2 n$ can be attained. Figure 2 shows the organization of a Transmission-Gate Conditional-Sum adder.

3 Logic Design of the Adders

CMOS technology allows for a wide choice of different logic types, such as fully complementary gates, domino logic, pseudo-nmos and pass transistors [3]. The types of logic chosen to implement the BLC adder and the TGCS adder are quite different. The specific logic configurations chosen are described below.

3.1 BLC adder logic

Initial investigations of a pass transistor (transmission gate) implementation of the BLC adder indicated that it would be difficult to introduce inverter buffers in the delay path. These buffers would have been required to implement the pass functions needed. For this reason a traditional, fully complementary gate logic was chosen. As can be seen from Figure 1 three different cells are needed to form a BLC adder. They are the

- G and P generator cell, labeled G.
- The o operator cell, labeled O.
- The final summation cell, labeled SO.

There is in addition 2 cells which provide interconnections between the three cells listed above. These routing cells are provided as a layout convenience and have no logical purpose. The G cell directly implements the logic functions given by Equation 4 and Equation 5. The O cell implements the function given by Equation 8, which is restated here in a more accessible form.

$$G = g + (p \cdot g') \quad (16)$$

$$P = p \cdot p' \quad (17)$$

The SO cell performs the final summation of the partial sum (P) of the current bit with the carry out of the previous bit, as given by Equation 7. It should be noted that this logic implementation differs from that described in [3,7]. In that implementation o and $\bar{o}$ are alternated every other column in the carry evaluation block. That implementation has also introduced inverter buffers in the locations where the O cells are missing. This

organization, while still regular does not grow in a true $\log_2 n$ fashion. For instance in a four bit adder three columns in the carry block are required, not two. In the implementation described in this paper the buffering is performed by the O cells themselves, and the carry block depth does grow as described.

3.2 TGCS adder logic

Conditional sum adders can be implemented in standard gate logic, just as the BLC adder has been. This implementation however uses pass transistor logic (transmission gates) instead. This particular Transmission-Gate Conditional Sum adder is based on work presented in [6]. This work differs from [6] however. Based on formal pass transistor design techniques [8,9,10] and the use of n-transistor only pass networks for arithmetic units [11], a smaller and simpler TGCS adder has been designed. As can be seen from Figure 2 seven different cells are needed to form this TGCS adder. They are the

- An input buffer cell, labeled IBUFF.
- The conditional sum cell, labeled CONSUM.
- A four section 2-1 multiplexer, labeled MUX2A.
- A two section 2-1 multiplexer, labeled MUX2B.
- A one section 2-1 multiplexer, labeled MUX1.
- An inverter buffer cell, labeled MUXBUFF.
- An output buffer, labeled OBUFF.

All routing required by this adder is contained within the cells listed above. Unlike the BLC adder described above no special interconnection cells are required. The IBUFF and OBUFF cells are simply buffering stages. If the adder was driven directly by a flip-flop and drove directly into a flip-flop these cells would not be required. The CONSUM cell directly implements the logic functions given by Equations 12, 14, 13 and 15. These equations are rewritten in the pass transistor format described by [8,9,11] as:

$$\begin{aligned} S_i^0 &= A_i(\bar{B}_i) + \bar{A}_i(B_i) \\ C_{i+1}^0 &= A_i(B_i) + \bar{A}_i(0) \\ S_i^1 &= A_i(B_i) + \bar{A}_i(\bar{B}_i) \\ C_{i+1}^1 &= A_i(1) + \bar{A}_i(B_i) \end{aligned}$$

The multiplexing cells MUX2A, MUX2B and MUX1 are used to select the proper output from the CONSUM cells. The MUXBUFF cell is used to buffer the outputs of certain multiplexer cells so as to drive other multiplexers.

4 Circuit Design and Layout of the Adders

The BLC and TGCS adders presented here were designed to be used in a multiply-accumulate (MAC) block required by an image processing chip. The MAC was organized as an 8 bit by 10 bit multiplier, with a 28 bit accumulator. The multiplier itself was organized as a carry-save multiplier operating at a 20MHz clock frequency. The adders were to be used as the final summation stage in the multiplier as well as performing the accumulation required by this application. The adders therefore needed to be designed to have a carry propagation delay of less than 25 nanoseconds under worst case conditions using a 1.6μm double-metal CMOS process. Worst case conditions for this project were:

- 4.5V supply, 0.2V noise on VDD and VSS.
- 140°C.
- Worst Case Parameter set.
- 2.0pF output load.

The circuit design and layout of the BLC adder was straight forward as it was designed with traditional CMOS logic gates. CAD tools were available to aid the design engineers in sizing transistors. The worst case propagation delay is 24 nanoseconds. The layout of the adder was also uncomplicated. The area required by the 28 bit BLC adder is 1,282μm X 452.8μm, which is 45.8μm X 452.8μm per bit.

The circuit design and layout of the TGCS adder were more complicated than for the BLC adder. The circuit design and sizing of pass transistor networks is still more of an art than a science. In this case the adder layout was required to be pitch matched with two memory blocks and a data path. Transistor sizes were chosen to fit into a cell layout with a fixed width, parasitic capacitances were extracted from the layout and fed into SPICE simulations of the critical path. This procedure was iterated until the specified speed was attained. It is not surprising that the worst case carry propagation delay in this adder was also 24 nanoseconds. The area required by the 28 bit TGCS adder is 963.2μm X 278.6μm, which is 34.4μm X 278.6μm per bit. This is only 46% of the area required by the BLC adder.

5 Summary and Conclusions

This paper describes two methods, Binary Lookahead Carry and Transmission Gate Conditional Sum, for performing high speed addition. The investigation of these adders was undertaken to find the best adder for a particular application, the design of a multiply-accumulate block in an image processor. At the outset neither method seemed to have an advantage over the other. When each adder was designed and met the performance requirements it was shown that the TGCS configuration has a significant area advantage over the BLC configuration.

Acknowledgement

The authors wish to acknowledge the NASA Space Engineering Research Center program and the Lawrence Livermoore Laboratory, whose support made this investigation possible.

References

[1] I. Flores, *The Logic of Computer Arithmetic*, Englewood Cliffs, New Jersey, Prentice-Hall, 1963.

[2] Kai Hwang, *Computer Arithmetic*, New York, New York, John Wiley and Sons, 1979.

[3] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*, Reading, Mass., Addison-Wesley, 1985.

[4] Y. Harata *et al.*, "A High-Speed Multiplier Using a Redundant Binary Adder Tree", IEEE JSSC, Vol. SC-22, February 1987, pp. 28-33.

[5] T. G. Noll *et al.*, "A Pipelined 330-MHz Multiplier", IEEE JSSC, Vol. SC-21, June 1986, pp. 411-416.

[6] A. Rothermel *et al.*, "Realization of Transmission-Gate Conditional-Sum (TGCS) Adders with Low Latency Time", IEEE JSSC, Vol. 24, June 1989, pp. 558-561.

[7] R. P. Brent and H. T. Kung, "A Regular Layout for Parallel Adders", IEEE Trans. Comput., Vol. C-31, March 1982, pp. 260-264.

[8] 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

[9] 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.

[10] C. Pedron and A. Stauffer, "Analysis and Synthesis of Combinational Pass Transistor Circuits", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 7, July 1988, pp. 775-786.

[11] J. Canaris, "A High Speed Fixed Point Binary Divider", Proceedings ICASSP-89, Glasgow, Scotland, May 1989, pp. 2393-2396.

C-2.

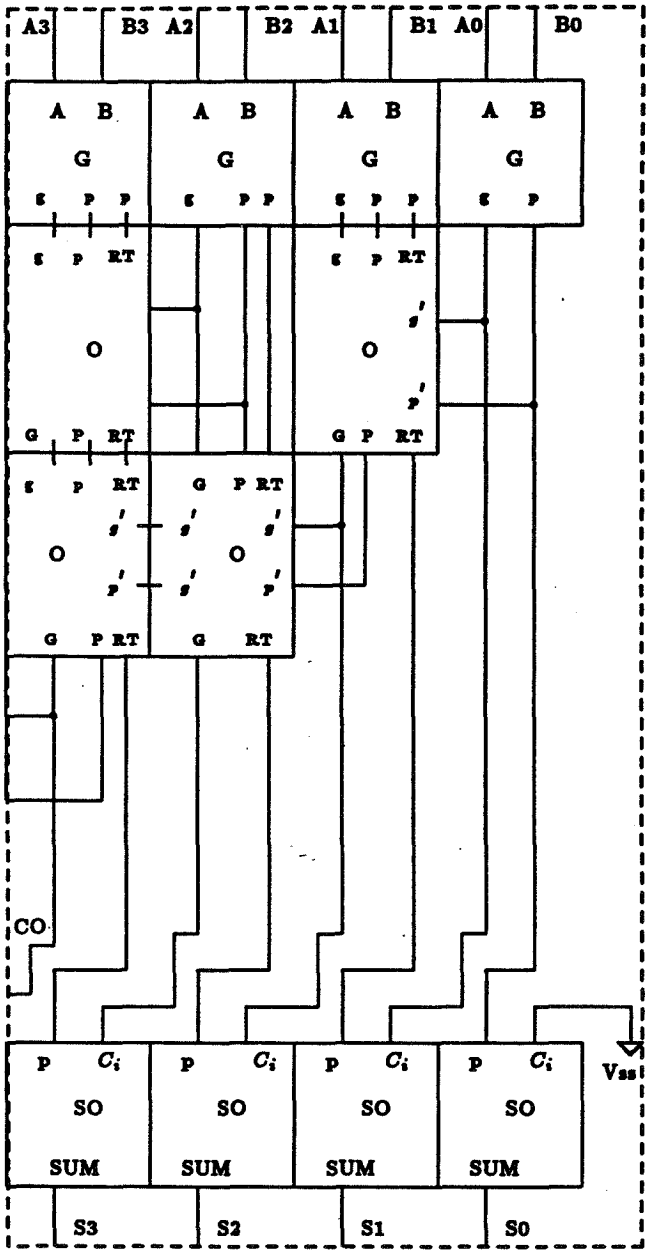


Figure 1: BLC adder block diagram.

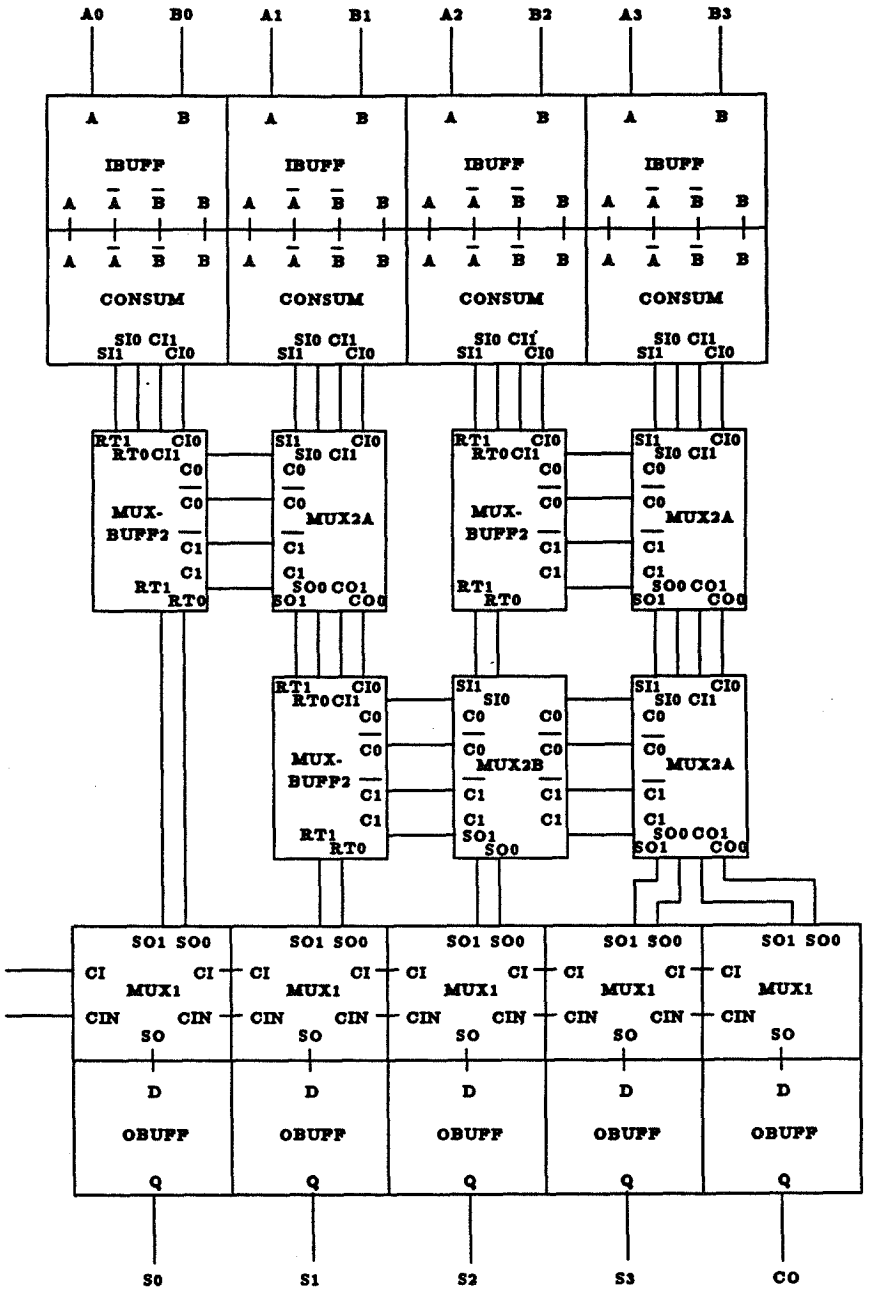


Figure 2: TGCS adder block diagram.

Self Arbitrated VLSI Asynchronous Sequential Circuits

S. Whitaker and G. Maki
NASA Engineering Research Center
for VLSI System Design
University of Idaho
Moscow, Idaho 83843

Abstract - **A new class of asynchronous sequential circuits is introduced in this paper. The new design procedures are oriented towards producing asynchronous sequential circuits that are implemented with CMOS VLSI and take advantage of pass transistor technology. The first design algorithm utilizes a standard Single Transition Time (STT) state assignment. The second method introduces a new class of self synchronizing asynchronous circuits which eliminates the need for critical race free state assignments. These circuits arbitrate the transition path action by forcing the circuit to sequence through proper unstable states. These methods result in near minimum hardware since only the transition paths associated with state variable changes need to be implemented with pass transistor networks.**

1 Introduction

Most control logic in current VLSI is realized as a synchronous sequential circuit. Major disadvantages in using synchronous machines include the clock and power distribution. Races are avoided in synchronous logic by selecting a clock rate that is slow enough to allow propagation of signals from the slowest logic block. This forces the clock rate to be governed by the slowest block on a VLSI chip. Moreover, the clock signal is assumed to transition simultaneously for all flip flops. Clock distribution on a VLSI circuit must account for the RC time delay caused by the finite resistance of metal lines. Another major design consideration for high speed circuits is power bussing. With synchronous logic, CMOS circuits have a peak current demand at the clock edge since many nodes transition simultaneously.

Asynchronous design avoids these limitations. Asynchronous designs are not often used because of the more complicated design procedures and the electronic considerations for proper operation. This paper introduces a new class of asynchronous circuits which eliminates the need for critical race free state assignments. Non-normal circuit operation is employed in the new architecture. A non-normal transition is characterized by allowing the circuit to assume a series of intermediate states prior to reaching the stable state[9]. The non-normal mode circuits take advantage of the ability of pass transistor networks to be tristated. The circuits are straight forward and easy to implement since they are free from hazard.

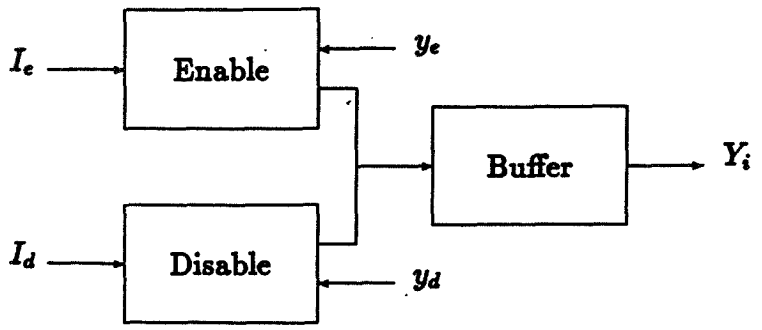


Figure 1: Enable-Disable Block Diagram.

y_i	Input	Y_i
0	0	0
0	1	1
1	0	0
1	1	1
0	Z	0
1	Z	1

Table 1: Buffer state table.

Section 2 of this paper introduces a general enable-disable pass transistor model. Section 3 presents a design procedure and a design example utilizing a Liu [1] state assignment. Section 4 introduces the theory for self arbitrated circuits and uses a one-hot-code to illustrate a handshaking protocol. Section 5 presents a general design procedure, then modifies the procedure to cover scale-of-two-loops [10] and presents a design example. Section 6 discusses hardware bounds and Section 7 summarizes the results of this work.

2 Circuit Model

The general model for the circuit is shown in Figure 1. There are two networks of pass transistors labeled *Enable* and *Disable* followed by a *Buffer* stage. The state variables values are held by a memory circuit in the buffer. The enable and disable pass transistor networks are designed to respond to input transitions that cause state variables to change logic states.

The buffer circuit is described by the state table shown in Table 1 where Z represents a high impedance state driving the input of the buffer. The next state follows the input when the input is either a 1 or a 0; $Y_i = y_i$ whenever the input to the buffer is tristated.

The pass transistor networks are driven by n input variables and constants such that the set of inputs, I, is described by,

$$I = [I_1, I_2, I_3, \ldots, I_n, 0, 1] \tag{1}$$

	I_p	y_1	y_2	y_3
A	A	1	1	0
B	A	1	0	0
C	A	1	0	0
D	D	1	1	1
E	D	1	0	1
F	F	0	1	0
G	F	0	0	0

Table 2: Example τ partition flow table.

A state machine is constructed from a set of circuits shown in Figure 1 where each circuit produces one next state variable, Y_i. The set of present state variables, y, allow the state machine to assume as many as 2^m unique states.

$$y = [y_1, y_2, \ldots, y_m] \tag{2}$$

The internal state of the machine may be described by an m-tuple consisting of each present state variable. S is the set of m-tuples which define the specified states of a sequential machine. Let the specified $q \leq 2^m$ internal states,

$$S = S_1, S_2, \ldots, S_q \tag{3}$$

Definition 1 *Let ρ_s be a partition that partitions states of set S from all other states under input I_p of a flow table.*

The states of S represent all the states of a transition path associated with S. S could be the elements of a transition pair or a k-set, depending on the situation. A k-set [3,6] consists of all next state entries in a column of a flow table that lead to the same stable state. For example $\rho_{abc} = ABC; DEFG$ under I_p of Table 2 partitions ABC from $DEFG$. In the hardware realization, partition ρ_s represents a path in the circuitry that forms the next state variable Y_i. In Table 2 there are 3 k-sets consisting of states ABC, DE, and FG under input I_p. Let $\tau_1 = ABCDE; FG$, $\tau_2 = AFD; BCGE$ and $\tau_3 = DE; ABCFG$. The product expression covering ρ_{abc} is then $y_1\,\overline{y_3}$. The path realizing ρ_{abc} would consist of two series transistors, one controlled by y_1 and the other by $\overline{y_3}$.

Several key concepts can be applied to ρ_s.

1. The set S is dependent on the state assignment method.
   (a) If the state assignment is a Tracey assignment, S consists of transition pairs under input I_p.
   (b) If the state assignment is a Liu assignment, then S consists of k-sets under input I_p.
2. Partition ρ_s can be expressed as a product of the partitioning variables of I_p.

Definition 2 *Let $\rho_{ij} I_k(x)$ represent a pass transistor network decoding the transition path that contains S_i S_j, S_j being stable, where ρ_{ij} covers the transition path containing S_i, S_j, and a qualifying input I_k passing x, where $x \in [0,1]$.*

In the enable-disable model, when the enable and disable pass networks are inactive, the input to the buffer is presented with a high impedance. The next state variable, Y_i, does not change under this input condition. An input only needs to be passed by the pass network when a state transition is required. The disable circuit provides a path to force $1 \rightarrow 0$ transitions of the state variable. The enable circuit provides a path to force $0 \rightarrow 1$ transitions.

Referring to Figure 1, the enable network is armed by present state information contained in $y_e \in y$ to respond to input I_j in $I_e \subset I$, which forces $Y_i \rightarrow 1$. The enable circuit is a set of pass implicants decoding each total state requiring $Y_i \rightarrow 1$. The states where $y_i = 0$ with next state $Y_i = 0$, are don't care state for the enable network logic.

When the circuit is in a state where $y_i = 1$, the disable network performs a similar function. The disable network is armed by present state information contained in $y_d \subset y$ to keep $y_i = 1$ until an input I_j in $I_d \subset I$ transitions causing the sequential machine to move to a new state with $y_i = 0$. The disable circuit could be a set of pass implicants decoding each total state requiring $y_i \rightarrow 0$. The states in which $y_i = 1$, but whose next state also require $y_i = 1$, are don't care states.

A simple double inverter buffer could be used as the buffering element and would function according to Table 1; however, the state information is dynamically stored as charge on the gates of the first buffering inverter. This would result in a minimum operating frequency due to leakage of the stored charge through the reversed biased junctions forming the drain regions of the pass transistors connected to the buffer gates. Also this circuit would be susceptible to potential accumulated loss of charge due to charge sharing with nodes internal to the pass network core. To avoid these restrictions two weak feedback transistors are added to each buffering circuit as illustrated in Figure 2. This forms a latch which the NMOS pass transistor network must overdrive to toggle. The weak PMOS feedback device also overcomes the threshold voltage drop across the NMOS pass network, thus avoiding potential high current draw by the first inverter in the buffer.

The enable-disable model is subject to race conditions as is any asynchronous sequential circuit. Whether a circuit is critical race free or not is determined by the state assignment.

3 STT Design Procedure

In general, critical race conditions can be avoided by state assignments in which transition paths between states are disjoint. Disjoint state assignments are in general known as single transition time (STT) state assignments [3].

In the following theorem, let the term $\rho_s I_p(x)$ where $x \in [0,1]$ in the expression for Y_i represents a series of pass transistors which are qualified by ρ_s and I_p to pass x to the buffer input.

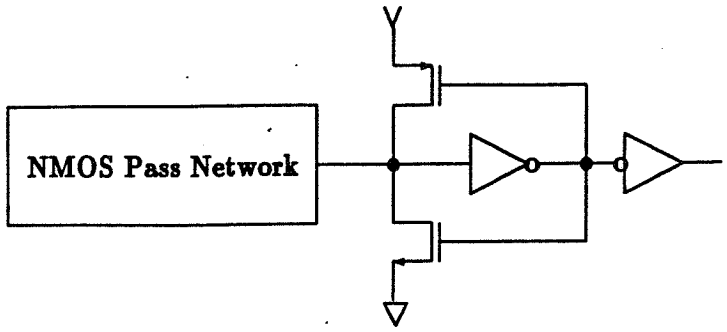


Figure 2: Buffer circuit with weak feedback devices.

Theorem 1 *Let s be the states of a transition path and let ρ_s partition the states of the transition path under I_p where Y_i must transition from a $0 \rightarrow 1$ or a $1 \rightarrow 0$. If Y_i contains $\rho_s I_p(1)$ or $\rho_s I_p(0)$ respectively, then Y_i will properly specify the state transitions of ρ_s.*

Proof If y_i is not required to change in the transition within s, that is y_i transitions $0 \rightarrow 0$ or $1 \rightarrow 1$, then $\rho_s I_p(1)$ or $\rho_s I_p(0)$ need not appear in the expression for Y_i since a high impedance presented at the input of the buffer does not change Y_i. Therefore, the absence of $\rho_s I_p(x)$ does not impact Y_i.

If y_i must transition $0 \rightarrow 1$ or $1 \rightarrow 0$, then Y_i must equal 1 or 0 respectively in the entire transition path of s. If $\rho_s I_p(1)$ or $\rho_s I_p(0)$ appears in Y_i, then Y_i is properly specified in all the states of the transition path of s. If all next state variables satisfy these conditions, the circuit will transition to the correct stable state. □

Another way this operation can be viewed is that partitioning variables partition transition paths and guarantee critical race free operation [9]. Partition variables must not change during transition from the unstable to the stable state. The conditions of Theorem 1 specify that only non-partitioning variables (those that change state) are excited to change; all others remain unchanged and hence the operation is guaranteed to be critical race free.

The following is a general design equation.

$$Y_i = \sum \rho_{disable} I_j(0) + \sum \rho_{enable} I_k(1) \tag{4}$$

Procedure 1 *A formal procedure for the design of an enable-disable realization for each state variable of an asynchronous sequential circuit follows:*

Design procedure.

1. Encode the flow table with an STT state assignment.
2. Identify each ρ partition, ρ_j, under each input variable, I_k, of the flow table.
3. List a table of transitions which cause the state variables to change states.

4. For each state variable y_i,
   (a) For any ρ_j under I_k in which y_i transitions $0 \rightarrow 1$, the term $\rho_j I_k(1)$ appears in Y_i.
   (b) For any ρ_j under I_k in which y_i transitions $1 \rightarrow 0$, the term $\rho_j I_k(0)$ appears in Y_i.
5. Find a covering for each ρ partition and substitute to derive the design equations.

Some asynchronous designs assume that only one input to the state machine is asserted at a time and one input must always be asserted. This is not a practical assumption since essential hazards may exist in any input forming logic. There will be overlaps of the input signals where two inputs will simultaneously be either 1's or 0's. However, the input forming logic can be designed to eliminate either 1–1 crossovers or 0–0 crossovers but not both.

Theorem 2 *When enable-disable model asynchronous sequential circuits are constructed using procedure 1, 0–0 cross over on the inputs is tolerated.*

Proof Consider an arbitrary $\rho_s I_p(x)$ circuit. When $I_p = 0$, the output is in a high impedance state. For a 0–0 cross over, all $I_p = 0$. Therefore all buffer inputs are high impedance. Since a high impedance input causes no change in state, circuit operation is unaffected. Therefore, the inputs may freely have 0–0 cross over. □

Theorem 3 *When enable-disable model asynchronous sequential circuits are constructed using procedure 1 and the inputs to the circuit are constrained to not have 1–1 crossovers, the circuits are free of essential hazards on the inputs.*

Proof Due to the nature of pass transistors, when the gate is driven low the output floats remaining unaffected, provided the effects of leakage currents are overcome. The latching buffer overcomes the effects of leakage currents and state variables will remain unaffected for any length of time that the inputs are all low. If the inputs are then constrained to be free of 1–1 cross over, essential hazards on the inputs have been eliminated for the circuit. □

Essential hazards have in the past been eliminated by either adding or limiting delay at selected points in a circuit [2]. The floating nature of the pass transistor can be viewed as adding a delay greater than the 0–0 crossover time.

Critical race free state assignments of Tracey [3], Liu [1] and Tan [4] are examples of STT state assignments. The following is an example of an asynchronous state machine designed with the Liu state assignment. The flow table is derived from Table 3.13 of [5].

Example 1 *Using design procedure 1, design the enable and disable pass networks to implement the flow table listed in Table 3.*

	I_1	I_2	I_3	y_1	y_2	y_3	y_4
A	A	B	A	0	0	0	0
B	C	B	E	1	0	0	1
C	C	F	C	1	0	1	0
D	D	G	C	1	1	0	0
E	E	F	E	0	0	1	1
F	D	F	E	1	1	1	1
G	G	G	A	0	1	0	0

Table 3: Flow table with Liu state assignment.

The flow table has been given a Liu state assignment covering the following ρ partitions under I_1.

$$\begin{aligned} \rho_a &= A;BCDEFG \\ \rho_{bc} &= BC;ADEFG \\ \rho_{df} &= DF;ABCEG \\ \rho_e &= E;ABCDFG \\ \rho_g &= G;ABCDEF \end{aligned} \tag{5}$$

Under I_2, there are three ρ partitions.

$$\begin{aligned} \rho_{ab} &= AB;CDEFG \\ \rho_{cef} &= CEF;ABDG \\ \rho_{dg} &= DG;ABCEF \end{aligned} \tag{6}$$

Under I_3, there are also three ρ partitions.

$$\begin{aligned} \rho_{ag} &= AG;BCDEF \\ \rho_{bef} &= BEF;ACDG \\ \rho_{cd} &= CD;ABEFG \end{aligned} \tag{7}$$

The enable-disable model requires paths in the enable circuit to allow $0 \rightarrow 1$ transitions and paths in the disable circuits to allow $1 \rightarrow 0$ transitions. Table 4 contains a summary of the transitions from the flow table. For Y_1, the enable circuit must sense $I_2 \rightarrow 1$ while the circuit is in state S_a or S_e. The disable circuit for Y_1 requires a path to sense $I_2 \rightarrow 1$ when the circuit is in state S_d to bring the machine to state S_g and must sense $I_3 \rightarrow 1$ when the circuit is in state S_b or S_f.

The enable circuits can now be formed by covering the $0 \rightarrow 1$ transitions.

$$\begin{aligned} Y_1 &= \rho_{ab}I_2(1) + \rho_{cef}I_2(1) \\ Y_2 &= \rho_{cef}I_2(1) \\ Y_3 &= \rho_{bc}I_1(1) + \rho_{bef}I_3(1) + \rho_{cd}I_3(1) \\ Y_4 &= \rho_{ab}I_2(1) + \rho_{cef}I_2(1) \end{aligned} \tag{8}$$

The disable circuits can then be formed by decoding the transition paths causing $1 \rightarrow 0$

	$0 \rightarrow 1$	Input	$1 \rightarrow 0$	Input
Y_1	$A \rightarrow B$	I_2	$B \rightarrow E$	I_3
	$E \rightarrow F$	I_2	$D \rightarrow G$	I_2
			$F \rightarrow E$	I_3
Y_2	$C \rightarrow F$	I_2	$D \rightarrow C$	I_3
	$E \rightarrow F$	I_2	$F \rightarrow E$	I_3
			$G \rightarrow A$	I_3
Y_3	$B \rightarrow C$	I_1	$F \rightarrow D$	I_1
	$B \rightarrow E$	I_3		
	$D \rightarrow C$	I_3		
Y_4	$A \rightarrow B$	I_2	$B \rightarrow C$	I_1
	$C \rightarrow F$	I_2	$F \rightarrow D$	I_1

Table 4: Summary of transitions for the Liu state assignment.

transitions in the state variables. Then the complete design equations are

$$\begin{aligned} Y_1 &= \rho_{ab} I_2(1) + \rho_{cef} I_2(1) + \rho_{bef} I_3(0) + \rho_{dg} I_2(0) \\ Y_2 &= \rho_{cef} I_2(1) + \rho_{cd} I_3(0) + \rho_{bef} I_3(0) + \rho_{ag} I_3(0) \\ Y_3 &= \rho_{bc} I_1(1) + \rho_{bef} I_3(1) + \rho_{cd} I_3(1) + \rho_{df} I_1(0) \\ Y_4 &= \rho_{ab} I_2(1) + \rho_{cef} I_2(1) + \rho_{bc} I_1(0) + \rho_{df} I_1(0) \end{aligned} \tag{9}$$

For implementation it is then necessary to construct the decode circuits for each of the state transition paths. The decode equations for these state transition paths are

$$\begin{aligned} \rho_{ab} &= \overline{y_2}\,\overline{y_3} \\ \rho_{cef} &= y_3 \\ \rho_{bef} &= y_4 \\ \rho_{dg} &= y_2\,\overline{y_3} \\ \rho_{cd} &= y_1\,\overline{y_4} \\ \rho_{ag} &= \overline{y_1}\,\overline{y_4} \\ \rho_{bc} &= y_1\,\overline{y_2} \\ \rho_{df} &= y_1\,y_2 \end{aligned} \tag{10}$$

Substituting the decode equations, the next state design equations become

$$\begin{aligned} Y_1 = {} & \overline{y_2}\,\overline{y_3}\,I_2(1) + \\ & y_3\,I_2(1) + y_4\,I_3(0) + y_2\,\overline{y_3}\,I_2(0) \\ Y_2 = {} & y_3\,I_2(1) + y_1\,\overline{y_4}\,I_3(0) + \\ & y_4\,I_3(0) + \overline{y_1}\,\overline{y_4}\,I_3(0) \\ Y_3 = {} & y_1\,\overline{y_2}\,I_1(1) + y_4\,I_3(1) + \\ & y_1\,\overline{y_4}\,I_3(1) + y_1\,y_2\,I_1(0) \\ Y_4 = {} & \overline{y_2}\,\overline{y_3}\,I_2(1) + y_3\,I_2(1) + \\ & y_1\,\overline{y_2}\,I_1(0) + y_1\,y_2\,I_1(0) \end{aligned} \tag{11}$$

Figure 3 shows the diagram of the circuit implemented with NMOS pass transistor networks.

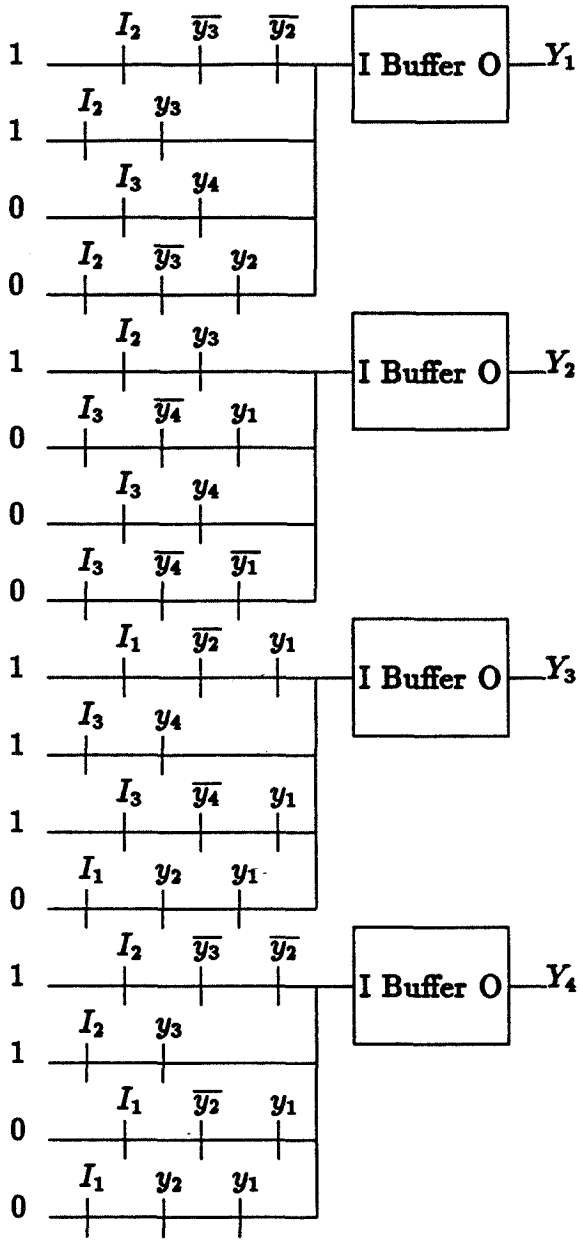


Figure 3: Circuit diagram for Liu state assignment.

4 Self Arbitrated Circuits

The state variable design equation for the enable-disable model is a summation of the enable terms plus a summation of the disable terms.

$$Y_i = \sum \rho_{disable} I_j(0) + \sum \rho_{enable} I_k(1) \tag{12}$$

where ρ_s is the covering for a transition path which leads to the change of a state variable.

The whole transition path is covered for next state variable Y_i by the ρ_s term. In reality, it is not necessary to cover the entire transition path. It is only necessary to cover the portion of the transition path that contains the unstable state. If S_u is the unstable state of a transition path and μ_s is the covering of only the unstable state in the transition path where state variable Y_i must experience either a $0 \rightarrow 1$ or $1 \rightarrow 0$ transition then

$$Y_i = \sum \mu_{disable} I_j(0) + \sum \mu_{enable} I_k(1) \tag{13}$$

Theorem 4 *Equation 13 is sufficient to create the proper asynchronous sequential circuit action.*

Proof Theorem 1 has shown that a high impedance input to the buffer causes the next state to remain equal the present state ($Y_i = y_i$) and no change occurs to state variable Y_i. State table transitions where Y_i does not change need not be covered in the design equation for Y_i as long as a high impedance is presented to the buffer input. Only transitions in Y_i need be effected by the enable and disable circuit. Therefore, only state transitions between unstable and stable states need be accounted for in the design equation.

If state transition $S_u \rightarrow S_s$ requires Y_i to transition from $0 \rightarrow 1$ or $1 \rightarrow 0$, then $\mu_s I_p(1)$ or $\mu_s I_p(0)$ respectively must appear in the equation for Y_i. Let the circuit begin in stable state S_r under I_k. When $I_k \rightarrow 0$ and $I_p \rightarrow 1$ unstable state S_u is entered and Y_i must experience a $0 \rightarrow 1$ transition as the circuit goes from $S_u \rightarrow S_s$. While $I_k = 1$, the term $S_u I_p(1)$ is disabled and passes a high impedance term to the buffer. Moreover, since only total circuit states that are unstable appear in the design equations, the buffer for Y_i has a high impedance input as long as $I_k = 1$. When the input switches to $I_p = 1$, $S_u I_p(1)$ passes a 1 to the input of Y_i forcing it to a 1 state. As soon as the circuit leaves state S_u, term $S_u I_p(1)$ outputs a high impedance state, but Y_i remains 1 to effect the transition for $y_i = 1$. Since the buffer for Y_i has a high impedance input for all the states of the transition path, y_i will assume the proper next state value and a critical race cannot occur. □

The above theorem allows a state assignment to be given that has critical races. The theorem shows that the next state variables are excited only when the circuit enters a total circuit state that is unstable. All other times, present and next state variables are the same and no state transitions occur. In a sense, the circuit is operating as a synchronous, or a self synchronized circuit [7,8]. In this case state variable transitions are arbitrated by the total unstable state.

If the circuit is constrained to allow only one state variable to change at a time as a circuit transitions between states, there will be no races and hence no critical races [5].

Using a series of unit distance state transitions is an acceptable mechanism to achieve state transitions. By using unit distant transitions between states, only one state variable is allowed to change at a time. This type of operation is called non-normal mode [9]. This is not an STT state assignment and therefore the penalty is slower operating circuit.

In the following assignment, each state has a distance 2 from all other states, and hence two successive state variable changes are required for each state transition. Thus the circuit operates at half the speed of an STT assignment.

The state assignment problem for non-normal mode operation is similar to that of STT operation and the critical concept is that transition paths of different k-sets (or transition pairs) must not contain states in common [9]. The transition paths must be partitioned to be disjoint. Transition paths for non-normal operation are more difficult to characterize than the STT assignment. For example, suppose state S_a transitions to S_b, and S_a is coded 000 and S_b 111. One transition path is $000 \rightarrow 001 \rightarrow 011 \rightarrow 111$. Another transition path is $000 \rightarrow 100 \rightarrow 110 \rightarrow 111$. There are a total of 6 unique paths in this case, all equally valid.

The state assignment used here will be a 1-hot-code where state S_i is coded by state $y_i = 1$ and all other state variables are 0.

Definition 3 *Let $[y_i\, y_j\, y_k \ldots y_n]$ represent a state where each element of the set is a state variable equal to 1 and all other state variables are 0.*

For example, $[y_2\, y_3]$ denotes those states where y_2 and y_3 are 1 and all other state variables are 0. If state S_i transitions to S_j, then the transition paths consists of the states $[y_i]$, $[y_i\, y_j]$ and $[y_j]$ such that $[y_i] \rightarrow [y_i\, y_j] \rightarrow [y_j]$. There are three states in the transition path. All states with more than two 1's are not members of any transition path.

Theorem 5 *All transition paths are disjoint for the case where no state is both a successor and predecessor state.*

Proof In a transition $S_i \rightarrow S_j$, S_i is the predecessor state of S_j and S_j is the successor state of S_i. The transition path for transition $S_i \rightarrow S_j$ is $[y_i]$, $[y_i\, y_j]$ and $[y_j]$. The state assignment and associated transition paths produce a valid design if the transition path for (S_i, S_j) is disjoint from all other transition paths. Clearly, the only state that is of concern is $[y_i\, y_j]$ and it must be shown that it is a member of only one transition path.

State variable y_j is excited only when a states transitions to S_j, that is $S_k \rightarrow S_j$ or when S_j transitions to some other state $Sj \rightarrow S_k$. In both cases, state $[y_j\, y_k]$ is entered. If $S_k \neq S_i$, then $[y_i\, y_j]$ is not the same state as $[y_j\, y_k]$ and the paths are disjoint. Moreover, since no state is both a successor and predecessor state to some other state, there cannot be a transition $S_j \rightarrow S_i$. Therefore, $[y_i\, y_j]$ is entered only for the transition $S_i \rightarrow S_j$ and no other time and cannot be an element of some other transition path. □

Theorem 6 *Under the conditions of theorem 5 where no state is both a successor and predecessor state, then proper operation occurs with design equation for Y_i as $y_k I_p(1) + y_i y_j(0)$, where S_k is the predecessor state to S_i, and S_j is a successor state of S_i.*

	I_1	I_2	y_1	y_2
A	A	B	1	0
B	-	B	0	1

Table 5: Design procedure flow table.

Proof Consider states S_i and S_k where $S_k \rightarrow S_i$ transitions under input I_p. The design equations for Y_i and Y_k are

$$\begin{aligned} Y_i &= y_k I_p(1) + y_i y_j(0) \\ Y_k &= y_i y_k(0) + \ldots \end{aligned} \tag{14}$$

When the circuit is in S_k, $y_k = 1$ and the circuit is in state $[y_k]$. When I_p is true, then $y_i \rightarrow 1$ and state $[y_i y_k]$ is assumed. When $[y_i y_k]$ is true then $y_i y_k(0)$ forces $Y_k \rightarrow 0$ and $[y_i]$ is entered. By theorem 5, $[y_i y_k]$ belongs only to the transition path of (S_k, S_i) and therefore the transition $S_k \rightarrow S_i$ is properly effected. Transitioning out of S_i occurs in the same manner as transitioning from S_k. When another input is true, $y_i = 1$ causes the circuit to assume $[y_i y_j]$ which forces $[y_i \rightarrow 0]$. □

5 Self Arbitrated Design Procedures

Procedure 2 *The following procedure can be used for the design of pass networks for each state variable of an asynchronous sequential circuit with a handshake operation using the enable-disable model.*

Design Procedure.

1. Encode the flow table with a one-hot-code state assignment
2. For each stable state S_i under I_j, with associated state variable $y_i = 1$,
   (a) For state transition, $S_k \rightarrow S_i$, introduce an enable term $y_k\, I_j(1)$.
   (b) For state transition out of the state, $S_i \rightarrow S_j$, introduces a disable term $y_i\, y_j(0)$.

For a transition from $S_a \rightarrow S_b$ under I_2 shown in the partial flow table of Table 5, design procedure 2 introduces an enable term for S_b such that $Y_2 = y_1\, I_2(1)$. This term will cause the transition $[y_1] \rightarrow [y_1\, y_2]$. The design procedure also introduces the disable term $Y_1 = y_1\, y_2(0)$ into state S_a thus forming a handshake.

The following relaxes the predecessor and successor requirements of Theorem 6.

Theorem 7 *A valid design equation for Y_i is $Y_i = y_j I_p(1) + y_i y_j I_n(0)$, where S_j is both a predecessor and successor state to S_i, such that $S_i \rightarrow S_j$ under I_p and $S_j \rightarrow S_i$ under I_r.*

	I_1	I_2	y_1	y_2
A	A	B	1	0
B	A	B	0	1

Table 6: Scale of two loop flow table.

Proof Assume that the transition path for transition $S_i \rightarrow S_j$ under I_k is $[y_i]$, $[y_i\, y_j]$ and $[y_j]$. Assume that the transition path for transition $S_j \rightarrow S_i$ under I_m is $[y_j]$, $[y_j\, y_i]$ and $[y_i]$. Clearly, $[y_i\, y_j] = [y_j\, y_i]$ and a problem exists.

If the input state is added to the pass implicant $y_i y_j(0)$ to form $y_i y_j I_k(0)$ then the transition path for $S_i \rightarrow S_j$ under I_k is $[y_i]$, $[y_i\, y_j]$ and $[y_j]$ and the transition path for $S_j \rightarrow S_i$ under I_m is $[y_j]$, $[y_j\, y_i]$ and $[y_i]$. As long as I_k and I_m are guaranteed to never be high at the same time, then the transition paths are disjoint. □

Scale-of-two-loops are common in flow tables [10]. If the flow table were altered to introduce a scale of two loop as shown in Table 6, the design procedure 2 would not produce valid circuits since S_a is both a predecessor and successor state of S_b and the transition paths are not disjoint. From Theorem 7, it can be seen that the input variable can be introduced in the design equation to partition the transition paths and hence, scale-of-two-loops do not present a problem.

Procedure 3 *The following procedure can be used to design pass networks for each state variable of an asynchronous sequential circuit using a handshake operation with the enable-disable model.*

Design procedure.

1. Encode the flow table with a one-hot-code state assignment
2. For each stable state S_i under I_j with associated state variable $y_i = 1$,
   (a) For state transition, $S_k \rightarrow S_i$, introduce an enable term $y_k\, I_j(1)$.
   (b) For state transition out of the state, $S_i \rightarrow S_k$, under I_m introduce a disable term $y_i\, y_k\, I_m(0)$.

For a transition from $S_a \rightarrow S_b$ under I_2 shown in the flow table of Table 6, design procedure 3 introduces an enable term for S_b such that $Y_2 = y_1\, I_2(1)$. This term will cause the transition $[y_1] \rightarrow [y_1\, y_2]$. The design procedure also introduces the disable term $Y_1 = y_1\, y_2\, I_2(0)$ into state S_a forming a handshake and causing the transition $[y_1\, y_2] \rightarrow [y_2]$. The transition from $S_b \rightarrow S_a$ under I_1 requires an enable term for S_a such that $Y_1 = y_2 I_1(1)$. This term will cause the transition $[y_2] \rightarrow [y_2\, y_1]$. The design procedure will also introduce the disable term $Y_2 = y_1\, y_2\, I_1(0)$ into state S_b thus forming a unique handshake and causing the transition $[y_2\, y_1] \rightarrow [y_1]$.

Example 2 *Design the enable and disable pass networks to implement the flow table listed in Table 7.*

	I_1	I_2	I_3	y_1	y_2	y_3	y_4	y_5	y_6
A	A	F	D	1	0	0	0	0	0
B	A	B	D	0	1	0	0	0	0
C	C	F	C	0	0	1	0	0	0
D	C	B	D	0	0	0	1	0	0
E	E	B	C	0	0	0	0	1	0
F	E	F	C	0	0	0	0	0	1

Table 7: Flow table with handshake state assignment.

The first step is to assign a one-hot-code to the flow table as shown in Table 7.

Derivation of the design equations can again be understood by studying the flow table. A state variable makes a $0 \rightarrow 1$ transition when entering the state which requires that variable to be asserted. This is accomplished by a term in the design equation qualified by the state from which the circuit is transitioning and the input under which the new state is stable passing a 1. For example, when the machine is in state S_a or S_c, if I_2 is asserted high, the machine will move towards state S_f. The enable terms of the design equation for state variable Y_6 can be written as $Y_6 = y_1 I_2(1) + y_3 I_2(1)$. To guarantee that the machine traverses states $S_a \rightarrow S_f$ such that $[y_1] \rightarrow [y_1\, y_6] \rightarrow [y_6]$ and $S_a \rightarrow S_d$ such that $[y_1] \rightarrow [y_1\, y_4] \rightarrow [y_6]$ when leaving stable state S_a, the disable terms $y_1 y_6 I_2(0)$ and $y_1 y_4 I_3(0)$ are introduced into the design equation for Y_1, thus forming a handshake.

First the enable terms are read from the flow table.

$$\begin{aligned} Y_1 &= y_2 I_1(1) \\ Y_2 &= y_4 I_2(1) + y_5 I_2(1) \\ Y_3 &= y_4 I_1(1) + y_5 I_3(1) + y_6 I_3(1) \\ Y_4 &= y_1 I_3(1) + y_2 I_3(1) \\ Y_5 &= y_6 I_1(1) + \\ Y_6 &= y_1 I_2(1) + y_3 I_2(1) \end{aligned} \tag{15}$$

The entire design equations with the disable terms are

$$\begin{aligned} Y_1 &= y_2 I_1(1) + y_1 y_6 I_2(0) + y_1 y_4 I_3(0) \\ Y_2 &= y_4 I_2(1) + y_5 I_2(1) + y_2 y_1 I_1(0) + y_2 y_4 I_3(0) \\ Y_3 &= y_4 I_1(1) + y_5 I_3(1) + y_6 I_3(1) + y_3 y_6 I_2(0) \\ Y_4 &= y_1 I_3(1) + y_2 I_3(1) + y_4 y_3 I_1(0) + y_4 y_2 I_2(0) \\ Y_5 &= y_6 I_1(1) + y_5 y_2 I_2(0) + y_5 y_3 I_3(0) \\ Y_6 &= y_1 I_2(1) + y_3 I_2(1) + y_6 y_5 I_1(0) + y_6 y_3 I_3(0) \end{aligned} \tag{16}$$

The logical implementation of the circuit is shown in Figure 4.

6 Hardware Bound

A hardware bound count can be established for the design of enable-disable sequential circuits using a handshake state assignment operation. The hardware count, T_i, is the

number of transistors required to build the enable circuits, T_e, plus the disable circuits, T_d, and the buffer circuits, T_b.

$$T_t = T_e + T_d + T_b \tag{17}$$

Using design procedure 2, there are two transistors in each enable term and two transistors in each disable term. There is an enable term and a disable term for every unstable state in the flow table. For a flow table with n states and i inputs, there are at most ni next state entries. If t transistors are required to build the buffer circuit then the hardware bound is such that

$$\begin{aligned} T_t &\leq 2ni + 2ni + nt \\ &\leq n(4i + t) \end{aligned} \tag{18}$$

Using design procedure 3, there two transistors in each enable term and three transistors in each disable term. Again there is an enable term and a disable term for every unstable state in the flow table. The hardware bound is

$$\begin{aligned} T_t &\leq 2ni + 3ni + nt \\ &\leq n(5i + t) \end{aligned} \tag{19}$$

An exact transistor count could be determined for a specific flow table. If there are u unstable states in an n row flow table with i inputs, then for design procedure 2

$$\begin{aligned} T_t &= 2(ni - u) + 2(ni - u) + nt \\ &= 4(ni - u) + nt \end{aligned} \tag{20}$$

For design procedure 3

$$\begin{aligned} T_t &= 2(ni - u) + 3(ni - u) + nt \\ &= 5(ni - u) + nt \end{aligned} \tag{21}$$

7 Summary

The enable-disable model allows efficient implementation of VLSI asynchronous sequential circuits. The circuit for each state variable is composed of three sections. First, the enable network for y_i which arms the circuit to look for input changes cause $0 \rightarrow 1$ transitions in y_i. Second, the disable network which arms the circuit to look for input transitions cause y_i to change from $1 \rightarrow 0$. Third, the buffer circuit which isolates the enable-disable network from the state variable load capacitance, restores the high level and provides a memory function to hold the state information when the pass network is tristated.

STT state assignments are used to provide critical race free operation. Only transitions which require state variable changes need to be covered by paths in the pass network. This eliminates the need to cover all ρ partitions for each state variable. The above procedures could be extended to the output states also. In this case the output states would be available from a buffer like the next state variables.

Two design procedures using a handshake operation with a 1-hot-code state assignment were presented and an example was given. The hardware bounds for both procedures were

established. The handshake code is free of critical races and hazards. The handshaking arbitrates differences in delays and forces the circuit through a unique sequence of states. No electronic circuit design constraints exist for the circuit. Since the circuit is based on a nonnormal operation, the handshake code will cause the circuits to function at half the speed of the STT state assignment based circuits.

The cross over constraint on the inputs was addressed by Theorem 2, which introduces a method to allow 0–0 cross overs. The 1–1 cross over can be solved in two ways. First, the flow table could be expanded to introduce columns which incorporate 1–1 cross overs. Second, 1–1 cross over could be eliminated in the input circuitry utilizing cross coupled Nor gates to logically eliminate 1–1 cross over.

References

[1] C. Liu, "A State Variable Assignment Method for Asynchronous Sequential Switching Circuits", JACM, Vol. 10, Apr. 1963, pp. 209-216

[2] C. Roth, *Fundamentals of Logic Design*, 3rd Ed., St. Paul, Minn., West Publishing, 1985, Unit 23-27

[3] J. Tracey, "Internal State Assignments for Asynchronous Sequential Machines", IEEE Transactions on Electronic Computers, Vol. EC-15, Aug. 1966, pp. 551-560

[4] C. Tan, "State Assignments for Asynchronous Sequential Machines", IEEE Transactions on Computers, Vol. C-20, No. 4, April 1971, pp. 382-391

[5] S. Unger, *Asynchronous Sequential Switching Circuits*, New York, NY, Wiley-Interscience, 1969

[6] G. Maki and D. Sawin, "Fault Tolerant Asynchronous Sequential Machines", IEEE Transactions on Computers, Vol. C-23, pp. 651-657, July, 1974

[7] H. Chuang and S. Das, "Synthesis of Multiple Input Change Asynchronous Machines using controlled Excitation and Flip-flops", IEEE Transactions on Computers, vol 22, December 1973, pp. 1103-1109

[8] H. Chuang, "Fail Safe Asynchronous Machines with Multiple Input Changes", IEEE Transactions on Computers, June 1976, pp. 637-642

[9] G. Maki and J. Tracey, "A State Assignment Procedure for Asynchronous Sequential Circuits", IEEE Transactions on Computers, June 1971, vol C-20, pp. 666-668

[10] L. Hollaar, "Direct Implementation of Asynchronous Control Units", IEEE Transactions on Computers, vol C-31, Dec. 1982, pp. 1133-1141

This work was supported in part by NASA under Contract NAGW-1406 and by the Idaho State Board of Education under Research Grant # 87-009.

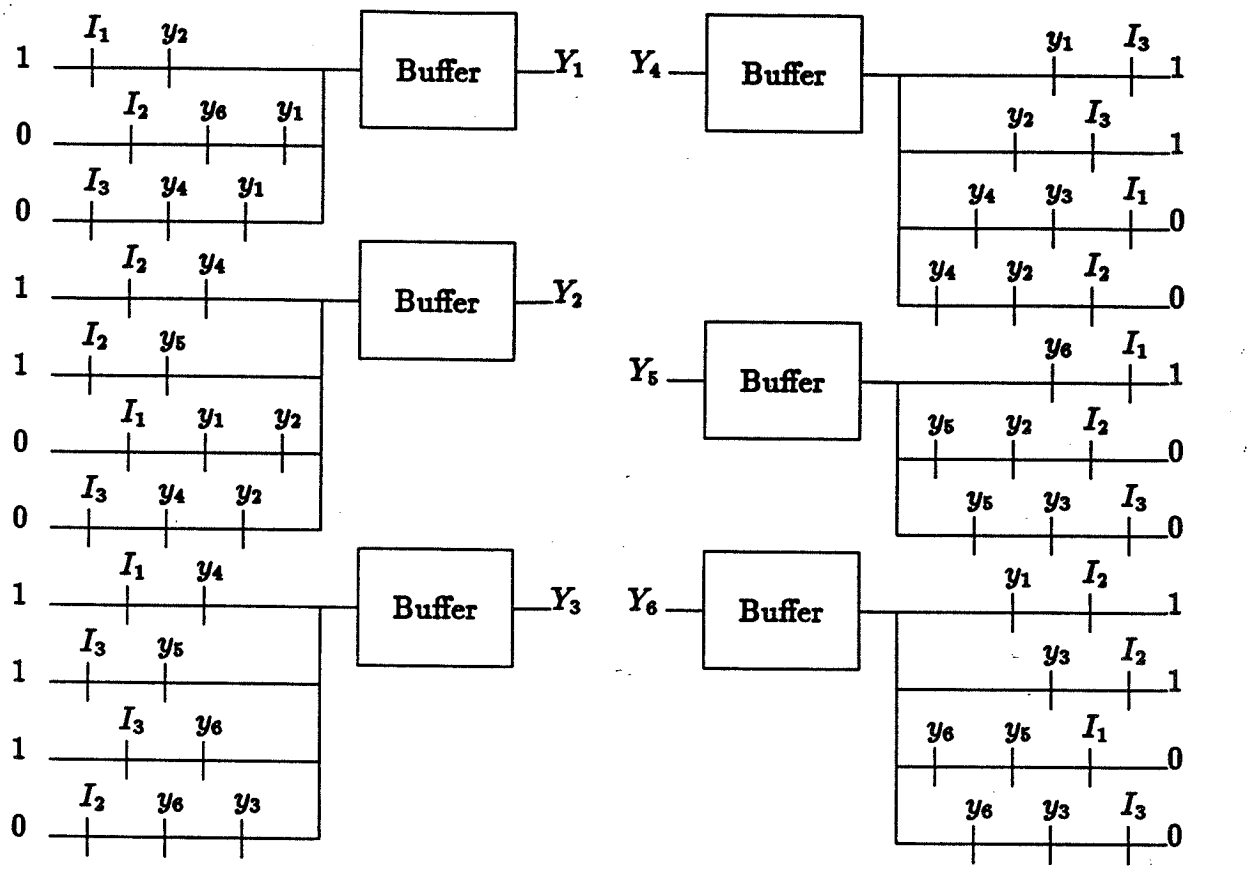


Figure 4: Circuit diagram for handshake state assignment.

Using Advanced Microelectronic Test Chips to Qualify ASIC'S for Space

M. G. Buehler, B. R. Blaes, and Y-S. Lin
Jet Propulsion Laboratory, MS 300-329
California Institute of Technology
Pasadena, California 91109

Qualification procedures for complex integrated circuits are being developed under a U. S. government program known as QML, Qualified Manufacturing Lines. This effort is focused at circuits designed by IC manufacturers and has not addressed application specific ICs (ASICs) designed at system houses. The qualification procedures described here are intended to be responsive to the needs of system houses who design their own ASICs and have them fabricated at Silicon foundries.

A particular focus of this presentation will be the use of the TID (Total Ionizing Dose) Chip to evaluate CMOS foundry processes and to provide parameters for circuit simulators. This chip is under development as a standard chip for qualifying the total dose aspects of ASICs. The benefits of standardization are that the results will be well understood and easy to interpret.

Data is presented and compared for 1.6-μm and 3.0-μm CMOS. The data shows that 1.6-μm CMOS is significantly harder than 3.0-μm CMOS. Two failure modes are explored: (a) the radiation-induced degradation of timing delays and (b) radiation-induced leakage currents.

In order to focus this effort, five critical questions have been formulated:

- WHAT ARE THE QUALIFICATION PROCEDURES FOR ASICs?
- HOW GOOD ARE RADIATION-HARDENED CIRCUIT SIMULATORS?
- DO LABORATORY IRRADIATIONS ACCURATELY SIMULATE SPACE?
- WHAT ARE THE RADIATION-HARDENED CIRCUIT DESIGN RULES?
- CAN NON RAD-HARD CMOS ASICs BE USED IN SPACE?

Initial answers to these questions are given at the end of this presentation.

An ASIC qualification scheme used for the fabrication of an ASIC Direct Memory Access Controller (DMAC) is illustrated in Figure 1. A set of test chips were placed next to the ASICs in order to verify the quality of the fabrication process. These chips analyze total dose hardness, single-event upset sensitivity, metal interconnect wire and contact reliability, and manufacturing yield.

The physics of total dose and single-event upset (SEU) radiation is illustrated in Figure 2 for the case of a static memory cell. The total dose effects are seen to introduce positive charge in the oxide and traps at the oxide-silicon interface. The Cosmic Rays create a

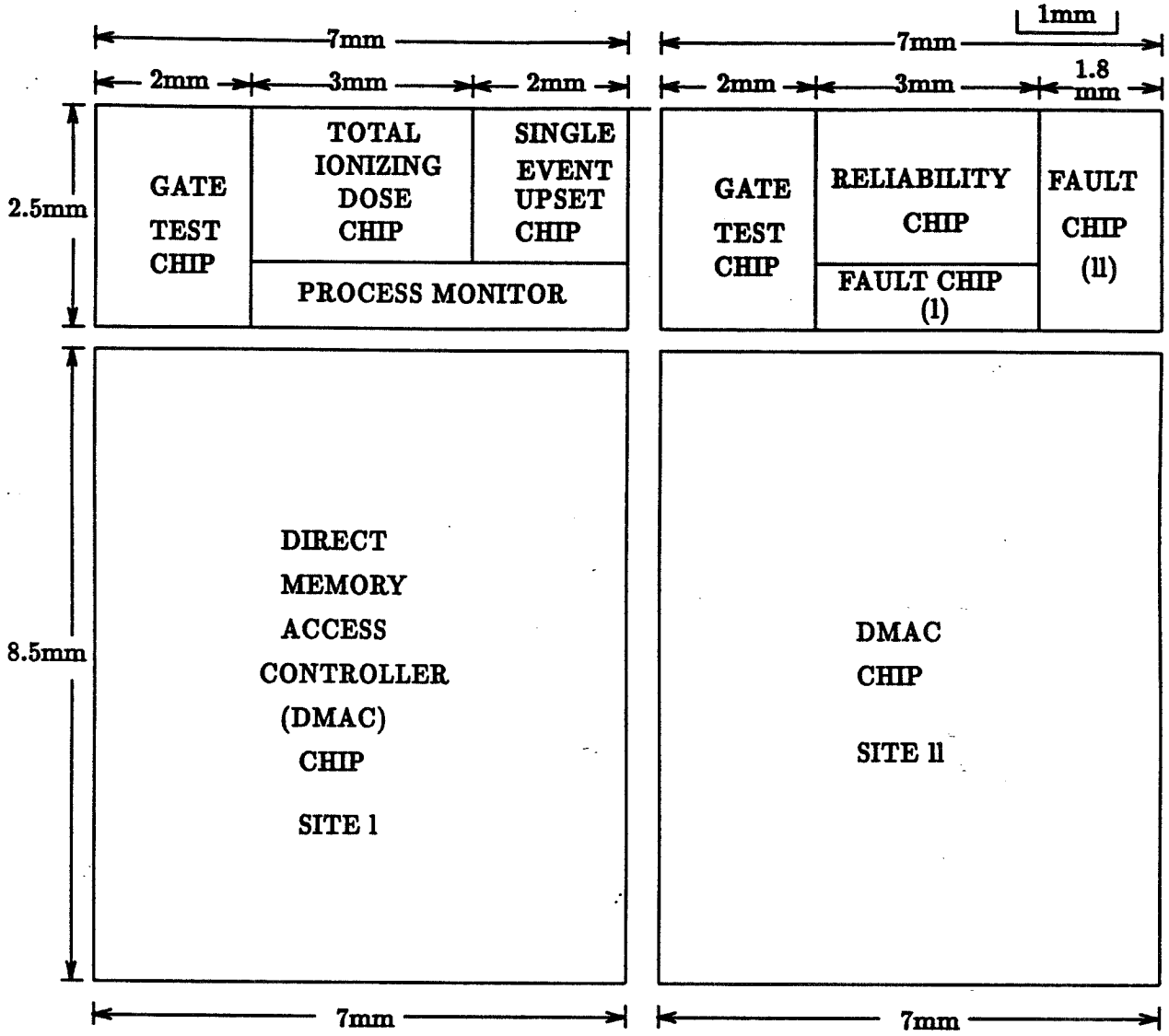


Figure 1: Microelectronic Test Chips for Space Qualified ASIC's

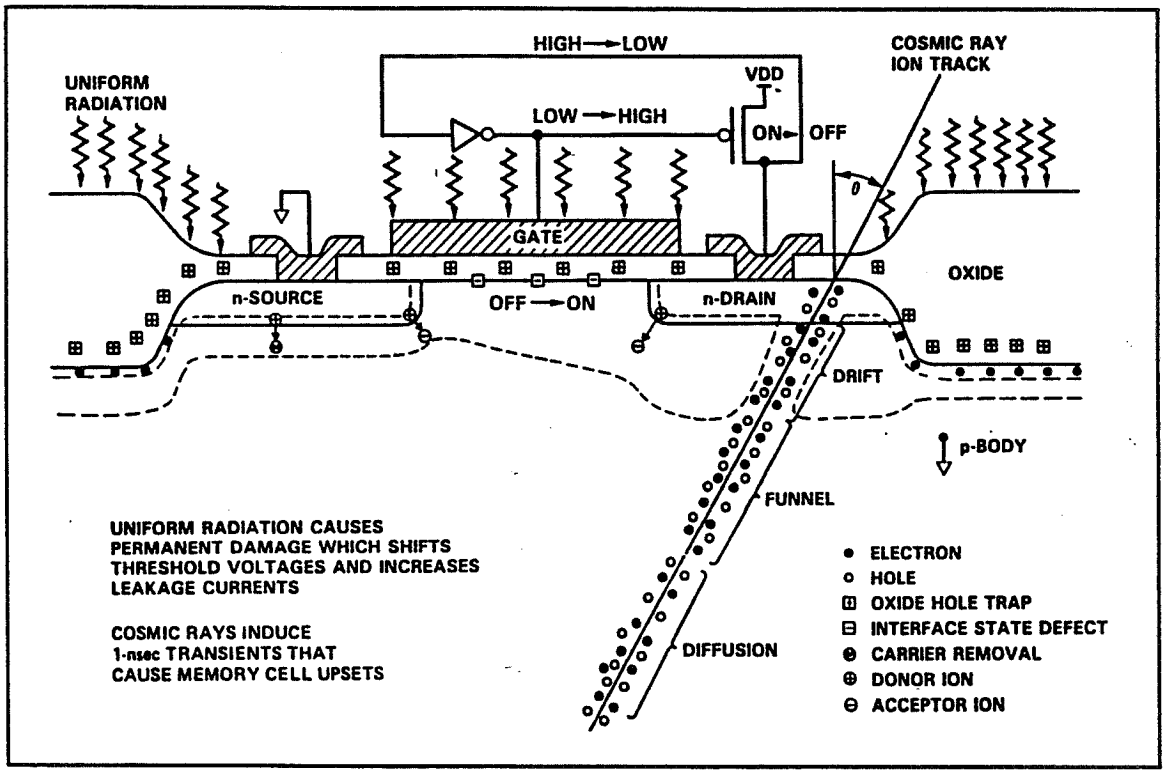


Figure 2: Total Dose and Cosmic Ray Effects in a Static Memory Cell

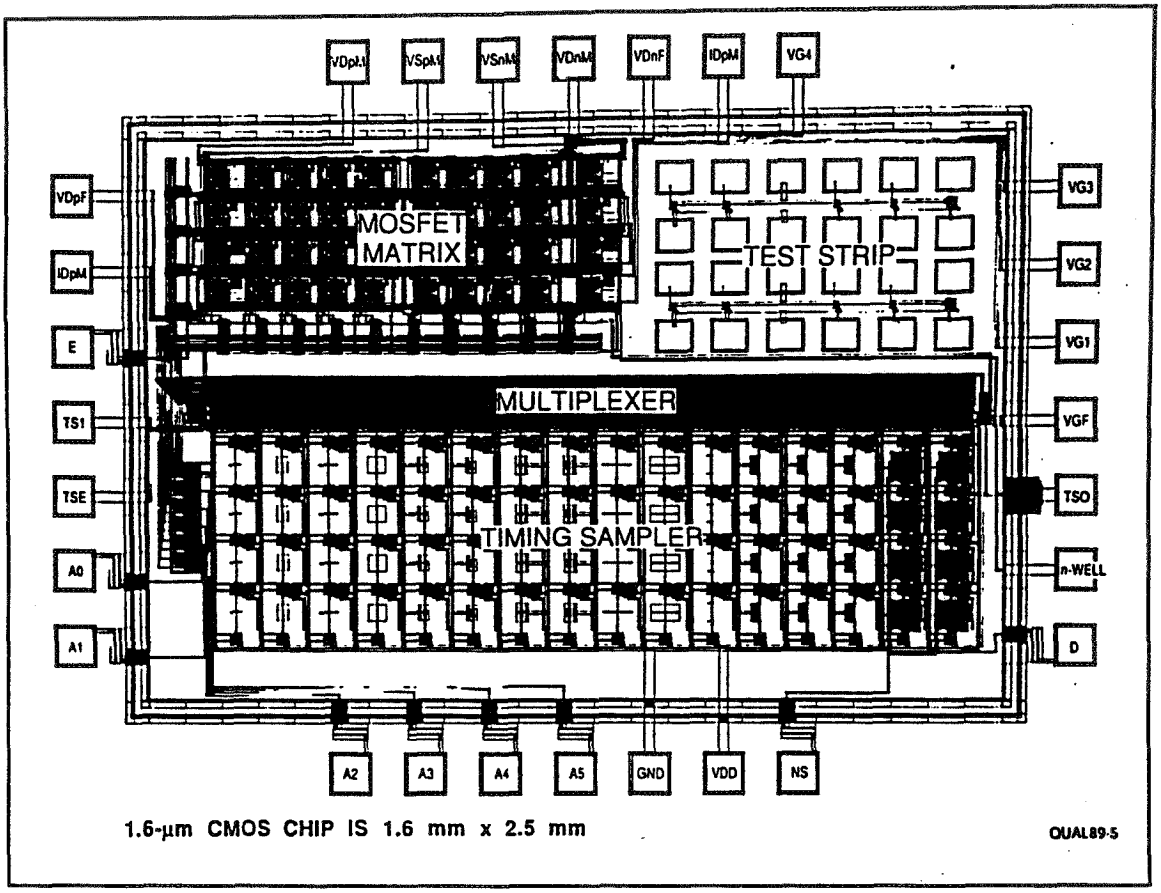


Figure 3: Total Ionizing Dose (TID) Chip

hole-electron plasma in the silicon which temporarily shorts out the struck junction and can cause the cell to change state.

The TID chip, shown in Figure 3, is packaged in a 28-pin DIP. The chip contains a MOSFET matrix [1] with up to 64 MOSFETs designed with a variety of geometries. The matrix contains both gate-oxide, field-oxide, and closed geometry MOSFETs. The chip also contains a Timing Sampler circuit [2] with 16 timing chains for determining inverter-pair delays at different capacitance loads.

p-MOSFET Matrix test results [2] are shown in Figure 4 for V_{TO}, KP, ΔW, and ΔL as a function of Cobalt-60 dose and anneal time. These results show a significant gate bias dependence for $V_{TO}(-1.37\ V/Mrad(Si))$ and $KP(-2.5\ \mu A/V^2 Mrad(Si))$. This is attributed to the build-up of positive charge in the oxide when gates are biased in the ON state. For gates biased in the OFF state, positive oxide charge and positive donor interface states cause even stronger shifts in $V_{TO}(-3.45\ V/Mrad(Si)))$ and $KP(-25.5\ \mu A/V^2 Mrad(Si))$.

n-MOSFET Matrix test results [2] are shown in Figure 5 for V_{TO}, KP, ΔW, and ΔL as a function of Cobalt-60 dose and anneal time. The shifts are due to the build up of positive

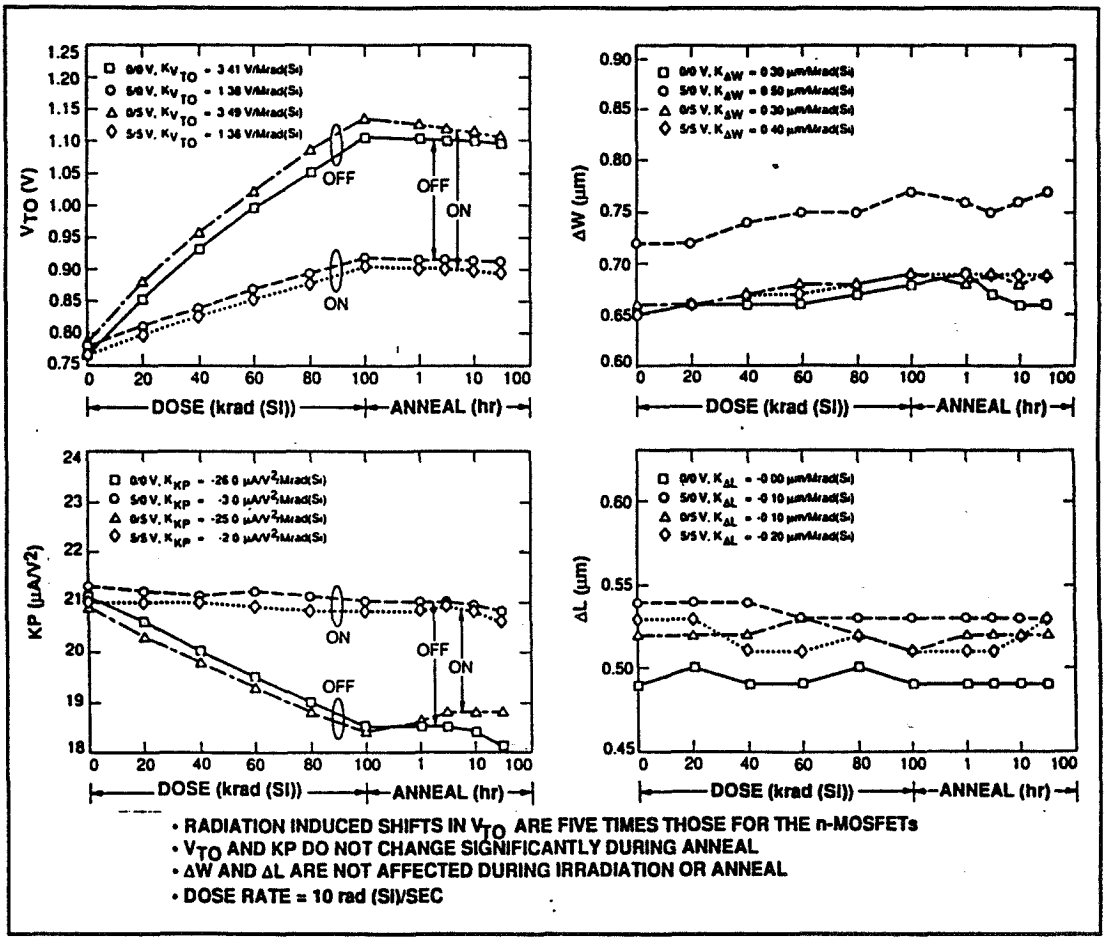


Figure 4: p-MOSFET Results (1.6-μm n-Well CMOS)

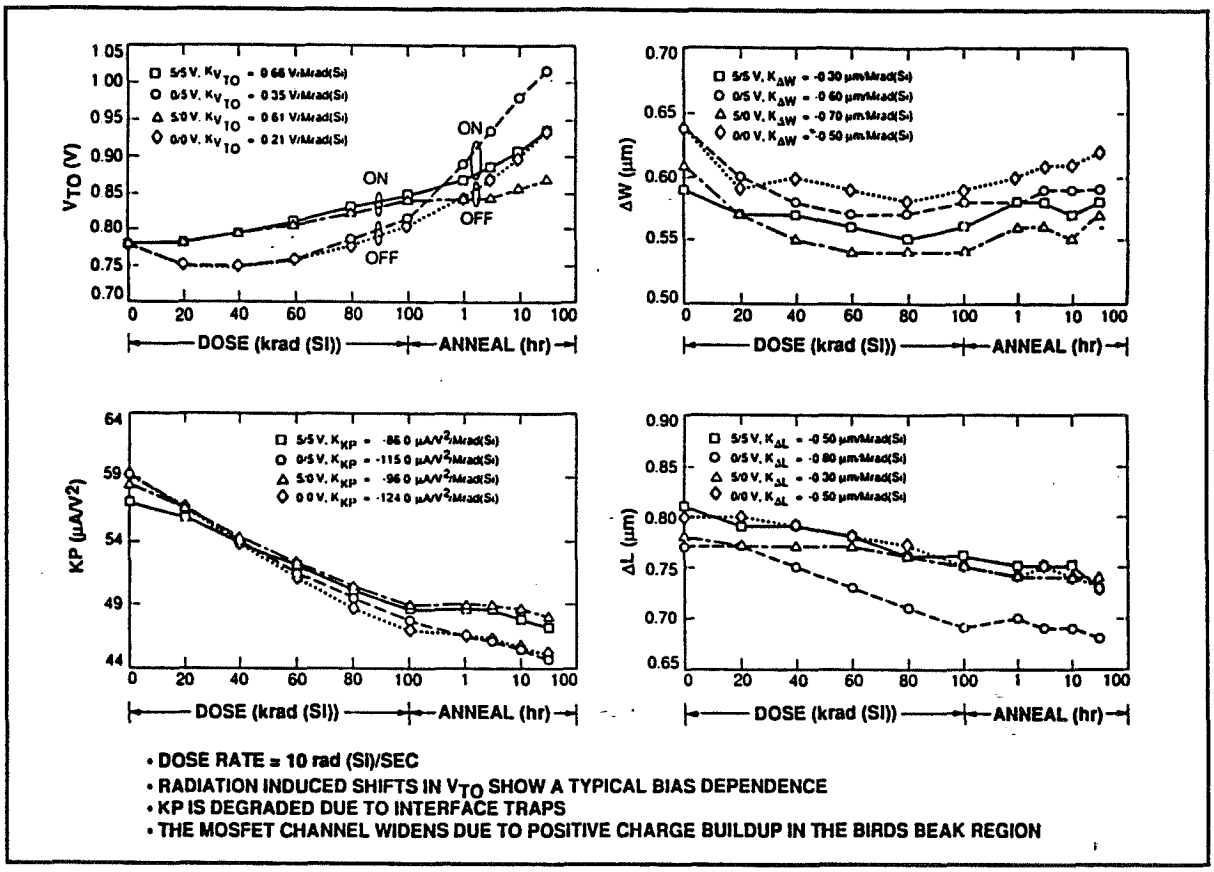


Figure 5: n-MOSFET Results (1.6- μ m n-Well CMOS)

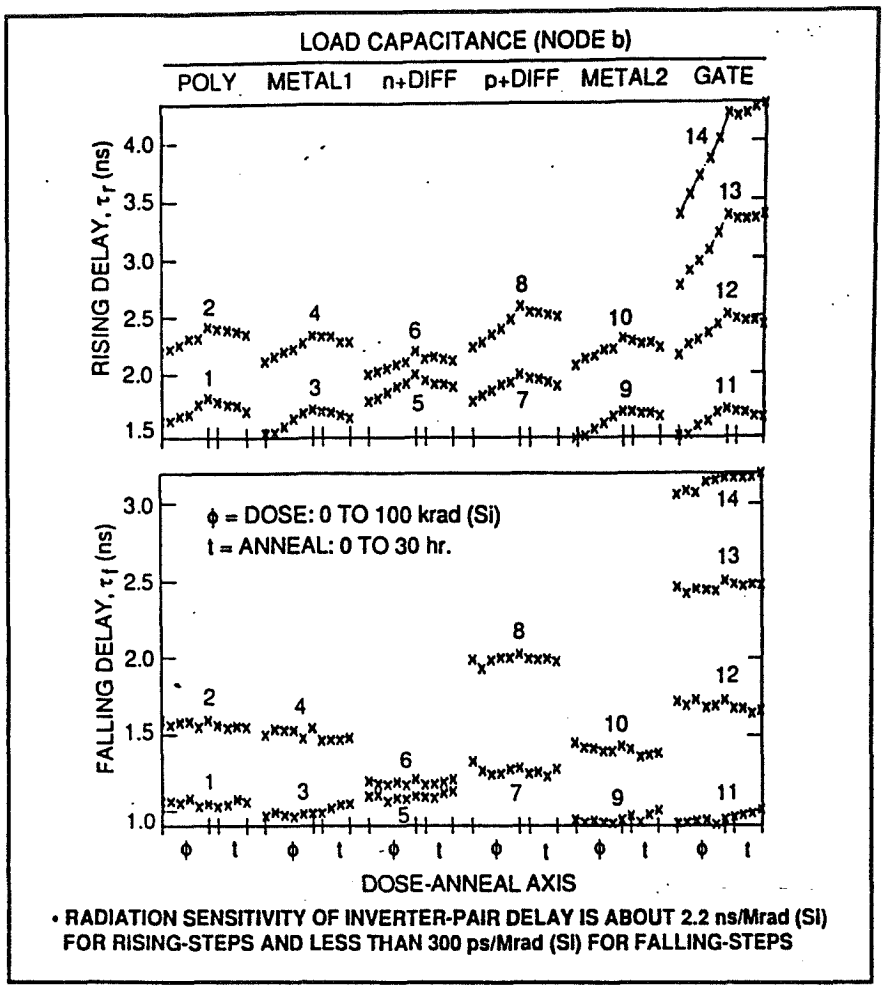


Figure 6: Timing Sampler Delay (1.6-μm n-Well CMOS)

oxide charge and negative acceptor interface states. The opposite signs of these charges cancels their effect on V_{TO}. The negative acceptor interface states cause a significant shift in $KP(-105\mu A/V^2 Mrad(Si))$.

Timing Sampler test results [2] are shown in Figure 6 for a variety of capacitive loads: Polysilicon on field oxide (1,2), Metal 1 (3, 4), Metal 2 (9, 10), n+Diffusion (5, 6), p+Diffusion (7, 8), Polysilicon on gate oxide (11, 12, 13, 14). The results are shown for Cobalt-60 dose up to 100 krad(Si) and an annealing time up to 30 hours. The rising-delay results are dominated by p-MOSFETs which pull-up the loaded nodes. The rising-delay shift with radiation is 2.32 $ns/Mrad(Si)$. These results are explained in terms of the radiation-induced shifts in the DC p-MOSFET parameters, V_{TO}, KP, ΔW, and ΔL. The falling-delay results are dominated by n-MOSFETs which pull-down the loaded nodes. The falling-delay shift with radiation is very small being 0.32 $ns/Mrad(Si)$. This result is unexpected and is not explained by radiation-shifts in the DC MOSFET parameters.

Node Current	Damage Factors
$I_D = C\frac{dV}{dt}$	$K_{V_{T0}} = \frac{\delta V_{T0}}{\delta \phi} \mid_{\phi_0}$
Mosfet Saturation Region Current	$K_{KP} = \frac{\delta KP}{\delta \phi} \mid_{\phi_0}$
$I_D = \frac{\beta(V_G - V_T)^2}{2}$	$K_{\Delta W} = \frac{\delta \Delta W}{\delta \phi} \mid_{\phi 0}$
$\beta = \frac{KP(W-\Delta W)}{L-\Delta L}, KP = \mu 0^{C_{ox}}$	$K_{\Delta L} = \frac{\delta \Delta L}{\delta \phi} \mid_{\phi_0}$
Rising Delay	$K_{\gamma/\phi} = \frac{\delta \gamma}{\delta \phi} \mid_{\phi_0}$
$\tau_\gamma = \frac{2V_{T_{ns}}(C_{b0}+C_{bs})}{\beta_{pb}(V_{DD}-V_{T_{pb}})^2}$	$K_{\gamma/C} = \frac{\delta \gamma}{\delta C} \mid_{C_0}$
Falling Delay	$K_{\frac{\gamma}{\phi C}} = \frac{\delta^2 \gamma}{\delta \phi \delta C} \mid_{\phi_0, C_0}$
$\tau_f = \frac{2V_{T_{ps}}(C_{b0}+C_{bs})}{\beta_{nb}(V_{DD}-V_{T_{nb}})^2}$	

Table 1: Model Results

Interface trapping effects are being proposed to explain these results.

The model, which couples results from the MOSFET Matrix and Timing Sampler, is shown in Figure 1. The Damage Factors used in this analysis are also defined in this figure. Damage Factors for 1.6-μm and 3.0-μm CMOS are listed in Figure 2. A comparison of the damage factors reveals that the 1.6-μm CMOS is much more radiation resistant than the 3.0-μm CMOS.

Another ASIC failure mode is radiation-induced leakage currents. The data, shown in Figure 7, were taken from four MOSFETs with different geometries. The results reveal that side-wall charging and acceptor interface states are responsible for the leakage. That is, the leakage scales with channel length, L, and is independent of channel width, W.

Some total dose test issues are outlined in Figure 7. The interpretation of ground test is limited by the following: (a) The particles used in ground tests are often different from space particles. (b) The dose rate of the ground tests is much higher than in space. Both of these issues argue for space tests that can verify the correctness or limitations of ground tests.

The conclusions, listed in Figure 3, indicate that the radiation-induced increase in rising delays can be modeled by radiation-induced shifts in MOSFET parameter. However, radiation-induced falling-delay degradation cannot be explained in terms of radiation-induced MOSFET parameter shifts. Interface-state trapping effects are suggested as the explanation of this effect.

Data was presented that shows that 1.6-μm CMOS is significantly harder than 3.0-μm CMOS. Leakage currents were identified as being due to leakage along the side-walls of the MOSFETs.

Responses to the critical questions posed at the beginning of this presentation are given in Figure 4. As the CMOS technology shrinks, mother nature seems to be cooperating by providing more radiation tolerant processes. However, caution must be exercised as trapping and side-wall charging effects could become more important at the smallest feature

TID CHIP RESULTS (1.6 – μm n-WELL CMOS)

Mossfet Type	State During Dosing	V_{T0} V	K_{VT} V/D	KP_0 $\mu A/V^2$	K_{kp} $\mu A/V^2D$	$\Delta W0$ μm	$K_{\Delta W}$ $\mu m/D$	ΔL_0 μm	$K_{\Delta L}$ $\mu m/D$	Delay	$\tau 00$ ns	$K_{\tau/\phi}$ ns/D	$K_{\frac{\tau}{\phi c}}$ $ns/pF*D$
n	ON	0.78	0.64	58.5	-91.0	0.62	-0.5	0.79	-0.4	Falling	0.90	0.30	0.18
n	OFF	0.78	0.28	58.5	-119.5	0.62	-0.55	0.79	-0.65	Rising			
p	ON	-0.78	-1.37	21.1	-2.5	0.67	0.45	0.52	-0.15	Falling			
p	OFF	-0.78	-3.45	21.1	-25.5	0.67	0.3	0.52	0	Rising	1.50	2.32	5.65

D = Mrad (Si), During Dosing TSI = Low, Run No. M91L

TID CHIP RESULTS (3.0 – μm n-WELL CMOS)

Mossfet Type	State During Dosing	V_{T0} V	K_{VT} V/D	KP_0 $\mu A/V^2$	K_{kp} $\mu A/V^2D$	$\Delta W0$ μm	$K_{\Delta W}$ $\mu m/D$	ΔL_0 μm	$K_{\Delta L}$ $\mu m/D$	Delay	$\tau 00$ ns	$K_{\tau/\phi}$ ns/D	$K_{\frac{\tau}{\phi c}}$ $ns/pF*D$
n	ON	0.94	-2.35	56.3	-434.7	1.32	0	1.40	0	Rising			
n	OFF	0.94	-2.54	56.3	-583.8	1.32	0	1.40	0	Falling	3.1	33.4	10.6
p	ON	-0.88	-14.40	19.2	0	2.40	0	1.43	0	Rising	3.94	23.8	37.7
p	OFF	-0.88	-14.03	19.2	0	2.40	0	1.43	0	Falling			

D = Mrad (Si), During Dosing TSI = High, Run No. M84M(MOSSFET) and M88F(Delay)

Table 2: CMOS DAMAGE FACTORS

- Dose-Shielding Codes Over Predict Total Dose by More Than Ten Times
- Correlation of Ground Radiation Sources using Gammas and X-Rays with Space Particles (Electrons and Protons) Has Not Been Established
- Ground Test Acceleration Factor Large (e.g. 2E9)
  1. ARACOR (X-RAY) = 6000 rad (Si)/sec
  2. ASTM (COBALT 60) = 100 rad (si)/sec
  3. SPACE(HIGH) = 3 milli-rad (Si)/sec
  4. SPACE(LOW) = 3 micro-rad (Si)/sec
- Test Conditions (Dose Rate and Gate Bias) Affect Parameter Degradation Due to Device Annealing

Table 3: TOTAL DOSE TEST ISSUES

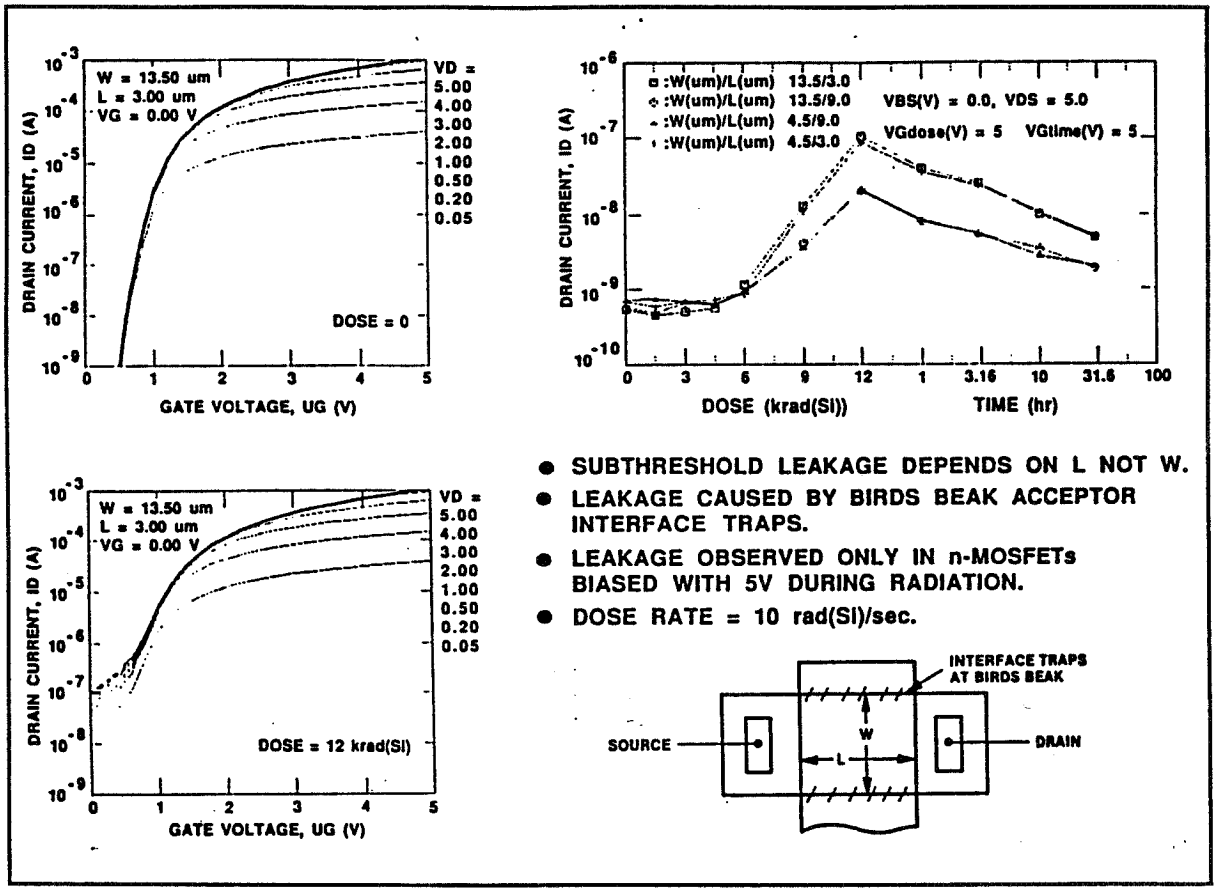


Figure 7: n-MOSFET Subthreshold Leakage Current

What Are The Qualification Procedures For ASIC's ?
Prior to fabrication, circuits are designed with circuit simulators to insure circuits meet performance goals. After fabrication, circuit tests are supplemented with tests from test chips which are fabricated along with the custom chips. Such tests can reveal the presence of hazards not simulated and often due to processing flaws

How Good Are Radiation-Hardened Circuit Simulators ?
Circuit timing simulations are based on radiation-induced shifts in DC MOSFET parameters, and accurate capacitance and resistor values. The simulators are only as good as the parameters put into them. Currently the simulators do not account for important second order effects such as trapping effects.

Do Laboratory Irradiations Accurately Simulate Space ?
Ground tests are highly accelerated and use single, mono-energetic particles. These particles are different from those found in space. Thus great care must be exercised in predicting circuit space performance from ground tests. Data from space is extremely limited. Only five experiments have been devoted to the testing of electronic components in space. More space test results are needed on current technologies to ensure their survivability in space.

What Are The Radiation-Hardened Circuit Design Rules ?
Are closed geometry MOSFETS required to eliminate leakage ? ...They tale too much for space ! Given a radiation scenario, what are the allowable timing margins? What are the rules for latch-up? Are cross strapped latches and memory cells really needed? This takes up room and requires additional processing steps for high value resistors.

Can Non Rad-Hard CMOS ASICs Be Used In Space ?
It appears that the answer is yes but more testing is required to verify the answer.

Table 4: Critical Questions

sizes. Thus a rigorous testing program should be undertaken to evaluate the latest CMOS processes. Such tests must be compared to results from space so as to validate the ground tests.

Test chips can play a vital role in providing first principle answers that can be used to evaluate processes and to provide ASIC design parameters. The TID chip is being fabricated at a number of CMOS foundries in preparation for its use in CMOS foundry qualification.

References

[1] B. R. Blaes, M. G. Buehler, and Y-S Lin, *A CMOS Matrix for Extracting MOSFET Parameters Before and After Irradiation,* IEEE Trans. Nuclear Science, NS-35,1529-1535 (1988).

[2] B. R. Blaes, M. G. Buehler, and Y-S Lin, *Radiation Dependence of Inverter Propagation Delay From Timing Sampler Measurement,* IEEE Trans. Nuclear Science, NS-36,(1989).

Acknowledgment

The authors are indebted to MOSIS of the Information Sciences Institute for brokering the fabrication of the TID Chips. The research described in this paper was carried out by the JET Propulsion Laboratory, California Institute of Technology, and was sponsored by the Defense Advanced Research Projects Agency and the National Aeronautics and Space Administration.

Real Time SAR Processing

A. B. Premkumar & J. E. Purviance
Department of Electrical Engineering
University of Idaho
Moscow, Idaho 88343

Abstract- **A simplified model for the SAR imaging problem is presented. The model is based on the geometry of the SAR system. Using this model an expression for the entire phase history of the received SAR signal is formulated. From the phase history, it is shown that the range and the azimuth coordinates for a point target image can be obtained by processing the phase information during the intrapulse and interpulse periods respectively. An architecture for a VLSI implementation for the SAR signal processor is presented which generates images in real time. The architecture uses a small number of chips, a new correlation processor, and an efficient azimuth correlation process.**

1 Introduction

Radar imaging of a scene requires processing the signals reflected off point targets comprising the scene. The received signals contain the two dimensional image information of the image and processing the two dimensional information is time consuming. Hence, imaging in real time becomes difficult. The basic processing functions to be implemented are convolution, magnitude detection, interpolation and system control. Due to dramatic improvements in fabrication techniques, the VLSI implementation of these functions can now be achieved and it is now possible to do real time imaging on-board the spacecraft. This will have significant impact on the capabilities of on-board signal processing functions allowing for more sophistication, flexibility and reliability.

The basic concept behind any radar system is that it illuminates its targets by transmitting bursts of microwave energy and collects the reflected signals from the targets. The illuminated target is said to be in the *footprint* of the radar beam. Each target in the footprint to be mapped has two coordinates, range and azimuth. The radar's resolving ability in the range dimension is inversely proportional to the bandwidth of the transmitted pulse. In the azimuth dimension it is inversely proportional to the length of the radar's antenna. By way of example, a radar bandwidth of 1GHz provides a range resolution of 15 cm and an antenna length of 1000 times the wavelength of the transmitted wave, provides an azimuth resolution of 10 m. As can be seen, there is a wide difference in range and azimuth resolutions. For equal resolutions in both azimuth and range the antenna size would have to be increased enormously.

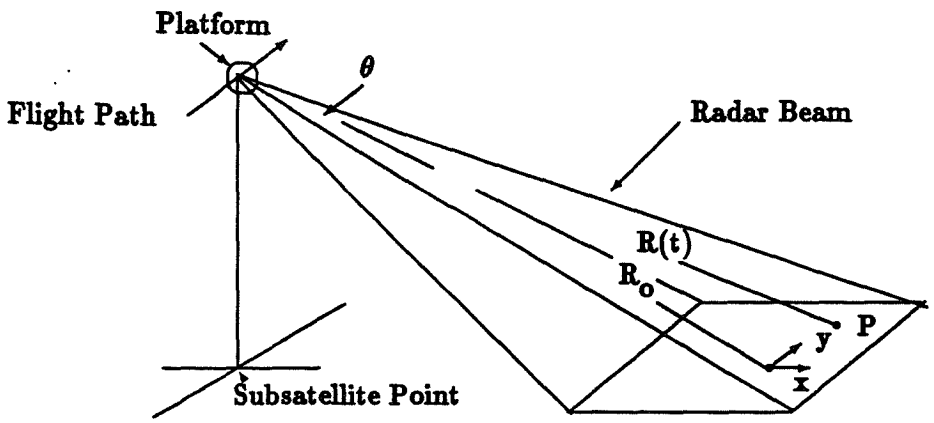


Figure 1: SAR Geometry

The *Synthetic Aperture Radar* (SAR) overcomes this problem by electronically synthesizing a long antenna from a physically small antenna. It does this by recording the intensity and the phase of the reflected signal and coherently processing the phase information. This synthesizes a large antenna aperture to obtain high azimuth resolution that is independent of range. The SAR achieves range resolution in much the same way as the conventional radar [1].

1.1 SAR Basics

In this section the phase function of the received signal will be derived. The expressions for range and azimuth resolutions in terms of the transmitted pulse bandwidth and the Doppler bandwidth, respectively, will also be derived. A simplified model of a SAR undergoing linear motion with velocity, V, with respect to a point target P is shown in Figure 1. The physical aperture of the radar generates a radio frequency signal (RF) and radiates it in the form of a beam of width θ. The time the point target lies in the footprint is the *dwell time* , T_d. The received signal contains the *radar reflectivity* (intensity) information of the point target as well as the radar carrier phase. The coherent radar determines the phase difference between the transmitted and the received pulses. The coherent detector in the receiver compares the received signal with the sum of the carrier frequency, f_c, and an offset frequency, f_{if}. The offset frequency centers the received spectrum about f_{if}. The slant range, $R(t)$, to the image point, P, is time varying. R_0 is the boresight range to the point target. The phase modulation on the carrier, $\phi(t)$, due to relative position between the platform and the point target is given by the following equation [2,3]:

$$\phi(t) = \frac{4\pi R(t)}{\lambda} \tag{1}$$

where λ is the wave length. From Figure 1:

$$R(t) = \sqrt{(R_0^2 + (x - x_0)^2)} \tag{2}$$

Substituting for $R(t)$ in equation (1):

$$\phi(t) = \frac{4\pi}{\lambda}\sqrt{R_0^2 + (x - x_0)^2} \tag{3}$$

$$\simeq \frac{4\pi}{\lambda}\{R_0 + \frac{(x - x_0)^2}{2R_0}\} \tag{4}$$

$$\simeq \frac{4\pi}{\lambda}\{R_0 + \frac{V^2(t - t_0)^2}{2R_0}\} \tag{5}$$

The shift in frequency, f_d, (Doppler Shift) which is the time varying phase is given by the following equation:

$$f_d = \frac{\delta\phi(t)}{\delta t} \tag{6}$$

$$= \frac{4\pi}{\lambda}\frac{V^2(t - t_0)}{R_0} \tag{7}$$

The output of the coherent detector is called the ***Doppler phase history*** .

This phase history for each point target is generated during the time the point target is in the physical aperture beam, that is, an interval T_d. Similar signal characteristics are obtained when the radar beam is directed at a different angle called the ***squint angle*** , θ_s. The corresponding Doppler frequency is given by the following equation [4]

$$f_d = -2V\frac{cos\theta_s}{\lambda} - 2V^2 sin^2\theta_s\frac{(t - t_0)}{\lambda R_0} \tag{8}$$

In order to obtain a fine range resolution, the transmitter produces a linear FM wave form represented as

$$f(t) = e^{i2\pi at^2} \qquad 0 \leq t \leq T_c$$
$$= 0 \qquad elsewhere \tag{9}$$

where a is the chirp rate and T_c is the FM pulse duration. When the detected signal is passed through a correlator or compressor, the output of the one dimensional correlator is the envelope of the function $\frac{sin(x)}{x}$. Due to the frequency coding of the pulse and proper phase matching in the correlator, the height of the correlated output is the amplitude of the original pulse multiplied by the square root of the time bandwidth product, aT_c^2. In the range direction the resolution is δR and in the azimuth direction the resolution is δAz. It is desirable for δR and δAz to be nearly equal. Expressions for the two dimensions and the derivations of their resolutions in the following chart are taken from Kovaly [4].

Range Direction	Azimuth Direction
Time Width τ_R of image at $-4db$ level: $\tau_R = 1/(\Delta f)_R$, where $(\Delta f)_R$ is the transmitting FM bandwidth	Time Width τ_{Az} of image at $-4db$ level: $\tau_{Az} = 1/(\Delta f)_{Az}$, where $(\Delta f)_{Az}$ is the Doppler bandwidth
Multiplying both sides of the above equation by $c/2$ produces: $c\tau_R/2 = c/2(\Delta f)_R$	Multiplying both sides of the above equation by Airborne Vehicle Velocity V produces: $V\tau_{Az} = V/(\Delta f)_{Az}$
Here $c\tau_R/2$ is defined as as Range Resolution, $\delta R = c/2(\Delta f)_R$	Here $V\tau_{Az}$ is defined as Azimuth Resolution, $\delta Az = V/(\Delta f)_{Az}$

The expressions have similar forms in their respective derivations. The resolutions are also seen to be inversely proportional to bandwidth. When these bandwidths are realized using signal processing, a synthetic aperture has been generated.

Depending on the application and also the system capability, a high resolution can be obtained by processing all of the Doppler bandwidth and compensating for the phase difference on a pulse-to-pulse basis. If all of the phase deviations are not compensated, a lower level of resolution is achieved called an *unfocussed aperture* . However, if phase correction is applied to each of the returned pulse, then a *focussed aperture* is obtained [4].

1.2 SAR Signal Processing

The physical measurement employed by SAR to achieve range resolution is the time delay introduced in the received signal while, in the azimuth direction it is the carrier phase history. Signal processing to implement SAR uses the measurements of time delay of the reflected signal to determine range of the target and the phase history of the carrier to give the angular coordinate of the target since the phase history is determined by the geometry of source and target. The range of a target varies at every instant, since the radar-to-target distance varies every instant. The processing is complicated further by the presence of several point targets in each beam footprint. The reflected signal is then the superposition of the reflections from all of the targets.

A general signal processing architecture is shown in Figure 2. In almost all SAR signal processors, the azimuth processing and the range processing are performed sequentially on the data. In reality the SAR signal is a two dimensional function of range and azimuth and hence the cross coupling between the two has to be considered. However, under certain conditions and with suitable compensation, the cross coupling can be made negligible and,

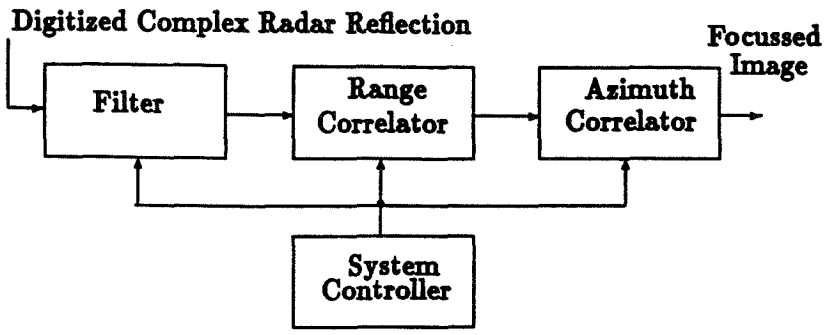


Figure 2: Block Diagram

to a first order, the two dimensional processing can be treated as two orthogonal one-dimensional processes. This makes processing a little easier since the compression in each direction is similar.

1.3 Goals

The primary goal of this paper is to present a simplified model for the SAR from the geometry. It will be shown from this model that the range and the azimuth coordinates of any point target can be determined from the intra and interpulse periods when the point target is in the footprint of the beam. The second goal is to present a VLSI architecture for real time SAR data processing using time domain methods in both range and azimuth directions.

2 Review of SAR Architectures

Historically, SAR architectures were ground based optical processors which were analog in nature. These processors used light sources and a series of lenses to perform two-dimensional processing. In these processors a film typically, is used as both input and output media. Although optical systems feature high throughput relative to digital processors, they are constrained by the dynamic range of the film and the limitations of the lenses. Although digital systems also suffer when the quality of the input data is poor, digital systems are more adaptable to specific data problems and can compensate for the poor quality data.

All of the existing architectures fall within the main frame-work of time domain and frequency domain architectures. However, more recently combined forms called " hybrid architectures " have been developed which exploit the time and frequency domain concepts. Whether the architectures are time domain, frequency domain or hybrid, all have to employ

parallel processing techniques to achieve real time imaging. A brief review of three types of existing digital architectures, and their merits and deficiencies are considered in the following sections.

2.1 Time Domain Architectures

Time domain architectures use algorithms that are based on the correlation or the convolution integral for range and azimuth compression. These take advantage of the repetitive nature of the correlation algorithm and employ high speed computers to implement the process. In these systems the range correlation is usually done before the azimuth correlation. However, since the azimuth correlation requires many received pulses before processing can begin, a considerable amount of memory is required to implement this architecture. Two of the important time domain architectures are discussed below.

2.1.1 Systolic Array Architecture

B Arambepola and S. R. Brooks [5] have suggested a real time systolic-array architecture employing a regular array of modules. The modules can be expanded to meet a variety of resolution and throughput requirements. The hardware uses a partitioned array of shift registers to process the data in both the range and azimuth directions. The system does not use any explicit corner turning and uses a minimum amount of two dimensional memory. The system degrades gracefully as the modules in the array fail and so no total failure occurs. However, the architecture is very complicated and the data processing rates are high.

2.1.2 Surface Acoustic Wave Compression Technique

A Surface Acoustic Wave (SAW) azimuth processor was suggested by Elachi [6]. This uses the quadratic nature of the azimuth phase history. This system uses charge coupled devices (CCD) for multiplexing and storing the data. This architecture tends to be slow because of the CCD devices used for storing and multiplexing. Since there are no prefilters in this system large amounts of data must be processed and stored.

2.2 Frequency Domain Architectures

The frequency domain architectures for SAR perform the range and azimuth correlation using the discrete Fourier transform technique (FFT). This technique has the advantage of converting convolution to multiplication and thereby possibly saves computation time. However, a large amount of memory is needed to store the data before and after this transformation. Corner turn memory is also required since the data has to be read in the azimuth direction before compression can be applied. Six kinds of the frequency domain architectures are examined in the following sections.

2.2.1 Linear Range Migration Approximation Architecture

M. Benson [7]has proposed a frequency domain architecture in which he approximated the range migration path by a straight line. In this research he skewed the range compressed data linearly to form a set azimuth lines on which azimuth compressions were done. Although this is a frequency domain approach, the algorithm can also be implemented in the time domain. The linear approximation for range migration requires calculating the FFT of the range compressed data. This process is computationally intensive. Since no prefiltering is done, there is too much data to be processed in real time.

2.2.2 The CRC Architecture

The Communication Research Company of Canada [8] implemented a true two-dimensional matched filter technique to process SAR data. This technique eliminates the assumption of no coupling between range and azimuth data. The advantage of the system is that it is capable of fine tuning itself by monitoring its own performance. However two dimensional processing is employed which requires large amounts of processing time and memory. Also there is no data reduction and as such the data to be processed is large. Removal of geometric distortion is more difficult due to the two dimensional processing.

2.2.3 Donier Systems

The Donier System, GmbH and the McDonald Detwiler Associates [9] have jointly developed a SAR processor based on the Specan Method. The Specan method employs the frequency domain approach. Donier system has been used in studies of image qualities, effects due to quantization errors, finite length registers, effects of interpolating filter sidelobes and ghost images. The architecture allows for parallel processing of sets of range compressed data by employing identical pipeline processors. No presumming is done on the data, which means that the data to be processed is large. Also because corner turning is done on the data, the memory requirements are high.

2.2.4 Single Instruction Multiple Data Systems

The single instruction multiple data system developed by B. Arampebola [10] uses an array architecture and is configured for high processing efficiency. The processor has high regularity and hence is suitable for VLSI implementation. The processing power can be increased by increasing the size of the array. Macros are used to increase the performance of the system. However, this system requires mass storage and needs extensive data routing. The random access memory is constructed as blocks and hence memory is not easily accessible between blocks. This architecture cannot be implemented in real time because there are no existing VLSI chips to implement it.

2.2.5 McDonald Detwiler Associates (MDA) Architecture

This architecture was developed by MDA for implementing the processing algorithm on a main frame system [11]. The computing facility of the MDA architecture consists of an Interdata 8/32 main-frame with limited fast storage and a large amount of disk storage. Several looks are processed and then individually interpolated in both the range and azimuth directions to accommodate the increase in the bandwidth due to the detection process. The final SAR image is generated by reducing the large radiometric dynamic range to 8 bits using square root law transformation preserving the dynamic range to $45db$. This architecture is accurate and fast and the range migration correction is efficiently implemented in the frequency domain. However, it needs a large memory and the data rates are high. Since a large amount of data is to be processed, a real time implementation is not possible

2.2.6 Template Controlled Image Processor

The template controlled image processor (TIP) developed by the Nippon Electric Company is essentially a data flow architecture and uses a special purpose, high speed processor to do signal processing [12]. Since the TIP uses a pipeline structure which is flexible, the same arrangement can be applied to all processes in the system. This architecture can be made real time by stacking up several TIP systems and applying segments of received data in parallel to all of them. The TIP system has a ring type architecture with one main and two major rings. Data are circulated into the rings and are processed by the modules connected to the rings until all operations on the data are complete. However, as in any frequency domain implementation this requires a large amount of memory for performing the corner turn operation. Real time implementation by stacking up the ring systems leads to other problems such as data handling capability, data flow, memory accessibility, processing capability and power.

2.3 Hybrid Architectures

The newly developed hybrid architectures use both time and frequency domain processing, combining the advantages of both. In these systems either the range or azimuth compression can be done first. The order in which the range and azimuth correlations are done gives rise to two different systems, with each system having some advantages. Two architectures which use this type of processing are discussed below.

2.3.1 Jet Propulsion Laboratory (JPL) Base Line Architecture

The JPL [13] has proposed a baseline architecture for a three frequency quad polarization system. The processor uses time domain parallel accumulators. The range correlation is done in the frequency domain and the azimuth correlation is done in the time domain. One of the significant features of this architecture is that it does not use bulk corner turn memory. It also does not have data transfer between the accumulator channels in

the azimuth processor. It generates precise Doppler reference functions and the range migration correction is programmable. The system achieves a data reduction rate of three. However, the number of azimuth reference samples is limited to 64 and this affects the compression ratio. Also since the range correlation is done in the frequency domain, some amount of memory is needed before the Fourier transformation. Most of the chips that would implement this architecture have yet to be designed.

2.3.2 Interim Digital Processor

The interim digital processor (IDP) with multiple execution developed by B. Barken, C. Wu and W. Karplus [14] uses a SEL 32/55 minicomputer and three AP 120B array processors. The array processors are controlled by executives which allow dynamic assignment and control of multiple array processors from a single control program. The processing is done on subtasks partitioned from the entire job. Two post-processing operations, namely data handling and multiple look overlay, are performed to derive the final image. This architecture is highly repetitive. The correlator software allows a number of array processors to perform in parallel. However, the system can suffer considerable time latency whenever the time taken for an operation is longer than the associated disk transfer time. This can cause a loss in efficiency.

3 Modeling of SAR

Any research into a VLSI implementation of the SAR processing algorithm requires an understanding of the principles involved in imaging the received data. Forming an image from the received data involves focusing in both the range and azimuth directions. The received signal, therefore, contains two-dimensional image information and hence a two dimensional matched filtering operation must be performed on the data. However, using the geometry of the SAR, it can be shown that the range and azimuth information can be obtained from processing the intra pulse and inter pulse data respectively. Furthermore, when the radiated modulated waveform is a periodic sequence of pulses, the range and the azimuth image information are contained in the reflected waveform in approximately independent ways. The following sections demonstrate how the two-dimensional received signal can be treated as two one-dimensional data for range and azimuth processing.

3.1 Transmitted and Received Signals

Let a point target, P, be located at a range x and an azimuth y with reference to the co-ordinate system shown in Figure 1. As the platform carrying the radar travels with a velocity, V, with respect to the point to be mapped, linear FM pulses of duration T_c are transmitted at some pulse repetition frequency (PRF). The time between the transmitted pulses is sufficiently long to prevent ambiguous range responses when there are many point targets. The time delay introduced in the received signal depends on the location of the point target and the location of the platform. The point target stays in the footprint of

the radar beam for a short time, T_d, during which time a number of pulses are transmitted and received. A single transmitted waveform, $s(t)$, translated with a carrier frequency, f_c, can be represented, in complex phasor form, by the following equation:

$$s(t) = e^{i2\pi(f_c t + 0.5at^2)}\{U(t) - U(t - T_c)\} \tag{10}$$

A single pulse travels a total distance of $2R(t)$ from the time of transmission to the time of reception, where $R(t)$ is the time varying distance of the point target to the platform. The received signal is the delayed version of $s(t)$ and is of the form $\sigma s(t - \tau)$, where σ is the reflectivity function, which is characteristic of the point target. Hence, the collected data comprise a set of reflectivity measurements in two dimensions and do not resemble the point target at all before processing. Rather the signals from the point target are dispersed in both range and azimuth. The effect of the carrier frequency in the reflected waveform can be removed (without destroying its phase) by coherently beating the signal down to an intermediate frequency, f_{if}. The received signal, $r(t)$, is translated to a lower frequency by low pass filtering. The filtered output is then,

$$r(t) = \sigma e^{i2\pi(-f_c\tau + f_{if}t + 0.5a(t-\tau)^2)}\{U(t - \tau) - U(t - T_c - \tau)\} \tag{11}$$

It is important for the phase coherence to be maintained between the transmitter and the receiver. The term $-f_c\tau$ in the exponent is the coherent carrier phase information.

3.2 Digital Processing of the Received Signal

The analog received signal can be sampled with a sampling frequency consistent with the Nyquist sampling requirements and this forms the basis for digital processing of the received signals. Let n be the number of pulses transmitted during the dwell time, T_d, and m be the total number of samples in the reflected signal. The image reflections are then sampled by $n \times m$ samples. The analog phase of the received signal for a single point image is given by the following equation

$$\phi = 2\pi\{-f_c\tau + f_{if}t + 0.5a(t - \tau)^2\}\{U(t - \tau) - U(t - T_c - \tau)\} \tag{12}$$

The platform moves a distance VT_p every time a pulse is transmitted and the total distance traveled during the dwell time is VnT_p, T_p being the interpulse period. The delay, τ, as stated before, is a function of the platform position and the target co-ordinates. Let j be the index for each of the samples in the received signal, and let k be the index for each of the transmitted pulses. The platform position, p, can be uniquely determined by the indices j and k, since the platform moves during every sample in the transmitted pulse train. The delay can, then, be written as:

$$\tau = d\{p(j,k), x, y\} \tag{13}$$

where $p(j,k)$, the platform position is determined by the jth sample index in the the received signal and kth pulse index in the pulse train. The sampled time, $t(j,k)$, corresponding to the platform position is also a function of j and k and is

$$t(j,k) = j\beta + kT_p$$

$$j = 1, 2, ..., m$$
$$k = 1, 2, ..., n \tag{14}$$

where β is the sampling period of the received signal. With this new notation for the delay, the sampled phase of the received signal sampled over the entire dwell time is:

$$\begin{aligned} \phi_{samp}\{t(j,k)\} &= 2\pi\{f_{if}(j\beta + kT_p) - f_c d\{p(j,k),x,y\} \\ &\quad +0.5a\{(j\beta + kT_p) - d\{p(j,k),x,y\}\}^2\} \\ &\quad \{U(j\beta + kT_p - \tau) - U(j\beta + kT_p - T_c - \tau)\} \\ j &= 1, 2, 3, ...m \\ k &= 1, 2, 3, ...n \end{aligned} \tag{15}$$

The above equation forms a coherent record of all the received pulses during the dwell time. Thus, a sampled record of the image is collected and stored in the SAR processor and the total number of samples stored are $n \times m$. The phase is a function of only the delay of the returned signal for a given set of parameters. An examination of the expression for the phase clearly reveals that it has the range and the azimuth information of the point target. We will show that the range information is present in the term $0.5a\{(j\beta + kT_p) - d\{p(j,k),x,y\}\}^2$ while the azimuth information is present in $-f_c d\{p(j,k),x,y\}$. The goal of the digital processing is to find the range and azimuth (x and y co-ordinates) information of the point target from this function. It is well known that the x and y co-ordinate information can be determined separately in independent processing. The details of this separation are given in the next section.

3.3 Range and Azimuth Processing

We will show that the delay in the received signal is a function of only x for intra PRF period samples and a function of only y for inter PRF period samples. However, the above independence is valid only when suitable compensations are applied for pitch, yaw and roll (attitude) changes of the platform and the data are corrected for range migration before the azimuth processing begins. As the platform moves between every sample, there is a positional displacement of the platform, $V\beta$, which gives rise to a different distance to the point target at every sampling instant. Therefore the distance, D_j, at the jth sampling instant is

$$D_j = \sqrt{(x^2 + (y - V\{j\beta + kT_p\})^2)} \tag{16}$$

since the platform is moving toward the target. Hence the delay corresponding to D_j is,

$$d_j\{p(j,k),x,y\} = 2\sqrt{(x^2 + (y - V\{j\beta + kT_p\})^2)}/c \tag{17}$$

where c is the speed of light.

3.3.1 Intrapulse Processing

When a pulse is transmitted, the parameter k is a constant until the reflected signal is completely received. Therefore for a fixed k and for j varying between 1 and m, the delay $d_j\{p(j,k),x,y\}$ varies as follows :

$$2\sqrt{(x^2+(y-V\{m\beta+kT_p\})^2)}/c \leq d_j\{p(j,k),x,y\} \leq 2\sqrt{(x^2+(y-V\{\beta+kT_p\})^2)}/c \quad (18)$$

Since V, β and T_p are known, the terms $V\{m\beta+kT_p\}$ and $V\{\beta+kT_p\}$ can be evaluated as constants. Using the seasat data, as an example, the number of pulses transmitted and received during the dwell time is 4509. Using a sampling frequency of $45MHz$ and a FM duration of $33.9\mu s$, m is 1525. Assume that a point target is at a range of 900 Km and and an azimuth of 10 Km. During the intra pulse duration for $k=0$, when the platform is at one extreme of the footprint, the distance expression $\sqrt{(x^2+(y-V\{m\beta+kT_p\})^2)}$ is 900.0555538 Km for $m=1$. It 900.0555510 for $m=1525$. In this case the distance is nearly equal to 900 for all m with an error in the $5th$ significant place. However, when k is 2254, that is, when the target is at the boresight of the platform, the distance when evaluated is 900.0000610 for $m=1$ and is 900.0000610 for $m=1525$. The distance for all m is again nearly equal to 900 Km with an error in the $8th$ significant digit. The value of the distance expression evaluated for $k=4508$, that is, when the platform is at the other extreme of the footprint is the same as that evaluated for $k=0$.

The numbers generated by evaluating the above example show that there is a small difference in the distance expressions when the platform is at either extreme of the footprint compared to when it is at the boresight. Therefore, the distance (and hence the delay) is not solely dependent on the range but is dependent on the azimuth as well. The dependence of the delay on both the range and azimuth is called the *range migration effect* . If the difference in the computed values is greater than half the range resolution, this effect has to be corrected. Thus, the data for any transmitted pulse during the dwell time will have to be corrected for range migration before they can be used to determine the delay and hence the range. When the range migration correction has been made, the delay expression can be approximated as:

$$d_j\{p(j,k),x,y\} \simeq 2\sqrt{(x^2)}/c \quad (19)$$

Therefore, the intrapulse delay and hence the phase profile variation is essentially only a function of the x co-ordinate of the image of the point target. This allows the x co-ordinate of the image to be determined using only the intrapulse processing.

3.3.2 Interpulse Processing

However, for every k, there is considerable motion of the platform toward or away from the point target. Therefore, with fixed j and varying k, (k varies from 0 to n) the delay will vary as follows:

$$2\sqrt{(x^2+(y-V\{j\beta+nT_p\})^2)}/c \leq d_k\{p(j,k),x,y\} \leq 2\sqrt{(x^2+(y-V\{j\beta+T_p\})^2)}/c \quad (20)$$

The delay illustrated by the above expression gives rise to a phase shift in the received signal. Processing for the y co-ordinate of the image, thus,involves processing this phase information. Note that as the platform passes the point target, the relative distance along the y direction between the platform and the point target decreases, becomes, zero and then increases. Hence, there is a point along the path of the platform, where the y distance to the point target is at a minimum and the corresponding delay of the returned signal is also at a minimum. The minimum delay in the returned signal is for a particular value of k, since k determines the relative position of the platform at any PRF instant. Assume, for now, that k is a continuous variable. The minimum delay for a particular k can be determined by minimizing the delay expression with respect to k. The k_{min} expression is:

$$k_{min} = (y/V - j\beta)/T_p \tag{21}$$

The y coordinate corresponding to this minimum delay is seen to be determined by both j and k. However, even when j is at a maximum, the contribution due to the term $j\beta$ is small. Hence, it is valid to state that k_{min} is a function of only the y coordinate. Hence, from the value of k_{min}, the y co-ordinate of the image can be determined. A correlator in the azimuth direction traditionally is used to determine k_{min}, including its fractional part and hence it also determines the y coordinate of the image point reflector. The determination of k_{min}forms the basis for the azimuth correlation processing.

3.4 Conclusion

We have presented a simplified geometrically based model for the entire SAR imaging problem. Using this model, we have formulated an expression for the entire phase history of the received SAR signal. We have shown that the azimuth and the range coordinate for a point target image can be obtained by processing the phase history during the interpulse and intrapulse periods respectively. Although we have considered a point target, the theory also applies to more generalized images.

4 A Real Time SAR architecture

The theory developed in the above section reveals that for seasat-type data range and azimuth processing can be done independently. Processing for range and azimuth involves compression techniques using correlation operations. The similarity in both the interpulse and intrapulse data processing enables similar hardware to be used for both operations. The correlation operation involves repetitive multiplications and additions. Although using the frequency domain approach, the correlation can be easily implemented by multiplying the frequency domain functions, the repetitive nature of the correlation algorithm suggests a time domain approach. Due to the recent availability of the technology and design capability, it is now possible to implement the correlation operation with VLSI chips in the time domain architecture. Hence, a time implementation is discussed in the following sections.

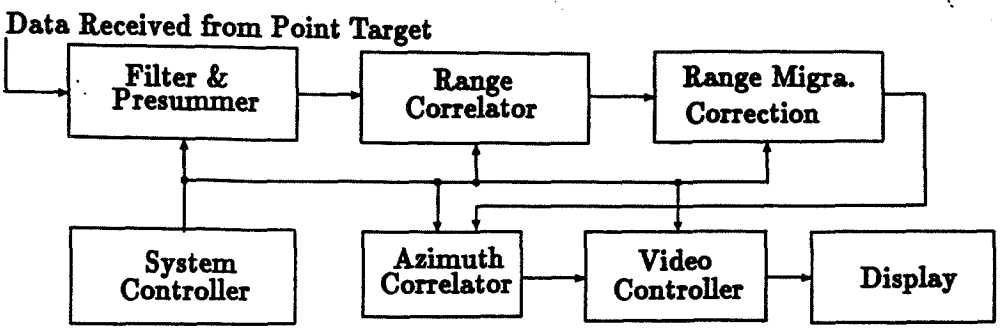


Figure 3: Time Domain Architecture

4.1 Block Diagram of Real Time SAR

A block diagram of the time domain architecture is shown in Figure 3. The data received from the target area is sampled and converted to digital form by an A to D converter (not shown). The system controller performs this operation using the system clock in order to retain the phase coherence of the received signal. The data is collected for every transmitted pulse and stored as a range line and is then sent to the filter and presumming circuit buffer. The following sections describe the block diagram briefly.

4.1.1 Filter and Presummer

The filter and presummer is shown in Figure 4 [15]. The azimuth time-bandwidth product required for the desired resolution is much lower than what is available in the received signal. In order to reduce the memory requirements of the system and the processing data rates, it is required by the system to filter the received data and use only the azimuth spectrum required for a particular resolution. Down sampling then is possible because a lower frequency spectrum is available for azimuth compression. The filtering and the presummer circuit performs this operation. The filtering operation is applied on the data in the azimuth direction. The presummer part of the circuit down samples the data by summing the range lines which are n lines apart with n channels running in parallel, n being the presum number. With presumming, the data rate is reduced by a factor of n.

4.1.2 Range Correlator

The range correlator shown in Figure 5 performs range compression on the presummed data in the time domain. Range compression compresses the image in the range direction. We used a new architecture to implement the range correlation. In this new architecture, the correlator consists of a number of multiplier cells operating in parallel on the incoming data. The high speed multipliers used here perform the correlation operation in real time. The data flow diagram of the correlator chip is shown in Figure 6.

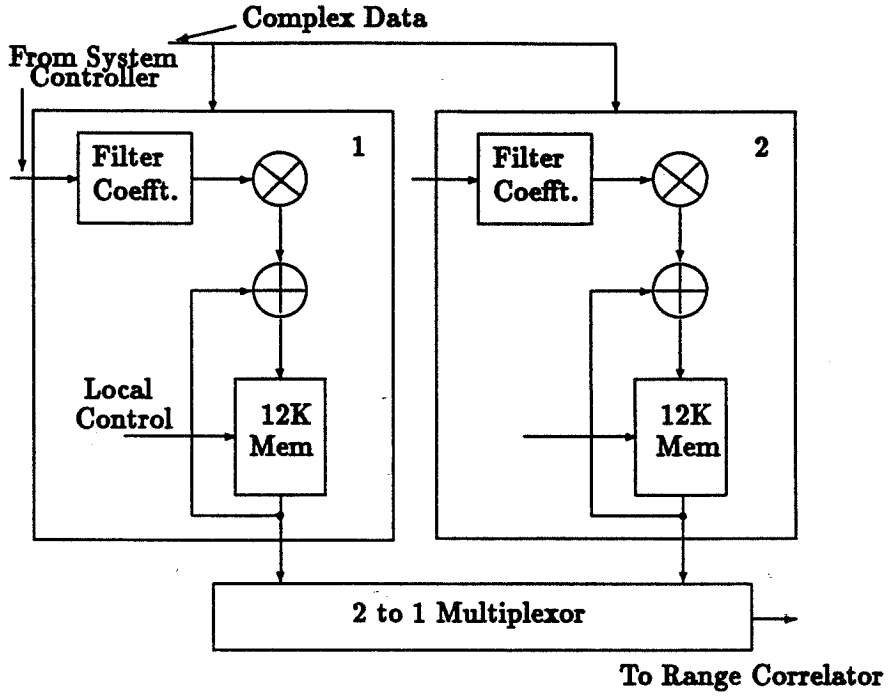


Figure 4: Filter & Presummer

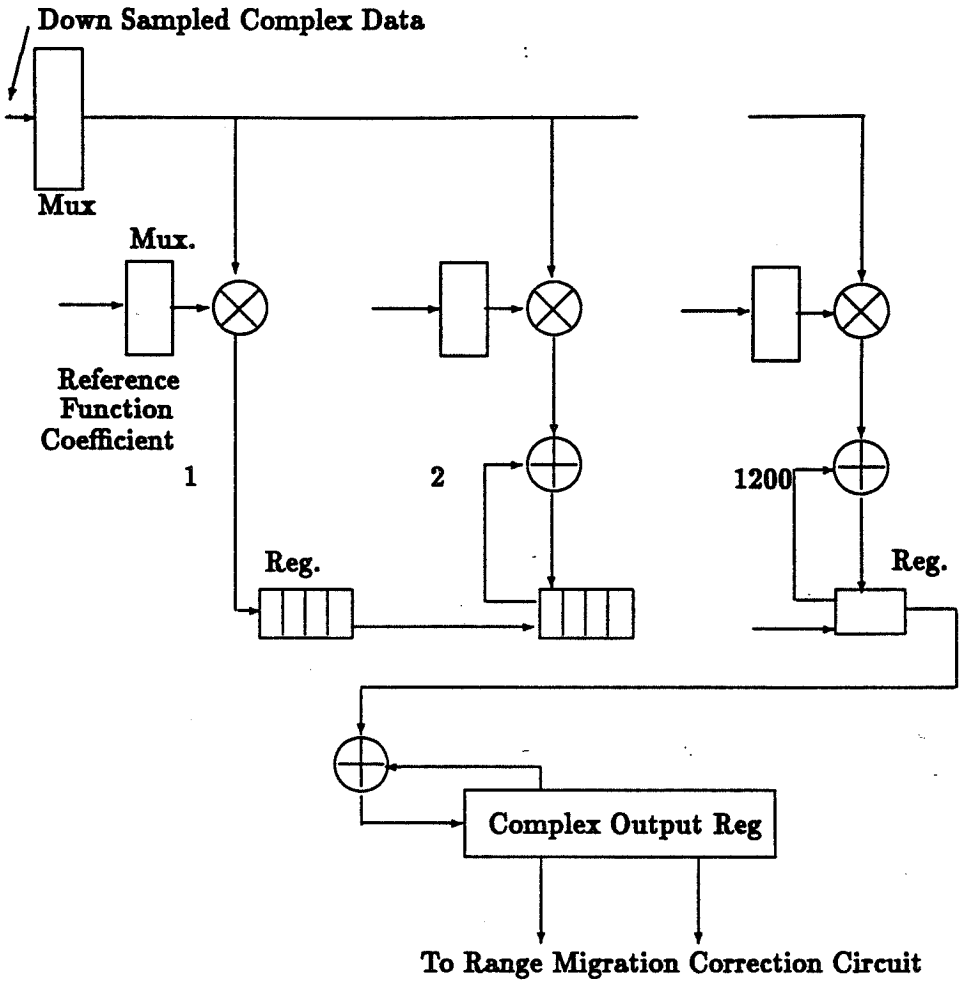


Figure 5: Range Correlator

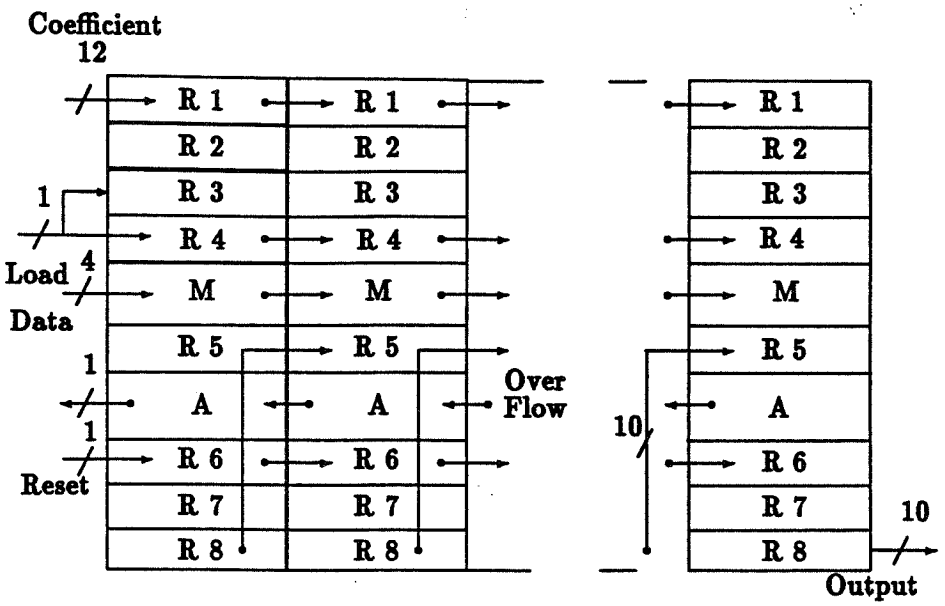


Figure 6: Data Flow Diagram of Range Correlator

4.1.3 Range Migration Correction Module

The range migration correction module (RMC) is shown in Figure 7. The received signal reflected off the point targets comprising the area to be mapped travels through several memory cells in the system memory. The range of the target varies with every transmitted pulse. Hence, the range compressed data must be interpolated according to the path that a particular point target takes through the memory. The range migration correction module performs this interpolation and sends out the corrected data to the azimuth correlator. Since the maximum range migration can be computed before correction is made, the correction is implemented by selecting one of the several sets of interpolated data. The interpolation is done in the RMC module. The correct set is chosen by the the system control processor.

4.1.4 Azimuth Correlator

The azimuth correlator with its associated video memory is shown in Figure 8. The azimuth compression is performed on the range migration corrected data to compress the spread of the image in the azimuth direction. This operation is performed in the time domain and requires the generation of a reference function which is essentially a matched filter in the azimuth direction. The matched filter characteristics are a function of the range and attitude parameters and hence, have to be computed. To save on time these could be precomputed and stored in the system controller. The selection of proper samples of

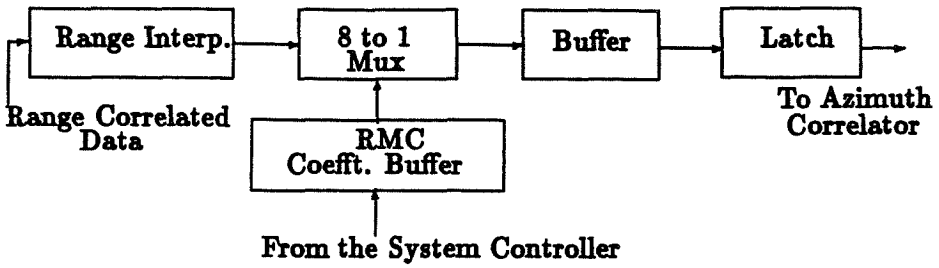


Figure 7: Range Migration Correction Module

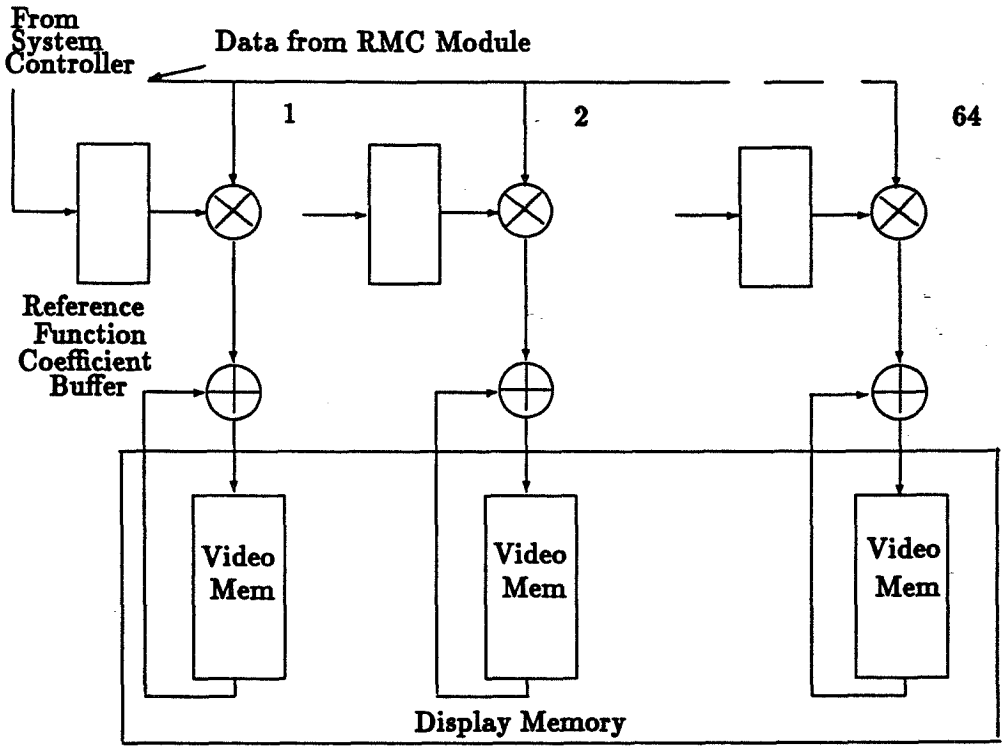


Figure 8: Azimuth Correlator

the matched filter characteristics for the correlation is done by the system controller and is based on the particular attitude parameters of the platform. The azimuth correlation is achieved by a single multiplier chip and part of the display memory.

4.1.5 Video Control Processor

The azimuth correlator uses the main display memory as part of its memory to do the correlation in the azimuth direction. Every time a multiplication is done, the previously computed and stored data is retrieved, summed with the present multiplied data and stored back in the same location in the video portion of the main display memory. Each location corresponds to a pixel in the image. This retrieval and storage operation is possible because the data rate has been considerably reduced and sufficient time is available between azimuth correlation operations. The video control processor performs this function of retrieving, summing and storing the data in the memory.

4.1.6 The System Controller

The system controller coordinates the operation of the entire system. It synchronizes the presumming operation so that proper data reduction is achieved. In addition, the system controller loads the coefficients for the filters and the RMC buffers and selects the proper interpolated data for the azimuth compression. The system controller also supervises the operation of the video control processor and the video output.

4.1.7 Video Output

This is the final interface for the image. The data stored in the video memory comprises the final focussed image which can be transmitted or displayed.

4.2 Significant Features of the Time Domain Architecture

1) It is a fast and efficient implementation.
2) Range processing is performed before azimuth processing.
3) No corner turn memory is required.
4) The time domain processing is performed on both range and azimuth.

4.3 VLSI Chip Descriptions

The following sections give overview descriptions of the VLSI chips used in the various parts of the time domain architecture. These descriptions are by no means exhaustive, but give a brief account of the different blocks that comprise each VLSI chip.

4.3.1 Filter and Presummer

The finite impulse response filter has twelve coefficients. A down sampling factor of 8 is achieved with two identical channels operating in parallel. This factor is achieved by

staggering the data coming into the two channels by 8 range lines between the two. The filter circuit consists of a buffer to store the filter coefficients, a complex multiplier, an adder and a 12 K memory. The memory is needed to apply the filtering operation in the azimuth direction. A controller which forms part of the chip synchronizes the operation of both channels and controls the flow of data from the memory during the filtering operation. A multiplexor connected to the channels enables the data to be output from both the channels sequentially.

4.3.2 Range Correlator

The range correlator uses the filtered and down sampled data and a reference function with 1200 samples. Each processor chip has 64 multiplier cells. Each multiplier cell has its own multiplier operating at $16MHz$, an adder and a 4 word register. The 4 word register is used for complex multiplication. With 1200 samples in the reference function, 20 such chips must be connected in cascade to perform real time correlation. The reference function samples are stored in buffers which are double buffered. This double buffering enables correlation with a different reference function. The final output of the correlator is taken out of a complex output register adder pair.

4.3.3 Range Migration Correction Module (RMC)

The RMC consists of an interpolator which is programmed to do interpolation on the range compressed data. The controller which performs the interpolation outputs a set of 8 interpolated data on a parallel bus. A multiplexor which forms part of the chip selects the data based on the range migration coefficients from the system controller that are stored in a buffer provided for that purpose. A second buffer is provided to store the selected interpolated data, since the data rate is changed after interpolation. The corrected data is output from this buffer at the system rate by the clock supplied by the system controller. A latch is provided at the output of the RMC module to latch the data before the azimuth processing.

4.3.4 Azimuth Correlator

The compression ratio for the azimuth processor is fixed at 64 for this example, although it is flexible and can be as high as 128. The azimuth correlation is done with a reference function with 64 samples. The samples are stored in double buffered registers which make it possible for an updated reference function to be loaded. These buffers and registers form part of the multiplier cell. The azimuth processor has 64 multiplier cells and adders associated with each multiplier. When the compression ratio is 128, two correlator chips have to be cascaded. The memory required for performing correlations in the azimuth direction is part of the display memory associated with the video control processor.

4.3.5 Video and System Controllers

The video controller is basically a microcontroller or a microprocessor which performs the complex function of retrieving processed data stored in the memory after every time the azimuth processor does a multiplication operation. The video controller also restores the summed data from the azimuth correlator in the original video memory location. The control program is loaded into the video controller program memory from the system controller at the beginning of the imaging process.

The system controller is a microcomputer coordinating the functions of all the subsystems of the architecture. It has its own program memory, buffers and peripheral devices to compute the reference functions, update the coefficients, etc. in the subsystems.

4.4 Total Chip Count

The real time SAR architecture discussed above has several chips which are dedicated in their functions. All of them are custom VLSI chips. A summary of the the number of each type of VLSI chip used in the architecture is given below. The total number of custom VLSI chips is 4.

Subsystem	Number of Chips
Filter & Presummer	2
Range Processor	20
Range Migration Correction	1
Azimuth Processor	1
Video Controller	1
System Controller	1

5 Summary

The theory behind the synthetic aperture radar signal processing has been presented. An alternative model for SAR from its geometry has been described. This geometrically based model shows that the range and azimuth information are present in the phase history of the signal received from the point targets comprising the area to be imaged. The model also describes how the range and azimuth information can be obtained by processing the information in the intrapulse and interpulse period samples. This makes range and azimuth processing independent of each other.

Some of the past work and architectures have been reviewed along with their merits and deficiencies. A real time architecture in the time domain using a new correlator has been described. The architecture uses only a small amount of memory and performs no corner turn. The architecture is fast and efficient and uses only five different types of custom

VLSI chips for its implementation. A brief description of each of these five VLSI chips has also been presented. This work was partially funded by the NASA SERC at the University of Idaho. Additionally, we, the authors, want to publically thank and praise the Lord Jesus Christ for influencing our lives and our work through His love, faithfulness, grace, death and resurrection.

References

[1] W. M. Brown and L. J. Porcello, "An Introduction to Synthetic Aperture Radar", *IEEE Spectrum*, Vol. 6, Sept. 1960, pp 52-62.

[2] J. P. Fitch, *Synthetic Aperture Radar*, Springler- Verlag, New York, 1988.

[3] R. O. Harger, *Synthetic Aperture Radar Systems: Theory and Design*, Academic Press, New York, 1970.

[4] J. J. Kovaly, *Synthetic Aperture Radar*, Artech House Inc., Dedham, MA, 1976.

[5] B. Arambepola and S. R. Brooks, "Systolic Array Architecture for Real Time SAR Processing", *Marconi research report* , pp 360-364.

[6] C. Elachi, *Spaceborne Radar Remote Sensing: Application and Techniques* , IEEE Press, Inc., New York, 1987.

[7] M. Benson, "Digital Processing of Seasat-A SAR Data Using Linear Approximations to the Range Cell Migration Curves", *IEEE International Radar Conference* , June 1980, pp 176-181.

[8] M. R. Vant, G. E. Haslam and W. E. Thorp, "The CRC SAR Digital Processor", Selected Papers, *European Space Agency Workshop*, Paris, France, 1979, pp 101-105.

[9] R. Schotter, R. Gunzenhauser and H. Holzi, "Real Time SAR Processor", 1982, pp FA-1, 2.1-2.6.

[10] R. J. Offen, *VLSI Image Processing* , McGraw-Hill Book Company, New York.

[11] J. R. Bennnet, I. G. Cumming and R. A. Deane, "The Digital Processing of Seasat Synthetic Aperture Radar Data", *Proceedings of IEEE International radar conference* , June 1980, pp. 168-174.

[12] H. Nohmi, N. Ito and S. Hanaki, " Digital Processing of Space-Borne Synthetic Aperture Radar Data", *Proceedings of the 3rd Seasat-SAR Workshop on SAR image Quality*, Frascati, Italy, Dec 1980, pp 47-49.

[13] K. Y. Liu and W. E. Arens, " Processing SAR Images on Board", *JPL Invention Report*, NPO-17195/6667. Jan 1989, pp 1.

[14] B. Barken, C. Wu, W. Karplus and D. Caswell, "Application of Parallel Array Processor for Seasat SAR Processing", *Geoscience and Remote Sensing*, Vol 1, pp 542-547.

[15] W. E. Arens and K. Y. Liu, "Flight SAR Processor Task", *FY 87 RTOP Report*, Jet propulsion Lab., 1987.

Using Algebra for Massively Parallel Processor Design and Utilization

Lowell Campbell
NASA Space Engineering Research Center
University of Idaho, Moscow, ID 83843

Michael R. Fellows
Department of Computer Science
University of Idaho, Moscow, ID 83843

Abstract- **This paper summarizes the authors' advances in the design of dense processor networks. Within is reported a collection of recent constructions of dense symmetric networks that provide the largest known values for the number of nodes that can be placed in a network of a given degree and diameter. The constructions are in the range of current potential engineering significance and are based on groups of automorphisms of finite-dimensional vector spaces.**

Key Words. **interconnection networks, Cayley graphs, linear groups**

1 Introduction

Two important objectives in the design of parallel processor interconnection networks are to minimize the number of wires connecting to a processor node and the number of nodes that a message must pass through to reach the destination node [11,12]. This is equivalent to the problem of constructing the largest possible graph when the degree and diameter of the graph are restricted [3].

Various approaches have been tried to construct graphs that improve these properties [4,7,6,8]. In many cases, our technique provides dramatic improvements over the best previously known constructions. Our new graphs are the largest known graphs of given degree and diameter for many of the degree and diameter pairs of engineering interest for large parallel processing systems [5]. These graphs are significantly better than current interconnection networks in terms of degree and diameter. An example comparison is the (degree=9,diameter=9) combination. For a 9 dimensional hypercube the number of nodes is $2^9 = 512$. For the our graphs, the (9,9) pair has 4,773,696 nodes. This is an increase of four orders of magnitude! This paper outlines these results and gives the generator sets for 35 new record constructions. For an overview of our new results see Table 1. Improved entries are shown in bold.

Our graphs are a type of graph called a Cayley Graph. Section II discusses the properties of these graphs that are relevant to processor design and section III discusses the construction technique.

	Diameter						
Deg	3	4	5	7	8	9	10
5	70	174	532	**4,368**	11,200	33,600	**123,120**
6	105	**355**	**1,081**	**13,104**	**50,616**	**202,464**	**682,080**
7	128	**506**	**2,162**	**39,732**	**150,348**	**911,088**	**4,773,696**
8	**203**	842	**3,081**	**103,776**	**455,544**	**2,386,848**	**7,738,848**
9	585	1,248	**6072**	215,688	**1,361,520**	**4,773,696**	**19,845,936**
10	650	1,820	**12,144**	**492,960**	**2,386,848**	**7,738,848**	47,059,200
11	715	3,200	14,625	898,776	**4,773,696**	**25,048,800**	179,755,200

Table 1: New Records

2 Algebraic symmetry as an organizing principle for parallel processing

There are important considerations apart from degree and diameter that must figure in any choice of network topology for parallel computation. Our approach yields *symmetric* constructions, and we believe that in this lies their greater value. Symmetry is one of the most powerful and natural tools to apply to the central problem of massively parallel computation: how to *organize* and *coordinate* computational resources.

The symmetries of the networks we describe are represented by simple algebraic operations (such as 2 by 2 matrix multiplications and modulo arithmetic). The main advantage of algebraic networks is that the developed mathematical resources of algebra are available to structure the problems of testing, data exchange, message routing, scheduling and the mapping of computations onto the network. The appeal of hypercubes, cube-connected-cycles, butterfly networks and others rests in large part on this same availability of easily computed (and comprehended) symmetries. These popular networks and those that we describe all belong to a class of algebraic networks based on vector spaces and their symmetry groups. For recent algebraic approaches to routing algorithms, deadlock avoidance, emulation and scheduling for algebraically described networks of this sort see [1,2,9,10].

The next section describes our basic approach and some examples of our constructions.

3 Technique and Example Constructions

A network is (*vertex-*) *symmetric* if for any two nodes u,v there is an automorphism of the network mapping u to v. Every Cayley network is symmetric (symmetries are given by group multiplication). If A is a group and $S \subseteq A$ is a generating set that is closed under inverses, i.e., $S = S \cup S^{-1}$, then the (undirected) *Cayley graph* (A, S) is the graph with vertex set A and with an edge between elements a and b of A if and only if $as = b$ for some $s \in S$. It is remarkable (but, indeed, natural) that most networks that have

been considered for large parallel processing systems (including hypercubes, grids, cube-connected-cycles and butterfly networks) are Cayley graphs. The degree of a Cayley graph (A, S) is $\Delta = |S|$ and the diameter of (A, S) is $D = max_{a \in A}\{min_t : a = s_1 \cdots s_t, s_i \in S \text{ for } i = 1, \ldots, t\}$.

Example 1 Degree 5, diameter 7 : 4368 vertices. (Best previous : 2988.)

This is a Cayley graph on the subgroup of GL[2,13] consisting of the matrices with determinant in the set {1,-1}. The generators are the following elements together with their inverses.

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} order\ 2 \quad \begin{bmatrix} 11 & 2 \\ 8 & 12 \end{bmatrix} order\ 52 \quad \begin{bmatrix} 11 & 4 \\ 7 & 5 \end{bmatrix} order\ 14$$

Example 2 Degree 6, diameter 10 : 682,080 vertices. (Best previous : 199,290.)

This is a Cayley graph on the group GL[2,29]. The generators are the following elements together with their inverses.

$$\begin{bmatrix} 28 & 10 \\ 8 & 8 \end{bmatrix} order\ 28 \quad \begin{bmatrix} 17 & 13 \\ 16 & 27 \end{bmatrix} order\ 28 \quad \begin{bmatrix} 3 & 4 \\ 27 & 14 \end{bmatrix} order\ 840$$

Example 3 Degree 10, diameter 5 : 12144 vertices. (Best previous : 10,000.)

This is a Cayley graph on the group SL[2,23]. The generators are the following elements together with their inverses.

$$\begin{bmatrix} 9 & 0 \\ 18 & 18 \end{bmatrix} order\ 11 \quad \begin{bmatrix} 13 & 10 \\ 18 & 21 \end{bmatrix} order\ 11 \quad \begin{bmatrix} 9 & 10 \\ 0 & 17 \end{bmatrix} order\ 22$$

$$\begin{bmatrix} 14 & 7 \\ 19 & 3 \end{bmatrix} order\ 22 \quad \begin{bmatrix} 18 & 13 \\ 17 & 20 \end{bmatrix} order\ 24$$

The tables at the end of the paper are a complete list of the new constructions. In them, B[2,n] denotes the Borel subgroup of GL[2,n].

4 Conclusions

We have presented new constructions for graphs with the largest number of nodes for a given degree and diameter. These graphs have potential for use in the design of parallel processing systems. They have the additional advantage of possessing algebraic symmetry simplifies processor design. These graphs and their properties are the subject of on going research.

References

[1] F. Annexstein, M. Baumslag and A.L. Rosenberg, "Group Action Graphs and Parallel Architectures," COINS Technical Report 87-133, Univ. of Mass., Amherst, 1987.

[2] S.B. Akers and B. Krishnamurthy, "On Group Graphs and Their Fault-Tolerance," *IEEE Trans. on Computers*, 36 (1987), pp. 885-888.

[3] J.C. Bermond, C. Delorme, and J.J. Quisquater, "Strategies for Interconnection Networks: Some Methods from Graph Theory", *Journal of Parallel and Distributed Computing* 3 (1986), pp. 433-449.

[4] J. Bond, C. Delorme, and W.F. de La Vega, "Large Cayley Graphs with Small Degree and Diameter", Rapport de Recherche no. 392, LRI, Orsay, 1987.

[5] L. Campbell, M. Fellows, et. al. , "Dense Symmetric Networks from Linear Groups", *The Second Symposium on Frontiers of Massively Parallel Computation*, Fairfax, Virginia, October 1988.

[6] G.E. Carlsson, J.E. Cruthirds, H.B. Sexton, and C.G. Wright, "Interconnection Networks Based on a Generalization of Cube-Connected Cycles", *IEEE Trans. on Computers*, Vol. C-34, No. 8, August 1985, pp. 769-772.

[7] D.V. Chudnovsky, G.V. Chudnovsky, and Denneau, "Regular Graphs with Small Diameter as Models for Interconnection Networks", *3rd Int. Conf. on Supercomputing*, Boston, MA, May 1988, pp. 232-239.

[8] K. Doty, "New Designs for Processor Interconnection Networks", *IEEE Trans. on Computers*, Vol. C-33, No. 5 May 1984, pp. 447-450.

[9] V. Faber, "Global Communication Algorithms for Hypercubes and Other Cayley Coset Graphs", Technical Report, Los Alamos National Laboratories, 1988.

[10] M. Fellows, "A Category of Graphs for Parallel Processing," Technical Report, University of Idaho, 1988.

[11] P. Mazumber,"Evaluation of Three Interconnection Networks for CMOS VLSI Implementation",*International Conference on Parallel Processing*, August 1986, pp. 200-207.

[12] A. Ranade and S. L. Johnson,"The Communication Efficiency of Meshes, Boolean Cubes and Cube Connected Cycles for Wafer Scale Integration", *International Conference on Parallel Processing*, August 1987, pp. 479-482.

Parameters	Order	Previous Record	Moore Bound	Group	Generators: order $S = S \cup S^{-1}$
degree 5 diameter 7	4,368	2,988	27,306	index 6 in $GL[2,13]$ $(det = r^6)$	[0,1,1,0]:2 [11,2,8,12]:52 [11,4,7,5]:14
degree 5 diameter 10	123,120	52,224	1,747,626	$GL[2,19]$	[0,1,1,0]:2 [16,11,2,0]:45 [11,16,0,15]:18
degree 6 diameter 4	355	320	937	index 14 in B[2,71]	[54,66,0,1]:5 [5,43,0,1]:5 [57,38,0,1]:5
degree 6 diameter 5	1,081	992	4,687	index 2 in $B[2,47]$	[7,20,0,1]:23 [6,33,0,1]:23 [9,42,0,1]:23
degree 6 diameter 7	13,104	13,056	117,187	index 2 in $GL[2,13]$ $(det = r^2)$	[10,12,10,9]:39 [12,3,9,8]:84 [8,11,5,0]:84
degree 6 diameter 8	50,616	32,256	585,937	$SL[2,37]$	[32,24,35,2]:19 [12,24,15,27]:37 [23,16,28,34]:36
degree 6 diameter 9	202,464	72,345	2,929,687	index 9 in $GL[2,37]$ $(det = r^9)$	[25,1,31,1]:36 [12,35,23,30]:76 [12,4,28,16]:152
degree 6 diameter 10	682,080	199,290	14,648,437	$GL[2,29]$	[28,10,8,8]:28 [17,13,16,27]:28 [3,4,27,14]:840

Table 2: Generators

Parameters	Order	Previous Record	Moore Bound	Group	Generators:order $S = S \cup S^{-1}$
degree 7 diameter 4	506	480	1,814	$B[2,23]$	[13,16,0,1]:11 [19,12,0,1]:22 [3,16,0,1]:11 [282,1,0,1]:2
degree 7 diameter 5	2,162	1,550	10,886	$B[2,47]$	[29,14,0,1]:46 [4,20,0,1]:23 [20,27,0,1]:46 [46,1,0,1]:2
degree 7 diameter 7	39,732	35,154	391,910	index 2 in $SL[2,43]$ $(/\{\pm 1\})$	[0,42,1,0]:2 [18,16,38,41]:22 [8,28,14,33]:43 [34,2,37,6]:22
degree 7 diameter 8	150,348	93,744	2,351,462	index 2 in $SL[2,67]$ $(/\{\pm 1\})$	[0,66,1,0]:2 [48,48,4,11]:66 [59,18,42,31]:33 [7,64,66,58]:134
degree 7 diameter 9	911,088	304,668	14,108,774	index 2 in $GL[2,37]$ $(det = r^2)$	[0,1,1,0]:2 [27,33,19,22]:684 [25,16,13,6]:36 [23,17,14,26]:18
degree 7 diameter 10	4,773,696	2,002,000	84,652,646	GL[2,47]	[0,1,1,0]:2 [18,2,15,32]:46 [13,12,7,25]:552 [36,29,37,29]:46

Table 3: Generators

Parameters	Order	Previous Record	Moore Bound	Group	Generators:order $S = S \cup S^{-1}$
degree 8 diameter 3	203	200	457	index 4 in $B[2, 29]$	[16,9,0,1]:7 [16,21,0,1]:7 [25,15,0,1]:7 [25,9,0,1]:7
degree 8 diameter 5	3,081	2,808	22,409	index 2 in $B[2, 79]$	[49,72,0,1]:39 [46,43,0,1]:13 [19,26,0, 1]:39 [13,13,0,1]:39
degree 8 diameter 7	103,776	89,280	1,098,057	SL[2,47]	[14,39,45,18]:24 [33,38,21,0]:46 [13,33,29,5]:16 [28,8,25,29]:23
degree 8 diameter 8	455,544	234,360	7,686,401	index 4 in $GL[2, 37]$ $(det = r^4)$	[28,32,33,33]:171 [9,34,25,16]:342 [21,9,17,5]:57 [0,26,3,1]:171
degree 8 diameter 9	2,386,848	1,822,176	53,804,809	index 2 in GL[2,47] $(det = r^{2k})$	[26,20,25,10]:1081 [8,23,21,33]:552 [20,37,31,28]:184 [33,4,25,44]:23
degree 8 diameter 10	7,738,848	3,984,120	376,633,665	GL[2,53]	[12,25,11,30]:52 [5,42,48,45]:1404 [33,1,39,42]:52 [26,39,22,51]:52

Table 4: Generators

Parameters	Order	Previous Record	Moore Bound	Group	Generators:order $S = S \cup S^{-1}$
degree 9 diameter 5	6,072	5,150	42,130	index 2 in $SL[2,23]$ $(/\{\pm 1\})$	[0,22,1,0]:2 [2,18,4,2]:11 [10,1,21,16]:24 [6,19,4,9]:24 [22,0,1,22]:46
degree 9 diameter 8	1,316,520	910,000	21,570,706	index 10 in GL[2,61] $(det = r^{10k})$	[60,1,0,1]:2 [10,21,19,8]:12 [11,15,4,51]:93 [51,7,43,60]:60 [50,1,18,26]:62
degree 9 diameter 9	4,773,696	3,019,632	172,565,650	GL[2,47]	[0,1,1,0]:2 [5,30,19,3]:23 [41,19,14,23]:46 [7,29,29,34]:46 [38,20,42,24]:368
degree 9 diameter 10	19,845,936	15,686,400	1,380,525,202	GL[2,67]	[66,1,0,1]:2 [4,19,19,48]:4488 [58,44,8,58]:11 [50,43,39,16]:66 [43,47,0,34]:66
degree 10 diameter 5	12,144	10,000	73,811	$SL[2,23]$	[9,0,18,18]:11 [13,10,18,21]:11 [9,10,0,17]:22 [14,7,19,3]:22 [18,13,17,20]:24

Table 5: Generators

Parameters	Order	Previous Record	Moore Bound	Group	Generators:order $S = S \cup S^{-1}$
degree 10 diameter 7	492,960	486,837	5,978,711	index 2 in $SL[2,79]$ $(/\{\pm 1\})$	[48,18,52,8]:78 [70,65,1,19]:80 [53,54,4,19]:78 [29,76,72,28]:80 [32,4,58,69]:80
degree 10 diameter 8	2,386,848	2,002,000	53,808,401	index 2 in GL[2,47] $(det = r^{2k})$	[29,5,0,22]:46 [23,8,3,12]:552 [19,7,11,19]:1104 [15,16,38,0]:46 [46,6,22,28]:23
degree 10 diameter 9	7,738,848	7,714,494	484,275,611	GL[2,53]	[5,42,48,45]:1404 [33,1,39,42]:52 [26,39,22,51]:52 [19,15,2,9]:2808 [16,15,4,28]:52
degree 11 diameter 8	4,773,696	4,044,492	122,222,222	GL[2,47]	[0,1,1,0]:2 [24,25,24,9]:736 [22,39,19,6]:23 [17,24,42,31]:1104 [19,8,9,37]:46 [32,31,37,32]:276

Table 6: Generators

Parameters	Order	Previous Record	Moore Bound	Group	Generators:order $S = S \cup S^{-1}$
degree 11 diameter 9	25,048,800	21,345,930	1,222,222,222	GL[2,71]	[70,1,0,1]:2 [69,44,41,53]:5040 [32,26,7,5]:70 [22,30,6,27]:5040 [54,37,60,8]:1260 [12,43,40,47]:1008
degree 12 diameter 5	24360	21320	193261	$SL[2,29]$	[1,18,3,26]:58 [17,20,14,8]:30 [15,6,20,10]:30 [23,28,28,19]:28 [26,14,11,16]:28 [16,7,28,25]:28
degree 12 diameter 8	9,922,968	8,370,180	257,230,657	index 2 in GL[2,67] ($det = r^{2k}$)	[13,26,63,49]:66 [50,44,6,19]:1122 [62,53,25,17]:22 [26,39,65,12]:2244 [13,33,1,4]:2244 [63,16,42,14]:11
degree 13 diameter 7	2,723,040	2,657,340	42,346,682	index 5 in GL[2,61] ($det = r^{5k}$)	[60,1,0,1]:2 [43,2,25,27]:93 [54,58,59,53]:60 [50,37,50,8]:30 [6,55,26,8]:744 [60,14,55,22]:62 [27,45,36,16]:60

Table 7: Generators

Parameters	Order	Previous Record	Moore Bound	Group	Generators:order $S = S \cup S^{-1}$
degree 13 diameter 8	13,615,200	10,257,408	508,160,186	GL[2,61]	[60,1,0,1]:2 [12,18,16,56]:3720 [48,50,27,6]:30 [1,29,38,15]:248 [56,30,37,42]:1830 [58,12,21,3]:120 [8,38,36,60]:620
degree 15 diameter 8	38,450,880	35,947,392	1,702,833,526	GL[2,79]	[78,1,0,1]:2 [9,16,25,36]:78 [40,37,27,41]:3120 [25,40,19,22]:2080 [67,67,46,59]:1560 [70,66,4,49]:1248 [12,57,78,56]:78 [72,74,12,46]:1248

Table 8: Generators

N94-71088

On Well-Partial-Order Theory and Its Application to Combinatorial Problems of VLSI Design

M. Fellows
Department of Computer Science
University of Idaho
Moscow, ID 88343

M. Langston
Department of Computer Science
University of Tennessee
Knoxville, TN 37996

Department of Computer Science
Washington State University
Pullman, WA 99164-1301

Abstract– **We nonconstructively prove the existence of decision algorithms with low-degree polynomial running times for a number of well-studied graph layout, placement and routing problems. Some were not previously known to be in $\mathcal{P}$ at all; others were only known to be in $\mathcal{P}$ by way of brute force or dynamic programming formulations with unboundedly high-degree polynomial running times. Our methods include the application of the recent Robertson-Seymour theorems on the well-partial-ordering of graphs under both the minor and immersion orders. We also briefly address the complexity of search versions of these problems.**

1 Introduction

Practical problems are often characterized by fixed-parameter instances. In the VLSI domain, for example, the parameter may represent the number of tracks permitted on a chip, the number of processing elements to be employed, the number of channels required to connect circuit elements or the load on communications links. In fixing the value of such parameters, we help focus on the physically realizable nature of the system rather than on the purely abstract aspects of the model.

In this paper, we employ and extend Robertson-Seymour poset techniques to prove low-degree polynomial-time decision complexity for a variety of fixed-parameter layout, placement and routing problems, dramatically lowering known time-complexity upper bounds. Our main results are summarized in Table 1, where n denotes the number of vertices in an input graph and k denotes the appropriate fixed parameter. * input restricted to graphs of maximum degree three

In the next section, we survey the necessary background from graph theory and graph algorithms that makes these advances possible. Sections 3, 4 and 5 describe our results on several representative types of decision problems, illustrating a range of techniques based on well-partially-ordered sets. In Section 6, we discuss how self-reducibility can be used

General Problem Area	Problem	Best Previous Upper Bound	Our Result
Circuit Layout	GATE MATRIX LAYOUT	open	$O(n^2)$ [FL1]
Linear Arrangement	MIN CUT LINEAR ARRANGEMENT	$O(n^{k-1})$	$O(n^2)$
	MODIFIED MIN CUT	$O(n^k)$	$O(n^2)$
	TOPOLOGICAL BANDWIDTH*	$O(n^k)$	$O(n^2)$ [FL2]
	VERTEX SEPARATION	$O(n^{k^2+2k+4})$	$O(n^2)$
Circuit Design and Utilization	CROSSING NUMBER*	open	$O(n^3)$ [FL2]
	MAX LEAF SPANNING TREE	$O(n^{2k+1})$	$O(n^2)$
	SEARCH NUMBER	$O(n^{2k^2+4k+8})$	$O(n^2)$
Embedding and Routing	2-D GRID LOAD FACTOR	open	$O(n^2)$
	BINARY TREE LOAD FACTOR	open	$O(n^2)$
	DISK DIMENSION	open	$O(n^3)$ [FL1]
	EMULATION	open	$O(n^3)$ [FL2]

Table 1: Main Results

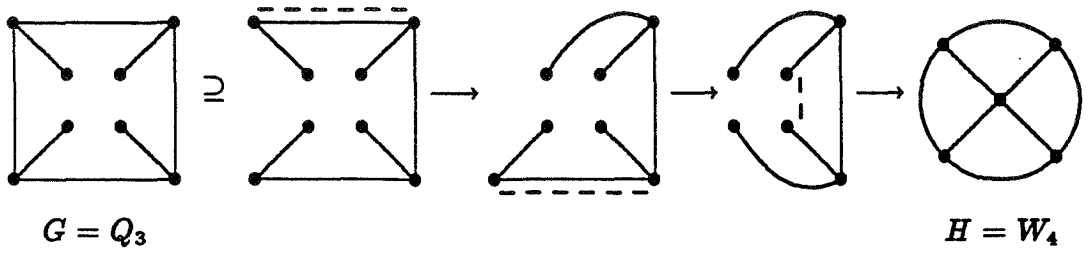


Figure 1: Construction demonstrating that W_4 is a minor of Q_3

to bound the complexity of search versions of these problems. A few open problems and related issues are briefly addressed in the final section.

2 Background

We consider only finite, undirected graphs. A graph H is less than or equal to a graph G in the *minor* order, written $H \leq_m G$, if and only if a graph isomorphic to H can be obtained from G by a series of these two operations: taking a subgraph and *contracting* an edge. For example, the construction depicted in Figure 1 shows that $W_4 \leq_m Q_3$.

Note that the relation $\leq_m$ defines a partial ordering on graphs. A family F of graphs is said to be *closed* under the minor ordering if the facts that G is in F and that $H \leq_m G$ together imply that H must be in F. The *obstruction set* for a family F of graphs is the set of graphs in the complement of F that are minimal in the minor ordering. Therefore, if F is closed under the minor ordering, it has the following characterization: G is in F if and only if there is no H in the obstruction set for F such that $H \leq_m G$.

Theorem 1. [RS5] (formerly known as Wagner's Conjecture [Wa]) Graphs are *well-partially-ordered* by $\leq_m$. That is, any set of graphs contains only a finite number of minor-minimal elements, and there are no infinite descending chains.

Theorem 2. [RS4] For every fixed graph H, the problem that takes as input a graph G and determines whether $H \leq_m G$ is solvable in polynomial time.

Theorems 1 and 2 guarantee only the *existence* of a polynomial-time decision algorithm for any minor-closed family of graphs. Moreover, no proof of Theorem 1 can be entirely constructive. For example, there can be no systematic method of computing the finite obstruction set for an arbitrary minor-closed family F from the description of a Turing machine that accepts precisely the graphs in F [FL5].

An interesting feature of Theorems 1 and 2 is the low degree of the polynomials bounding the decision algorithms' running times (although the constants of proportionality are enormous). Letting n denote the number of vertices in G, the time required to recognize F is $O(n^3)$. If F excludes a planar graph, then F has bounded tree-width [RS2] and the time complexity decreases to $O(n^2)$.

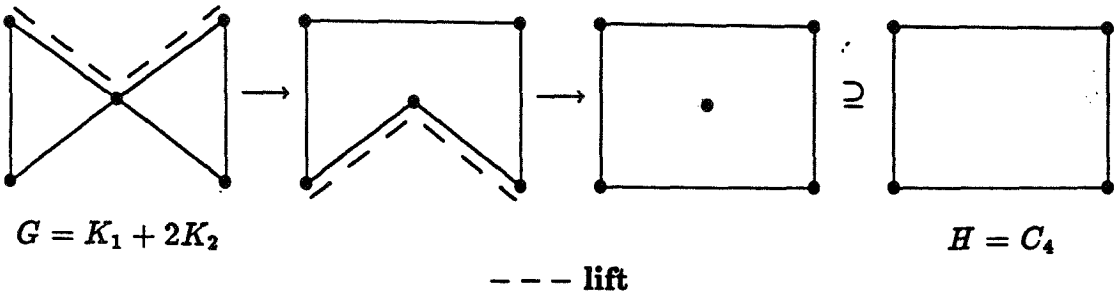


Figure 2: Construction demonstrating that C_4 is immersed in $K_1 + 2K_2$

A graph H is less than or equal to a graph G in the *immersion* order, written $H \leq_i G$, if and only if a graph isomorphic to H can be obtained from G by a series of these two operations: taking a subgraph and *lifting* [Ma3] a pair of adjacent edges. For example, the construction depicted in Figure 2 shows that $C_4 \leq_i K_1 + 2K_2$ (although $C_4 \not\leq_m K_1 + 2K_2$).

The relation $\leq_i$, like $\leq_m$, defines a partial ordering on graphs with the associated notions of closure and obstruction sets.

Theorem 3. [RS1] (formerly known as Nash-Williams' Conjecture [Na]) Graphs are well-partially-ordered by $\leq_i$.

The proof of the following result is original, although it has been independently observed by others as well [Rob].

Theorem 4. For every fixed graph H, the problem that takes as input a graph G and determines whether $H \leq_i G$ is solvable in polynomial time.

Proof. Letting k denote the number of edges in H, we replace $G = \langle V, E \rangle$ with $G' = \langle V', E' \rangle$, where $|V'| = k|V| + |E|$ and $|E'| = 2k|E|$. Each vertex in V is replaced in G' with k vertices. Each edge e in E is replaced in G' with a vertex and $2k$ edges connecting this vertex to all of the vertices that replace e's endpoints. We can now apply the disjoint-connecting paths algorithm of [RS4], since it follows that $H \leq_i G$ if and only if there exists an injection from the vertices of H to the vertices of G' such that each vertex of H is mapped to some vertex in G' that replaces a distinct vertex from G and such that G' contains a set of k vertex-disjoint paths, each one connecting the images of the endpoints of a distinct edge in H. □

Theorems 3 and 4, like Theorems 1 and 2, guarantee only the existence of a polynomial-time decision algorithm for any immersion-closed family F of graphs. The method we have used in proving Theorem 4 yields an obvious time bound of $O(n^{h+6})$, where h denotes the order of the largest graph in F's obstruction set. (There are $O(n^h)$ different injections to consider; the disjoint-paths algorithm takes cubic time on G', a graph of order at most n^2.) Thanks to the next theorem due to Mader, however, we find that the bound immediately reduces to $O(n^{h+3})$, because the problem graphs of interest permit only a linear number of distinct edges.

Theorem 5. [Ma1] For any graph H there exists a constant c_H such that every simple graph $G = \langle V, E \rangle$ with $|E| > c_H|V|$ satisfies $G \geq_i H$.

We shall show in Section 4 that, by exploiting excluded-minor knowledge on immersion-closed families, the time complexity for determining membership can in many cases be reduced to $O(n^2)$.

3 Exploiting the Minor Order

Given a graph G of order n, a *linear layout* of G is a bijection ℓ from V to $\{1, 2, \ldots, n\}$. For such a layout ℓ, the *vertex separation* at location i, $s_\ell(i)$, is $|\{u : u \in V, \ell(u) \leq i$, and there is some $v \in V$ such that $uv \in E$ and $\ell(v) > i\}|$. The vertex separation of the entire layout is $s_\ell = \max\{s_\ell(i) : 1 \leq i \leq n\}$, and the vertex separation of G is $vs(G) = \min\{s_\ell : \ell$ is a linear layout of $G\}$.

Given both G and a positive integer k, the $\mathcal{NP}$-complete VERTEX SEPARATION problem [Le] asks whether $vs(G)$ is less than or equal to k. It has previously been reported that VERTEX SEPARATION can be decided in $O(n^{k^2+2k+4})$ time [EST], and is thus in $\mathcal{P}$ for any fixed value of k. We now prove that the problem can be solved in time bounded by a polynomial in n, the degree of which does not depend on k.

Theorem 6. For any fixed k, VERTEX SEPARATION can be decided in $O(n^2)$ time.

Proof. Let k denote any fixed, positive integer. We shall show that the family F of "yes" instances is closed under the minor ordering. To do this, we must prove that if $vs(G) \leq k$ then $vs(H) \leq k$ for every $H \leq_m G$. Without loss of generality, we assume that H is obtained from G by exactly one of these three actions: deleting an edge, deleting an isolated vertex, or contracting an edge.

If H is obtained from G by deleting an edge, then $vs(H) \leq vs(G) \leq k$ because the vertex separation of any layout of G either remains the same or decreases by 1 with the removal of an edge. If H is obtained from G by deleting an isolated vertex, then clearly $vs(H) \leq k$.

Suppose H is obtained from G by contracting the edge uv. Let ℓ denote a layout of G whose vertex separation does not exceed k and assume $\ell(u) < \ell(v)$. We contract uv to u in the layout ℓ' of H as follows: we set $\ell'(x) = \ell(x)$ if $\ell(x) < \ell(v)$ and set $\ell'(x) = \ell(x) - 1$ if $\ell(x) > \ell(v)$. Let us consider the effect of this action on the vertex separation at each location of the layout. Clearly, $s_{\ell'}(i) = s_\ell(i)$ for $1 \leq i < \ell(u)$. If there exists a vertex w with $\ell(w) > \ell(u)$ and either $uw \in E$ or $vw \in E$, then $s_{\ell'}(\ell(u)) \leq s_\ell(\ell(u))$. Otherwise, $s_{\ell'}(\ell(u)) \leq s_\ell(\ell(u)) - 1$. Similar arguments establish that $s_{\ell'}(i) \leq s_\ell(i)$ for the ranges $\ell(u) < i < \ell(v)$ and $\ell(v) \leq i < n$. Therefore, the vertex separation of ℓ' does not exceed k and $vs(H) \leq k$.

We conclude that, in any case, H is in F and hence F is minor-closed. It remains only to note that there are trees with arbitrarily large vertex separation. □

Given a graph G and a positive integer k, the $\mathcal{NP}$-complete SEARCH NUMBER problem [Par] asks whether k searchers are sufficient to ensure the capture of a fugitive who is free to move with arbitrary speed about the edges of G, with complete knowledge of the location of the searchers. More precisely, we say that every edge of G is initially *contaminated.* An edge $e = uv$ becomes *clear* either when a searcher is moved from u to v (v to u) while another searcher remains at u (v) or when all edges incident on u (v) except e are clear and a searcher at u (v) is moved to v (u). (A clear edge e becomes recontaminated if the movement of a searcher produces a path without searchers between a contaminated edge and e.) The goal is to determine if there exists a sequence of *search steps* that results in all edges being clear simultaneously, where each such step is one of the following three operations: 1) place a searcher on a vertex, 2) move a searcher along an edge, or 3) remove a searcher from a vertex. It has been reported that SEARCH NUMBER is decidable in $O(n^{2k^2+4k+8})$ time [EST]. As has been independently noted by Papadimitriou [Pap], however, minor-closure can be applied to reduce this bound.

Theorem 7. For any fixed k, SEARCH NUMBER can be decided in $O(n^2)$ time.

Proof. Straightforward, by showing that, for fixed k, the family of "yes" instances is closed under the minor ordering and by observing that there are excluded trees. □

Consider next the $\mathcal{NP}$-complete MAX LEAF SPANNING TREE problem [GJ]. Given a graph G and a positive integer k, this problem asks whether G possesses a spanning tree in which k or more vertices have degree one. This problem can be solved by brute force in $O(n^{2k+1})$ time. (There are $\binom{n}{k}$ ways to select k leaves and $O(n)$ possible adjacencies to consider at each leaf. For each of these $O(n^{2k})$ candidate solutions, the connectivity of the remainder of G can be determined in linear time because there can be at most a linear number of edges.) Although this means that MAX LEAF SPANNING TREE is in $\mathcal{P}$ for any fixed k, we seek to exploit minor-closure so as to ensure a low-degree polynomial running time.

Theorem 8. For any fixed k, MAX LEAF SPANNING TREE can be decided in $O(n^2)$ time.

Proof. Let k denote any fixed, positive integer. Consider a proper subset of the "no" instances, the family F of graphs none of whose connected components has a spanning tree with k or more leaves. F is clearly closed under the minor ordering, from which the theorem follows because one need only test an input graph for connectedness and membership in F. □

4 Exploiting the Immersion Order

An *embedding* of an arbitrary graph G into a fixed *constraint graph* C is an injection $f\colon V(G) \to V(C)$ together with an assignment, to each edge uv of G, of a path from

$f(u)$ to $f(v)$ in C. The *minimum load factor* of G relative to C is the minimum, over all embeddings of G in C, of the maximum number of paths in the embedding that share a common edge in C.

For example, for the case in which C is the infinite-length one-dimensional grid, the minimum load factor of G with respect to C is called the *cutwidth* of G. In the $\mathcal{NP}$-complete MIN CUT LINEAR ARRANGEMENT problem [GJ], we are given a graph G and an integer k, and are asked whether the cutwidth of G is no more than k. Related $\mathcal{NP}$-complete problems address the cutwidth of G relative to C when C is the infinite-length, fixed-width two-dimensional grid (2-D GRID LOAD FACTOR) or when C is the infinite-height binary tree (BINARY TREE LOAD FACTOR).

Theorem 9. For any fixed k and any fixed C, the family of graphs for which the minimum load factor relative to C is less than or equal to k is closed under the immersion ordering.

Proof. Let an embedding f of G in C with load factor no more than k be given. Suppose $H \leq_i G$. If $H \subseteq G$, then the embedding that restricts f to H clearly has load factor no more than k. If H is obtained from G by lifting the edges uv and vw incident at vertex v, then an embedding for H can be defined by assigning to the resulting edge uw the composition of the paths from u to v and from v to w in C. This cannot increase the load factor. □

Corollary. For any fixed k, MIN CUT LINEAR ARRANGEMENT, 2-D GRID LOAD FACTOR and BINARY TREE LOAD FACTOR can be decided in polynomial time.

This result has previously been reported for MIN CUT LINEAR ARRANGEMENT, using an algorithm with time complexity $O(n^{k-1})$ [MaS]. We now prove that it is sometimes possible to employ excluded-minor knowledge on immersion-closed families to guarantee quadratic-time decision complexity.

Theorem 10. For any fixed k, MIN CUT LINEAR ARRANGEMENT, 2-D GRID LOAD FACTOR and BINARY TREE LOAD FACTOR can be decided in $O(n^2)$ time.

Proof. For MIN CUT LINEAR ARRANGEMENT, it is known that there are binary trees with cutwidth exceeding k for any fixed k [CMST]. Let T denote such a tree. Because T has maximum degree three, it follows that $G \geq_m T$ implies $G \geq_i T$. Thus no $G \geq_m T$ can be a "yes" instance (recall that the "yes" family is immersion closed) and we know from [RS2] that all "yes" instances have bounded tree-width. (Tree-width and the associated metric branch-width are defined and related to each other in [RS3].) Now one needs only search for a satisfactory tree-decomposition, using the $O(n^2)$ method of [RS4]. Testing for obstruction containment in the immersion order can be done in linear time on graphs of bounded tree-width, given such a tree-decomposition.

Sufficiently large binary trees are excluded for 2-D GRID LOAD FACTOR as well (recall that both k and the grid-width are fixed).

For BINARY TREE LOAD FACTOR, it is a simple exercise to see that all "yes"

instances have bounded tree-width, by building a tree-decomposition with width at most $3k$ from a binary tree embedding with load factor at most k. (The decomposition tree T can be taken to be the finite subtree of C that spans the image of G. For vertex $u \in V(T)$, the set S_u contains the inverse image of u if one exists, and every vertex $v \in V(G)$ with an incident edge that is assigned a path in C that includes u. It follows that $|S_u| \leq 3k + 1$.) □

5 Other Methods

The application of Theorems 1 through 4 directly ensures polynomial-time decidability. A less direct approach relies on the well-known notion of polynomial-time transformation, as we now illustrate with an example. The $\mathcal{NP}$-complete MODIFIED MIN CUT problem was first introduced in [Le]. Given a linear layout ℓ of a simple graph G, the *modified cutwidth* at location i, $c_\ell(i)$, is $|\{e : e = uv \in E \text{ such that } \ell(u) < i \text{ and } \ell(v) > i\}|$. The modified cutwidth of the entire layout is $c_\ell = \max\{c_\ell(i) : 1 \leq i \leq n\}$, and the modified cutwidth of G is $mc(G) = \min\{c_\ell : \ell \text{ is a linear layout of } G\}$. Given both G and a positive integer k, the MODIFIED MIN CUT problem asks whether $mc(G)$ is less than or equal to k. Observe that, while the MIN CUT LINEAR ARRANGEMENT problem addresses the number of edges that cross any cut *between* adjacent vertices in a linear layout, the MODIFIED MIN CUT problem addresses the number of edges that cross (and do not end at) any cut *on* a vertex in the layout.

When k is fixed, neither the family of "yes" instances nor the family of "no" instances for MODIFIED MIN CUT is closed under either of the available orders. Nevertheless, we can employ a useful consequence of well-partially-ordered sets.

Consequence. [FL2] If $(S, \leq)$ is a well-partially-ordered set that supports polynomial-time order tests for every fixed element of S, and if there is a polynomial-time computable map t: $D \rightarrow S$ such that for $F \subseteq D$

a) $t(F) \subseteq S$ is closed under $\leq$ and
b) $t(F) \cap t(D - F) = \emptyset$

then there is a polynomial-time decision algorithm to determine for input z in D whether z is in F.

To use this result on fixed-k MODIFIED MIN CUT, observe that if any vertex of a simple graph G has degree greater than $2k + 2$, then G is automatically a "no" instance. Given a simple graph G with maximum degree less than or equal to $2k + 2$, we first augment G with loops as follows: if a vertex v has degree $d < 2k + 2$, then it receives $(2k + 2) - d$ new loops. Letting G' denote this augmented version of G, we now replace G' with the Boolean matrix M, in which each row of M corresponds to an edge of G' and each column of M corresponds to a vertex of G'. That is, M has $|E'|$ rows and n columns, with $M_{ij} = 1$ if and only if edge i is incident on vertex j. M and $k' = 3k + 2$ are now viewed as input to the GATE MATRIX LAYOUT problem [DKL], in which we are asked whether the columns of M can be permuted so that, if in each row we change to $*$ every

0 lying between the row's leftmost and rightmost 1, then no column contains more than k 1s and $*$s. Thus a permutation of the columns of M corresponds to a linear layout of G. For such a permutation, each $*$ in column i, $1 < i < n$, represents a distinct edge crossing a cut at vertex i in the corresponding layout of G.

Theorem 11. For any fixed k, MODIFIED MIN CUT can be decided in $O(n^2)$ time.

Proof. We apply the consequence, using the set of all graphs for S, $\leq_m$ for $\leq$, the set of simple graphs of maximum degree $2k+2$ for D, the family of "yes" instances in D for F, and the composition of the map just defined from graphs to matrices with the map of [FL1] from matrices to graphs for t. Testing for membership in D and computing t are easily accomplished in $O(n^2)$ time. That $t(F)$ is closed under $\leq_m$ and excludes a planar graph for any fixed k is established in [FL1]. Finally, condition b) holds because, for any G in D, $t(G)$ is a "yes" instance for GATE MATRIX LAYOUT with parameter $3k+2$ if and only if G is a "yes" instance for MODIFIED MIN CUT with parameter k. □

6 Search Problems

Given a decision problem Π_D and its search version Π_S, any method that pinpoints a solution to Π_S by repeated calls to an algorithm that answers Π_D is termed a *self-reduction.* This simple notion has been formalized with various refinements in the literature, but the goal remains the same: use the existence of a decision algorithm to prove the existence of a search algorithm.

It sometimes suffices to *fatten up* a graph by adding edges to isolate a solution. For example, this strategy can be employed to construct solutions to (fixed-k) GATE MATRIX LAYOUT, when any exist, in $O(n^4)$ time [BFL]. It follows from the proof of Theorem 11 that the same can be said for MODIFIED MIN CUT as well. We leave it to the reader to verify that such a scheme works for the search version of (fixed-k) VERTEX SEPARATION, by attempting to add each edge in $V \times V - E$ in arbitrary order, retaining in turn only those whose addition maintains a "yes" instance, and at the end reading off a satisfactory layout (from right to left) by successively removing a vertex of smallest degree. This self-reduction automatically solves the search version of SEARCH NUMBER too (see the discussion of "2-expansions" in [EST]).

Conversely, it is sometimes possible to *trim down* a graph by deleting edges so as to isolate a solution. It is easy to see that this simple strategy yields an $O(n^4)$ time algorithm for the search version of (fixed-k) MAX LEAF SPANNING TREE, by attempting to delete each edge in E in arbitrary order, retaining in turn only those whose deletion does not maintain a "yes" instance.

Another technique involves the use of graph *gadgets.* A simple gadget, consisting of two new vertices with k edges between them, is useful in constructing a solution to (fixed-k) MIN CUT LINEAR ARRANGEMENT, when any exist, in $O(n^4)$ time [BFL]. A similar use of gadgets enables efficient self-reductions for load factor problems. (On BINARY TREE LOAD FACTOR, for example, one can begin by using two k-edge gadgets uv and

wx to locate a vertex y of the input graph that can be mapped to a leaf of the constraint tree by identifying u, w and y.)

In addition to these rather straightforward techniques, faster but more elaborate methods are described in [FL4, FL5].

7 Concluding Remarks

The range of problems amenable to an approach based on well-partially-ordered sets is remarkable. Although the problems we have addressed in this paper are all fixed-parameter versions of problems that are $\mathcal{NP}$-hard in general, we remind the reader that by fixing parameters one does not automatically trivialize problems and thereby obtain polynomial-time decidability (consider, for example, GRAPH k–COLORABILITY [GJ]). Moreover, the techniques we have employed can be used to guarantee membership in $\mathcal{P}$ for problems that have no associated (fixed) parameter [FL2].

The results we have derived here immediately extend to hypergraph problem variants as long as hypergraph instances can be efficiently reduced to graph instances. For example, such reductions are known for HYPERGRAPH VERTEX SEPARATION and HYPERGRAPH MODIFIED MIN CUT [MiS, Su]. Nevertheless, Table 1 suffers from one notable omission, namely, BANDWIDTH [GJ]. The only success reported to date has concerned restricted instances of TOPOLOGICAL BANDWIDTH. Both BANDWIDTH and the related EDGE BANDWIDTH problem [FL3] have so far resisted this general line of attack. Clearly, BANDWIDTH is at least superficially similar to other layout permutation problems we have addressed, and fixed-k BANDWIDTH, like the others, is solvable in (high-degree) polynomial-time with dynamic programming [GS]. But perhaps BANDWIDTH really is different; it is one of the very few problems that remain $\mathcal{NP}$-complete when restricted to trees [GGJK].

Finally, we observe that even partial-orders that fail to be well-partial-orders (on the set of all graphs) may be useful. For example, although it is well known that graphs are not well-partially-ordered under the topological order, it has been shown [Ma2] that all graphs without h vertex-disjoint cycles *are* well-partially-ordered under topological containment. Also, polynomial-time order tests exist [RS4]. Problems such as (fixed-k) TOPOLOGICAL BANDWIDTH, therefore, are decidable in polynomial time as long as the input is restricted to graphs with no more than h disjoint cycles (for fixed h). Similarly, one might employ the result [Se] that graphs without a path of length h, for h fixed, are well-partially-ordered under subgraph containment.

8 Bibliography

[BFL] D. J. Brown, M. R. Fellows and M. A. Langston, "Polynomial-Time Self-Reducibility: Theoretical Motivations and Practical Results," *Int'l J. of Computer Mathematics*, to appear.

[CMST] M-J Chung, F. Makedon, I. H. Sudborough and J. Turner, "Polynomial Time Algorithms for the Min Cut Problem on Degree Restricted Trees," *SIAM J. on Computing* 14 (1985), 158–177.

[DKL] N. Deo, M. S. Krishnamoorthy and M. A. Langston, "Exact and Approximate Solutions for the Gate Matrix Layout Problem," *IEEE Trans. on Computer-Aided Design* 6 (1987), 79–84.

[EST] J. A. Ellis, I. H. Sudborough and J. S. Turner, "Graph Separation and Search Number," *Proc. 21st Allerton Conf. on Communication, Control and Computing* (1983), 224–233.

[FL1] M. R. Fellows and M. A. Langston, "Nonconstructive Advances in Polynomial-Time Complexity," *Information Processing Letters* 26 (1987), 157–162.

[FL2] ———, "Nonconstructive Tools for Proving Polynomial-Time Decidability," *J. of the ACM* 35 (1988), 727–739.

[FL3] ———, "Layout Permutation Problems and Well-Partially-Ordered Sets," *Proc. 5th MIT Conf. on Advanced Research in VLSI* (1988), 315–327.

[FL4] ———, "Fast Self-Reduction Algorithms for Combinatorial Problems of VLSI Design," *Proc. 3rd Aegean Workshop on Computing* (1988), 278–287.

[FL5] ———, "On Search, Decision and the Efficiency of Polynomial-Time Algorithms," *Proc. 21st ACM Symp. on Theory of Computing* (1989), 501–512.

[GGJK] M. R. Garey, R. L. Graham, D. S. Johnson and D. E. Knuth, "Complexity Results for Bandwidth Minimization," *SIAM J. on Applied Mathematics* 34 (1978), 477–495.

[GJ] M. R. Garey and D. S. Johnson, Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness, Freeman, San Francisco, CA, 1979.

[GS] E. M. Gurari and I. H. Sudborough, "Improved Dynamic Programming Algorithms for Bandwidth Minimization and the Min Cut Linear Arrangement Problem," *J. of Algorithms* 5 (1984), 531–546.

[Le] T. Lengauer, "Black-White Pebbles and Graph Separation," *Acta Informatica* 16 (1981), 465–475.

[Ma1] W. Mader, "Hinreichende Bedingungen für die Existenz von Teilgraphen, die zu einem vollständigen Graphen homöomorph sind," *Math. Nachr.* 53 (1972), 145–150.

[Ma2] ———, "Wohlquasigeordnete Klassen endlicher Graphen," *J. Combinatorial Theory Series B* 12 (1972), 105–122.

[Ma3] ———, "A Reduction Method for Edge-Connectivity in Graphs," *Annals of Discrete Mathematics* 3 (1978), 145–164.

[MaS] F. S. Makedon and I. H. Sudborough, "On Minimizing Width in Linear Layouts," *Lecture Notes in Computer Science* 154 (1983), 478–490.

[MiS] Z. Miller and I. H. Sudborough, "Polynomial Algorithms for Recognizing Small Cutwidth in Hypergraphs," *Proc. 2nd Aegean Workshop on Computing* (1986), 252–260.

[Na] C. Nash-Williams, "On Well-Quasi-Ordering Infinite Trees," *Proc. Cambridge Philosophical Society* 61 (1965), 697–720.

[Pap] C. H. Papadimitriou, private communication.

[Par] T. D. Parsons, "Pursuit-Evasion in a Graph," in Theory and Application of Graphs (Y. Alavi and D. R. Lick, eds.), Springer-Verlag, 1976, 426–441.

[Rob] N. Robertson, private communication.

[RS1] N. Robertson and P. D. Seymour, "Graph Minors IV. Tree-Width and Well-Quasi-Ordering," *J. Combinatorial Theory Series B*, to appear.

[RS2] ———, "Graph Minors V. Excluding a Planar Graph," *J. Combinatorial Theory Series B* 41 (1986), 92–114.

[RS3] ———, "Graph Minors X. Obstructions to Tree-Decomposition," to appear.

[RS4] ———, "Graph Minors XIII. The Disjoint Paths Problem," to appear.

[RS5] ———, "Graph Minors XVI. Wagner's Conjecture," to appear.

[Se] P. D. Seymour, private communication.

[Su] I. H. Sudborough, private communication.

[Wa] K. Wagner, "Uber Einer Eingeshaft der Ebener Complexe," Math. Ann. 14 (1937), 570–590.

9 Footnote

- A preliminary version of a portion of this paper [FL3] was presented at the Fifth MIT Conference on Advanced Research in VLSI held in Cambridge, Massachusetts, in March, 1988.

- Michael R. Fellow's research has been supported in part by the National Science Foundation under grants MIP–8603879 and MIP–8919312, by the Office of Naval Research under contract N00014–88–K–0456, and by the National Aeronautics and Space Administration under engineering research center grant NAGW–1406.

- Michael A. Langston's research has been supported in part by the National Science Foundation under grants MIP–8603879 and MIP–8919312, and by the Office of Naval Research under contract N00014–88–K–0343.

Burst Error Correction Extensions for LARGE Reed Solomon Codes

P. Owsley
Advanced Hardware Architectures
Moscow, Idaho 83843

Abstract– **Reed Solomon codes are powerful error correcting codes that include some of the best random and burst correcting codes currently known. It is well known that an (n,k) Reed Solomon code can correct up to $(n-k)/2$ errors. Many applications utilizing Reed Solomon codes require correction of errors consisting primarily of bursts. In this paper, it is shown that the burst correcting ability of Reed Solomon codes can be increased beyond $(n-k)/2$ with an acceptable probability of miscorrect.**

1 Decoding Burst Errors in a Reed Solomon Code

The random error correcting ability of a code is set by the minimum Hamming distance of the code, d^*. The Hamming distance between two codewords is the number of symbols in which the codewords are different. A code can correct all error patterns of t or fewer symbols where

$$2t \leq d^* - 1 \tag{1}$$

An upper bound for the minimum distance of a code given the number of parity symbols added to form a linear (n,k) code is given by the Singleton Bound

$$d^* - 1 \leq n - k \tag{2}$$

The minimum number of random errors that a code can correct is found at the point of equality. Any code that meets the bound with equality is said to be a Maximum Distance code. Combining equations 1 and 2 gives

$$2t \leq d^* - 1 \leq n - k$$

with the result,

$$2t \leq n - k \tag{3}$$

Equation 3 says that a code that corrects all patterns of t or fewer symbol errors requires at least $2t$ parity symbols.

The burst correcting ability of a linear code can also be related to the number of parity symbols added to form the (n,k) codeword. The Rieger Bound [2] states: "In order to correct all bursts of length t or less, a linear block code must have at least $2t$ parity symbols." Any code that meets the Singleton Bound with equality also meets the Rieger Bound with equality. The Reed Solomon code is such a code.

A maximum distance code, such as the Reed Solomon code, can correct all combinations of $(n-k)/2$ random symbol errors. It can also correct all bursts of length $(n-k)/2$ symbols, which is a subset of the random error patterns that can be corrected. The Reed Solomon code can correct t errors, either randomly placed or contiguously placed as in a burst.

Many applications that require burst error correction use a Reed Solomon code, however there is a significant amount of information that is not being used by the code in a burst error environment. When t errors occur at random, there are 2t unknowns: the t locations and the t magnitudes. However, when a burst of length t occurs, then the t magnitudes and the location of the burst are unknown. This accounts for only t+1 variables that need to be solved for. Less information is required to solve for the burst.

In this section, Reed Solomon codes are studied as burst error correcting codes. The amount of extra information that is available when the errors are known to be bursts a priori is found. The added burst correcting ability of the code is studied and the cost of increasing the burst length to be corrected beyond $(n-k)/2$ is found.

1.1 Definitions and Conjectures

To begin the study, some terms need to be defined. Also, some assumptions concerning the occurrence of burst errors are stated.

Definition 1 *Reed Solomon codes are linear codes and therefore are a subspace of* $GF(q)^n$ *[1].*

$GF(q)^n$ is the n dimensional vector space over the finite field $GF(q)$. As a linear code, Reed Solomon codes satisfy the properties of a group. As such, the difference between any two codewords is also a codeword. Consequently, the relationship of any particular codeword to all other points in the space is isomorphic to the relationship of any other codeword to the vector space. The distance properties of any one codeword to all other codewords is also isomorphic to the distance properties for all codewords. Because of this, a study of the distance properties of any single codeword provides all the distance information for the complete code.

Definition 2 *A burst polynomial,* $b(x)$*, representing a burst of length* l*, located at position* c *is defined as follows:*

$$b(x) = \sum_{i=0}^{n-1} b_i x^i \tag{4}$$

where $b_i = 0$ *for* $i < c$ *and* $i > c + l - 1$, $b_c \neq 0$, *and* $b_{c+l-1} \neq 0$.

A burst can have many zero coefficients within the burst itself. The length of the burst is the number of coefficients from the first error symbol to the last error symbol inclusive in the block. A burst is considered to be located at the position of its least significant coefficient.

Definition 3 *The burst distance* d_b *between any two codewords is the burst length of the difference between the two words.*

This is analogous to the Hamming distance [1] between two code words. The Hamming distance between two codewords is the number of symbols in which they differ. The burst distance between two codewords is the smallest burst that can be added to one to get the other.

The Hamming distance measures distance with no constraints on location, much as a random noise source corrupts data with equal probability. The Hamming distance is a metric for the random error correcting ability of an error correcting code. The burst distance measures distance with the additional constraint that the locations must be consecutive. The burst distance is a metric for the burst error correcting ability of an error correcting code.

Theorem 1 *The minimum burst distance d_b^* of a linear code is the burst length of the nonzero codeword with smallest burst length.*

Proof For a linear code, the difference between any two codewords is another codeword. Therefore, the minimum burst distance of the code d_b^* is equal to the burst length of the nonzero codeword with the smallest burst length. □

This is analogous to the minimum Hamming distance of a code. A burst error correcting code can correct all burst patterns of length t or less where

$$2t \leq d_b^* - 1 \tag{5}$$

The Singleton Bound still holds for burst distance, because the burst distance between two codewords can never be greater than the Hamming distance. Therefore

$$d_b^* - 1 \leq n - k \tag{6}$$

Combining equations 5 and 6 gives

$$2t \leq d_b^* \leq n - k$$

with the result

$$2t \leq n - k \tag{7}$$

which is the Rieger bound.

As seen previously, the Rieger Bound states that a code cannot correct all bursts of length t if less than $2t$ parity are added to form the codeword.

Definition 4 *A burst shell of radius r contains all of the points that are equal to a burst distance of r from the code polynomial at the center.*

Definition 5 *A burst sphere of radius r contains all of the points that are a burst distance of r or less from the code polynomial at the center.*

A "normal" sphere, as distinguished from a burst sphere, of radius r contains all points that are Hamming distance r or less from the center point. A burst sphere of radius t is a subset of a sphere of radius t, since bursts of length t or less are included in the set of words of weight t or less. It should be noted that a point in the sphere of radius two could be a point in the outer shell of the burst sphere of radius $n/2$ where n is the blocklength of the code.

Definition 6 *A decoding sphere is a sphere with a codeword as the center point.*

A t random error correcting code corrects all points within a sphere of radius t about the codewords. Likewise for a burst error correcting code, the code corrects all points within the burst sphere.

An error correcting code has a decoding sphere about each of the codewords: a "normal" sphere for a random error correcting code and a burst sphere for a burst correcting code. Current theory assumes that the spheres about different codewords do not overlap. It is the purpose of this work to show that choosing to correct errors of a magnitude such that the decoding spheres overlap is of benefit. Because there is an overlap, there will be points in the space that could be corrected to more than one code word. Any point within this intersection region of two decoding spheres could be corrected to the codeword at the center of either sphere. Decoding decisions for these points are presented in a Section 1.2.

Definition 7 *l_t is the radius of the largest non-overlapping burst spheres.*

For Reed Solomon codes l_t is equal to $(n-k)/2$

If the radii of the burst decoding spheres is allowed to grow beyond l_t, then the spheres begin to overlap. Overlap occurs when the Rieger bound is violated. Points within the area of overlap might not be correctable because they belong to more than one sphere. If a point that is at burst distance greater than t from a code word, but an element of only one sphere, then that point can be corrected unambiguously. The following term is be used to define the radius of the expanded sphere.

Definition 8 *The maximum burst that the code attempts to correct is l_{max}.*

l_{max} is the length of the longest burst that the decoder attempts to correct. As l_{max} increases beyond l_t, the burst spheres begin to overlap. Bursts of length greater than l_{max} should be detected by the decoder as an uncorrectable error condition.

Since the spheres are allowed to overlap, the decoder will not always be able to correct bursts of length l_{max}. The following two definitions name the conditions when the decoder fails. The decoder is designed to correct bursts of length l_{max}.

Definition 9 *A miscorrect occurs when the decoder fails to correct to the right codeword when a correctable burst of length l_{max} or less has occurred.*

Definition 10 *A misdetect occurs when a burst of length greater than l_{max} has occurred and the decoder corrects to a codeword.*

When an error occurs that is larger than the code can correct, i.e. greater than l_{max}, then the desired decoding action would be error detection. If the received vector is in a decoding sphere and a decision to correct is made, then a misdetect has occurred. To summarize, if a burst error of length less than or equal to l_{max} occurs and the decoder does not correct to the right codeword, then a miscorrect has occurred. If a burst error of length greater than l_{max} occurs, and the decoder corrects to a codeword, then a misdetect has occurred.

Definition 11 *The minimum sized burst that can be miscorrected is l_{min}.*

The length l_{min} is the radius of the points that do not overlap with any other sphere. This does not mean that all points of distance greater than l_{min} overlaps with other spheres, but all points of distance less than or equal to l_{min} do not overlap with any other spheres.

For $n-k$ even,

$$l_{min} = n - k - l_{max} + 1 \tag{8}$$

For $n-k$ odd,

$$l_{min} = n - k - l_{max} \tag{9}$$

Conjecture 1 *All bursts of length l_i are equally likely.*

If this is not true, then a code can be created that puts the more likely burst patterns into more favorable regions. The arguments developed in this section assume that burst errors of identical length are equally likely.

Conjecture 2 *For all l_m and l_n, and for some l_t, such that $l_t \leq l_n$, and $l_m < l_n$ the probability of a burst of length l_n is less than the probability of a burst of length l_m.*

The second conjecture assumes that longer bursts are less likely than shorter bursts as long as the length of the longer burst is greater than l_t.

1.2 Decoding Decisions for Points in the Overlap Region

Any burst pattern that moves a codeword into one of the regions of overlap is a pattern that cannot unambiguously be detected. However it might be possible to identify all of the spheres of which it is an element. If a received word is in the area of overlap, then there are five possibilities.

1. The received word is within a burst sphere of radius l_t with the sent codeword at the center of the sphere. By Conjecture 2, the decision should be made to correct to the closer codeword and the decoder would function correctly.

2. The burst distance between the received word and the sent codeword is less than the burst distance between the received word and any other codeword. If all of the spheres can be identified, then by Conjecture 2, the shorter burst should be chosen, and again the decoder would function correctly.

3. The burst distance between the received word and the sent codeword is less than or equal to the burst distance between the received word and any other codeword and equal to at least one of the other burst distances. In this case, an unambiguous decoding choice can not be made. However, the burst distance between the received word and the sent codeword is less than or equal to l_{max} and should have been correctable. The proper decoding choice is to detect an uncorrectable error. This is a miscorrect in the sense that the burst is a length that is considered correctable, but the decoder cannot make an unambiguous decoding decision.

4. The burst distance between the received word and the sent codeword is greater than the burst distance between the received word and another codeword, but less than l_{max}. The decoder will not correct to the sent codeword and a miscorrect occurs.

5. The burst distance between the received word and the sent codeword is greater than l_{max} but within another burst decoding sphere. The error should be detected, but will be corrected to the other codeword. This is a misdetect. It should be noted that all decoders suffer from this problem.

The next task is to find the probability of miscorrect given that a burst error of length l_j has occurred. This is done by finding the number of burst errors of length l_j that exist. Then the number of those errors that result in miscorrects is found. The ratio between the two is the probability of miscorrect given that a burst of length l_j has occurred. The first number is equal to the volume of the burst shell of radius l_j. The second number is equal to the volume of the overlap region of that shell with all other shells with radius less than or equal to l_j.

1.3 Volumes for Burst Shells

Two volumes need to be quantified. The first is the volume of the shell of burst radius l_j. This represents the number of ways that a burst error of length l_j can occur. The second volume is the intersection of a burst shell of radius l_j with all burst shells of radius l_k. This is the number of potential received words within the first shell that are a distance l_k from another codeword.

1.3.1 Burst Shell Volume

The objective in this section is to find the volume of a burst shell about a code polynomial. The shell volume is a function of three parameters:

1. n, the blocklength of the code.

2. q, the number of symbols in the field over which the code is defined.

3. l_j, the radius of the shell.

For a full length Reed Solomon code, $n = q - 1$.

Any burst of length l_j added to the codeword at the center results in a point that is in the shell. Each point represents a received word that is burst distance l_j from the sent codeword and the sum of all such points in the shell is the volume of the shell.

Consider the all zero codeword. The volume of the shell of burst radius l_j about the all zero codeword is equal to the number of bursts of length l_j. The method for finding the total number of bursts of length l_j is combinatoric. The following theorem gives the volume of a shell of burst radius l_j.

Theorem 2 *The volume of V_{l_j}, a burst shell of burst radius l_j is:*

$$V_0 = 1 \tag{10}$$

$$V_1 = n(q-1) \tag{11}$$

and for $l_j > 1$

$$V_{l_j} = (n - l_j + 1)(q-1)^2 q^{l_j - 2} \tag{12}$$

Proof The shell of burst radius 0 includes only one point, therefore the volume is 1. This would correspond to no errors.

The shell of burst radius 1 contains all of the bursts of length 1. There are n possible locations and in each location the value must be non-zero, therefore there are $q-1$ possible values for each location. This would correspond to one random error.

For a burst of length greater than 1, there are $n - l_j + 1$ different ways to place the burst, $b(x)$. For each placement of the burst, the two endpoints of the burst which are the coefficients b_0 and b_{l_j-1}, must be nonzero, therefore there are $(q-1)^2$ ways to choose the two endpoints. The $l_j - 2$ interior points of the burst which are the coefficients b_1 through b_{l_j-2}, can take on any of q values. Therefore there are q^{l_j-2} ways to choose the interior points for each burst location. Moreover, the total volume for a burst shell of radius l_j is $(n - l_j + 1)(q-1)^2 q^{l_j-2}$. □

1.3.2 Volume of Overlap

Points that are in the overlap region are the points that can cause a decoding failure. If the volume of the region is small compared to the total volume of the shell, then the probability of landing in the overlap region is also small.

In investigating the overlap region, the codeword at the center of a burst sphere C_j of radius l_j is considered to be the sent codeword; the codeword at the center of any other sphere C_i, $j \neq i$ is considered to be erroneous. The study is from the point of view of the sent codeword. Burst shells about the sent codeword are studied to find how many points in the shell are a burst distance equal to l_k from another codeword where l_k is any burst distance less than or equal to l_j.

The following two definitions define the two different types of bursts.

Definition 12 *The real burst, $b_j(x)$, is the difference between the point in the region of overlap and the sent codeword $c_j(x)$.*

Definition 13 *The phantom burst, $b_k(x)$, is the difference between the point in the region of overlap and a codeword, $c_k(x)$, which is not the sent codeword.*

It will be shown that if the difference of the two bursts is a codeword, then the point is in the overlap region. The probability of miscorrect that is calculated is conditioned on the probability of the real burst occurring. The above two definitions provide a mechanism to distinguish between the actual error and the error that is miscorrected.

The overlap region is the intersection between the two spheres, one of burst radius l_j and the other of burst radius l_k. In this section, the volume of the intersection of the two shells is found as a function of the blocklength n, the two burst lengths l_j and l_k, the size of the symbol space q, and the number of parity symbols $n-k$.

Theorem 3 *For an (n,k) Reed Solomon code, any set of $n-k$ symbols can be expressed as $n-k$ functions dependent on the other k symbols.*

Theorem 4 *The minimum Hamming weight of a Reed Solomon codeword with $n-k$ parity symbols is $n-k+1$.*

Theorem 5 *The difference between two codewords is a codeword.*

The above three theorems have been proved numerous times. For example, see [1][2].

Theorem 6 *If a received polynomial $v_j(x) = c_j(x) + e_j(x)$ is in the region of overlap, then $v_j(x) = c_k(x) + e_k(x)$ where the difference of the two errors is also a codeword $c_i(x)$, $c_j(x) \neq c_k(x)$ and $c_i(x)$ has at least $n-k+1$ non-zero coefficients.*

Proof For the first part of the theorem,

$$c_j(x) + e_j(x) = c_k(x) + e_k(x)$$
$$c_j(x) - c_k(x) = e_k(x) - e_j(x)$$

By Theorem 5,

$$c_j(x) - c_k(x) = c_i(x)$$

where $c_i(x)$ is a codeword. Therefore

$$c_i(x) = e_k(x) - e_j(x)$$

For the second part, by Theorem 4, the minimum Hamming weight of a Reed Solomon codeword with $n-k$ parity is $n-k+1$. If $e_k(x) - e_j(x)$ is a codeword, then its Hamming weight must be greater than or equal to $n-k+1$. □

If the code alphabet is a characteristic 2 alphabet, i.e., the Galois field is GF(2^m) where m is a positive integer, then subtraction in the field is identical to addition. In that case, if the sum of two bursts is a codeword, then either of the bursts added to a codeword will be in the region of overlap.

Theorem 7 *If a point in the region of overlap is a burst distance l_j from a codeword c_j, and a burst distance l_k from a code word c_k, then*

$$l_k + l_j > n - k \tag{13}$$

Proof By theorem 6 the sum of the two bursts must be a codeword. The Hamming weight w_j of b_j must be

$$1 < w_j \leq l_j$$

The Hamming weight w_k of b_k must be

$$1 < w_k \leq l_k$$

Therefore

$$l_j + l_k \geq w_k + w_j$$

The weight of the codeword is $w_k + w_j$. By theorem 4, all codewords have a weight greater than $n - k$. Therefore

$$l_j + l_k \geq w_j + w_k > n - k$$

□

Enough groundwork has now been laid to find the volume of the region of overlap between a burst shell of radius l_j and all other burst shells of radius l_k. This region is defined below.

Definition 14 *$V_{l_j \cap l_k}$ is the intersection of the burst shell of radius l_j about the code polynomial $c_j(x)$ with all other burst shells of radius l_k.*

In the following theorem, let $c_j(x)$ represent the sent codeword, $b_j(x)$ represent the actual burst error, and $v_j(x)$ represent the received word which is in the overlap region. Let $b_k(x)$ represent the phantom burst and let $c_k(x)$ represent the codeword at the center of the intersecting shell of burst radius l_k. In this case

$$v_j(x) = c_j(x) + b_j(x) = c_k(x) + b_k(x).$$

Theorem 8

$$V_{l_j \cap l_k} \leq \left[2 \sum_{i=0}^{l_k - 1} (n - l_j - l_k - i + 1) + (n - 2l_k - l_j + 1)(n - l_j - 2l_k + 2) \right] \times (q^{l_j + l_k - (n-k)} - 1) \quad (14)$$

Proof By Theorem 6, $v_j(x)$ is in the region of overlap if the difference of $b_j(x)$ and $b_k(x)$ is a code polynomial. The right side of Equation 14 consists of two parts. The first part is the number of ways that two bursts, one of length l_j and the other of length l_k, can be placed in a block of length n without overlap. The second part of Equation 14 counts the number of polynomials for each placement that is a code polynomial.

First, there are $n - l_j + 1$ ways that the burst $b_j(x)$ of length l_j can be placed in a block of length n. This is broken into two cases. For the first case, the distance between the edge of the block and $b_j(x)$ is less than l_k. This is true for $2l_k$ of the locations. In this case the number of ways to place the burst $b_k(x)$ is $n - l_j - l_k - i + 1$ where i is the distance between the edge of the block and $b_j(x)$. This is the first term in Equation 14 summed over all $i < l_k$ for both ends of the block.

In the second case, the burst $b_j(x)$ is located far enough from the edge such that $b_k(x)$ can be located on either side of it. There are $n - 2l_k - l_j + 1$ of these locations. For each of these locations there are $n - l_j - 2l_k + 2$ ways to locate $b_k(x)$. This is the second term of Equation 14.

For each of the ways of placing the bursts, there are $l_k + l_j$ coefficients that are included in either of the two bursts. By theorem 6 the sum of the two bursts must be a code word. By theorem 3, for the sum to be a codeword, $n - k$ of the coefficients must be uniquely specified. Of the $(q-1)^4 q^{l_k+l_j-4}$ ways of choosing the 2 bursts, only $q^{l_k+l_j-(n-k)} - 1$ of them are nonzero codewords. The all zero codeword is not a possibility, because a burst did occur. This is the third term of Equation 14.

The number of points in a shell of burst radius l_j that are in the overlap region is the product of the ways to place the bursts and the number that are codewords. □

With the volume of the intersection between a burst shell of radius l_j with any burst shell of radius l_k, it is possible to evaluate the probability of decoding failure. Every point within the overlap region represents one error of burst length l_j that is also a burst error of length l_k from another codeword.

1.4 Probability of Miscorrect and Misdetect

A desirable feature of an error correcting decoder is that uncorrectable errors are detectable. With any error correcting code, there is always the chance that an uncorrectable error can cause a misdetect.

Any error correction code that does not try to correct errors beyond that which the distances in Equations 1 and 5 allow should never miscorrect. However, if correction of errors greater than these are attempted, then there is a possibility of miscorrect. In this section these probabilities are found given that a burst of length l_j which violates the Rieger bound occurs.

1.4.1 Miscorrect

A miscorrect occurs if a burst of length less than or equal to l_{max} is not corrected to the sent codeword. The conditions for which this occurs were outlined in section 1.2. To summarize, the decoder miscorrects if the burst distance from the sent codeword to the received word (the real burst) is greater than or equal to the burst distance to any other codeword (the phantom burst).

Theorem 9 *Given that a burst of length l_j for*

$$l_t < l_j \leq l_{max}$$

has occurred, the conditional probability of a miscorrect is

$$P(\text{miscorrect}|l_j) \leq \sum_{l_k=n-(k-1)-l_j}^{l_j} \frac{V_{l_j \cap l_k}}{V_{l_j}} \tag{15}$$

Proof The points in the region of overlap that cause a miscorrect are those points that are a burst distance less than or equal to l_j from another codeword. If $l_j \leq l_t$, then there is no l_k that can satisfy both bounds on the summation. Any burst of length less than $n-(k-1)-l_j$ has fewer than $n-k$ nonzero coefficients and cannot be a codeword. Any burst of length greater than l_j will not cause a miscorrect because of the assumptions made in Conjecture 2. The numerator is a count of the points that cause miscorrect with a burst of length l_j and the denominator is a count of the total number of points that are in a burst shell of radius l_j. By Conjecture 1, each of the points in the shell are equally likely. □

The conditional probability of miscorrect given a burst of length l_j is not a function of the maximum burst that the decoder will attempt to correct. It is a function of the the length of the burst that is actually found.

The bound given in Theorem 9 can be simplified through approximation. First, for the numerator the following approximations can be made to Equation 15.

$$\begin{aligned} V_{l_j \cap l_k} &\leq \left[2\sum_{i=0}^{l_k-1}(n-l_j-l_k-i+1)+(n-2l_k-l_j+1)(n-l_j-2l_k+2)\right] \\ &\quad \times(q^{l_j+l_k-(n-k)}-1) \\ &< n^2(q^{l_j+l_k-(n-k)}-1) \\ &< q^2 q^{l_j+l_k-(n-k)} \\ &< q^{l_j+l_k+2-(n-k)} \end{aligned}$$

The term in the brackets enumerates the number of ways a burst of length l_j and a burst of length l_k can be placed in a block of length n. There are less than n ways for each burst, therefore the total value of the bracketed term is less than n^2. For a Reed Solomon code, $n \leq q-1$, and n^2 can be replaced with q^2 and the remaining follows. The denominator of Equation 15 is simplified below.

$$\begin{aligned} V_{l_j} &= (n-l_j+1)(q-1)^2 q^{l_j-2} \\ &< nq^2 q^{l_j-2} \\ &< q^{l_j+1} \end{aligned}$$

The bound given in Equation 15 can be approximated as

$$P(\text{miscorrect}|l_j) \approx \sum_{l_k=n-(k-1)-l_j}^{l_j} q^{-(n-k-l_k-1)} \tag{16}$$

Equation 16 *is not* a bound but can be used as a rough guess.

Finally, the greatest conditional probability of miscorrect occurs when a burst that is the same length as the real error occurs. In this case the bound is

$$P(\text{miscorrect}|l_j) \approx q^{-(n-k-l_j-1)} \tag{17}$$

Since the approximations of both the numerator and the denominator of Equation 15 are less than the original quantities, it is not possible to state that the approximation given in Equation 17 is less than the conditional probability. However, it appears that if $q \gg l_j$, the approximation is a good one.

¿From the above, once the channel error statistics are known, the conditional probability of miscorrect can be calculated. This conditional probability can then be used to specify the error correction scheme for the channel.

1.4.2 Misdetect

A misdetect can occur when a burst of length greater than l_{max} occurs. A burst of this length is considered uncorrectable. If the burst vector moves the original codeword into another coding sphere, then the decoder corrects to the wrong codeword instead of detecting an uncorrectable error.

Theorem 10 *Given a burst of length greater than l_{max} the conditional probability of a misdetect given a burst decoding sphere of radius l_{max} for an (n,k) code is*

$$P(\text{misdetect}|l_{max}) < \frac{q^k \sum_{l_j=0}^{l_{max}} V_{l_j}}{q^n} P(l_j > l_{max}) \tag{18}$$

Proof The numerator is an upper bound on the total number of polynomials within all of the burst decoding spheres of radius l_j. Some polynomials belong to more than one sphere. The summation is the volume of one sphere. There are q^k codewords and consequently q^k spheres. The denominator is the total number of polynomials. □

The conditional probability of misdetect is a strong function of the maximum length burst that the decoder attempts to correct.

1.5 Example

It is now possible to determine the conditional probability of miscorrect for a decoder that violates the Rieger Bound for an (n,k) Reed Solomon code.

Example 1 *The (255,223) Reed Solomon code over $GF(2^8)$ is used to protect satellite communication channels [4]. It is concatenated with a convolutional inner code. The purpose of the convolutional code is to correct random errors. When the raw bit error rate is too high for the convolutional decoder, it creates a burst of errors in the output. This data is then passed to a Reed Solomon decoder for burst correction. The code corrects all bursts of length 16 that occur within a block. If a burst of length greater than 16 occurs, then the conditional probability of a misdetect from Equation 18 is less than $4.7(10)^{-14}$. This means that a miscorrect occurs with a probability*

$$P(\text{misdetect}) < 4.7(10)^{-14} P(l_j > 16).$$

Given that the code is being used on a burst channel, the number of errors that the code can correct could be increased. The bound on the conditional probabilities of miscorrect can be found through direct application of Equation 15. The approximation conditional as given by Equation 17 is given as a reference. As can be seen in the table, the approximation is greater than the actual conditional probability. It is also well within an order of magnitude of the value it is approximating. The results are summarized in Table 1. One interesting

Burst length l_j	P(miscorrect$\|l_j$) *Eq. 2.15*	P(miscorrect$\|l_j$) *Eq 2.17*
17	$1.4(10)^{-34}$	$1.9(10)^{-34}$
18	$3.6(10)^{-32}$	$4.9(10)^{-32}$
19	$9.1(10)^{-30}$	$1.3(10)^{-29}$
20	$2.3(10)^{-27}$	$3.2(10)^{-27}$
21	$5.8(10)^{-25}$	$8.3(10)^{-25}$
22	$1.4(10)^{-22}$	$2.1(10)^{-22}$
23	$3.6(10)^{-20}$	$5.4(10)^{-20}$
24	$9.1(10)^{-18}$	$1.4(10)^{-17}$
25	$2.3(10)^{-15}$	$3.6(10)^{-15}$
26	$5.7(10)^{-13}$	$9.1(10)^{-13}$
27	$1.4(10)^{-10}$	$2.3(10)^{-10}$
28	$3.6(10)^{-8}$	$6.0(10)^{-8}$
29	$9.0(10)^{-6}$	$1.5(10)^{-5}$
30	$2.3(10)^{-3}$	$3.9(10)^{-3}$
31	1.0	1.0

Table 1: Probability of miscorrect for a (255,223) Reed Solomon code with the given burst lengths.

result is that the conditional probability of miscorrect for a burst of length 25 is less than the conditional probability of misdetect for a normal decoder. This is possible because the normal decoder allows for the occurrence of random errors.

Another interesting problem would be how many parity symbols are really needed to correct a burst of length 16 with a conditional probability of miscorrect less than 10^{-10}***? A*** $(255, 234)$ ***code requiring*** 21 ***parity symbols has a conditional probability of miscorrect equal to*** $1.8(10)^{-10}$***. A common metric for measuring the efficiency of a code is the rate*** $R = k/n$***. The rate of the original Reed Solomon code is*** 87%***. The new rate is*** 92%***, a significant improvement.***

1.6 Summary

For a burst error environment, the error correcting ability of a Reed Solomon code can be extended beyond the Rieger bound with a high degree of confidence that the bursts that

are found are the bursts that occurred. This is significant in that the code rate of a code can be reduced without much if any reduction in the burst correcting ability of a code that only corrects bursts that meet the Rieger Bound.

The decoder that performs this burst correcting does not correct any random errors that occur outside of the bursts that are being corrected. One very common use for a burst correcting code is as an outer code for a random error correcting code as illustrated in the above example. When the inner random error correcting code fails, it creates a large burst error, but the inner code corrects all of the random errors. In this case, the increased burst correcting ability of the Reed Solomon codes is valuable.

When the size of the bursts to be corrected is increased such that the Rieger bound is violated, the possibility of a miscorrect is non-zero. Significant improvement of the burst error correcting ability of a Reed Solomon code can be accomplished while maintaining a negligible conditional probability of miscorrect given that a burst of l_{max} occurs. A bound on the conditional probability of miscorrect given that an error of magnitude l_{max} occurs was found to be approximated by $q^{-(n-k-l_{max}-1)}$.

2 A Decoder for Bursts that Violate the Rieger Bound

Reed Solomon codes are a special case of the BCH codes. Any decoding algorithm that works for BCH codes also works for the Reed Solomon codes.

Reed Solomon decoding is a computationally complex process. Since the first decoding algorithms were defined, most of the research in the area has been focused at reducing the complexity rather than improving the correcting ability of the code. The results in the previous section indicate that in a burst error environment, the error correcting ability can be improved beyond what the Rieger bound would indicate.

The possibility of extending the burst correcting ability of a Reed Solomon code was developed in the last section. For a given amount of information and a given maximum burst size to be corrected, the number of parity symbols could be reduced significantly to achieve essentially the same burst correcting ability.

Error trapping, a decoding algorithm first identified in 1964 by Rudolph and Mitchell [3], decodes extended burst errors. This algorithm identifies the error polynomial $e(x)$ and then corrects the received polynomial by subtracting $e(x)$ to get the code polynomial. In the following sub-sections, the error trapping algorithm and a decoder for trapping extended bursts is described.

2.1 The Syndrome

The polynomial received by the decoder, $v(x)$, is the sent codeword added to the error induced by the channel.

$$v(x) = c(x) + e(x) \tag{19}$$

As can be seen in Equation 19, the error is additive and once found, can be subtracted from $v(x)$ to get the original code word.

The syndrome, as described below, is a function of the error polynomial and is independent of the sent code word. There are two forms of the syndrome: the partial syndromes and the syndrome polynomial. Most of the current methods for decoding Reed Solomon codes use partial syndromes. The partial syndromes are discussed in a later section. The error trapping decoder is based on the syndrome polynomial.

Definition 15 *The syndrome polynomial $s(x)$ is given by the equation*

$$s(x) = R_{g(x)}[v(x)] \tag{20}$$

The syndrome generator for calculation of $s(x)$ is developed in Section 2.3.

The syndrome is a function only of the error polynomial and not of the code polynomial. This is true only because $g(x)$ divides $c(x)$.

$$\begin{aligned} s(x) &= R_{g(x)}[c(x) + e(x)] && (21)\\ &= R_{g(x)}[c(x)] + R_{g(x)}[e(x)] && (22)\\ &= 0 + R_{g(x)}[e(x)] && (23)\\ &= R_{g(x)}[e(x)] && (24) \end{aligned}$$

¿From Equation 24 it can be seen that when the degree of the error polynomial is less than $n-k$, the syndrome polynomial is equal to the error polynomial. When the syndrome polynomial equals the error polynomial, the special condition known as the error trap occurs. An equivalent statement is when the error pattern is wholly contained within the $n-k$ lowest degree coefficients, then the error is trapped.

All bursts of length l_j such that

$$l_j < 2t$$

are trapped if the $n-k-l_j$ coefficients that are not part of the burst are equal to zero.

If a burst is not wholly contained in locations 0 through $n-k-1$, then it is not trapped. However, if the received polynomial can be cyclically shifted, and the syndrome for that shifted polynomial found, then any burst error can be trapped. The Meggitt Theorem provides such a mechanism for cyclically shifting the received polynomial and updating the syndrome polynomial to correspond to the new polynomial.

2.2 The Meggitt Theorem and Trapping the Error

The major significance of the Meggitt Theorem is that it allows for a simple method of calculating the syndrome of a cyclically shifted codeword given that the syndrome of the unshifted codeword is known. The following is the Meggitt Theorem as stated by Blahut [1].

Theorem 11 *If*

$$g(x) | (x^n - 1)$$

and

$$R_{g(x)}[v(x)] = s(x)$$

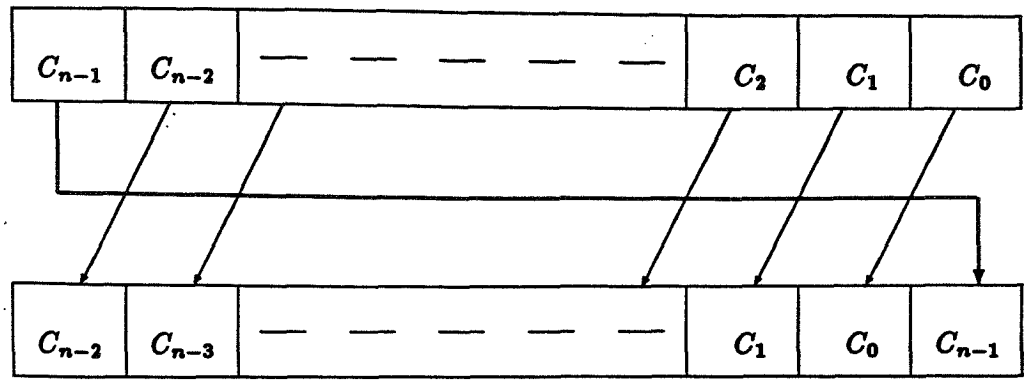


Figure 1: Cyclic shift caused by a multiplication by x mod $(x^n - 1)$.

then

$$R_{g(x)}[R_{x^n-1}\{xv(x)\}] = R_{g(x)}[xs(x)] \tag{25}$$

The left side of Equation 25 is the received polynomial cyclically shifted to the left. The most significant coefficient becomes the least significant and the degree of all of the other coefficients is increased by one. This is illustrated in Figure 1.

On the right side of the equation, the syndrome for the new, cyclically shifted polynomial is found. The original syndrome has been multiplied by x and the residue with respect to $g(x)$ found. This can be done as follows:

$$R_{g(x)}[xs(x)] = xs(x) - s_{n-k-1}g(x) \tag{26}$$

where s_{n-k-1} is the most significant coefficient of $s(x)$.

In Section 2.3 it is shown that the syndrome generator accomplishes this operation.

Theorem 12 *If*

$$v_j(x) = c_j(x) + x^r b_j(x)$$

where

$$|b_j(x)| < n - k - 1$$

then

$$R_{g(x)}\{R_{x^n-1}[x^{-r}v_j(x)]\} = b_j(x) \tag{27}$$

Proof

$$\begin{aligned} R_{g(x)}\{R_{x^n-1}[x^{-r}v_j(x)]\} &= R_{g(x)}\{R_{x^n-1}[x^{-r}(x^{-r}c(x) + x^{-r}x^r b(x)]\} && (28)\\ &= R_{g(x)}\{R_{x^n-1}[x^{-r}(x^{-r}c(x)]\} + R_{g(x)}\{R_{x^n-1}[b_j(x)]\} && (29)\\ &= 0 + R_{g(x)}\{R_{x^n-1}[b_j(x)]\} && (30) \end{aligned}$$

□

Theorem 12 specifies the direction a codeword with an additive burst error must be shifted to get the burst within the window. If the burst is offset from the zeroth coefficient by r places, then $v(x)$ must be divided by $x^r \bmod x^n - 1$ to get the syndrome equal to the burst. This is illustrated in Figure 2. A polynomial is multiplied by $x \bmod x^n - 1$ and is rotated counter clockwise in the figure. An $n-k$ coefficient window, located on coefficients 0 through $p-1$ is fixed where p is the number of parity symbols for the code. The coefficients are shifted through the window in the direction shown.

2.3 Implementation Considerations

The functions that an error trapping decoder has to implement are calculation of the syndrome, cyclically shifting the received vector and calculating the shifted syndrome, trapping the error, and applying the correction. Circuits for implementing each of these is described below. These circuits are commonly known [2] [1].

2.3.1 Syndrome

The syndrome is the remainder of the division $v(x)/g(x)$. A circuit for dividing two polynomials is shown in Fig 3. In this implementation of the divider, the $n-k$ s registers are initialized to zero. The received polynomial is input to the circuit, most significant coefficient first. After $n-k$ clocks, the first coefficient of the quotient appears on the feedback line. After n clocks, the registers s_0 to s_{n-k-1} contain the respective coefficients of the remainder, which is the syndrome polynomial.

The same circuit can be used to find the syndrome of the shifted code word as described in Equation 25. Each shift performs the multiplication by x residue $g(x)$.

This circuit does not perform efficiently for Reed Solomon codes that have been shortened. A full length Reed Solomon code has $2^m - 1$ symbols, where m is the number of bits in each symbol. A code can be shortened to blocklength n by letting the $2^m - 1 - n$ most significant symbols be equal to zero[1] . It is not necessary to send these zeros, as the syndrome generator is in the same state after the zeros have been shifted in as it is initially.

After the received polynomial has been input and the syndrome polynomial generated, the *burst decoding window* is the least significant $n-k$ coefficients of $v(x)$. As the shifting of the received word and the syndrome begins, the most significant coefficients are shifted into the window. This can be seen referring to Figure 2.

This works fine unless the code has been shortened. For the shortened codeword, the most significant coefficients were not sent and are known to be zero. The syndrome generator wastes $2^m - 1 - n$ clock pulses searching the most significant coefficients for the burst error.

The problem of wasted clock pulses can be solved in one of two ways. The first is to reposition the burst decoding window to the most significant coefficients of the actual codeword. This can be accomplished by cyclically shifting $v(x)$ so that the most significant coefficient becomes coefficient $n-k-1$ and coefficient $k-1$ becomes the zeroth coefficient.

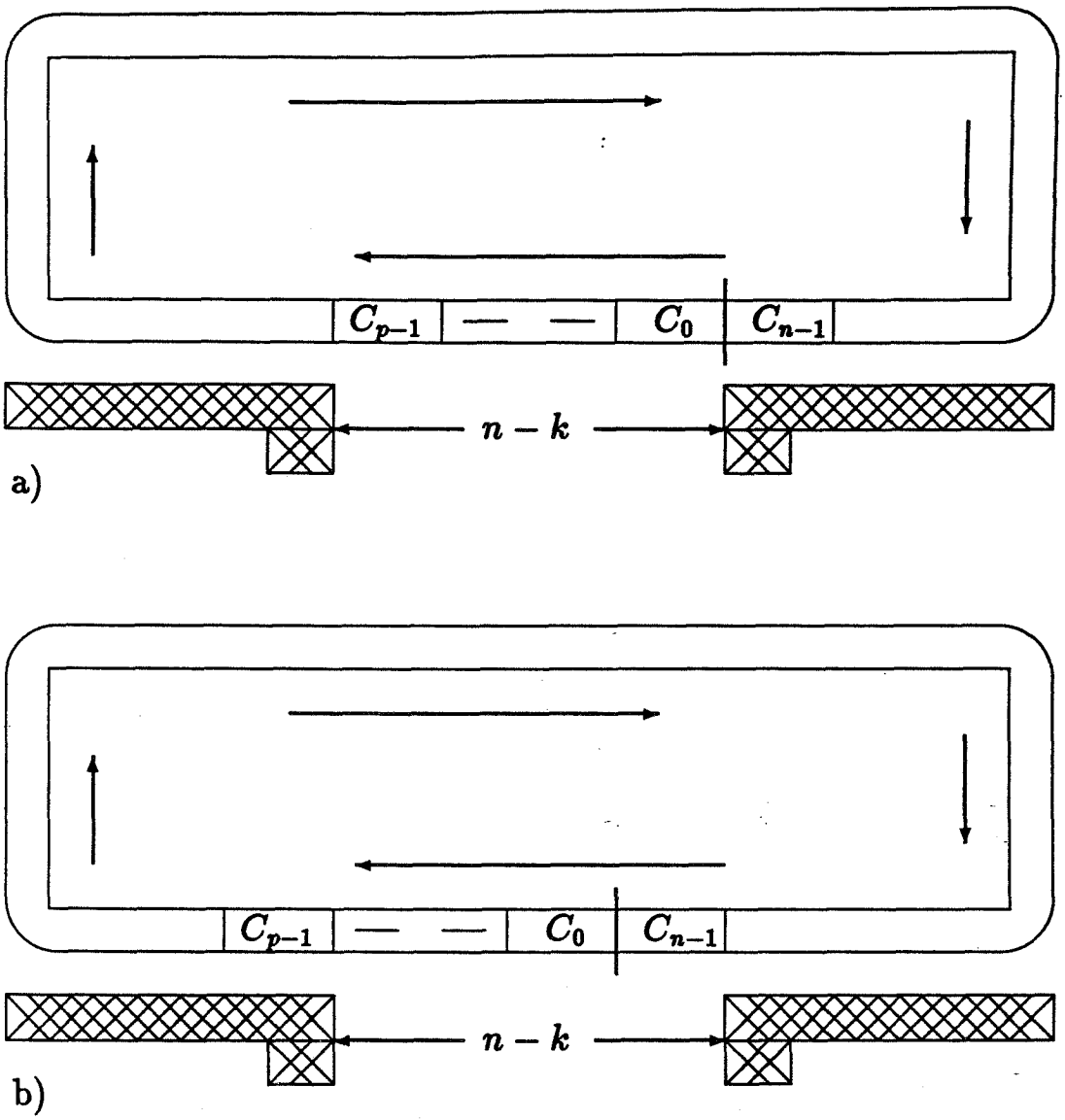


Figure 2: Cyclic shift of the codeword through the burst decoding window with a) before the shift; and b) after the shift due to a multiplication by x.

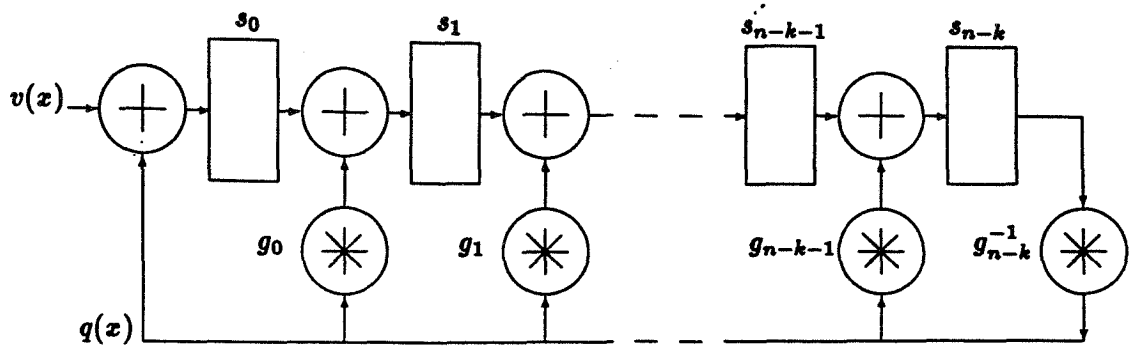


Figure 3: Polynomial Division Circuit

Let $v'(x)$ be the shifted $v(x)$. Then

$$v'(x) = R_{x^n-1}[x^{2^m-1-n}v(x)] \tag{31}$$

and its syndrome, $s'(x)$ is

$$s'(x) = R_{g(x)}[v'(x)] \tag{32}$$

The shifted syndrome can be modified into a form equal to the residue with respect to $g(x)$ of the original received polynomial multiplied by a new polynomial. A circuit will then be shown that performs this as the received polynomial is shifted in. Let

$$m(x) = R_{g(x)}\{R_{x^n-1}[x^{2^m-1-n}]\}$$

then

$$\begin{aligned} s'(x) &= R_{g(x)}[v'(x)] && (33)\\ &= R_{g(x)}\{R_{x^n-1}[x^{2^m-1-n}v(x)]\} && (34)\\ &= R_{g(x)}[R_{g(x)}\{R_{x^n-1}[x^{2^m-1-n}]\}R_{g(x)}\{R_{x^n-1}[v(x)]\}] && (35)\\ &= R_{g(x)}[m(x)v(x)] && (36) \end{aligned}$$

where the order of the polynomial $m(x)$ is less than $n-k$. Circuits that divide by a polynomial (syndrome generator) and multiply by a polynomial can be concatenated [2]. The one that performs the operation described in Equation 36 is shown in Figure 4.

The second solution is to change the direction that the burst decoding window slides over $v(x)$. This can be done by a modifying Equation 25 as follows:

$$R_{g(x)}\{R_{x^n-1}[x^{-1}v(x)]\} = R_{g(x)}[x^{-1}s(x)] \tag{37}$$

Since $v(x)$ is multiplied by x^{-1} instead of x, it is equivalent to shifting the codeword in the opposite direction indicated by Figure 3. The advantage is that the decoding window starts on the least significant coefficients of $v(x)$ and slides toward the most significant coefficients.

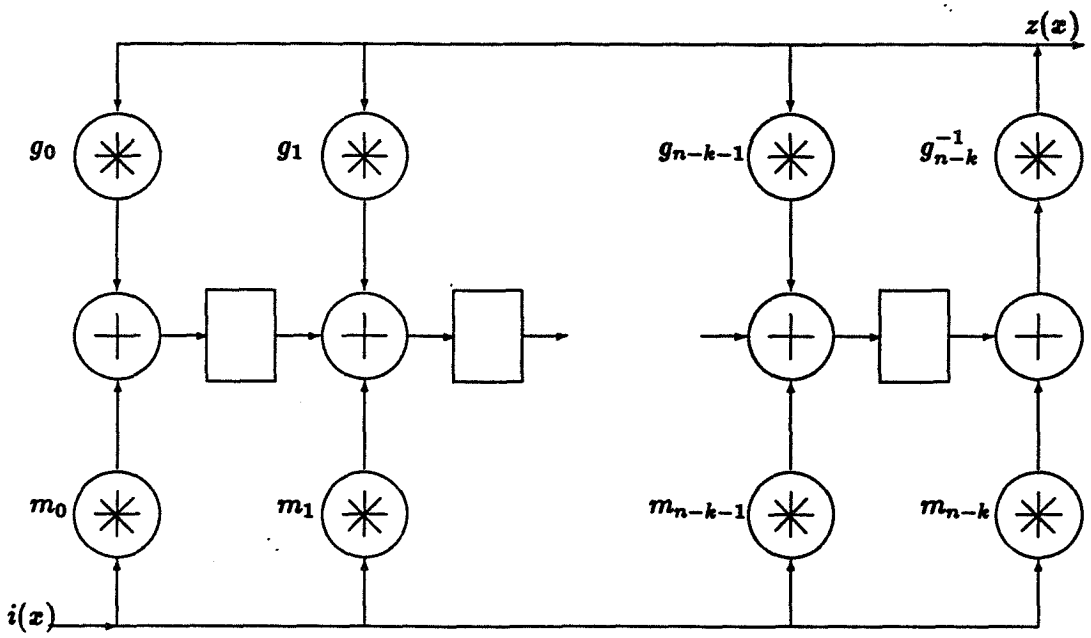


Figure 4: Circuit that performs the operation $z(x) = i(x)m(x)/g(x)$.

This function can be accomplished by reversing the direction of the syndrome generator. When the circuit is run in reverse, then two constant multipliers must be changed. The least significant multiplier is the multiplicative inverse of g_0. The most significant multiplier is equal to g_{n-k} instead of the multiplicative inverse of g_{n-k}. The circuit for clocking backward is shown in Figure 5.

A circuit that incorporates both the syndrome generator and the ability to shift the syndromes in a reverse direction is shown in Figure 6. The circuit must be initialized to all zeros. For the first n clock pulses, $v(x)$ is shifted into the shift register, most significant coefficient first. The syndrome generator registers shift to the left, the leftmost multiplexor selects g_0, and the rightmost multiplexor selects g_{n-k}^{-1}. The circuit in this configuration performs the syndrome generation.

For the second n clock pulses, the syndrome generator registers shift to the right, the leftmost mux selects g_0^{-1}, and the rightmost mux selects g_{n-k}. The circuit in this configuration performs a division by x modulo the generator polynomial.

After the first n clock pulses, the burst decoding window is in the $n-k$ lowest degree coefficients. Each clock pulse the window is shifted one coefficient towards the higher degree. At the end of $n-k$ clock pulses, the window is on the $n-k$ highest order coefficients.

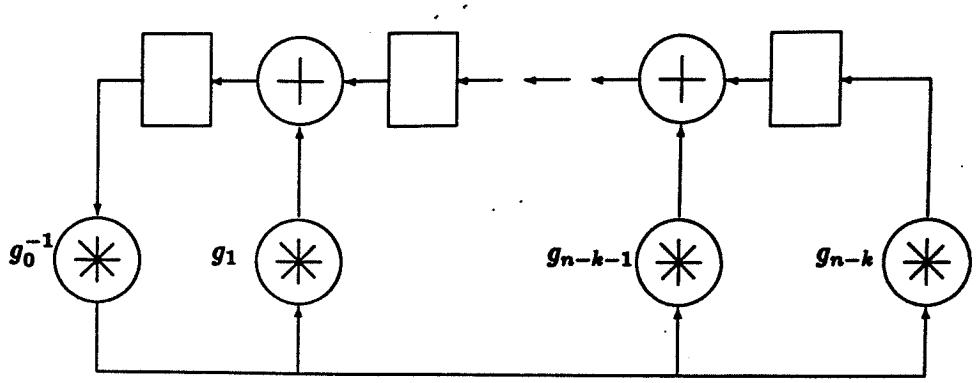


Figure 5: Circuit that shifts the syndromes in the reverse direction

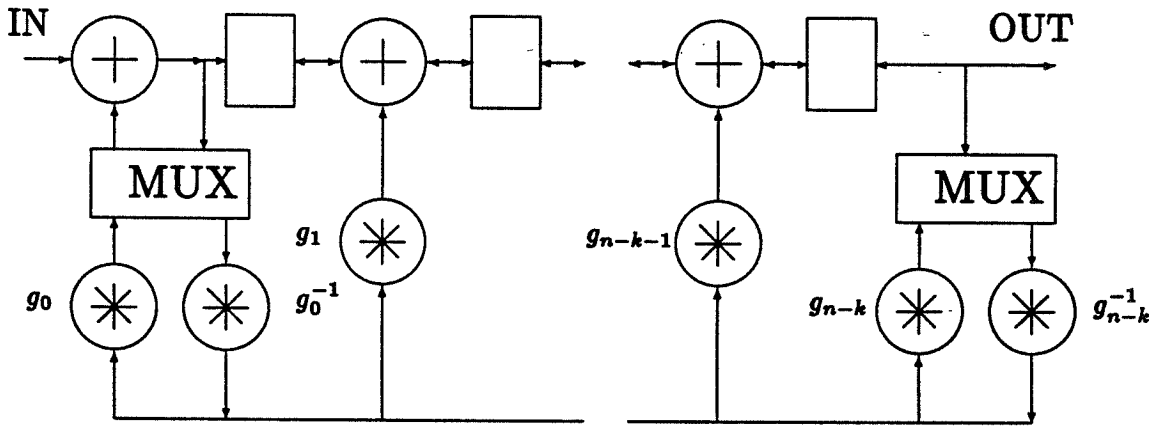


Figure 6: Circuit that combines both the forward and reverse syndrome calculation

2.3.2 Recognizing the Error

The syndrome register cycles through the code, i.e., the observable window moves across $v(x)$. When the syndrome is equal to the burst then an error has been trapped. The detection circuitry used to recognize the burst is a pattern recognizer. It has to recognize valid bursts.

A valid burst is recognized whenever 1 or more of the coefficients at the outer extreme of the window is equal to zero. If the burst is length l_j then $n-k-l_j$ zeros occur in the window. If the window is moving from low order coefficients of $v(x)$ to high order, the zeros first appear in the low order coefficients of the syndrome. As the burst shifts through the syndrome, the zeros shift from the low order coefficients to the high. If the window is moving from the high order coefficients of $v(x)$ to the low, then the opposite situation occurs.

In a normal error trapping decoder, if a valid error pattern is found, correction can proceed immediately because only one error pattern is possible if the Rieger bound is not violated. When the Rieger bound is violated, then the decoder must be capable of trapping all possible error patterns and choosing the most likely error from them. When a burst is found it should be saved, and its position and length recorded. As the search through $v(x)$ continues, if another burst is found that is more likely than any previous, then it should be saved. After searching through the whole code, the most likely burst, if one exists, has been found.

During the time that the burst is shifting through the syndrome generator, the feedback line is equal to zero. The number of consecutive clock pulses that the line is zero determines the length of the burst. The end of the burst is located when the feedback line becomes nonzero. At this point, the burst is valid on the outputs of the syndrome registers and can be latched into a holding register.

The latching should be conditioned to the length of the burst being less than the length of any burst that was found previously. If they are of the same length, then there is no clear choice, and in the previous section the decision was identified as a detected but not correctable error condition.

2.3.3 Correcting the Error

Work has been done to build error trapping decoders that load $v(x)$ into a shift register as the syndrome is created, and shift out $v(x)$ as the syndrome is shifted, and have the burst shift out of the syndrome generator coincident with the symbols of $v(x)$ which are in error [1]. These circuits do not work when more than one burst is present. The location and values of the burst are not known until the burst decoding window has traversed the entire received polynomial. For this reason, control circuitry is needed to apply the corrections at the right time.

Both $v(x)$ and the burst $e(x)$ need to be stored in memory. Correction can be applied to $v(x)$ as it is read from memory.

2.4 Summary

It was shown in the previous section that the burst error correcting ability of a Reed Solomon code is much better than thought previously.

In this section, a decoder that finds the large bursts has been identified. The error trapping decoder is a well known and simple algorithm that accomplishes the error correction. It has been shown that it also finds and corrects the bursts that violate the Rieger bound.

For codes that are designed to protect against bursts exclusively, significant savings in decoder cost, as well as increased performance can be achieved over the Reed Solomon decoders in current use. The core engine of the decoder is the same as the systematic encoder circuit. This allows the decoder to also serve as the encoder.

References

[1] R. E. Blahut, *Theory and Practice of Error Control Codes*, Reading, MA, Addison-Wesley, 1983

[2] W. W. Peterson, E. J. Weldon, *Error-Correcting Codes*, Cambridge, MA, MIT Press, 1972

[3] L. D. Rudolph and M. E. Mitchell, "Implementation of Decoders for Cyclic Codes," *IEEE Trans. on Inf. Theory*, IT-10 pp. 259-260, 1964.

[4] *Consultative Committee for Space Data Systems*, Telemetry Channel Coding "Blue Book", 1984.

N94-71090

Performance Comparison of Combined ECC/RLL Codes

C. French
Department of
Electrical Engineering
University of Idaho
Moscow, ID 88343

Y. Lin
Center for
Magnetic Recording Research
University of California, San Diego
La Jolla, CA 92093

Abstract- **In this paper, we present a performance comparison of several combined error correcting/run-length limited (ECC/RLL) codes created by concatenating a convolutional code with a run-length limited code. In each case, encoding and decoding are accomplished using a single trellis based on the combined code. Half of the codes under investigation use conventionally (d,k) run-length limited codes, where d is the minimum and k is the maximum allowable run of 0's between 1's. The other half of the combined codes use a special class of (d,k) codes known as distance preserving codes. These codes have the property that pairwise Hamming distances out of the (d,k) encoder are at least as large as the corresponding distances into the encoder (i.e., the codes preserve distance). Thus a combined code, created using a convolutional code concatenated with a distance preserving (d,k) code, will have a free distance (d_{free}) no smaller than the free distance of the original convolutional code. It should be noted that this does not hold if the (d,k) code was not distance preserving. A computer simulation is used to compare the performance of these two types fo codes over the binary symmetric channel for various (d,k) constraints, rates, free distances, and numbers of states. Of particular interest for magnetic recording applications are codes with run-length constraints (1,3), (1,7) and (2,7).**

1 Creating Combined Codes

In recent work on combined ECC/RLL trellis codes [1,2], it has been demonstrated that some of the best codes, in the sense of lowest decoded error probability, are codes created by concentrating a convolutional code with a RLL code, and then decoding using a single trellis based on the combined code. In this work, we will be dealing exclusively with such concatenated coding schemes. As an example of a concatenated code, consider the trellis for the rate 1/4, $d_{free} = 10$, 4-state convolutional code shown in Figure 1 (a). Here, free distance is defined to be the minimum Hamming distance between any two sequences out of the encoder that diverge in one state and remerge in another state. Notice in Figure 1 (a) that each branch of the trellis has a label of the form X/Y, where X is the encoder input (1 bit long, in this case) and Y is the encoder output (4 bits long). We wish to concatenate this code with the rate 1/2, 7-state, (2,7) code described by Adler, Coppersmith & Hassner

in their paper on (d,k) code construction [3]. The trellis for the (2,7) code is shown in Figure 1 (b). Initially, it would be expected that the rate 1/8 combined code would have 28 states (i.e., $4 * 7 = 28$). However, the trellis for the combined code can be simplified to 10 states, as shown in Figure 1 (c). In addition, the combined code now satisfies the more stringent (2,5) constraint. The last parameter to be determined is the free distance of the combined code. Since it is difficult to determine the free distance of a non-linear code such as this, we will give the smallest distance found (and the free distance is then less than or equal to this smallest distance). For this case, two paths separated by a distance of 6 were found. These paths go through the sequences of states 5-3-1-4-9 and 5-7-5-7-9. The free distance of the combined code is thus $d_{free} \leq 6$. As is clear from the example described above, a combined ECC/RLL code can have a lower free distance than the original convolutional code. This is due to the manner in which the RLL code was constructed [3]. There has been some work recently involving a class of RLL codes known as distance preserving codes [4,5]. As the name suggests, distance preserving codes have the property that the Hamming distance between any two encoder outputs is at least as large as the Hamming distance between the corresponding inputs. Thus, when a distance preserving RLL code is concatenated with a convolutional code, the combined code will have a free distance greater than or equal to the free distance of the convolutional code. The trade-off is that, in general, distance preserving RLL codes will have lower rates than classical RLL codes, due to the additional requirement that the code preserve distance. For this reason, a higher rate convolutional code is usually required to create a combined code when using a distance preserving RLL code. One of the main reasons for this study is to determine whether this decrease in rate is balanced by the preservation of free distance. As an example of a combined code created using a distance preserving RLL code, consider the convolutional code and the distance preserving (2,7) code shown in Figure 2 (a) and 2(b) respectively. It is not hard to show that the rate 3/8 (2,7) code is indeed a distance preserving code, as discussed in [5]. The combined code has a rate equal to 1/8, as in the previous example. The trellis for the combined code can be simplified from 8x2=16 states to 10 states, as shown in Figure 2 (c). Also, the combined code satisfies a (2,6) constraint. Finally, since the RLL code was a distance preserving code, the overall free distance is $d_{free} \geq 10$. These parameters are summarized in Table 1, along with the parameters from the previous example.

The parameters for third code, to be described in the next section, are also included in the table. We would expect that the combined code with the higher free distance (i.e. the codes labeled 1b–the code that utilized a distance preserving RLL code) would perform better than the other code (labeled 1a). In Section 3, we will verify this by comparing decoded probability of error for each code.

2 An Interesting Special Case

As discussed in the previous section, the free distance of a general convolutional code can be preserved with an appropriate choice of a (d,k) code. While experimenting with codes

of this type, we have run across some interesting examples of combined codes created using the rate 1/4 convolutional code of Figure 1 (a). Notice that this convolutional code utilizes the codewords 0000, 1000, 0111 and 1111, and no others. Thus, when choosing a distance preserving (d,k) mapping for use with this code, we need only concern ourselves with these 4 sequences (instead of all 16 4-bit sequences). The pairwise distances between these sequences are as follows:

	0000	1000	0111	1111
0000	0	1	3	4
1000		0	4	3
0111			0	1
1111				0

As an example of a distance preserving (2,6) mapping for the above sequences, consider the following:

0000→00001001
1000→00010001
0111→00100010
1111→00100100

Note that the mapping has a rate equal to 4/8, thus the rate of the combined code is 1/8. The pairwise distance between the (2,6) sequences are as follows:

	00001001	00010001	00100010	00100100
00001001	0	2	4	4
00010001		0	4	4
00100010			0	2
00100100				0

Comparing this to the pairwise distances of the 4-bit sequences, we see that we have achieved a distance preserving mapping. Thus the overall free distance of the combined code is bounded by $d_{free} \geq 10$. The parameters for this code are also summarized in Table 1 (code 1c). Notice that this code is comparable to code 1b, except that it has only 4 states instead of 10. As a second example, consider the rate 4/6 (1,4) mapping below:

000→010101
1000→000101
0111→010010
1111→001010

It is easy to show that this mapping is also distance preserving. In this case, the resulting combined code will have a rate equal to 1/6, and $d_{free} \geq 10$. In Section 3, this code will be compared to some other rate 1/6 (1,k) codes. An interesting thing to note is that the rates of the above two mappings are larger than the capacities of the corresponding (d,k) constraints. Specifically, the (2,6) and (1,4) capacities are 0.4979 and 0.6174, respectively compared to rates of 0.5 and 0.6667 for the mappings. This is due to the fact that we need only 4(d,k) sequences (instead of 16) for each mapping.

3 Performance of Combined Codes

A computer simulation was utilized to compare the codes in Table 1 over the binary symmetric channel. In each case, the Viterbi algorithm was utilized in decoding. The results are shown in Figure 3. Notice that, as expected, the codes that used a distance preserving RLL code (codes 1b and 1c have a lower decoded probability of error than the other code (code 1a).

As another example, consider the rate 1/4 and rate 2/8 (2,k) codes listed in Table 2(b) with three different convolutional codes to create combined codes with rates all equal to 2/8, and with different free distances. For comparison, we also include a rate 1/4 code that utilizes the (2,7) code from Figure 1(b). In Figure 4 we give the probability of error curves for these codes. From the figure we see that, at low channel bit error probability, the codes created using the distance preserving RLL code (codes 2b, 2c, and 2d) all perform better than the other code (code 2a).

For the next set of comparisons, we are interested in codes that satisfy a (1,k) constraint. In Table 3 we give the parameters for a rate 1/6 and a rate 2/12 (1,k) code. The rate 1.6 code (code 3a) uses the 5-state, rate 2/3 (1,7) code in [6]. This is the (1,7) code used in many existing recording systems. The rate 2/12 code (code 3b) utilizes the rate 2/4 distance preserving (1,5) code from [5]. This code is really a block code, thus the trellis has only one state. The last code listed in Table 3 is the 4-state (1,4) code described in Section 32. In Figure 5 we give the probability of error curves for codes 3a, 3b, and 3c.

As a final comparison, consider the rate 1/4 (1,k) codes in Table 4. Codes 4a and 4b utilize the Miller code (also known as MFM), and codes 4c and 4d utilize the distance preserving (1,5) code from [5]. Although the Miller code was not constructed specifically to be distance preserving, it happens to satisfy the distance preserving criterion. Thus, all the codes in 4 were constructed from distance preserving RLL codes. In 6 we compare the performance of these four codes.

4 Summary

We have given decoded probability of error curves for several concatenated codes that satisfy a run-length constraint in addition to providing error correction capabilities. We showed that the use of distance preserving RLL codes was beneficial in terms of decoder performance.

References

[1] P. Lee & J.K. Wolf, "Combined Error Correction/Modulation Codes," IEEE Transactions on Magnetics, Sept. 1987.

[2] Y. Lin & J.K. Wolf, "Combined ECC/RLL Trellis and Tree Codes," IEEE Transaction on Magnetics, Nov. 1988.

[3] R. Adler, D. Coppersmith, M. Hassner, "Algorithms for Sliding Block Codes," IEEE Transactions on Information Theory, Jan. 1983.

[4] H.C. Ferreira, D.A. Wright & A.l. Nel, "Hamming Distance Preserving Mappings and Trellis Codes with Constrained Binary Symbols," IEEE Transactions on Information Theory, July 1989.

[5] C.A. French, "Distance Preserving Run-Length Limited Codes," IEEE Transactions on Magnetics, Sept. 1989.

[6] P.H. Seigel, "Recording Codes for Digital Magnetic Storage," IEEE Transactions on Magnetics, Sept. 1985.

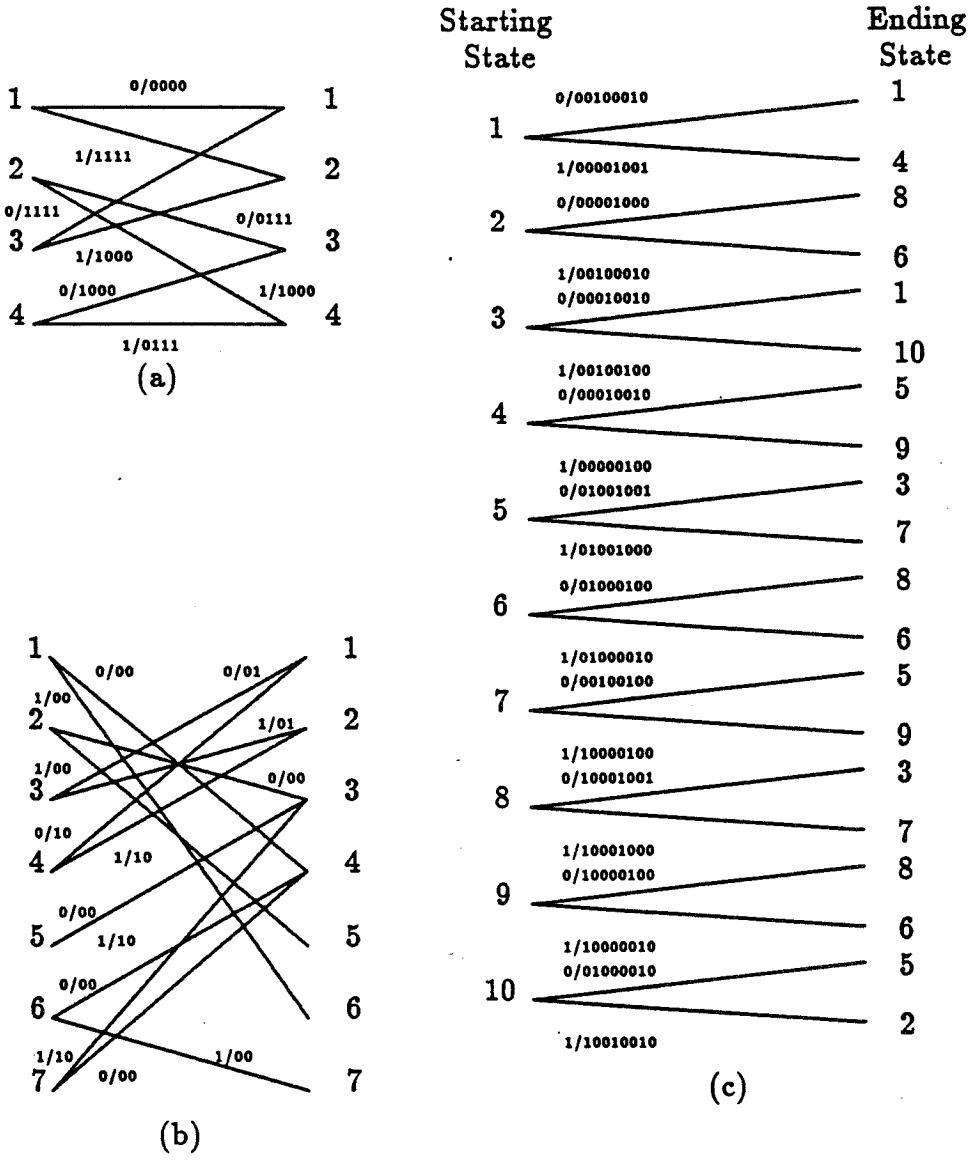


Figure 1: Trellis for (a) rate $1/4 d_{free} = 10$, 4 -state convolutional code, (b) rate 1/2 (2,7) code and (c) Trellis for combined code

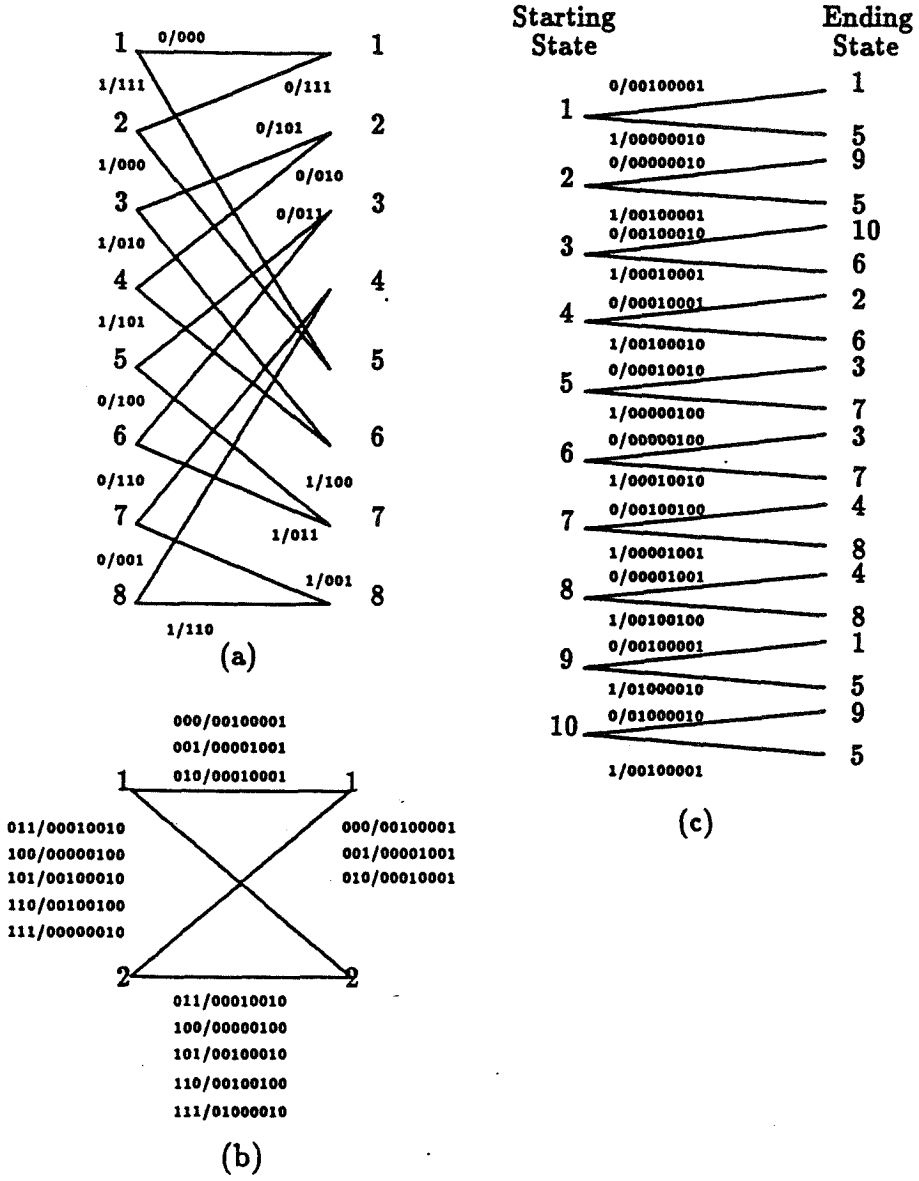


Figure 2: Trellis for (a) rate $1/3 d_{free} = 10$, 8-state convolutional code, (b) rate 3/8 (2,7) code and (c) combined code

	ECC Code			(d,k) Code				Combined Code			
	Free Dis-tance	Rate	No. of States	(d,k)	Rate	No. of States	Dist. Preser-ving	(d,k)	Free Dis-tance	Rate	No. of States
1a	10	1/4	4	(2,7)	1/2	7	No	(2,5)	≤ 6	1/8	10
1b	10	1/3	8	(2,7)	3/8	7	Yes	(2,6)	10-13	1/8	10
1c	10	1/4	4	(2,6)	4/8	1	Yes	(2,6)	10-12	1/8	4

Table 1: Rate 1/8 (2,k) combined codes

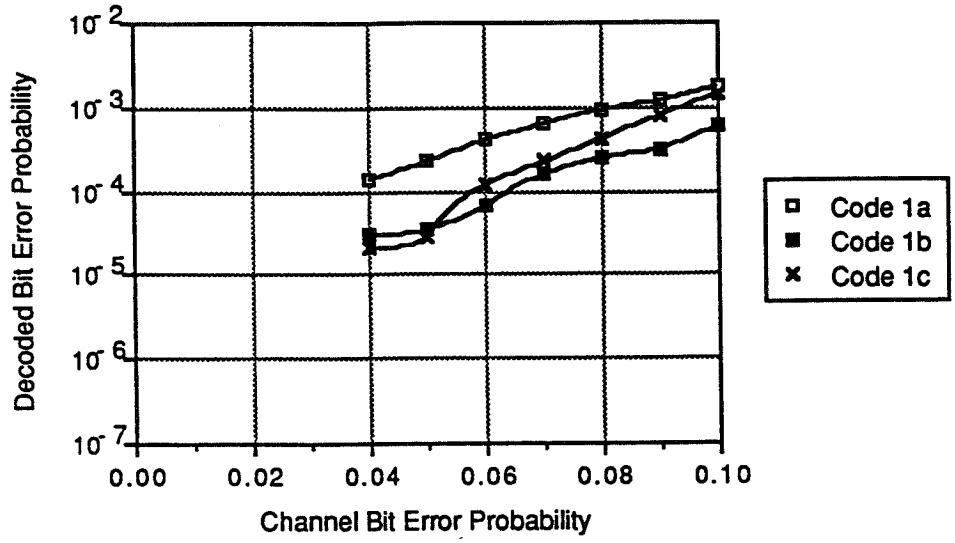


Figure 3: Performance of rate 1/8 (2,k) combined codes

	ECC Code			(d,k) Code				Combined Code			
	Free Dis-tance	Rate	No. of States	(d,k)	Rate	No. of States	Dist. Preser-ving	(d,k)	Free Dis-tance	Rate	No. of States
2a	5	1/2	4	(2,7)	1/2	7	No	(2,7)	≤ 2	1/4	18
2b	3	2/3	4	(2,7)	3/8	2	Yes	(2,7)	3-5	2/8	6
2c	4	2/3	8	(2,7)	3/8	2	Yes	(2,7)	4	2/8	12
2d	5	2/3	16	(2,7)	3/8	2	Yes	(2,7)	5-7	2/8	24

Table 2: Rate 1/4 and 2/8 (2,k) combined codes

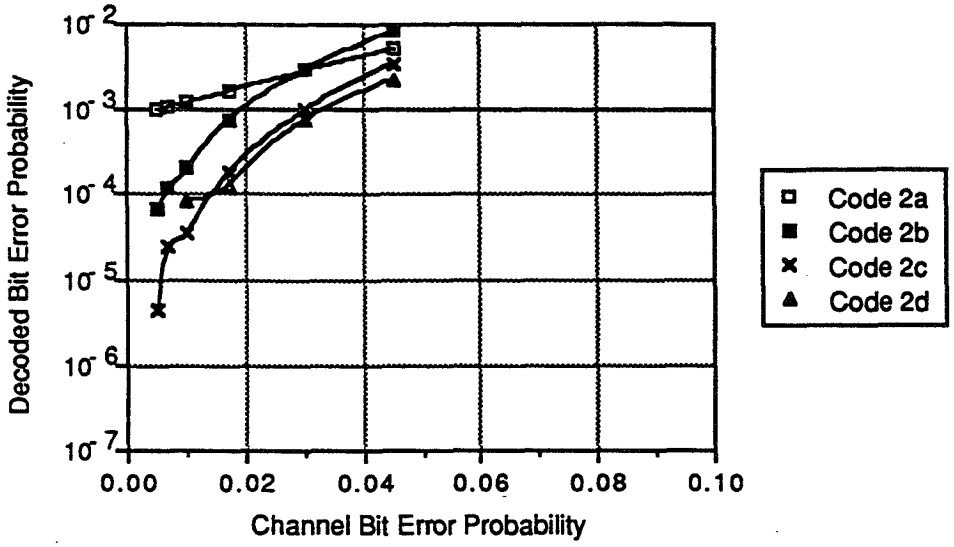


Figure 4: Performance of rate 1/4 and 2/8 (2,k) combined codes

	ECC Code			(d,k) Code				Combined Code			
	Free Dis-tance	Rate	No. of States	(d,k)	Rate	No. of States	Dist. Preser-ving	(d,k)	Free Dis-tance	Rate	No. of States
3a	10	1/4	4	(1,7)	2/3	5	No	(1,5)	≤8	1/6	9
3b	10	1/3	8	(1,5)	2/4	1	Yes	(1,5)	10-14	2/12	8
3c	10	1/4	4	(1,4)	4/6	1	Yes	(1,4)	10-12	1/6	4

Table 3: Rate 1/6 and 2/12 (1,k) combined codes

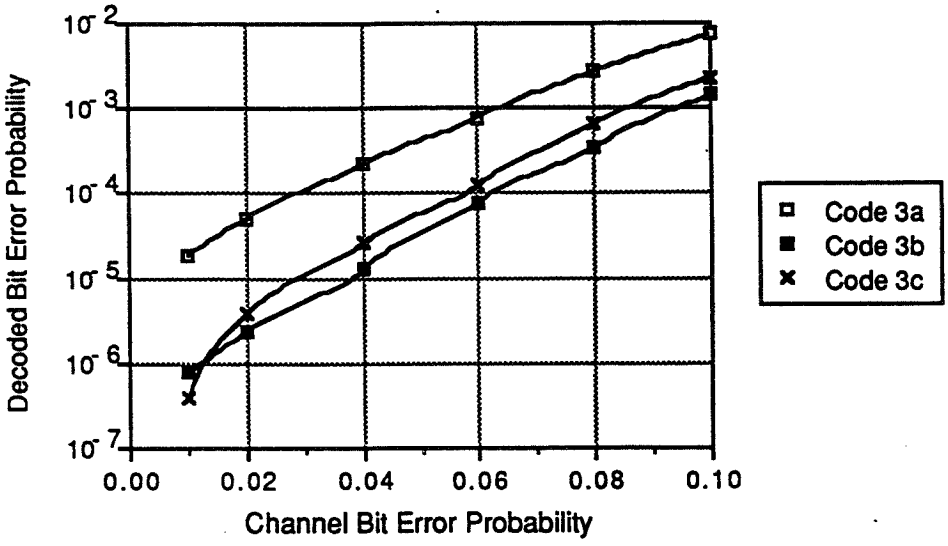


Figure 5: Performance of rate 1/6 and 2/12 (1,k) combined codes

	ECC Code			(d,k) Code				Combined Code			
	Free Dis-tance	Rate	No. of States	(d,k)	Rate	No. of States	Dist. Preser-ving	(d,k)	Free Dist-ance	Rate	No. of States
4a	3	1/2	2	(1,3)	1/2	2	Yes	(1,3)	3	1/4	4
4b	5	1/2	4	(1,3)	1/2	2	Yes	(1,3)	5-7	1/4	8
4c	5	1/2	4	(1,5)	2/4	1	Yes	(1,5)	5-6	1/4	4
4d	6	1/2	8	(1,5)	2/4	1	Yes	(1,5)	6-7	1/4	8

Table 4: Rate 1/4 (1,k) combined codes

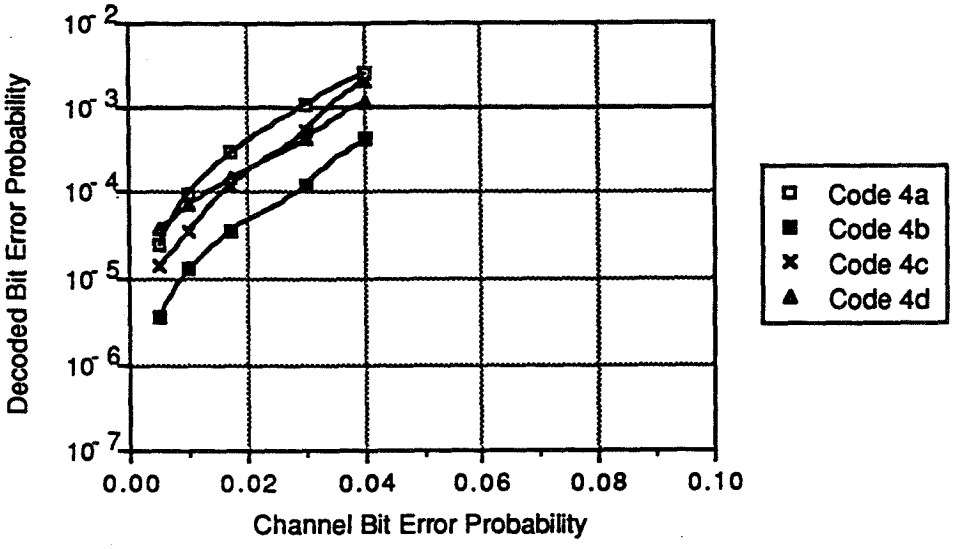


Figure 6: Performance of rate 1/4 (1,k) combined codes

Serial Multiplier Arrays for Parallel Computation

Kel Winters Department of Electrical Engineering
Montana State University
Bozeman, Montana

Abstract- **Arrays of systolic serial-parallel multiplier elements are proposed as an alternative to conventional SIMD mesh serial adder arrays for applications that are multiplication intensive and require few stored operands. The design and operation of a number of multiplier and array configurations featuring locality of connection, modularity, and regularity of structure are discussed. A design methodology combining top-down and bottom-up techniques is described to facilitate development of custom high-performance CMOS multiplier element arrays as well as rapid synthesis of simulation models and semicustom prototype CMOS components. Finally, a differential version of NORA dynamic circuits requiring a single-phase uncomplemented clock signal is introduced for this application.**

1 Introduction

Single instruction/multiple datapath (SIMD) computer arrays were proposed for high performance processing of large planar data structures with Unger's Spatial Computer proposal in 1958 [16], the Solomon array proposal in 1962 [14], and later the ILLIAC IV project at the University of Illinois in the sixties and seventies [1]. These early machines, however, failed to gain commercial acceptance over vector based supercomputers in scientific applications. The technology did not exist to exploit the inherent modularity of the architecture or the locality of reference provided by the mesh interconnection network.

With the advent of Very Large Scale Integrated (VLSI) circuit methodologies in the late 1970s, SIMD array architectures re-emerged tailored primarily for image processing applications. This new generation of machines, like the original Unger and Slotnick designs, featured bit-serial arithmetic and I-O operations, rather than the word-wide arrangement of the ILLIAC processing elements. Bit-serial architectures, such as the early CLIP, Digital Array Processor (DAP), and Massively Parallel Processors (MPP) [7], avoided much of the functional complexity and interconnect cost of the larger-grained ILLIAC IV, at the expense of arithmetic and I-O throughput. Subsequent SIMD mesh arrays, such as Blitzen [4] and the Geometric Arithmetic Parallel Processor (GAPP) [6], have remained close to the DAP/MPP architecture.

While SIMD mesh processor arrays have evolved into a set of highly similar designs, there is in fact a continuum of possible configurations with respect ot word width, interconnection, and functionality. Optimization of the architecture for a particular domain of

applications is a matter of balancing the ratio of IC area allocated to logic, memory, and interconnection, to the requirements of the application set.

For example, algorithms requiring few stored arguments favor very fined-grained PEs with a high logic/memory area ratio. On the other hand, the PE must have sufficient storage to hold all arguments required by the application algorithms without wasting IC memory or running short. One solution is to define a processor array as a matrix of elements that may be flexibly allocated to data elements (pixels, for instance) of the problem space in groups. Thus, an array of fixed size could serve as a large array of very fine-grained element groups (later referred to as virtual processors) for applications with few operands, or as a smaller array of larger groups for problems requiring more storage per problem element.

For multiplication intensive massively parallel problems requiring relatively little operand storage, arrays of multipliers can offer better performance and better resource utilization than DAP/MPP style adder arrays. Adder arrays typically have a large random access memory store (1K bits for the MPP) to accommodate varying word widths and operand storage requirements. The access time of RAM storage can significantly reduce clock speed. In these applications, much of this memory capacity can go un-utilized, while insufficient arithmetic resources are available to exploit bit-level parallelism. An alternative approach is a mesh array of serial multipliers where:

1. The ratio of arithmetic logic to memory silicon area is higher than that for conventional SIMD adder arrays.

2. The function set is optimized for serial multiplication rather than serial addition to better serve multiplication intensive applications.

3. Operand storage primarily consists of high-speed shift registers, rather than RAM to enable higher clock rates.

4. Multiplier elements are of a fixed word width, but may be logically concatenated to accommodate multiple word operations without degradation of performance.

2 Bit Serial Multiplication

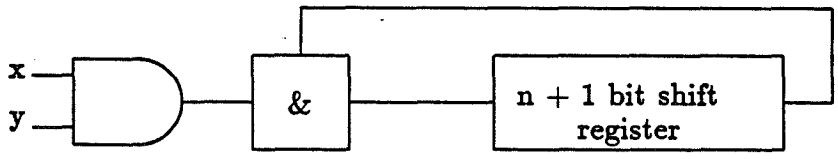


Figure 1: Simple Serial-Parallel Multiply-Accumulator

A very simple serial multiply-accumulator is shown in Figure 1, consisting of an AND gate, a shift register (or equivalent in random accessed memory), and a bit-serial adder.

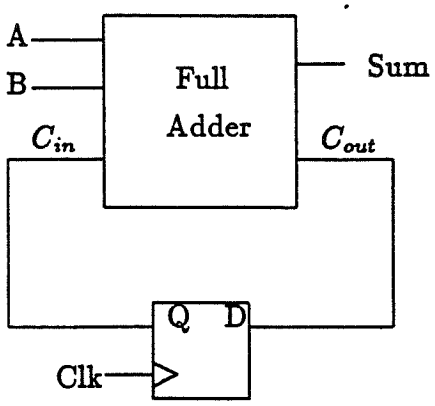


Figure 2: Serial Adder Module

The serial adder, designated by &, consists of a full adder and a carry-save flip-flop as shown in Figure 2. Arguments x and y, n bits in length, are fed serially such that y is repeated m times as each bit of x is shifted in one bit per n clock cycles. Partial products are accumulated in the shift register, which is n+1 bits in length so that each successive partial product is effectively multiplied by two before it is summed in the shift register. n^2 clock cycles are required to complete a multiplication operation.

Despite the slow speed of this configuration, it is the basis of integer multiplication in virtually all current SIMD processor arrays, including the DAP and MPP and their successors. This can create a substantial performance bottleneck given the multiplication-intensive nature of image and signal processing applications typically run on these machines.

High performance serial multipliers for VLSI processor arrays should (a) be modular in structure, (b) have minimal internal signal fanout, and (c) require no asynchronous carry propagation that would constrain the clock rate, (d) are extensible in word width, and (e) require a minimum number of clock cycles.

The problems of input loading, adder delay, and extensibility for serial-parallel multipliers may be addressed by pipelining both the multiplicand input path and the product accumulation path. Figure 3 illustrates a fully pipelined or systolic multiplication network [17]. On the first clock cycle, x_0 is shifted into the multiplier pipeline and x_0y_0 is shifted into the product pipeline, and appears at the product output n cycles later. This circuit requires 2n clock cycles to multiply two n-bit operands. An addend, a, may be summed with the product by shifting it into the product pipeline concurrent with the serial multiplicand, x.

An alternative multiplier , shown in Figure 4, is a fully systolic adaptation of an early serial-parallel multiplier introduced by Daniel Hampel, et al., in 1975 [10]. In this configuration, the multiplicand is pipelined through n 2 gate inputs. This systolic multiplier also requires 2n clock cycles. The least significant bit of the product x_0y_0, appears at the output n clock cycles into the multiplication sequence. External addends must be preshifted

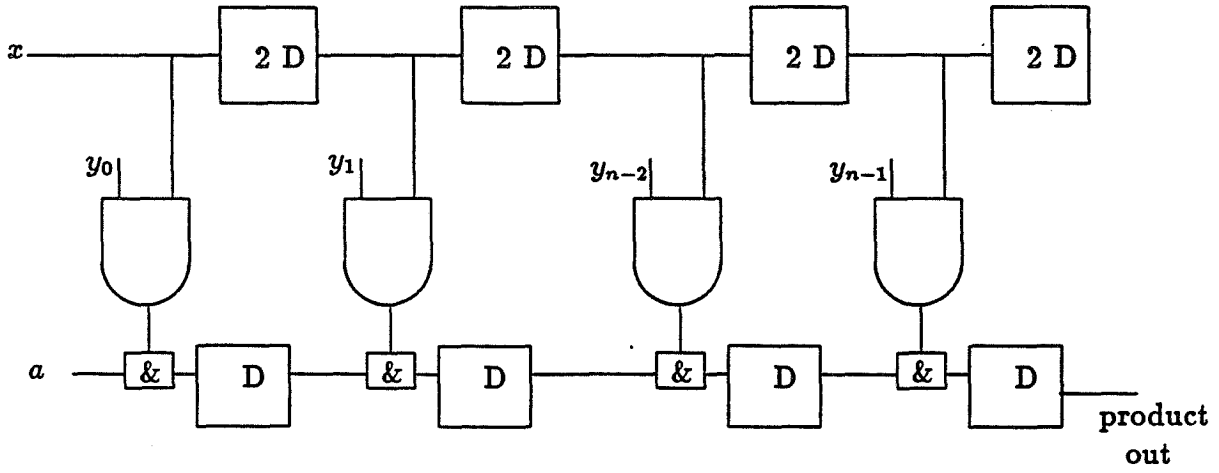


Figure 3: Fully Systolic Pipelined-Multiplier Configuration

n bits into the product pipeline prior to a multiplication.

An interesting property of the second systolic multiplication circuit is that it contains n bits of storage for the multiplicand and 2n bits for the product. An array of these circuits would have the proper ratio of operand versus product storage. Thus, 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.

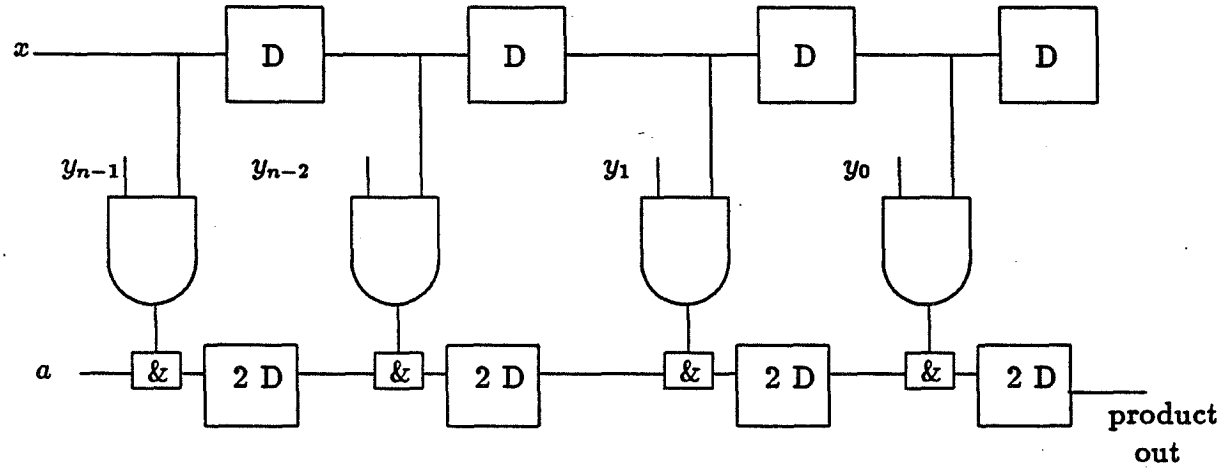


Figure 4: Fully Systolic Hampel Multiplier

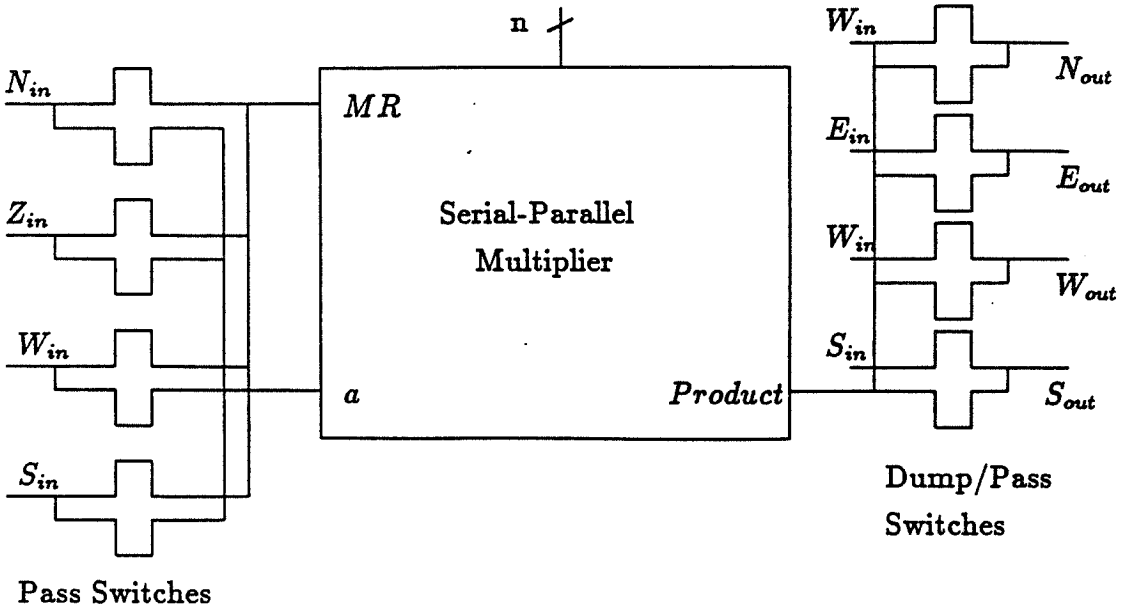


Figure 5: Multiplier Array Element

3 Multiplier Array Topology

The following examples illustrate preliminary configurations intended for sum of products evaluation involving multiplication by a constant, such as convolution. [1]

In the first configuration, serial-parallel multipliers are interconnected to four nearest neighbors with dynamically segmented serial busses that allow communication between non-adjacent neighbors. This facilitates on-the-fly allocation of multiplier elements (MEs) to the application. Control signal and parallel multiplier (constant) inputs to the MEs are routed by individual diagonal array rows. Thus, each diagonal row of MEs is controlled independently, allowing constrained multiple-instruction (MIMD) execution.

At the system level, an Instruction Sequencer provides control signals to each diagonal row of MEs. These in turn are controlled by a Host Interface Controller, whose function is to manage the communication channel to the host scalar computer and decode array instructions.

The multiplier element, shown in Figure 5, in the array has four input ports (NIN, EIN, WIN, SIN) and four output ports (NOUT, EOUT, WOUT, SOUT). The output ports are in one of two modes, dump or pass. In the dump mode, the multiplier element drives the output port, while in the pass mode, a CMOS switch passes the value of the corresponding input port to the output port.

As only N-fet pass transistors are required for data switching to N-logic NORA stages, three control lines are required, Load, Dump, and NotDump, for each input/output pair.

[1] Features necessary for more general purpose application will be added at a later time.

This scheme enables communication between nonadjacent MEs passing through intermediate MEs, or configurable length array tesselation. Since diagonal rows of MEs are controlled independently, traversals across arbitrary vertical column or horizontal row distances are supported. Without I/O port pipelining, the propagation delay increases with the square of the number of series pass devices [13], or, in this case, the square of the array distance traversed.

Configurable length tesselation of the register array with independent control of diagonal register rows enables MEs to be dynamically allocated to the application in groups or virtual processors (VPs). Register elements within a virtual processor can then be randomly accessed by neighboring virtual processors. For example, Figure 6 illustrates the ME allocation to a two-dimensional problem requiring three one-word arguments, A, B, and C. Column communication between any combination of A, B, and C in adjacent VPs are supported.

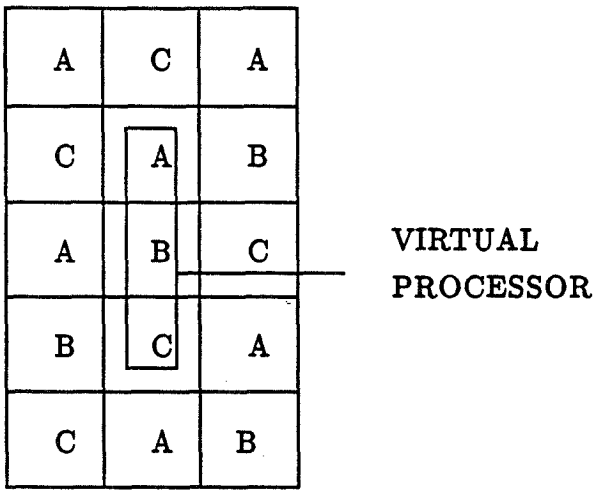


Figure 6: Virtual Processor Partitioning

Horizontal communication is more constrained. The diagonal control routing serves to offset horizontally adjacent virtual processors by one row. This allows horizontal communication between different argument registers in adjacent VPs, but not random register access. In this example, B may only communicate with A in the next left VP neighbor or with C in the next right VP. Other combinations require data shifting to reorder the arguments. Despite this constrained horizontal VP communication, this arrangement is suited to a good number of applications, particularly where products are summed by column, then collected by row.

To illustrate, a two-dimensional convolution may be described by the function:

$$c(x,y) = \sum_{i=0}^{m-1} /sum_{j=0}^{m-1} w_{i,j} p_{x-i,y-j} \qquad (1)$$

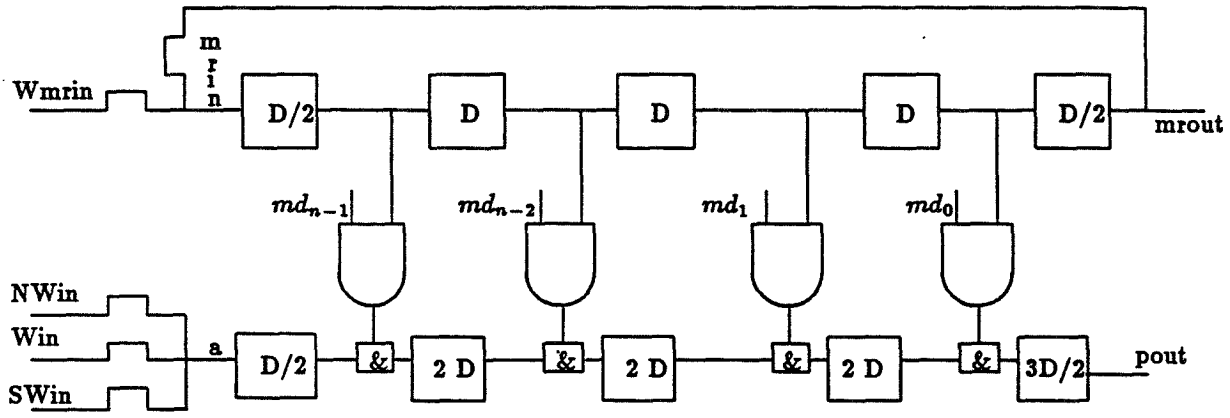


Figure 7: SASM Element

where c is the resulting convolution array, w is the weight or mask matrix, and p is the input array data (pixels, if an image application). A configurable-length tesselation array can perform an m-by-m convolution in $2nm_2$ clock cycles if two MEs are allocated per pixel, one to hold the pixel datum. [2] , the other to accumulate the product-sum. Thus, a 3-by-3 by 8-bit convolution would require 144 clock cycles, versus 795 for the MPP [8]. Allocating four MEs per pixel, the same convolution could be performed in as little as $2(2m-1)n$ or 80 clock cycles, by executing column multiplications in the weight matrix in parallel, then summing the columns. However the area-time performance is diminished.

Spice [3] suggests that inter-element transfers at an array distance of three elements is practical up to a clock rate of about 50 MHz in 2-micron CMOS. While faster than current SIMD mesh arrays, this is considerably below the performance of the multiplier circuits investigated. An alternative approach is to add a pipeline latch to each input and output port, These add a delay of one clock cycle, for each element traversed, to inter-ME transfers, eliminating the need for wait cycles in long distance transfers where RC delays through I/O pass gates would be greater than a single clock cycle allows. While enabling high clock rates, adding pipeline delays to the configurable length interconnection scheme does add considerable control complexity, as serial word boundaries are no longer aligned between array elements. Multiple word-width operations would suffer significant performance reduction to I/O pipeline delays.

A configuration for systolic arrays of systolic multipliers (SASM) is shown in Figure 7. Here the systolic multiplier of Figure 4 is used to store both data and an accumulated result. Data operands are stored in the multiplier or mr pipeline, which is loaded from itself or its West (left) neighbor. The product accumulator pipeline, 2n stages in length, may be loaded from the Northwest, West, or Southwest neighbors. Thus, the array is connected

[2]If an n-bit shift register were added to each ME to hold pixel data, only one ME per pixel would be required for this convolution.

[3]conducted by D. Wall and C. Hsiaochi at Montana State University with slow speed device models for the MOSIS 2-micron (drawn gate length) SCMOS process, T=100 degrees C, Vdd=4.5V.

as a shuffle-exchange network, rather than a mesh as were the previous examples. The multiplier array boundaries must be connected to form a cylinder or torus.

This multiplication circuit requires the multiplier operand to be shifted into the multiplier pipeline in n cycles and out in n additional cycles for a total of 2n clock cycles per multiplication. Nominally, the multiplier pipeline should be cleared at the beginning and end of each multiplication. To perform this while storing the multiplicand operand in an n-bit shift register with no external storage, the product logic must effectively ignore the multiplier operand on every second pass through its shift register. This is provided by ANDing the parallel multiplicand operand, md, with the sequence 1000...0, 1100...0, 1110...0, ..., 1111...1, 0111...1, 0011...1, and finally 0000...0 during each multiplication sequence. Such sequences are easily generated with Johnson-Ring or Mobius [15] counters. Multiplicand enabling would occur in the control logic external to the multiplier array.

In this array, operands in the mr pipeline and products in the product pipeline may be moved in two dimensions relative to each other, not the physical array. To move products east relative to the mr data, mrout is fed back to the MR register in the same multiplier element. To move the products west relative to the mr data, mr is fed from its west neighbor, so that the mr operands travel across the array eastbound at twice the rate of the product pipeline. Northwest and southwest input switching enable vertical movement.

$$
\begin{array}{l}
w_{00}mobius \rightarrow md, mrout \rightarrow mrin, win \rightarrow a; (ncycles) \\
w_{00}mobius \rightarrow md, mrin \rightarrow mr, win \rightarrow a; (ncycles) \\
w_{01}mobius \rightarrow md, mrin \rightarrow mr, win \rightarrow a; (2ncycles) \\
w_{02}mobius \rightarrow md, mrin \rightarrow mr, win \rightarrow a; (ncycles) \\
w_{02}mobius \rightarrow md, mrout \rightarrow mr, nwin \rightarrow a; (ncycles) \\
w_{12}mobius \rightarrow md, mrin \rightarrow mr, nwin \rightarrow a; (ncycles) \\
w_{12}mobius \rightarrow md, mrout \rightarrow mr, win \rightarrow a; (ncycles) \\
w_{11}mobius \rightarrow md, mrout \rightarrow mr, win \rightarrow a; (2ncycles) \\
w_{10}mobius \rightarrow md, mrout \rightarrow mr, win \rightarrow a; (ncycles) \\
w_{10}mobius \rightarrow md, mrout \rightarrow mr, nwin \rightarrow a; (ncycles) \\
w_{20}mobius \rightarrow md, mrin \rightarrow mr, nwin \rightarrow a; (ncycles) \\
w_{20}mobius \rightarrow md, mrin \rightarrow mr, win \rightarrow a; (ncycles) \\
w_{21}mobius \rightarrow md, mrin \rightarrow mr, win \rightarrow a; (2ncycles) \\
w_{22}mobius \rightarrow md, mrin \rightarrow mr, win \rightarrow a; (ncycles) \\
w_{22}mobius \rightarrow md, mrin \rightarrow mr, win \rightarrow a; (ncycles)
\end{array}
$$

Table 1: SASM Convolution

Table 1 illustrates a 3-by-3 by n-bit convolution. The array is initialized with the product shift registers cleared, the mr registers containing the pixel data, and the mobius enable register in the control sequencer [4] cleared. In this example, only one ME per pixel is required, so identical control and multiplicand information is sent to all MEs

[4] $w_{00}mobius \rightarrow md$ means that the multiplier, W_{00} is ANDed with the output of an n-bit Johnson-ring counter, as described above.

simultaneously (SIMD operation). Control and multiplicand information is indicated with the transfer operator, $\rightarrow$.

The 3-by-3 integer convolution requires 18n clock cycles or 144 for 8-bit operands. This array configuration is fully pipelined for all communication within and between the multiplier elements.

4 Design Methodology

The design methodology adopted for the development of high-performance multiplier arrays is two-faceted, combining bottom-up and top-down approaches. First, a library of custom multiplier datapath cells based on differential single-clock NORA circuits was developed that could be utilized in a variety of array configurations. Second, hardware description languages (HDLs), logic synthesis, logic and switch simulation, and module compilation tools are used for top-down definition and verification of multiplier array systems and rapid prototyping of semi-custom CMOS models.

Custom datapath cells were developed using the Tekspice circuit simulator and graphics editor Quickic from Tektronix, Inc. Scalable CMOS design rules from the National Science Foundation MOSIS program were used to provide portability to a number of silicon foundries, compatibility with cell libraries from the academic community, and access to economical multi-project CMOS prototyping through the MOSIS service. Design rule verification is done with SDRC, provided by the Northwest Laboratory for Integrated Systems (NWLIS) at the University of Washington.

The OCT toolset from the University of California, Berkeley, is used for top-down behavioral and structural HDL modeling, logic and switch simulation, CMOS standard-cell prototype synthesis, and custom layout. The toolset is an integrated system for VLSI design, including tools and libraries for multi-level logic synthesis, standard cell placement and routing, programmable logic array and gate matrix module generation, custom cell design, and utility programs for managing design data. Most tools are integrated with the OCT data-base manager and the X-based VEM graphical user interface.

The OCT tools currently use non-industry standard hardware description languages BDS for behavior (originally from the Digital Equipment Corporation) and Bdnet for structure. EDIF support has been recently introduced and VHDL support is under development. Unlike some recent commercial VHDL tools, BDS descriptions cannot be directly simulated but must first be compiled into logic netlists. While this has not been a major inconvenience, direct HDL modeling is also under development at Berkeley.

A behavioral description of an 8-bit serial-parallel multiplier of the pipelined-product configuration is listed in Table 2.

Behavioral models are structurally decomposed in a top-down fashion. For instance, the multiplier of Table 2 decomposed into the eight instances of the bit-multiplier cell whose behavior is defined in Table 3.

In turn, this bit-multiplier cell is used to define the function of the custom differential single-clock NORA circuit described in the next section.

Datapath module compilers to assemble tiled arrays of multiplier elements are under development locally in the OCT environment. Composite placement and routing of custom and semi-custom components is done with a combination of OCT toolset place-and-route and symbolic layout programs. Final mask verification is performed using software from the Berkeley OCT and Washington NWLIS toolsets. Masks are released for fabrication via the MOSIS service in Cal-Tech Intermediate (CIF) format. Currently, all custom and semi-custom components are designed under 2-micron design rules for fabrication using economical high-volume commercial processes. MOSIS scalable design rules (SCMOS Revision 6) are currently supported to a minimum feature size of 1.2 μm, which could be used to fabricate the custom datapath library without modification.

5 Differential Single-Clock NORA CMOS Circuits

High performance serial multiplier arrays require CMOS circuit realizations with properties complementing those of the multiplier architecture. Specifically, these circuits should have low input loading, low internal signal fanout, high locality of interconnection, minimal series device delays, and little clock skew.

At high clock speeds, in excess of 50 MHz, clock skews and associated hold time margins occupy a significant portion of the clock cycle timing budget for synchronous systems. For this reason, two phase non-overlapped clocking schemes (a favorite technique of NMOS circuit designers) are not commonly used at clock rates above 40 MHz. Above 100 MHz, skew between a single phase clock and its complement can become significant. One solution is to eliminate complemented clocks by using circuits requiring only a single phase uncomplemented clock. One such circuit was proposed by Yuan Ji-Ren, et. al., in 1987 [11] as an improvement to NORA, or "No-Race" logic [9]. With this technique, synchronous systems may be constructed from alternating precharged P-fet and N-fet logic stages separated by clocked inverters. This scheme, like its predecessors NORA and Domino [12] logic, is free of precharge race failures, yet eliminates the need for a complemented clock and associated skewing problems at very high clock rates. This method appears to have good potential for future CMOS control [18] and datapath applications if the following design constraints are imposed:

1. Series logic fets are minimized. NOR functions are preferable to NAND in N-logic stages; NAND is preferable to NOR in P-logic.

2. Series inverters to form complemented gate outputs are eliminated from critical timing paths. Unlike Domino logic, the sense of dynamic logic stage output transitions is not important in preventing a precharge race condition in successive stages. Therefore, inverters may be used to complement logic stage outputs. These, however, add significant output delay. In critical timing paths, it is preferable to construct differential logic stages that output complemented output pairs with equal delay. In many cases, programming fets may be shared between the true and complemented logic networks.

3. Logic functions are split between successive P-fet and N-fet logic stages which also serve as synchronous master and slave delay elements. This serves to reduce series logic delays (a). Where practical, layout area and input loading may be reduced by realizing the largest logic stages in N-logic rather than P.

A high-performance serial multiplication cell is under development in $2-\mu m$ (gate length) CMOS using differential single-clock NORA techniques. A circuit diagram is shown in Figure 8. The P-logic stages at left precharge low during *clock* and evaluate during $\overline{clockbar}$ and serve as master storage latches. The N-logic stages at right precharge high during $\overline{clockbar}$ and evaluate during *clock*, serving as slave dynamic storage latches. The clocked inverters at each stage output insure that the inputs to successive stages only change during the precharge phase and are stable during evaluation, eliminating precharge race conditions.

In this circuit, a maximum of two series logic fets are used in any dynamic stage. Differential logic stage configurations are used wherever there are two logic fets in series and complemented outputs are required, such as the P-logic and N-logic differential XOR stages. Discrete output inverters are allowed only where one series logic fet exists in a dynamic stage, such as the $md-AND-mr$ P-logic stage. N-logic stages are used to drive the cell outputs to take advantage of higher N mobility in critical inter-cell communication timing.

The multiplier cell was simulated. [5] at clock rates in excess of 100 MHz with SPICE using 2-micron design rules. Substantially higher speeds should be possible using more advanced processes.

A preliminary layout of the multiplier cell, with most metal-2 bussing removed, is shown in Figure 9. It is 120 μm by 150 μm or 18,000 μm_2, compared to 99,000 μm_2 for the standard-cell prototype implementation.

6 Conclusions

Arrays of systolic serial-parallel multiplier elements have been proposed as an alternative to SIMD mesh arrays of serial adders for multiplication intensive parallel applications requiring relatively little operand storage capacity. This type of machine is suited for a narrower class of problems than conventional SIMD mesh arrays but broader than for special purpose systolic machines. Targeted applications include image processing and compression, particularly those involving convolution based algorithms, such as Laplacian pyramid encoding [5].

A new variation of single-clock NORA CMOS circuits is presented for application in high speed systolic multiplication networks. A design methodology is proposed that emphasizes matching the properties of the array design at the system, register (multiplier

[5]Spice simulations and layout by M. Feister, D. Virag, and D. Mathews, of Montana State University, using Tektronix, Inc. Tekspice, Tektronix MFET Level 2 device models, and slow speed device parameters for the MOSIS 2-micron (drawn gate length) SCMOS process, Revision 6 design rules, at T=100 degrees C and Vdd=4.5V.

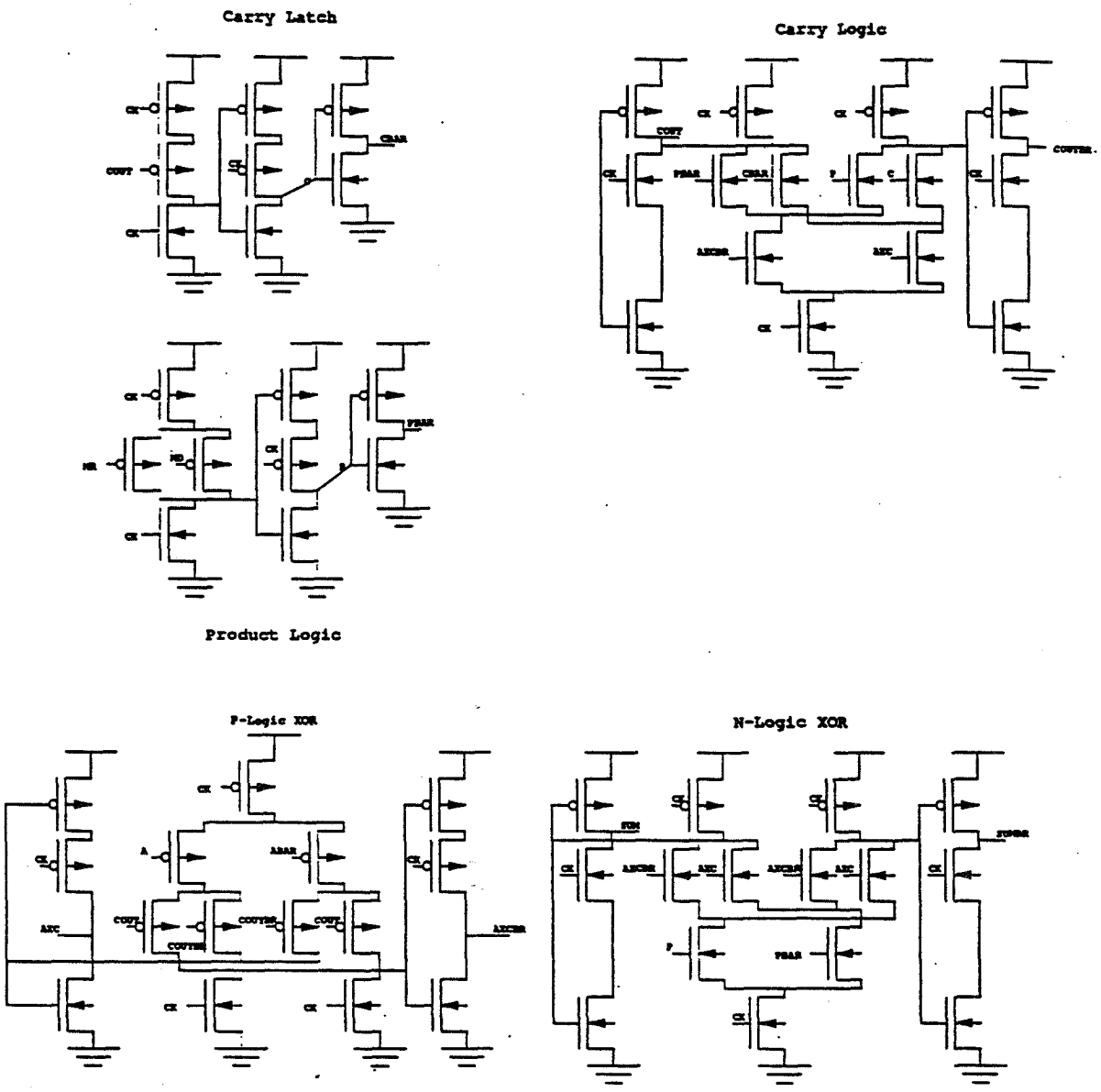


Figure 8: Circuit Diagram

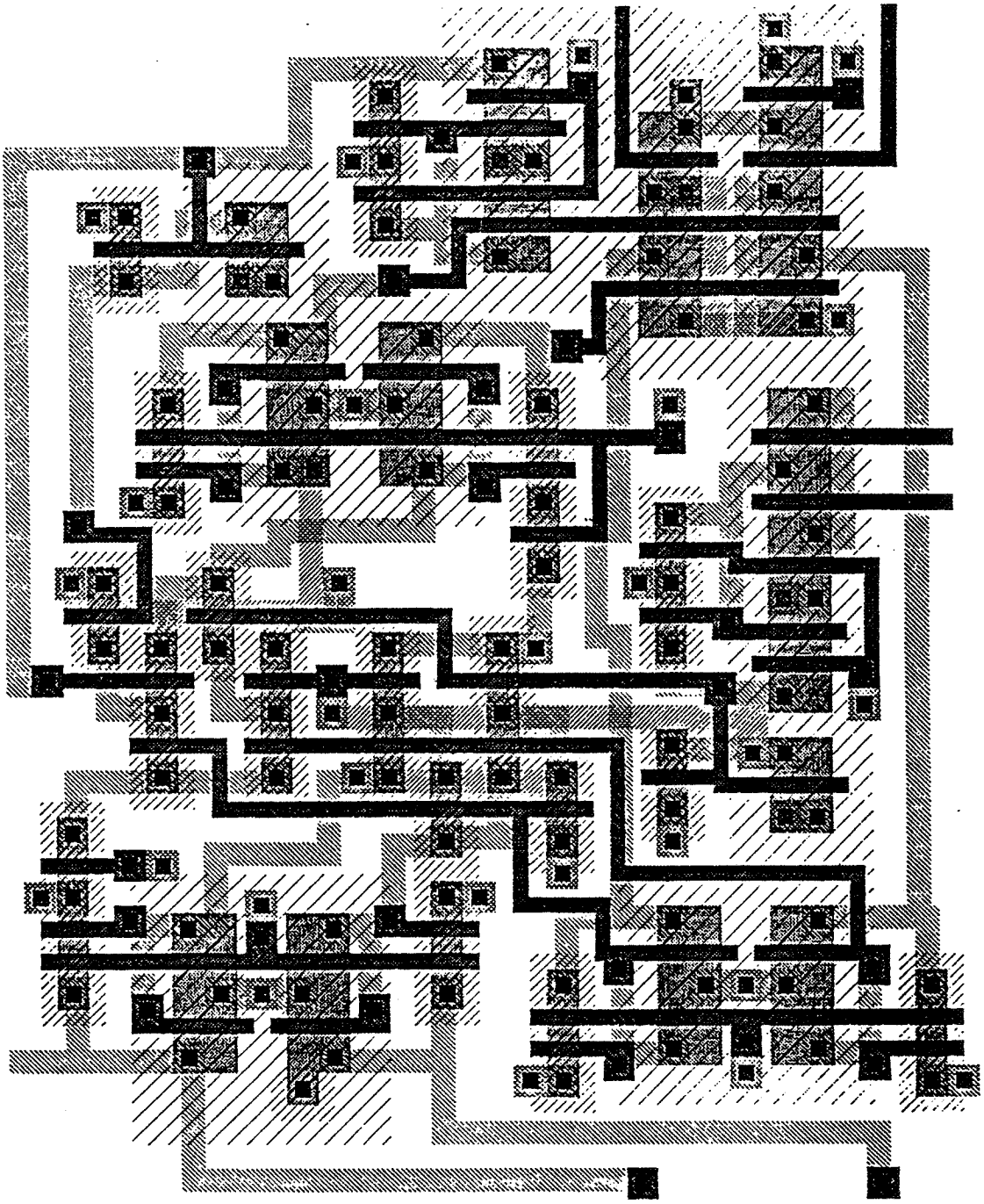


Figure 9: Layout Diagram

element), and circuit levels. The Berkeley OCT tools set is used to facilitate both custom VLSI design as well as the rapid development of simulation models and semicustom prototype components using logic synthesis methods. The circuit elements, serial multiplier modules, and design methodology are intended to serve as a set of building blocks to facilitate development of processing arrays for a variety of applications.

Currently, custom CMOS modules and semicustom prototypes for high performance systolic arrays of systolic multipliers are under development. It is hoped that this work will lead to the implementation of a large scale parallel array prototype for image and integer matrix processing. Future investigations will also include the application of these circuit modules and methodology to other types of computing arrays, including application specific systolic arrays, digital neural networks, and error correction encoders/decoders.

This work was supported by grants from the NASA Space Engineering Research Center at the University of Idaho, Moscow, and the Montana State University Engineering Experiment Station. The author would like to thank Dr. Gary Maki, Diane Mathews, and the students of EE501 at Montana State University for their invaluable contribution to this work.

References

[1] G. Barnes, R. Brown, M. Kato, D. Kuck, D. Slotnick, R. Stokes, "The ILLIAC IV Computer," IEEE Trans., C-17, vol. 8, pp. 746- 757, August, 1968.

[2] K. Batcher, "Design of a Massively Parallel Processor," IEEE Trans. Computers, vol. C-29, no. 9, Sept. 1980, pp. 836-840.

[3] K. Batcher, " The Architecture of Tomorrow's Massively Parallel Computer," Proc. 1st Symposium on the Frontiers of Massively Parallel Scientific Computation, Sept. 24, 1986, Greenbelt, MA, pp. 151-157.

[4] D. Blevins, E. Davis, R. Heaton, J. Reif, "BLITZEN: A Highly Integrated Massively Parallel Machine," Proc. 2nd Symposium on the Frontiers of Massively Parallel Computation, Oct. 10, 1988, Fairfax, VA, pp. 399-406.

[5] P. Burt and E. Adelson, "The Laplacian Pyramid as a Compact Image Code," IEEE Trans. on Communications, Vol. COM-31, No. 4, April 1983.

[6] E. L. Cloud, "The Geometric Arithmetic Parallel Processor," Proc. 2nd Symposium on the Frontiers of Massively Parallel Computation, Oct. 10, 1988, Fairfax, VA, pp. 373-381.

[7] T. Fountain, "A Survey of Bit-Serial Array Processor Circuits," Computing Structures for Image Processing, M. Duff ed., Academic Press, 1983.

[8] F. A. Gerritsen "A Comparison of the CLIP4, DAP, and MPP Processor-Array Implementations," Computing Structures for Image Processing, M. J. Duff ed., Academic Press, 1983.

[9] N. Goncalves and H. De Man, "NORA: A Racefree Dynamic CMOS Technique for Pipelined Logic Structures," IEEE J. Solid-State Circuits, vol. SC-18, no. 3, June 1983, pp. 261-266.

[10] D. Hampel, K McGuire, and K. Prost, "CMOS/SOS Serial-Parallel Multiplier," IEEE J. on Solid-State Circuits, Vol. SC-10, No. 5, October 1975.

[11] Y. Ji-Ren, I. Kaarlson, and C. Svensson, "A True Single- Phase-Clock Dynamic CMOS Circuit Technique," IEEE J. Solid-State Circuits, vol. SC-22, no. 5, October 1987, pp. 899-901.

[12] R. Krambeck, C. Lee, and H. Law, "High-Speed Compact Circuits with CMOS," IEEE J. Solid-State Circuits, vol. SC-17, June 1982, pp. 614-619.

[13] C. Mead and L. Conway, Introduction to VLSI Systems, Addison-Wesley, 1980.

[14] D. Slotnick, W. Borck, R. McReynolds, "The Solomon Computer," Proc. of AFIPS Fall Joint Comp. Conf., Wash. DC, 1962, pp. 97-107.

[15] H. Taub and D. Schilling, Digital Integrated Electronics, McGraw-Hill Inc., New York, 1977, pp. 349-355.

[16] S. Unger, "A Computer Oriented Toward Spatial Problems," Proceedings of the IRE, vol. 46, no. 10, pp. 1744-1750, October, 1958.

[17] N. Weste and K. Eshraghian, Principles of CMOS VLSI Design, a Systems Perspective, Addison-Wesley, 1985.

[18] K. Winters, EES Quarterly Report: Project 16233045, Bridger Processor Array Investigation, Montana State University, Feb. 12, 1988.

```
! MULREG multiplier register logic BDS model
! File: mulreg.bds
! Kel Winters
! Rev: 8-16-89
!
! Variables: nextprod & presentprod - product state variables
! a - addend, mr - multiplier, md - multiplicand
!
model mulreglog nextprod< 7 : 0 >= a < 0 >, mr < 0 >, md < 7 : 0 >, presentprod< 7 : 0 >;
routine cycle
        nextprod = (presentprodSR01) + (128 × a) + (mr × md);
endroutine;
endmodel;
```

Table 2: BDS Description of 8-bit Multiplier

```
! MULBIT multiplier register bit module BDNET description
! File : mulbit.bds
! Kel Winters
! Rev: 9-5-89
!
! Variables: a - addend, mr - multiplier, md - multiplicand,
! cin - carry in, cout - carry out, sum - sum of prod.
! p - product of mr and md.
!
model mulbit sum< 0 >, cout< 0 > = a< 0 >, mr< 0 >, md< 0 >, cin< 0 >;
routine cycle;
state p< 0 >;
p = mr AND md;
sum = a XOR p XOR cin;
cout = (a AND p) OR (a AND cin) OR (p AND cin);
endroutine;
endmodel;
```

Table 3: BDS Description of Bit-Multiplier Cell

PLA Realizations for VLSI State Machines

S. Gopalakrishnan, S. Whitaker, G. Maki and K. Liu
NASA Engineering Research Center
for VLSI System Design
University of Idaho
Moscow, Idaho 83843

Abstract - **A major problem associated with state assignment procedures for VLSI controllers is obtaining an assignment that produces minimal or near minimal logic. The key item in PLA area minimization is the number of unique product terms required by the design equations. This paper presents a state assignment algorithm for minimizing the number of product terms required to implement a finite state machine using a PLA. Partition algebra with predecessor state information is used to derive a near optimal state assignment. A maximum bound on the number of product terms required can be obtained by inspecting the predecessor state information. The state assignment algorithm presented is much simpler than existing procedures and leads to the same number of product terms or less. An area-efficient PLA structure implemented in a $1.0\mu m$ CMOS process is presented along with a summary of the performance for a controller implemented using this design procedure.**

1 Introduction

Most VLSI can be partitioned into data path and control logic. In many applications the control logic is realized with a PLA. Most chips perform operations that are synchronized with a clock and can be modeled as a synchronous machine. The control logic can be formally modeled as a synchronous sequential circuit. Once a formal description of the control logic is given in the form of a flow table or equivalent representation, a state assignment procedure is invoked to generate a binary encoding of the states.

A main problem associated with the state assignment procedure is obtaining an assignment that produces minimal or near minimal logic. Finding a valid state assignment that produces a valid hardware realization of the flow table is trivial; however, finding an assignment that has near minimal hardware is not. The state assignment process is combinatorial in nature and fits into the category of NP- complete problems. The lower bound on the number of assignments for a flow table with 2^q states is $2^q!$. Enumeration, even for medium sized machines, is not practical.

The state assignment problem has been a research subject for a long time [1,2,3], however, the problem has gained renewed interest with the increasing complexity of VLSI circuits [4,5,6,7] that realize sequential circuits. Sequential circuits are often realized with Programmable Logic Arrays (PLA). The size of a PLA is related to the number of unique

product terms required in the design equations. The number of literals in each product term is not the important issue. Hence finding a MSP expression does not necessarily lead to a minimal PLA but producing a solution that uses a minimum number of product terms does.

This paper presents a state assignment algorithm that can be used to reduce the number of product terms needed to implement a state machine using a PLA and also presents a dynamic, area-efficient PLA architecture. Section 2 of this paper deals with the design algorithms. The state assignment algorithm given in this paper is much simpler than the existing procedures [6,7] and leads to the same number of product terms or less. Section 3 applies the state assignment algorithm to an example. Section 4 deals with the PLA architecture and Section 5 reports the performance of a controller designed using the algorithms described in this paper.

2 Design Algorithms

Before demonstrating the design algorithms, some basic definitions must be reviewed [8,9] and established.

Definition 1 *A τ partition is a two block partition that partitions the states that are coded 1 by a state variable from those coded 0.*

If $\tau_1 = \{S_iS_j; S_kS_l\}$, where S_i, S_j, S_k and S_l are internal states of a state machine, then S_i and S_j are coded 1 by y_1 and the states S_k and S_l are coded 0 by y_1.

Definition 2 *A total circuit state of a sequential machine is a pair (S_i, I_p), where S_i is an internal state of the machine and I_p is a member of the set of input states.*

The total circuit state of the machine uniquely specifies the current state of the machine, ie. the internal state and the input state.

Definition 3 *An internal state S_j has a total predecessor state $S_k * I_p$, if the circuit transitions from S_k to S_j when the input I_p is applied.*

Predecessor states specify the next state values for each next state variable Y_i and output state variable Z. If y_i encodes state S_j 1(0), then everywhere S_j appears in the flow table, $Y_i = 1(0)$. The total predecessor states uniquely specify the next state entries of each Y_i and Z.

Partition algebra can be used to formalize the design equations. Let τ_i code S_j and S_k with a 1. Let the total predecessor states for S_j and S_k be $S_l * I_p$, $S_m * I_q$ and $S_n * I_r$, $S_o * I_s$ respectively. Then the next state partition of Y_i is $\{S_l * I_p, S_m * I_q, S_n * I_r, S_o * I_s; \ldots\}$. The design equation for state variable Y_i is given below.

$$Y_i = S_l \cdot I_p + S_m \cdot I_q + S_n \cdot I_r + S_o \cdot I_s \tag{1}$$

This equation specifies that Y_i consists of four product terms, each covering a total circuit state. Since y_i codes states S_j and S_k with a 1, it should attain a value of 1 whenever the circuit transitions to the states S_j and S_k. These transitions occur when the circuit assumes the total predecessor states for either S_j or S_k. Covering all predecessor states where $Y_i = 1$, produces a sum-of-products expression for Y_i.

Each predecessor state corresponds to one next state entry; the total number of predecessor states equals the number of specified entries in the flow table. Since each predecessor state produces a unique product term, the maximum bound on the number of unique product terms is equal to the number of specified entries in the flow table. If product terms can be shared in generating Y_i and Z_j, then the number of entries in the flow table is an upper bound on the number of product terms needed in a PLA.

Whenever a state variable y_i assigns a value 0 to a state, the corresponding product term for Y_i need not be generated. This produces two important observations:

1. States which are coded with more 0's than 1's require fewer product terms.
2. The state that appears most often (the greatest number of predecessor states) ought to be coded all 0.

Outputs play an important role in the state assignment process which seeks to minimize the total number of product terms. A product term is needed for all total circuit states that require an output of 1 and its presence is independent of the state assignment. The state assignment only determines which product term is needed. Since product terms are needed for every total circuit state with an output = 1, these states are ignored in the first step of the state assignment procedure shown next. The design procedure for a Mealy type machine [10,11] is outlined next.

State Assignment Procedure

1. Identify all the predecessor terms necessary to generate the outputs and remove them from the predecessor table.
2. Assign the all zero state as the state with the maximum number of remaining predecessor terms in the predecessor table. If the inputs are decoded using the same PLA, then expand all the inputs to their corresponding expressions and count the number of product terms that should be generated.
3. Apply the Armstrong-Humphrey adjacency conditions [2] to complete the remaining part of the state assignment. The conditions that are applicable for PLA based implementation are:
   (a) States that have the same next state for a given input should be given adjacent assignments (In general an assignment which allow a single product term covering for the corresponding predecessor state terms).
   (b) States that have the same output for the given input should be given adjacent assignments (In general an assignment which allow a single product term covering for the corresponding predecessor state terms).

	I_1	I_2	I_3
A	C/10	B/00	E/10
B	E/11	B/00	D/10
C	A/10	B/00	E/10
D	A/00	D/01	B/10
E	C/00	D/00	C/01

Table 1: Sample Flow Table

Each output variable Z_j can be treated the same as the next state variable Y_i. The expression for Z_j covers the predecessor terms where $Z_j = 1$. The number of product terms for each Z_j is independent of the state assignment. This fact is useful in selecting the state assignment. The objective of the state assignment generation is to derive an assignment which minimizes the number of product terms that need to be covered. Since predecessor states that are present in the generation of Z_j must be covered, the state assignment selection can impact only those predecessor states that are not contained in any Z_j. The first step in the algorithm incorporates this action. When using this algorithm for a Moore type machine [10], the designer should try to assign states such that they map into outputs.

3 Design Example

Table 1 gives the flow table is used to demonstrate this algorithm. The first step is to list the predecessor states for the state machine of Table 1. Table 2 shows this list. The product terms required to generate the outputs have been listed in Table 3. These terms are necessary for the outputs and are removed from consideration for state assignment selection. The reduced predecessor table is given in Table 4. Following the second step in the state assignment algorithm, State B is selected as the all zero state since it contains the largest number of predecessor terms. Applying the adjacency conditions, the state assignment of Equation 2 is derived.

$$\begin{aligned} \tau_1 &: \ ED; ABC \\ \tau_2 &: \ AE; BCD \\ \tau_3 &: \ ACDE; B \end{aligned} \tag{2}$$

From the τ partitions in Equation 2, state A is coded 0 by y_1 and 1 by y_2, y_3. Y_1 is coded 1 by the states E and D, leading to the following design equation for Y_1.

$$Y_1 = \ B \cdot I_1 + A \cdot I_3 + C \cdot I_3 + D \cdot I_2 + E \cdot I_2 + B \cdot I_3 \tag{3}$$

Similarly, the design equations for the other state variables and outputs are given below.

State	Total Predecessor State
A	$C*I_1, D*I_1$
B	$A*I_2, B*I_2, C*I_2, D*I_3$
C	$A*I_1, E*I_1, E*I_3$
D	$D*I_2, E*I_2, B*I_3$
E	$B*I_1, A*I_3, C*I_3$

Table 2: Predecessor Table

Output	Product terms
Z_1	$A*I_1, B*I_1, C*I_1, A*I_3,$ $B*I_3, C*I_3, D*I_3$
Z_2	$B*I_1, D*I_2, E*I_3$

Table 3: Output Terms

$$
\begin{aligned}
Y_2 = \;& B\cdot I_1 + A\cdot I_3 + C\cdot I_3 + \\
& C\cdot I_1 + D\cdot I_1 \\
Y_3 = \;& B\cdot I_1 + A\cdot I_3 + C\cdot I_3 + \\
& C\cdot I_1 + D\cdot I_1 + B\cdot I_3 + \\
& D\cdot I_2 + E\cdot I_2 + A\cdot I_1 + \\
& E\cdot I_1 + E\cdot I_3 \\
Z_1 = \;& A\cdot I_1 + B\cdot I_1 + C\cdot I_1 + \\
& A\cdot I_3 + B\cdot I_3 + C\cdot I_3 + \\
& D\cdot I_3 \\
Z_2 = \;& B\cdot I_1 + D\cdot I_2 + E\cdot I_3
\end{aligned}
\tag{4}
$$

The total number of product terms generated is 12, including the outputs. The state assignment uses 3 state variables. This is the same number of product terms generated using KISS [6]. By making use of the adjacencies in the state assignment, the following reduced design equations can be obtained.

State	Total Predecessor State
A	$D*I_1$
B	$A*I_2, B*I_2, C*I_2$
C	$E*I_1$
D	$E*I_2$
E	

Table 4: Reduced Predecessor Table

$$
\begin{aligned}
Y_1 &= B \cdot I_1 + (A + C + B)I_3 + D \cdot I_2 + E \cdot I_2 \\
Y_2 &= B \cdot I_1 + (A + C)I_3 + (C + D)I_1 \\
Y_3 &= I_1 + (A + B + C)I_3 + +D \cdot I_2 + E \cdot I_2 + E \cdot I_3 \\
Z_1 &= (A + B + C)I_1 + (A + B + C)I_3 + D \cdot I_3 \\
Z_2 &= B \cdot I_1 + D \cdot I_2 + E \cdot I_3
\end{aligned} \tag{5}
$$

This leads to 10 unique product terms to realize the circuit. The design equations in terms of the state variables are as follows.

$$
\begin{aligned}
Y_1 &= y_1'y_2'y_3' \cdot I_1 + y_1' \cdot I_3 + \\
&\quad y_1y_2'y_3 \cdot I_2 + y_1y_2y_3 \cdot I_2 \\
Y_2 &= y_1'y_2'y_3' \cdot I_1 + y_1'y_3 \cdot I_3 + \\
&\quad y_2'y_3 \cdot I_1 \\
Y_3 &= I_1 + y_1' \cdot I_3 + y_1y_2'y_3 \cdot I_2 + \\
&\quad y_1y_2y_3 \cdot I_2 + y_1y_2y_3 \cdot I_3 \\
Z_1 &= y_1' \cdot I_1 + y_1' \cdot I_3 + y_1y_2'y_3 \cdot I_3 \\
Z_2 &= y_1'y_2'y_3' \cdot I_1 + y_1y_2'y_3 \cdot I_2 + y_1y_2y_3 \cdot I_3
\end{aligned} \tag{6}
$$

4 PLA Architecture

The architecture for the PLA implementation is shown in the block diagram of Figure 1. The main paths are represented schematically in Figure 2. The AND and OR planes are both precharged to a 1 by p-channel transistors when $\phi = 1$. During precharge, n-channel evaluate transistors for the OR and AND planes disconnect the ground structures to avoid a ratioed DC current path. The input latch is enabled to capture new state and input information during the precharge time. The input lines to the AND core must propagate and settle before $\phi \rightarrow 0$. The OR plane precharge transistors are gated by a dummy line through the AND plane. This dummy line is constructed to be the slowest possible configuration for an AND plane term such that the OR plane remains in a precharge state after $\phi \rightarrow 0$ until all the AND plane output lines are settled. This self-timed concept avoids charge sharing and races between the AND and OR planes during evaluation and is the same concept used in RAM design.

When $\phi \rightarrow 0$, first, the AND plane evaluates; then the OR plane evaluates. The output of the OR plane is then captured in the output Flip Flops. Two Flip Flop designs are used for the state and output registers. One is set and the other is reset by a control signal. The control input is tied to the reset condition for the state machine and an appropriate latch is chosen for each state variable and output such that the initial state is reached under any reset condition.

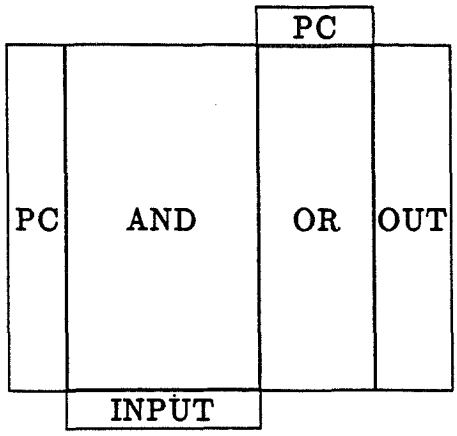


Figure 1: PLA Block Diagram

5 Results

The state assignment procedure, design techniques and PLA implementation were applied to the design of the PLA controller shown in Figure 3 [12]. The controller required 13 states, 4 state variables, 20 outputs and 139 PLA product terms. The circuit was drawn in a $1.0\mu m$, double metal CMOS process utilizing minimum transistor sizes in the core. The resulting PLA core was $270.2\mu m$ by $642.2\mu m$. Capacitance information was then extracted, and SPICE simulations were run to determine the operating frequency and margins. The controller operated at 25 MHz under 3σ worst case speed parameters at $100°C$ and $V_{dd} = 4.5V$. The AND plane inputs run in both polysilicon and Metal 2 through the core, and poly was occasionally shorted to avoid significant RC time delays for propagating the input signals. The width of the OR core was sufficiently small for this PLA such that no periodic shorting was needed. Speed of operation is limited by the self-timing circuit. If timing signals were available for the AND and OR plane, the implementation would operate at 35 MHz under 3σ worst case speed parameters at $100°C$ and $V_{dd} = 4.5V$.

References

[1] J. Hartmanis, "On The State Assignment Problem for Sequential Machines", IEEE Transactions on Electronic Computers, Vol. EC-10, pp. 157-167, Jun., 1961

[2] D. Armstrong, "Efficient Assignment of Internal Codes to Sequential Machines", IRE Transactions on Electronic Computers, vol EC-11, pp. 611-622, Oct., 1962

[3] J. R. Story, H. J. Harrison and E. A. Reinhard, "Optimum State Assignment for Synchronous Sequential Circuits", IEEE Transactions on Computers, Vol. C-21, pp. 1365-1373, Dec., 1972

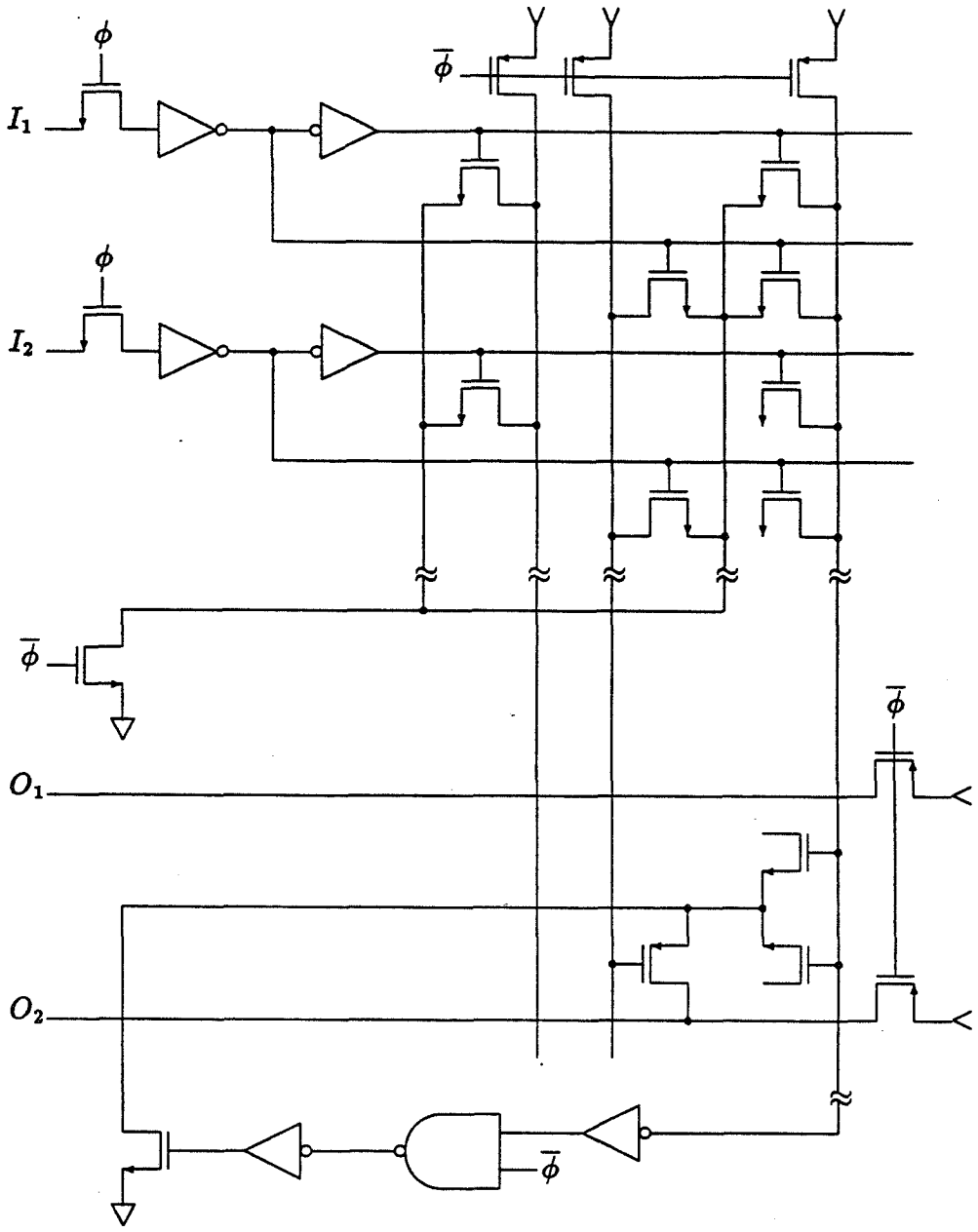


Figure 2: PLA Path Schematic

[4] G. Michelli, R. K. Brayton and A. L. Sangiovanni-Vincentelli, "Computer-aided Synthesis of PLA-based Finite State Machines", Proceedings of ICCAD-84, pp. 154-157, Sep., 1984

[5] T. Sasao, "Input Variable Assignment and Output Phase Optimization of PLA's", IEEE Transactions on Computers, Vol. C-33, pp. 879-894, Oct., 1984

[6] G. Michelli, R. K. Brayton and A. L. Sangiovanni-Vincentelli, "Optimal State Assignment for Finite State Machines", IEEE Transactions on CAD, Vol. CAD-4, pp. 269-285, Jul., 1985

[7] R. Amann and U. G. Baitinger, "Optimal State Chains and State Codes in Finite State Machines", IEEE Transactions on CAD, Vol. CAD-8, pp. 153-170, Feb., 1989

[8] J. Tracey, "Internal State Assignments for Asynchronous Sequential Machines", IEEE Transactions on Electronic Computers, Vol. EC-15, pp. 551-560, Aug., 1966

[9] G. Maki, D. Sawin and B. Jeng, "Improved State Assignment Selection Tests", IEEE Transactions on Computers, Vol. C-21, pp. 1443-1449, Dec., 1972

[10] C. Roth, *Fundamentals of Logic Design*, 3rd Ed., St. Paul, Minn., West Publishing, 1985

[11] D. Lewin, *Design of Logic Systems*, England, Van Nostrand Reinhold, 1985

[12] S. Gopalakrishnan, S. Whitaker, G. Maki and J. Gibson, "Simple Partition Algebra Based State Assignments", Hewlett-Packard VLSI Design Technology Conference, Portland, Ore., May 1989

This work was supported in part by NASA contract NAGW-1406.

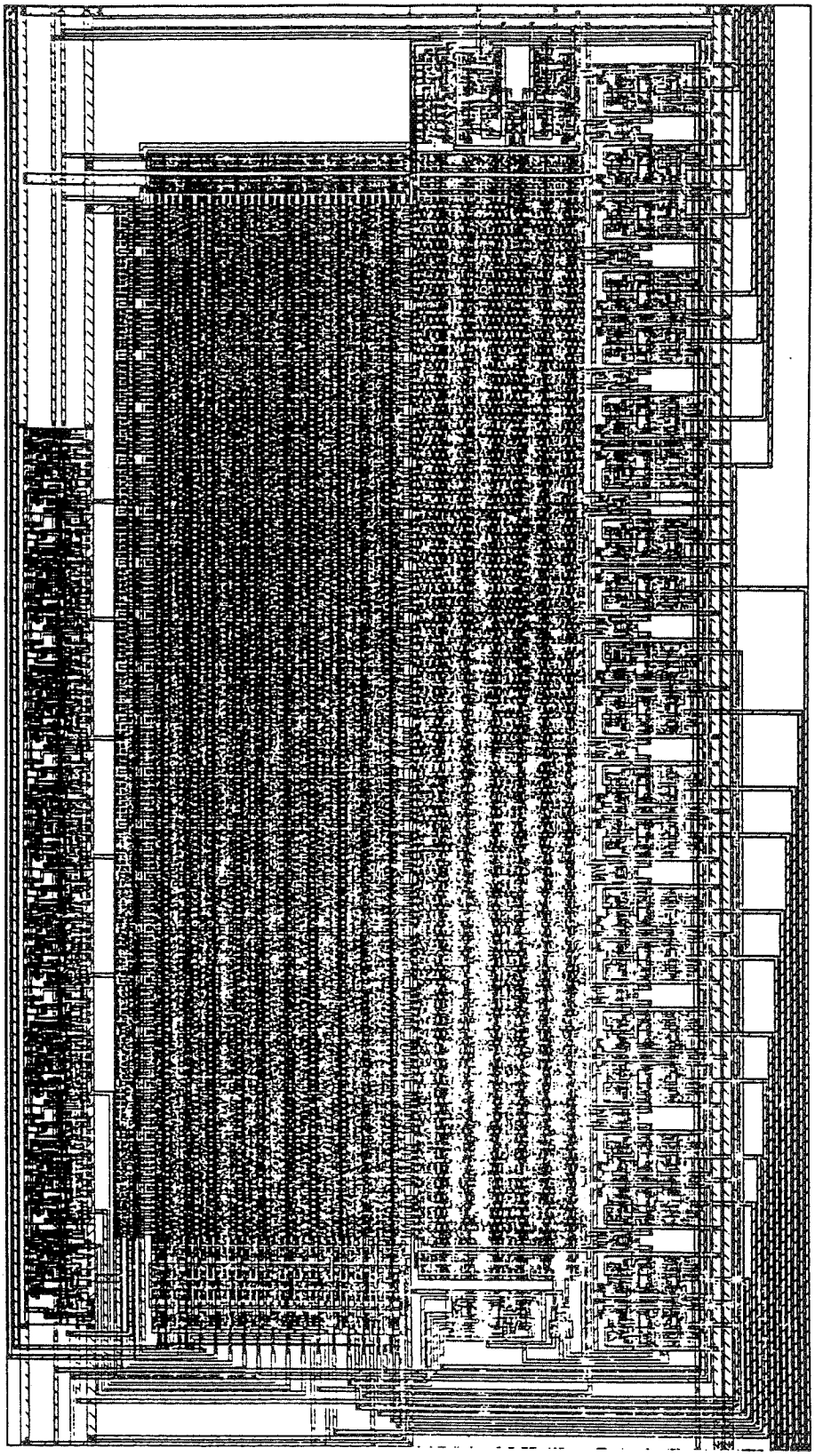


Figure 3: PLA Layout Plot

A Programmable Architecture for CMOS Sequential Circuits

S. Whitaker, G. Maki and M. Canaris
NASA Engineering Research Center
for VLSI System Design
University of Idaho
Moscow, Idaho 83843

Abstract - **This paper presents a programmable architecture for sequential pass transistor circuits. The resulting circuits are such that a state machine with N states and M outputs is constructed using a single layout replicated $N + M$ times.**

1 Introduction

Control circuits in digital logic are often designed as finite state machines. Control often occupies a small portion of the overall chip area but a major portion of the logic design effort. Layout of these controllers is often random in nature. Pass transistors have been studied over the past several years resulting in high speed and high density, practical combinational logic circuits [1]. This paper utilizes new design methods for sequential pass transistor circuits, which result in circuits such that the realization for each next state variable and output variable is identical for a given flow table [2]. Thus, a state machine with N states and M outputs can be constructed using a single layout replicated $N + M$ times. The personalization of each state variable is made in the input pass variables applied to the circuit. The number of paths in the network for each state variable is a function of the flow table, not the state assignment.

Synchronous sequential circuits can be drawn with nicely structured PLA architectures. Random logic in VLSI is normally avoided because of the increased cost of layout, verification, and design when compared with a regular architecture. Structured designs also are more easily set up for programmatic generation. Attempts at structured asynchronous sequential circuit design have been pursued in the past [3,4]. This architecture can reduce the required design effort and lends itself to programmatic generation.

2 Design Equations

The circuits developed in [2] using the Tracey, Liu and Tan state assignments resulted in networks which were identical in structure for each state variable. The personalization of each state variable was made in the input pass variables applied to the circuit. The number of paths in the network for each state variable is a function of the flow table, not the state assignment and is equal to the number of ρ partitions. This architecture is suitable for

	I_1	I_2	I_3	y_1	y_2	y_3
A	**A**	**F**	**E**	0	1	0
B	A	**B**	D	0	0	0
C	A	**C**	D	0	0	1
D	**D**	B	**D**	1	0	0
E	D	C	**E**	1	1	1
F	D	**F**	E	1	1	0

Table 1: Flow table with Liu assignment.

	Y_1	Y_2	Y_3
x_7	0	1	0
x_6	1	0	0
x_5	1	1	0
x_4	0	0	0
x_3	0	0	1
x_2	1	1	1
x_1	1	0	0

Table 2: Liu circuit inputs.

establishing a structured layout. Since Liu assignments resulted in the most economical circuits, this assignment will be employed.

The next state design equations for the flow table shown in Table 1 are

$$\begin{aligned} Y_1 = \; & \overline{y_1}\, I_1(0) + y_1\, I_1(1) + y_2\, \overline{y_3}\, I_2(1) + \\ & \overline{y_2}\, \overline{y_3}\, I_2(0) + y_3\, I_2(0) + y_2\, I_3(1) + \\ & \overline{y_2}\, I_3(1) \end{aligned} \tag{1}$$

$$\begin{aligned} Y_2 = \; & \overline{y_1}\, I_1(1) + y_1\, I_1(0) + y_2\, \overline{y_3}\, I_2(1) + \\ & \overline{y_2}\, \overline{y_3}\, I_2(0) + y_3\, I_2(0) + y_2\, I_3(1) + \\ & \overline{y_2}\, I_3(0) \end{aligned} \tag{2}$$

$$\begin{aligned} Y_3 = \; & \overline{y_1}\, I_1(0) + y_1\, I_1(0) + y_2\, \overline{y_3}\, I_2(0) + \\ & \overline{y_2}\, \overline{y_3}\, I_2(0) + y_3\, I_2(1) + y_2\, I_3(1) + \\ & \overline{y_2}\, I_3(0) \end{aligned} \tag{3}$$

The circuit diagram of Figure 1 shows the logic for the next state variable Y_i. The logic is replicated three times and the inputs are driven by the next state information as shown in Table 2 to form the total circuit diagram.

By observing the circuit diagram of Figure 1 and the circuit input matrix in Table 2, three distinct sections are shown and a forth is implied. The input section is a coding of 1's and 0's to program the state assignment for a given state variable. The ρ partition section programs the structure of the flow table into the sequential circuit and is identical for each state variable. The buffer section restores the threshold drop on the 1 level out

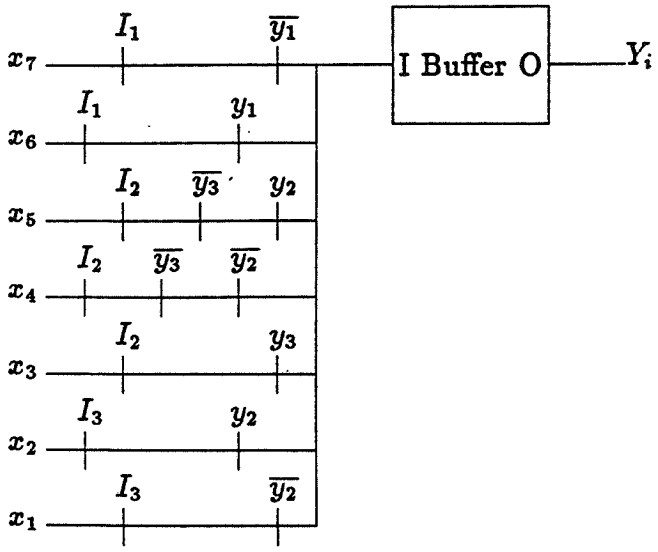


Figure 1: Circuit diagram for Liu state assignment.

of the pass network and eliminates essential hazards on 0–0 cross over of the inputs. The fourth section that is implied if the circuit is to be made programmatically generated, is a programmable feed back section. The block diagram of the architecture is shown in Figure 2.

By overlaying the architectural block diagram with the logic, the layout form can be envisioned. Figure 3 shows the general logic/layout form. The buffer section requires one input and two outputs along with power supply lines. The latching buffer is driven by the output of the pass transistor network in the ρ partition section and feeds back to drive two of the state variable lines in that section. This would be drawn as a cell such that it fits the height of a minimum number of ρ partition variable lines. The feedback section has both signals from the buffer in one layer of interconnect arranged such that a contact can be dropped to the layer driving potential gates in the partition section. The ρ partition section would be an array such that series structures driven by state variables and primary inputs can be programmed. The programming could be accomplished with transistor structures and jumper connections. The input section would consist of Vdd and Vss supply lines which would be programmed by contacts on the input node lines to the pass array.

The programming features needed by the architecture shown in Figure 3 can be more easily seen by an example. Figure 4 shows an overlay of the logic for state variable y_1 from the Liu state assignment circuit. The feedback section has contacts programmed to connect the buffer outputs to the y_1 and $\overline{y_1}$ lines driving through the pass network. Transistors and jumpers are programmed in the ρ partition section to create the required pass network. The input variables shown in Table 2 are programmed as connections to the Vdd and Vss supply lines running through the input section.

The other state variables could likewise be formed by changing the feed back and input

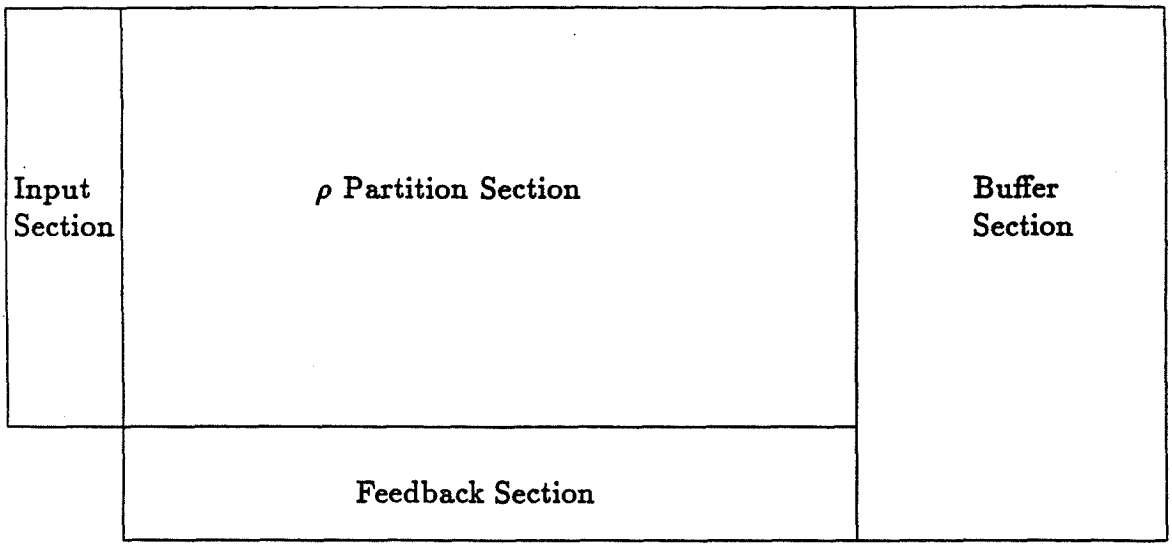


Figure 2: Architecture for programmable controller.

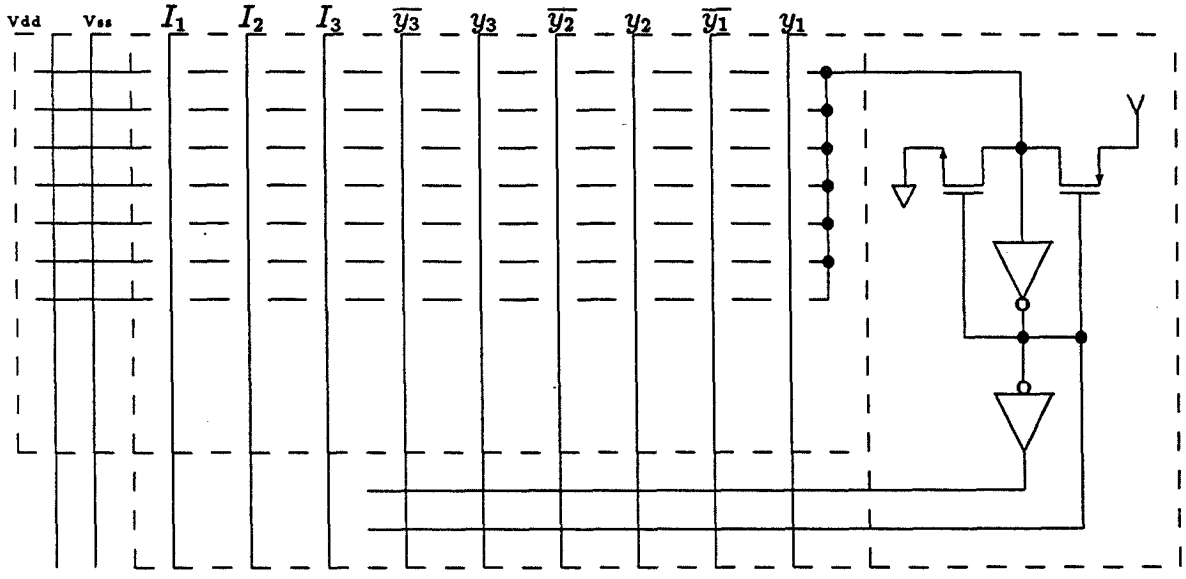


Figure 3: Architecture with logic overlay.

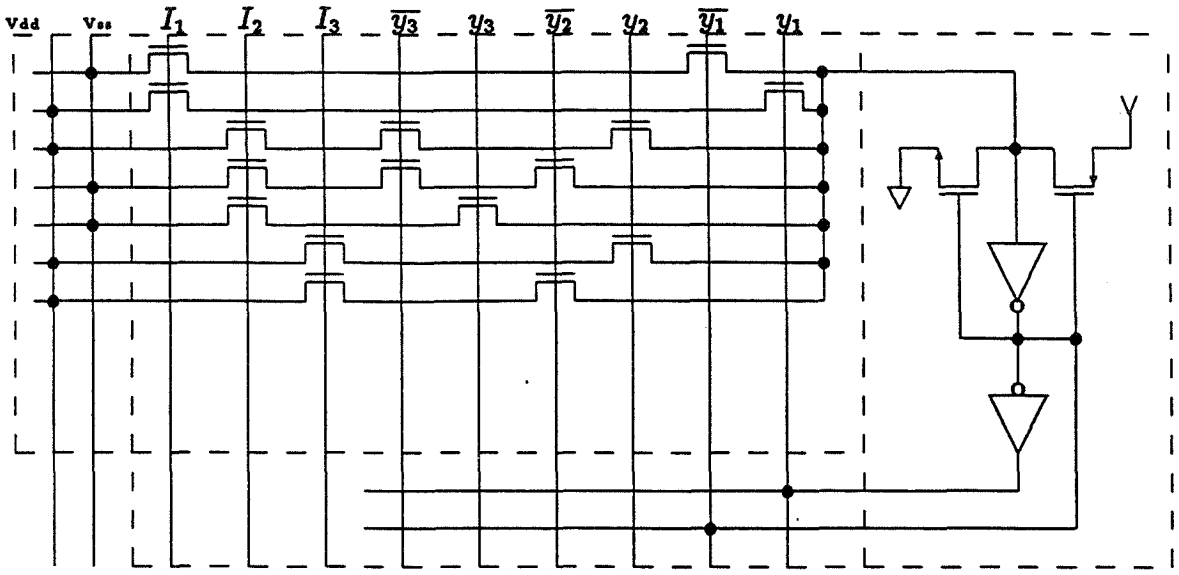


Figure 4: Architecture with y_1 logic overlay.

programming and abutting the cells together. The complete circuit has been drawn in a $1.6\mu m$ CMOS double metal N-well process and is shown in Figure 5.

The feed back lines are metal 1 which is programmed by placing a contact to the poly lines feeding the gates in the ρ partition section. The size of the machine (93.2 μm by 121.6 μm) allowed the state lines and input lines to run in poly. No metal 2 was used in the ρ partition section so that these lines could be run in metal 2 for machines requiring a large number of state variables or a large number of partition lines.

The pass transistor matrix is programmed with either a diffusion – contact – metal 1 transistor structure or a metal 1 jumper. The transistors in the pass transistor network are sized such that the metal overlap of the contact rule is just met, forming minimum capacitance structures thus allowing maximum speed.

The input section has Vdd and Vss running in metal 2 with the programming vias dropping to metal 1 lines feeding the pass array. The state variable metal 1 lines are passed out of the cell under the Vdd and Vss lines of the input section to drive external requirements.

3 Performance

The pass network transistors were sized to minimize the node diffusion at $W_n = 3.2\mu m$. The first buffer inverter was sized with a $W_p = 6.4\mu m$ p-channel transistor. The n-channel transistor was the same size to lower the switch point of the inverter in order to compensate for the threshold loss on the 1 level out of the ρ partition pass transistor array. The second inverter in the buffer was also sized with $W_p = W_n = 6.4\mu m$ to minimize 1–1 cross over of the state variables and avoid any potential essential hazards. The feedback devices were weak transistors with $W_p = 2.8\mu m$, $L_p = 5.0\mu m$, $W_n = 2.8\mu m$ and $L_n = 10.4\mu m$. The

sizes were set to insure proper operation when these devices ratio with the pass network.

The state machine occupies an area 93.2 μm by 121.6 μm. The layout density is a very respectable 171.7 μm^2 (0.266 mil^2) per transistor or 54.75 μm^2 (0.085 mil^2) per transistor site. For a perspective, a single standard cell D flip flop in this same 1.6 μm double metal CMOS process is 70.4 μm by 139.2 μm [5]. The layout of Figure 5 which contains a 3 input, 5 state, 3 state variable state machine occupies an area only 1.16 times that of a single standard cell D flip flop drawn in the same process.

Parasitic capacitances were then extracted from the layout and a SPICE simulation was run to determine the operating frequency of the state machine. Worst case speed 3σ parameters were used in the simulations along with high temperature, $T_j = 100 \deg C$, low power supply, $Vdd = 4.5V$ and supply bus drops of $0.2V$. The inputs were assumed to have a rise and fall time of 1.0 nsec. Under these assumptions, the circuit ran in fundamental mode for 30 MHz input changes.

Typical speed parameters were then used in a simulations along with room temperature, $T_j = 25 \deg C$, and typical power supply, $Vdd = 5V$. The inputs were assumed to have a rise and fall time of 1.0 nsec. Under these assumptions, the circuit ran in fundamental mode for 100 MHz input changes.

3.1 Improvements

The operating speed can be improved by two means. First, the buffer could be sized to increase the speed of the state variables at the cost of increased dynamic power. An improvement in speed with no penalty could be achieved by laying out the ρ partition section such that the transistors driven by the circuit inputs would be next to the output. This places the last arriving signal next to the pass transistor network output node and maximizes the operating speed [6].

If speed needs to be optimized at the expense of programmability, then another improvement would be to reduce the logic. The reduction in logic for the ρ partition section would reduce the total node capacitance that must be charged in the pass array and would also reduce the gate capacitance driven by the state variable buffers.

References

[1] 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

[2] S. Whitaker and G. Maki, "Pass Transistor Asynchronous Sequential Circuits", IEEE JSSC, Vol. SC-24, Feb., 1989, pp.

[3] J. Jump, "Asynchronous Control Arrays", IEEE Transactions on Computers, vol C-23, no 10, Oct. 1974, pp. 1020-1029

[4] R. David, "Modular Design of Asynchronous Circuits Defined by Graphs", IEEE Transactions on Computers, vol C-26, no 8, Aug. 1977, pp. 727-737

[5] *EDS40 CMOS40 Standard Cell User Manual*, Santa Clara, California, Hewlett-Packard Company, 1986

[6] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*, Reading, Mass., Addison-Wesley, 1985, pp. 55-57

This work was supported in part by NASA under Contract NAGW-1406 and by the Idaho State Board of Education under Research Grant # 87-009.

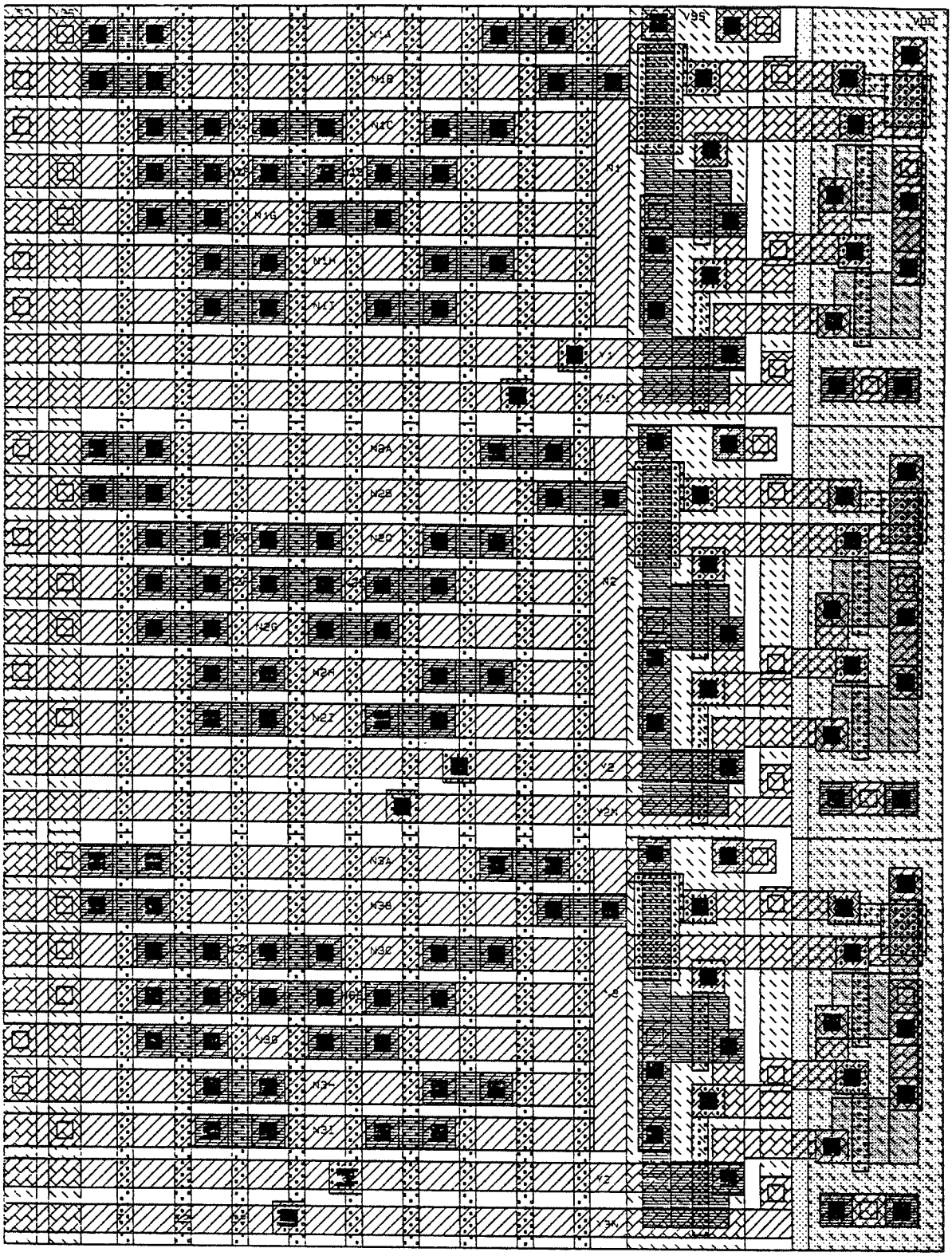


Figure 5: Layout for example Liu state machine.

A Bit Serial Sequential Circuit

S. Hu and S. Whitaker
NASA Engineering Research Center
for VLSI System Design
University of Idaho
Moscow, Idaho 83843

Abstract–**Normally a sequential circuit with n state variables consists a total of n unique hardware realizations, one for each state variable. All variables are processed in parallel. This paper introduces a new sequential circuit architecture that allows the state variables to be realized in a serial manner using only one next state logic circuit. The action of processing the state variables in a serial manner has never been addressed before.**

This paper presents a general design procedure for circuit construction and initialization. Utilizing pass transistors to form the combinational next state forming logic in synchronous sequential machines, a bit serial state machine can be realized with a single NMOS pass transistor network connected to shift registers. The bit serial state machine occupies less area than other realizations which perform parallel operations. Moreover, the logical circuit of the bit serial state machine can be modified by simply changing the circuit input matrix to develop an adaptive state machine.

1 Introduction

Most controllers in digital circuits are modeled as sequential circuits. The controller dictates the activity of a data path and interacts with external control signals. Since most VLSI circuits are controlled by a clock, the controller is represented as a synchronous sequential circuit.

In general, a sequential circuit processes all state variables in parallel with unique hardware realizations for each state variable. With n next state variables, n unique hardware circuits are needed to generate the variable values. Whitaker [1] has introduced a new design that produces identical hardware for each state variable. If the hardware is identical for each state variable, then the logical question to address centers around the concept of trying to develop a new architecture that has only one next state hardware circuit that is used to generate all state variables. The advantage is much less hardware circuitry. The disadvantage results from the same hardware being used n different times to calculate the n state variables. Thus the hardware savings in next state generating logic is a factor of n, while the time to calculate each next state increases by a factor of n.

It has been shown that pass transistor realizations of digital circuits possess the advantages of high density and speed [2]. A unique form of pass transistor circuits is known as

	I_1	I_2	I_3
A	C	B	A
B	D	C	B
C	E	D	C
D	F	E	D
E	A	F	E
F	B	A	F

Table 1: Example flow table.

Binary Tree Structure (BTS) circuits [3]. BTS circuits possess the property of often requiring fewer transistors and producing regular layouts. BTS networks are used to implement the combinational portion of the next state equations in the design presented here.

The next chapter of this work shows the design method used to realize traditional pass transistor sequential networks. Chapter 3 establishes the design procedure of bit serial state machines. Chapter 4 discusses adaptable circuit realization.

2 Regular State Machine Design

2.1 Pass Transistor State Machine Design

The logic that forms each next state equation Y_i consists of the following elements: a storage device (normally a flip-flop), next state excitation circuitry which generates the next state values to the flip-flop, and input logic. In classical state machine realizations, all next state equations are evaluated in parallel with independent logic. Present state information is fed back by state variables y_i to the excitation logic. The excitation logic is a combinational logic function of the input and state variable information. Current input and state variables select the specific next state value.

Whitaker has derived a unique realization for asynchronous sequential circuits that has identical circuitry for all of the next state equation circuitry[1]. This concept can be extended to synchronous sequential circuits discussed in this paper by replacing the transition paths of the asynchronous circuits with predecessor states. In the asynchronous case, a transition path defines the set of states that must have the same next state entry and a pass implicant covers each transition path, resulting in a unique network of pass transistors in the next state equations. In the synchronous case, predecessor states define the states that have the same next state entry and a pass implicant covers the predecessor states and is realized as a unique network of pass transistors in the next state equations. This notion of using predecessor states is described next.

Predecessor states for state S_i under an input, I_p, are states which have S_i as a next state entry. In Table 1, the predecessor state for C under I_1 is A. Whenever the circuit is in a predecessor state of S_i, the next clock pulse will effect a transition to S_i. Moreover, the next state is present at the input of the flip-flops prior to the clock pulse. At the next clock pulse, the flip-flops will assume the value of the code associated with S_i.

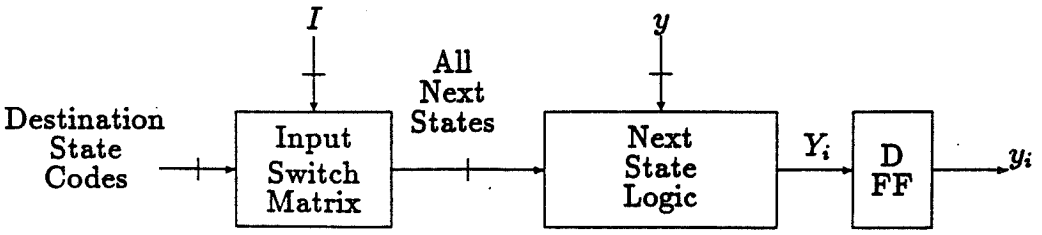


Figure 1: General block diagram.

Conceptionally, the new architecture operates as follows: For each predecessor state of S_i, there exists a pass transistor network in the excitation network that presents the next state value, S_i, to the flip-flops. The pass transistor network consists of a single pass implicant that covers each predecessor state such that when any state is entered, a unique pass transistor path is enabled that passes the proper next state value to the flip-flops. For example, in Table 1, there is a pass transistor path that presents the code for state C to the flip-flops when $I_1 = 1$ and the circuit is in state A.

All pass transistor implementations for the synchronous sequential circuits described in this paper operate as described above, including those that use PLA pass transistor networks to generate the next state equations. All next state equations are identical when they are realized with general pass networks that completely decode all the internal states. That is, if there are n state variables, then the pass network must decode all 2^n states. The value for the next state entries for each predecessor state for S_i is the code for S_i and the constants for this code are input to the pass network.

A parallel implementation for an architecture to realize sequential circuits is shown in Figure 1, where the next state logic is the general pass network. The destination state codes contain all possible destination states for the sequential circuit. Since every state of the flow table appears as a next state entry, all the specified states are contained in the destination code set. These codes are input to the input switch matrix which generates all the destination codes for a given input I_p as specified by I_p. The next state logic selects the unique destination code as specified by the current state variables y.

The following illustrates specifically how this architecture works. In general, let the circuit be in state S_k with input I_j. Let S_k be the predecessor for S_i. All destination states codes are normally input to the input switch matrix. Input I_j selects the codes for the destination states under I_j to be presented to the next state logic. The feedback that specifies that the present state is S_k enables one and only one path in the pass network to select the correct destination code, S_i, and presents this value to the flip-flops. This is depicted in Figure 2.

In summary, the input state selects the set of potential next states that the circuit can

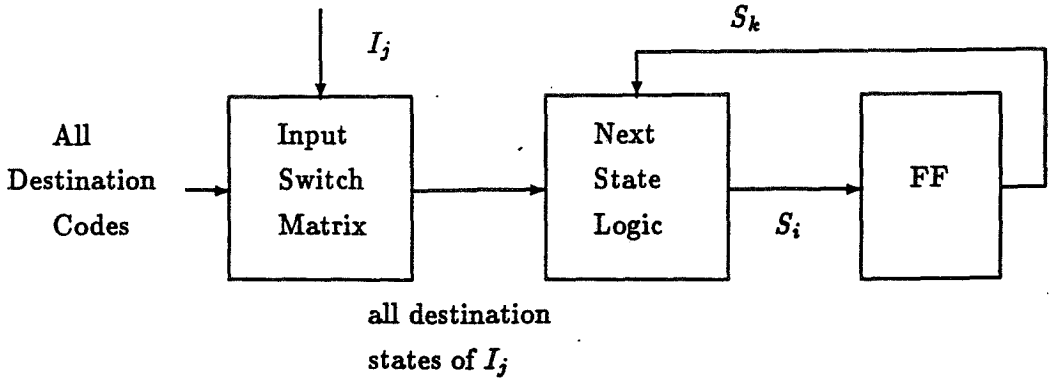


Figure 2: State machine operation.

y_1	y_2	y_3		I_1	I_2	I_3
0	0	0	A	010	001	000
0	0	1	B	011	010	001
0	1	0	C	100	011	010
0	1	1	D	101	100	011
1	0	0	E	000	101	100
1	0	1	F	001	000	101

Table 2: Example flow table with next state entries.

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 at the next clock pulse.

2.2 Design Procedure

The first step in the design process is to generate the state assignment. Any state assignment can be used, as long as no two distinct states have the same code. Consider the state assignment and next state entries shown in Table 2.

Let ρ_{S_i} define a state partition that partitions state S_i from all other states of a flow table. If states $S_0, S_1, \ldots S_j$ are the destination states for a given input, then state partitions $\rho_{S_0}, \rho_{S_1}, \ldots \rho_{S_j}$ must be covered in the design equations. For Table 2, the partitions are:

$$\begin{aligned}
\rho_a &= A; BCDEF \\
\rho_b &= B; ACDEF \\
\rho_c &= C; ABDEF \\
\rho_d &= D; ABCEF \\
\rho_e &= E; ABCDF \\
\rho_f &= F; ABCDE
\end{aligned} \tag{1}$$

The general design equation for each next state variable consists of a term for each of the ρ partitions and is as follows:

$$\begin{aligned} Y_i = \; & \rho_a I_1() + \rho_b I_1() + \rho_c I_1() + \rho_d I_1() + \rho_e I_1() + \rho_f I_1() + \\ & \rho_a I_2() + \rho_b I_2() + \rho_c I_2() + \rho_d I_2() + \rho_e I_2() + \rho_f I_2() + \\ & \rho_a I_3() + \rho_b I_3() + \rho_c I_3() + \rho_d I_3() + \rho_e I_3() + \rho_f I_3() \end{aligned} \tag{2}$$

By rearranging the partitions, the general next state design equation can be written as follows:

$$Y_i = \sum_{k=1}^{m} C(\rho_{S_k})[\sum_{l=1}^{p} I_l(x_{S_k i})] \tag{3}$$

where $C(\rho_{S_k})$ is the product expression covering state S_k, m is the number of states, p is the number of inputs, and $x_{S_k i}$ is the ith bit of the code for the destination state of S_k for y_i under input I_l. For this example,

$$\begin{aligned} Y_i = \; & \rho_a[I_1() + I_2() + I_3()] + \rho_b[I_1() + I_2() + I_3()] + \\ & \rho_c[I_1() + I_2() + I_3()] + \rho_d[I_1() + I_2() + I_3()] + \\ & \rho_e[I_1() + I_2() + I_3()] + \rho_f[I_1() + I_2() + I_3()] \end{aligned} \tag{4}$$

The next step specifies the constants that are passed. To do this, enter $\rho_j I_k(1)$ for the destination state under ρ_j where $Y_i = 1$ and to enter $\rho_j I_k(0)$ for the destination state under ρ_j where $Y_i = 0$. For example, the next state under the input I_1 for present state A or partition ρ_a is C. C has the code $y_1 = 0$, $y_2 = 1$ and $y_3 = 0$ and this information can be entered into the design equations as shown:

$$\begin{aligned} Y_1 = \; & \rho_a[I_1(0) + I_2() + I_3()] + \rho_b[I_1() + I_2() + I_3()] + \\ & \rho_c[I_1() + I_2() + I_3()] + \rho_d[I_1() + I_2() + I_3()] + \\ & \rho_e[I_1() + I_2() + I_3()] + \rho_f[I_1() + I_2() + I_3()] \end{aligned}$$

$$\begin{aligned} Y_2 = \; & \rho_a[I_1(1) + I_2() + I_3()] + \rho_b[I_1() + I_2() + I_3()] + \\ & \rho_c[I_1() + I_2() + I_3()] + \rho_d[I_1() + I_2() + I_3()] + \\ & \rho_e[I_1() + I_2() + I_3()] + \rho_f[I_1() + I_2() + I_3()] + \end{aligned} \tag{5}$$

$$\begin{aligned} Y_3 = \; & \rho_a[I_1(0) + I_2() + I_3()] + \rho_b[I_1() + I_2() + I_3()] + \\ & \rho_c[I_1() + I_2() + I_3()] + \rho_d[I_1() + I_2() + I_3()] + \\ & \rho_e[I_1() + I_2() + I_3()] + \rho_f[I_1() + I_2() + I_3()] \end{aligned}$$

By inspection of Table 2, the destination state codes for the remaining ρ partitions are incorporated into the design equations.

$$
\begin{aligned}
Y_1 = \; & \rho_a[I_1(0) + I_2(0) + I_3(0)] + \rho_b[I_1(0) + I_2(0) + I_3(0)] + \\
& \rho_c[I_1(1) + I_2(0) + I_3(0)] + \rho_d[I_1(1) + I_2(1) + I_3(0)] + \\
& \rho_e[I_1(0) + I_2(1) + I_3(1)] + \rho_f[I_1(0) + I_2(0) + I_3(1)] \\
\\
Y_2 = \; & \rho_a[I_1(1) + I_2(0) + I_3(0)] + \rho_b[I_1(1) + I_2(1) + I_3(0)] + \\
& \rho_c[I_1(0) + I_2(1) + I_3(1)] + \rho_d[I_1(0) + I_2(0) + I_3(1)] + \\
& \rho_e[I_1(0) + I_2(0) + I_3(0)] + \rho_f[I_1(0) + I_2(0) + I_3(0)] \\
\\
Y_3 = \; & \rho_a[I_1(0) + I_2(1) + I_3(0)] + \rho_b[I_1(1) + I_2(0) + I_3(1)] + \\
& \rho_c[I_1(1) + I_2(1) + I_3(0)] + \rho_d[I_1(0) + I_2(1) + I_3(1)] + \\
& \rho_e[I_1(0) + I_2(0) + I_3(1)] + \rho_f[I_1(1) + I_2(0) + I_3(0)]
\end{aligned}
\tag{6}
$$

The final step is to derive the covering equations for each ρ partition. Let $C(\rho_s)$ define the product term that partitions state S from all other states. For example, $C(\rho_a)$ is $\overline{y_1}\,\overline{y_2}\,\overline{y_3}$. With a unique code for each state, there is a unique product term for each state partition.

$$
\begin{aligned}
C(\rho_a) &= \overline{y_1}\,\overline{y_2}\,\overline{y_3} & C(\rho_d) &= \overline{y_1}\,y_2\,y_3 \\
C(\rho_b) &= \overline{y_1}\,\overline{y_2}\,y_3 & C(\rho_e) &= y_1\,\overline{y_2}\,\overline{y_3} \\
C(\rho_c) &= \overline{y_1}\,y_2\,\overline{y_3} & C(\rho_f) &= y_1\,\overline{y_2}\,y_3
\end{aligned}
\tag{7}
$$

The circuit in Figure 3 shows the logic for implementing state variable y_3. The logic of Figure 3 is replicated three times and the inputs are driven by the destination state information as shown in Table 2. Except for constant input values driving the input switch matrix, the design equations and pass transistor realizations for each state variable are identical.

3 Serial Implementation

A serial machine consists of a single next state logic circuit to calculate each next state variable. In order to utilize one hardware circuit to calculate all next state variables, the hardware must be identical for each next state variable. It is shown that the design presented in the previous section achieves the property of having identical hardware for each state variable.

An architecture that can be used to realize a bit serial sequential circuit is depicted in Figure 4. The circuit consists of a single next state variable calculator, a D flip-flop to store each next state variable value, a shift register, a state latch and a storage means to hold the destination codes.

Let the number of state variables be n, the number of states in the flow table be m and the number of inputs be p. Since the state variables are calculated serially, a shift register is used to store and generate the entire next state one bit at a time. The shift register length must be n. The next state of the sequential circuit is a function of the present state and the inputs. In the bit serial circuit, n clock pulses are needed to generate

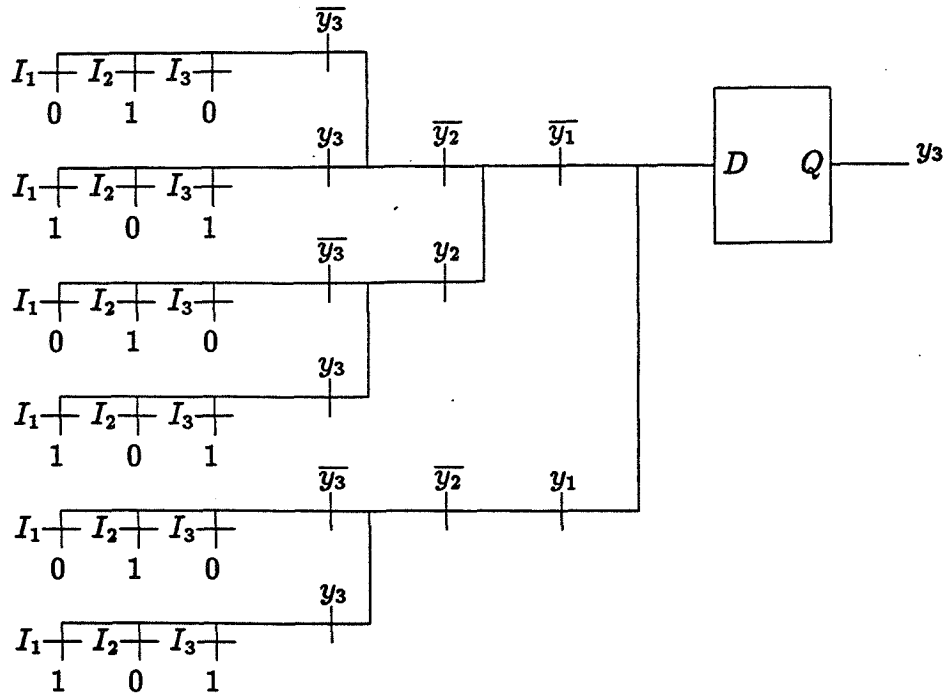


Figure 3: Example next state equation for y_3.

the next state. Since the present state of the machine is needed to generate each new bit of the next state, the present state must be held in a register until the entire next state is generated. A state latch is used to hold the present state value until the next state is generated. When the next state is completely formed, this latch receives a new input from the shift register.

The input state I_p selects one of the destination code set blocks which pass all the destination states in I_p to the next state logic. The destination state codes reside in a storage means such as a ROM, RAM or register stack. Let the bits of each destination codes set be noted as D_{ji}, where i denotes next state variable Y_i and j denotes the state in the flow table; j takes on values 1,2, ... m and i takes on values 1,2,... n. The destination code sets are organized such that the next state variables assume the columns and the next state codes assume the rows. Below is the destination code set for input I_1 for Table 1.

State	y1	y2	y3
A	0	1	0
B	0	1	1
C	1	0	0
D	1	0	1
E	0	0	0
F	0	0	1

Destination Code Set for I_1

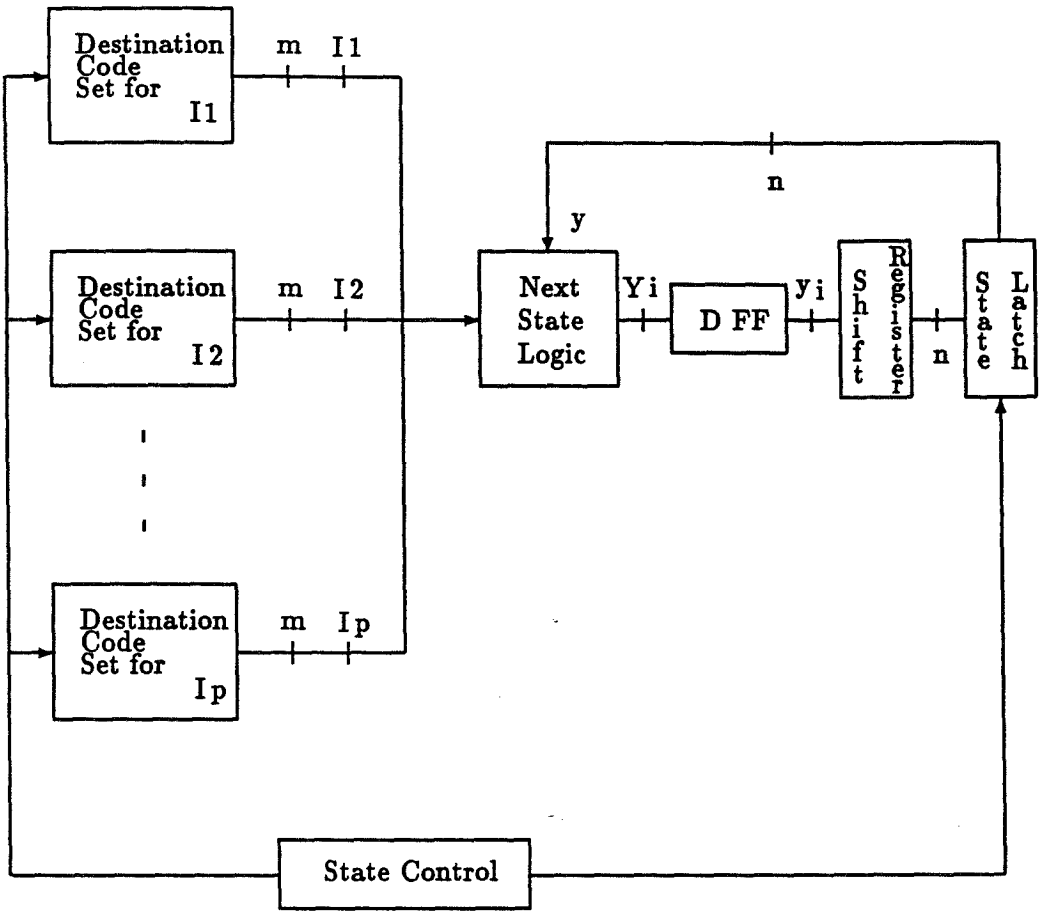


Figure 4: Bit serial architecture

The operation of the serial circuit can be described as follows: Assume the initial state is in the state latch. Next state variable Y_1 is calculated first. The input state passes the appropriate destination code set to the next state logic. The first column representing the next state values for y_1 is selected and next state variable Y_1 is calculated and gated into the D flip-flop. The second column in the destination code set then is selected, representing y_2, is input to the next state logic and Y_2 is calculated. Y_2 is gated into the D flip-flop at the same time as Y_1 is gated into the shift register. The process continues n times to calculate the next state. When the next state is generated, it is gated into the state latch which denotes the new present state of the sequential circuit. At this time, a new input state can be accepted for the circuit to begin the process again. The operation of the entire process is controlled by a state controller which can be implemented with a simple ring counter.

Consider the example from flow table Table 1 to illustrate the operation of this circuit. Let the initial state be B and input I_1. The state latch would have initial value 001, the code for B. To calculate Y_1, 001100 from the destination code set for I_1 is passed to the next state logic. With present state B, path $\overline{y_1}\overline{y_2}y_3$ passes 0 to the input to the flip-flop. At the next clock pulse, 0 is gated into the flip-flop and value 110000 is presented to the pass network from the destination code set to calculate Y_2. With present state B, value 1 is passed to the input of the flip-flop. At the next clock pulse, 1 is input to the flip-flop and the previous flip-flop value is gated to the shift register. Value 010101 is passed to the next state logic to calculate Y_3, passing 1 to the flip-flop. At the next clock pulse, 1 is input to the shift register which now holds value 10. The flip-flop assumes a value 1 for Y_3. At the next clock pulse, 1 from the flip-flop is gated to the shift register which now holds value 011, the next state of the machine. This value is gated to the state latch signifying that the next state is D. The process repeats with a new input state.

4 Adaptable Circuit

The architecture described above can implement any sequential circuit with a maximum of m internal and p input states. Therefore, this architecture could be termed universal.

In addition to being universal, this architecture allows for adaptable operation. If the destination code sets are changed, the circuit will assume a different set of destination states. Simply changing these sets change the operation of the circuit. In the above example, if the destination code set for I_1 had column 000100 for y_1 instead of 001100, the next state for C under I_1 would be A (000) instead of E (100).

Therefore, one circuit configuration can be adapted to implement different flow tables. Moreover, it is possible to modify the flow table dynamically which opens the possibility making changes to provide some level of fault tolerance.

5 Conclusion

A bit serial synchronous sequential circuit has been presented that enables all state variables to be generated with single next state logic circuitry. Circuit operation can be changed to implement different flow tables by simply changing input constants that are fed to the next state equation circuitry.

References

[1] S. Whitaker and G. Maki, "Pass-Transistor Asynchronous Sequential Circuits," IEEE Journal of Solid State Circuits, pp. 71-78, Feb. 1989.

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

[3] 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, pp. 139-144, Oct. 1984.

Footnotes

This research was supported in part by NASA under the NASA Space Engineering Research Center grant NAGW-1406 and by the Idaho State Board of Education under grant 88-038.

Sequence Invariant State Machines

S. Whitaker and S. Manjunath
NASA Engineering Research Center
for VLSI System Design
University of Idaho
Moscow, Idaho 83843

Abstract - **A synthesis method and new VLSI architecture are introduced to realize sequential circuits that have the ability to implement any state machine having N states and m inputs, regardless of the actual sequence specified in the flow table. A design method is proposed that utilizes BTS logic to implement regular and dense circuits. A given state sequence can be programmed with power supply connections or dynamically reallocated if stored in a register. Arbitrary flow table sequences can be modified or programmed to dynamically alter the function of the machine. This allows VLSI controllers to be designed with the programmability of a general purpose processor but with the compact size and performance of dedicated logic.**

1 Introduction

Most digital systems include a controller. This is usually either a general machine such as a microprocessor, or a dedicated, custom designed sequential state machine. The trade-off between these two respective approaches is breadth of applicability and ease of reconfiguration versus cost, performance and complexity. This work describes an architecture that retains the traditional strengths of dedicated state machines, but offers the adaptability of a microcontroller. This paper describes the concept and structure of this new architecture which produces sequence invariant machines. The controllers on the full custom data compression chips for NASA are being implemented using this new architecture [1].

The advantage of a hardware realization of a sequence invariant sequential machine is that it can implement any flow table without a change in the logic equations. This type of circuit is also known in the literature as a universal state machine [2]. The only parameters needed to realize this circuit are the maximum number of input states m and the maximum number of internal states 2^n. The hardware realization of a given circuit for m and n can implement any circuit with a maximum of m input states and 2^n internal states. This capability to transition through all possible circuit sequences requires a hardware realization that yields design equations that can be adaptable. A new design procedure for asynchronous sequential circuits [3] has produced identical design equations and this procedure is modified in this paper for synchronous machines.

A sequential circuit consists of a combinational logic excitation network which implements the state variable equations and storage elements, which in this case are D flip-flops. A sequential circuit is often defined in terms of a flow table like the example used in a

	I_1	I_2	I_3
A	C	B	A
B	D	C	B
C	E	D	C
D	F	E	D
E	A	F	E
F	B	A	F

Table 1: Example flow table.

later section shown in Table 1. The input states are noted as I_p, internal states are S_j, next states as N_j, present state variables as y_i and next state variables as Y_i.

Pass transistor logic design is known for producing circuits that have high density and speed [4]. When these circuits are realized in as Binary Tree Structure (BTS) networks, they also have unique properties which often produce minimal transistor circuits and possess fault detecting properties [5]. Since these properties are attractive, BTS networks are used in the realization of the design equations in this paper.

Section 2 of this paper introduces the combinational logic realized as BTS networks and specifies a general BTS structure that is used in the design equations. Section 3 presents the design procedure using the general BTS structure along with an example. The example is then altered demonstrating the adaptive nature of this architecture.

2 Binary Tree Structured Logic

Pass transistor logic has the significant advantages over gate logic of high speed and density [4,5]. A pass transistor network realized in a BTS form often requires fewer transistors and displays attractive fault detection characteristics [5]. A unique form of a BTS network, called a general BTS network, is used here to formulate the next state equations for the sequential circuit.

In general a BTS circuit is characterized by having a maximum of $2^n - 1$ nodes and each node has exactly two branches. One branch is controlled by variable X_i and the other by $\overline{X_i}$. The maximum number of transistors in a BTS network is $2^{n+1} - 2$, and therefore only constants 0 and 1 are input to the network. A general BTS network contains the maximum number of transistors and represents a complete decoding of an input space. Figure 1 shows a general BTS network which implements all three-variable functions. In this general network, all three-variable functions can be realized by simply changing the pass variable constants 1(0) at the input of the appropriate branch. A general BTS network is a fully decoded binary tree and is used in the design which follows.

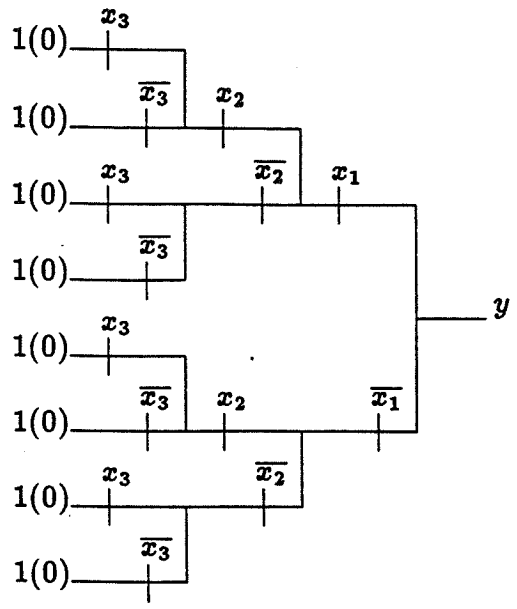


Figure 1: General three variable BTS network.

3 State Machine Design

The logic that forms each next state equation Y_i consists of the following elements: a storage device (normally a flip-flop), next state excitation circuitry which generates the next state values to the flip-flop, and input logic. Present state information is fed back by the state variables y_i to the excitation logic. The excitation logic is a combinational logic function of the input and the state variable information. In general, the information needed to generate all possible next state values that can be assumed by the circuit at the next clock pulse is resident within the excitation circuitry. The specific next state value is selected by the current input and state variables.

A sequence invariant sequential machine can implement any flow table without a change in hardware. In this paper, a hardware circuit can realize any flow table which requires equal to or less than m input states and/or 2^n internal states. In other words, only m and n are needed to specify the hardware necessary to realize all possible state transitions in a sequential circuit.

In order for the next state circuitry to implement sequence invariant operation, it must assume a unique form to allow a circuit to implement an arbitrary state transition. First the circuitry for each next state variable must be identical. Second, specific next state information must not be hardwired into the logic that forms the next state equations. Rather, specific next state values must be presented from an external source.

Whitaker has derived a unique realization for asynchronous sequential circuits where all of the next state equations have identical circuitry[3]. This concept can be extended to synchronous sequential circuits by replacing transition paths of the asynchronous circuits

with predecessor states. In the asynchronous case, a transition path defines the set of states that must have the same next state entry and a pass implicant covers each transition path resulting in a unique network of pass transistors in the next state equations. In the synchronous case, predecessor states also define the states that have the same next state entry and a pass implicant covers the predecessor states and is realized as unique network of pass transistors in the next state equations. This notion of using predecessor states is described next.

The predecessor states for state S_i under an input I_p are states which have S_i as a next state entry. In Table 1, the predecessor state for C under I_1 is A. Whenever the circuit is in a predecessor state of S_i, the next clock pulse will effect a transition to S_i. Moreover, the next state entry is present at the input of the flip-flops prior to the clock pulse. At the next clock pulse, the flip-flops will assume the value of the code associated with S_i.

Conceptionally, the new architecture operates as follows: For each predecessor state of state S_i, there exists a pass transistor network in the excitation network that presents the next state value, S_i, to the flip-flops. The pass transistor network consists of a single pass implicant that covers each predecessor state such that when any state is entered, a unique pass transistor path is enabled that passes the proper next state value to the flip-flops. For example, in Table 1, there is a pass transistor path that presents the code for state C to the flip flops when $I_1 = 1$ and the circuit is in state A.

All pass transistor implementations for the synchronous sequential circuits described in this paper operate as described above, including those that use PLA pass transistor networks to generate the next state equations. The circuit becomes invariant when all state equations are identical. All equations are identical when they are realized with general BTS networks that completely decode all the internal states. That is, if there are n state variables, then the BTS network must decode all 2^n states. The value for the next state entries for each predecessor state for S_i is the code for S_i and the constants for this code are input to the BTS network.

A specific implementation of an architecture to realize sequential circuits is shown in Figure 2, where the next state decoder is a general BTS network. The destination state codes represent all of the possible destination states for the sequential circuit. Since every state of the flow table appears as a next state someplace in the flow table, all of the specified states appear in the destination code set. These codes are input to the input switch matrix which generates all of the destination codes for a given input I_p as specified by I_p. The next state decoder selects the unique destination code as specified by the current state variables y.

The following illustrates specifically how this architecture works. In general, let the circuit be in state S_k with input I_j. Let S_k be the predecessor for S_i. All destination states codes are normally input to the input switch matrix. Input I_j selects the codes for the destination states under I_j to be presented to the next state decoder. The feedback that specifies that the present state is S_k enables one and only one path in the BTS network to select the correct destination code, S_i, for S_k under I_j and presents this value to the flip-flops. This is depicted in Figure 3.

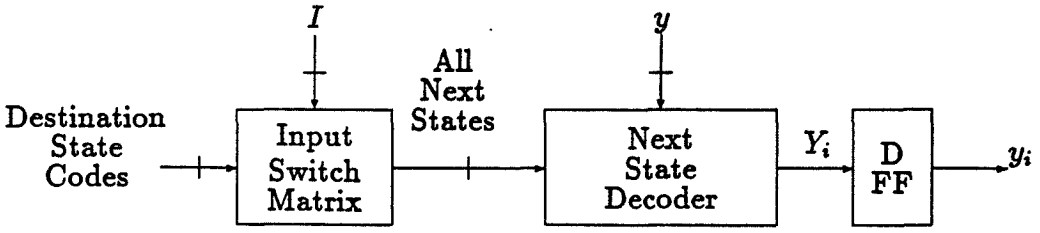


Figure 2: General block diagram.

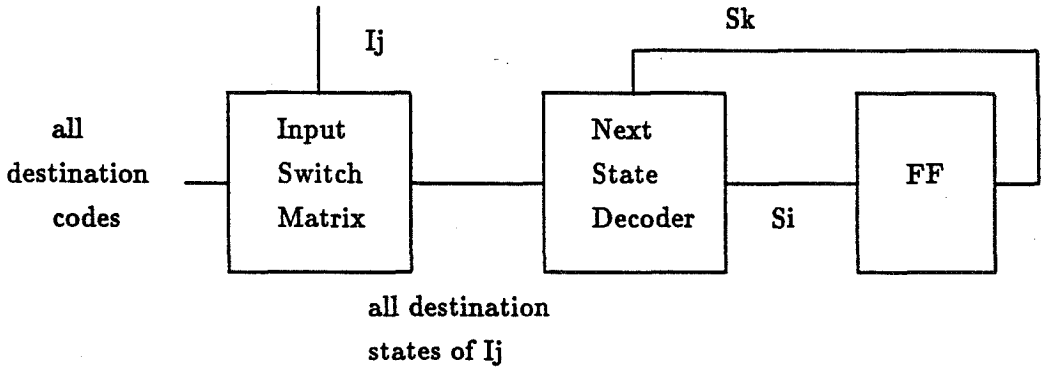


Figure 3: Sequential circuit operation.

	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}

Table 2: General six-state three-input flow table.

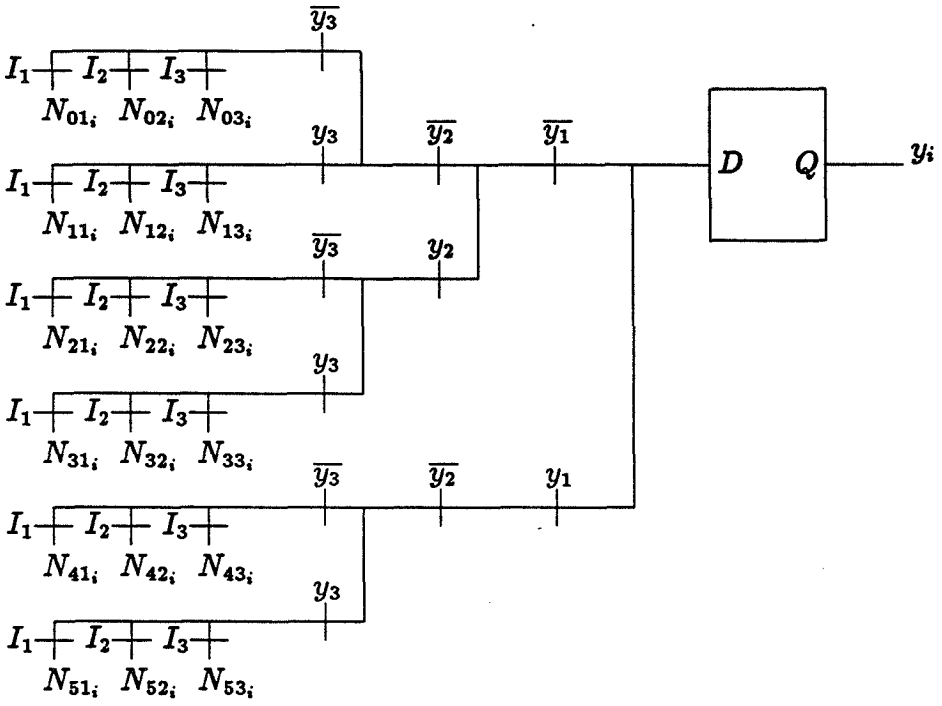


Figure 4: General six-state three-input next state equation bit.

Let the general six-state and three-input flow table as shown in Table 2 depict an example for a general m state machine. I_1, I_2 and I_3 are the inputs, $S_0 \ldots S_5$ are the present states of the sequential circuit and $N_{S_0 I_1}, N_{S_0 I_2} \ldots N_{S_5 I_3}$ are the next states. This can be generalized so that $N_{S_i I_j}$ are the next states for S_i under input I_j. $N_{S_i I_j}$ has been abbreviated as N_{ij}. Let the state assignment be $S_0 = 000, S_1 = 001, S_2 = 010, \ldots, S_5 = 101$. The next state decoder is a general BTS circuit with paths that decode each state. The input switch matrix is a pass transistor matrix, that passes the destination state codes to the next state pass network as shown in Figure 4.

The circuit realization of the next state pass network depicted in Figure 4 operates in the following manner: All of the destination state codes N_{ij} are presented to the input switch matrix. For each input state I_i, all of the destination states in I_i are presented to the next state decoder. The present state variables, y, select one and only one next state

y_1	y_2	y_3		I_1	I_2	I_3
0	0	0	A	010	001	000
0	0	1	B	011	010	001
0	1	0	C	100	011	010
0	1	1	D	101	100	011
1	0	0	E	000	101	100
1	0	1	F	001	000	101

Table 3: Example flow table with next state entries.

entry. This single next state entry passed to the flip-flip is determined by the present state of the circuit. If the machine is in state S_1 and input I_2 is asserted, then N_{12_i} would be passed to the input of the flip-flop for next state variable Y_i.

In summary, the input state 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 at the next clock pulse.

If the machine is to be adaptive, the circuitry must also be able to implement any flow table and therefore must be independent of the sequences in a flow table. This architecture can implement different flow tables by changing only the destination state information driving the input switch matrix. Hence, the pass transistor hardware remains constant. Changing constants can be implemented by programming the supply connections to provide the 1's and 0's or through using enabling transistors that present data from a register array. With a register array, the destination state information could be changed by a host controller to allow dynamic reconfiguration.

Design Procedure The first step in the design process is to generate a state assignment. Any synchronous state assignment could be used, as long as no two distinct states have the same assignment. For ease of illustration, consider the flow table shown in Table 1 with the state assignment and next state entries shown in Table 3.

Let ρ_S define a state partition that partitions state S from all other states of the flow table. If states $S_0, S_1, \ldots S_j$ are all the destination states for a given input, then state partitions $\rho_{S_0}, \rho_{S_1}, \ldots \rho_{S_j}$ must be covered in the design equations. For this flow table, the partitions are

$$\begin{aligned} \rho_a &= A; BCDEF \\ \rho_b &= B; ACDEF \\ \rho_c &= C; ABDEF \\ \rho_d &= D; ABCEF \\ \rho_e &= E; ABCDF \\ \rho_f &= F; ABCDE \end{aligned} \tag{1}$$

The general design equation for each of the next state variables consists of a term for each ρ partition and is as follows:

$$\begin{aligned} Y_i = \; & \rho_a I_1() + \rho_b I_1() + \rho_c I_1() + \rho_d I_1() + \rho_e I_1() + \rho_f I_1() + \\ & \rho_a I_2() + \rho_b I_2() + \rho_c I_2() + \rho_d I_2() + \rho_e I_2() + \rho_f I_2() + \\ & \rho_a I_3() + \rho_b I_3() + \rho_c I_3() + \rho_d I_3() + \rho_e I_3() + \rho_f I_3() \end{aligned} \tag{2}$$

By rearranging the partitions, the general next state design equation can be written as follows:

$$Y_i = \sum_{k=1}^{m} C(\rho_{S_k})[\sum_{l=1}^{p} I_l(x_{S_k i})] \tag{3}$$

where $C(\rho_{S_k})$ is the product expression covering state S_k, m is the number of states, p is the number of inputs and $x_{S_k i}$ is the ith bit of the code for the destination state of S_k for y_i under input I_l. For this example,

$$\begin{aligned} Y_i = \; & \rho_a[I_1() + I_2() + I_3()] + \rho_b[I_1() + I_2() + I_3()] + \\ & \rho_c[I_1() + I_2() + I_3()] + \rho_d[I_1() + I_2() + I_3()] + \\ & \rho_e[I_1() + I_2() + I_3()] + \rho_f[I_1() + I_2() + I_3()] \end{aligned} \tag{4}$$

The next step is to enter $\rho_j I_k(1)$ for the destination state under ρ_j that has $Y_i = 1$ and to enter $\rho_j I_k(0)$ for the destination state under ρ_j that has $Y_i = 0$. For example, the next state under the input I_1 for present state A or partition ρ_a is C. C has the code $y_1 = 0$, $y_2 = 1$ and $y_3 = 0$ and this destination state can be entered into the design equations as shown:

$$\begin{aligned} Y_1 = \; & \rho_a[I_1(0) + I_2() + I_3()] + \rho_b[I_1() + I_2() + I_3()] + \\ & \rho_c[I_1() + I_2() + I_3()] + \rho_d[I_1() + I_2() + I_3()] + \\ & \rho_e[I_1() + I_2() + I_3()] + \rho_f[I_1() + I_2() + I_3()] \end{aligned}$$

$$\begin{aligned} Y_2 = \; & \rho_a[I_1(1) + I_2() + I_3()] + \rho_b[I_1() + I_2() + I_3()] + \\ & \rho_c[I_1() + I_2() + I_3()] + \rho_d[I_1() + I_2() + I_3()] + \\ & \rho_e[I_1() + I_2() + I_3()] + \rho_f[I_1() + I_2() + I_3()] + \end{aligned} \tag{5}$$

$$\begin{aligned} Y_3 = \; & \rho_a[I_1(0) + I_2() + I_3()] + \rho_b[I_1() + I_2() + I_3()] + \\ & \rho_c[I_1() + I_2() + I_3()] + \rho_d[I_1() + I_2() + I_3()] + \\ & \rho_e[I_1() + I_2() + I_3()] + \rho_f[I_1() + I_2() + I_3()] \end{aligned}$$

By inspection of the flow table, the destination state codes for the remaining ρ partitions are incorporated in the design equations.

$$
\begin{aligned}
Y_1 = \; & \rho_a[I_1(0) + I_2(0) + I_3(0)] + \rho_b[I_1(0) + I_2(0) + I_3(0)] + \\
& \rho_c[I_1(1) + I_2(0) + I_3(0)] + \rho_d[I_1(1) + I_2(1) + I_3(0)] + \\
& \rho_e[I_1(0) + I_2(1) + I_3(1)] + \rho_f[I_1(0) + I_2(0) + I_3(1)] \\
\\
Y_2 = \; & \rho_a[I_1(1) + I_2(0) + I_3(0)] + \rho_b[I_1(1) + I_2(1) + I_3(0)] + \\
& \rho_c[I_1(0) + I_2(1) + I_3(1)] + \rho_d[I_1(0) + I_2(0) + I_3(1)] + \\
& \rho_e[I_1(0) + I_2(0) + I_3(0)] + \rho_f[I_1(0) + I_2(0) + I_3(0)] \\
\\
Y_3 = \; & \rho_a[I_1(0) + I_2(1) + I_3(0)] + \rho_b[I_1(1) + I_2(0) + I_3(1)] + \\
& \rho_c[I_1(1) + I_2(1) + I_3(0)] + \rho_d[I_1(0) + I_2(1) + I_3(1)] + \\
& \rho_e[I_1(0) + I_2(0) + I_3(1)] + \rho_f[I_1(1) + I_2(0) + I_3(0)]
\end{aligned}
\tag{6}
$$

The final step is to find the covering equations for each ρ partition. Let $C(\rho_s)$ define the product term that partitions state S from all other states. For example, $C(\rho_a)$ is $\overline{y_1}\,\overline{y_2}\,\overline{y_3}$. With a unique code for each state, there is a unique product term for each state partition.

$$
\begin{aligned}
C(\rho_a) &= \overline{y_1}\,\overline{y_2}\,\overline{y_3} & C(\rho_d) &= \overline{y_1}\,y_2\,y_3 \\
C(\rho_b) &= \overline{y_1}\,\overline{y_2}\,y_3 & C(\rho_e) &= y_1\,\overline{y_2}\,\overline{y_3} \\
C(\rho_c) &= \overline{y_1}\,y_2\,\overline{y_3} & C(\rho_f) &= y_1\,\overline{y_2}\,y_3
\end{aligned}
\tag{7}
$$

The circuit in Figure 4 shows the logic for implementing each state variable y_i. Each $C(\rho_s)$ is a path through the BTS network forming the next state decoder. The logic of Figure 4 is replicated three times and the inputs are driven by the destination state information shown in Table 3. Figure 5 shows the programming of the input switch matrix for state variable y_3. Except for the constant input values that are driving the input switch matrix, the design equations and pass transistor realizations for each state variable are identical.

A major result of this work is that it is no longer necessary to derive the pass logic configurations for each next state equation. The next state information is only used as the input pattern to the input switch matrix. The actual realization or hardware is therefore independent of the next state information. This allows for an adaptive circuit realization for the state machine. Since the next state information is stored in the input switch matrix, only the programming of the destination state codes need be changed to implement a different flow table.

To illustrate the ease of adapting a flow table to a different transition sequence, consider the flow table in Table 4. This flow table differs from Table 1 in columns I_2 and I_3. The new next state information is shown in Table 5 and the circuitry is shown in Figure 6. Note that the next state circuitry is identical to that of Figure 5 and only the destination state codes are changed at the input of the input switch matrix. Figure 6 shows next state variable Y_3 reprogrammed with the next state information from the modified example flow table.

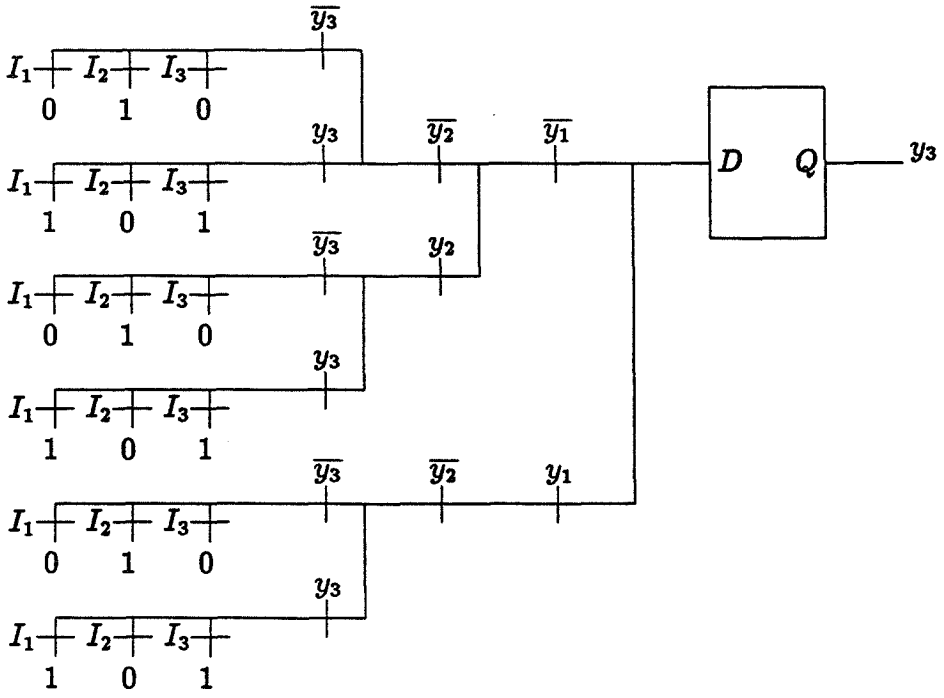


Figure 5: Example next state equation for Y_3.

	I_1	I_2	I_3
A	D	C	B
B	E	C	C
C	F	D	D
D	A	E	D
E	B	F	E
F	C	A	E

Table 4: Modified example flow table.

y_1	y_2	y_3		I_1	I_2	I_3
0	0	0	A	011	010	001
0	0	1	B	100	010	010
0	1	0	C	101	011	011
0	1	1	D	000	100	011
1	0	0	E	001	101	100
1	0	1	F	010	000	100

Table 5: Modified example flow table with next state entries.

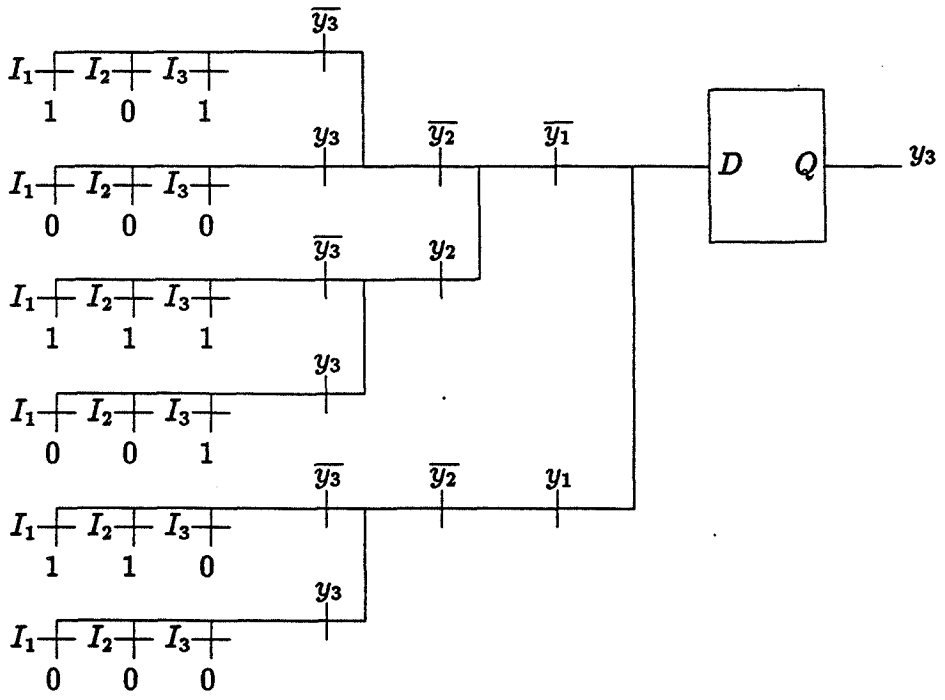


Figure 6: Modified example next state equation for Y_3.

4 Conclusion

This research introduces a new VLSI architecture which realizes sequential circuits. This general architecture provides a hardware realization which is independent of any state transition that may be specified in a flow table. The architecture promotes an easy and straight forward way to design synchronous machines and this design can be accomplished by inspection. Traditional design steps are not needed such as state table generation and design equation realizations from K-maps.

It is shown that a given hardware implementation can realize any m row flow table by simply changing a set of input constants. The design equations have identical hardware resulting in easy VLSI replication. From a flow table, the destination state codes can be programmed directly into the input switch matrix. This allows designers to produce compact, programmable VLSI controllers. The state machine architecture is also unique because it requires a minimal amount of extra hardware to introduce adaptability into the state machine. If the next state information is stored in a register that drives the input matrix, any sequence can be changed dynamically to alter the function of the circuit by rewriting the register. This produces an adaptable sequential state machine that can be modified at will.

References

[1] J. Venbrux, and N. Liu, "VLSI Architectures for Data Compression using the Rice Algorithm", accepted for publication at NASA Space Engineering Research Center VLSI Symposium, January 1990.

[2] S. H. Unger, *Asynchronous Sequential Switching Circuits*, Robert E. Krieger Publishing Company, Inc., 1983

[3] S. Whitaker and G. Maki, "Pass-Transistor Asynchronous Sequential Circuits", IEEE Journal of Solid State Circuits, pp. 71-78, Feb., 1989

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

[5] 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, pp. 139-144, Oct., 1984

[6] T. Martinez and J. Vidal, "Adaptive Parallel Logic Networks", Journal of Parallel and Distributed Computing, pp. 26-58, Feb. 1988

[7] I. Aleksander and E. Mamdani, "Adaptive Logic Elements in Pulse Control Systems", Proceedings of the Symposium on Pulse-Rate and Pulse-Number Signals in Automatic Control, pp. 486-493, Apr. 1968

Footnotes

This research was sponsored in part by NASA under the NASA Space Engineering Research Center Grant NAGW-1406 and by the Idaho State Board of Education under research grant 88-038.

Pass Transistor Implementations of Multivalued Logic

G. Maki and S. Whitaker
NASA Engineering Research Center
for VLSI System Design
University of Idaho
Moscow, Idaho 83843

Abstract - **A simple straight-forward Karnaugh map logic design procedure for realization of multiple-valued logic circuits is presented in this paper. Pass transistor logic gates are used to realize multiple-valued networks. This work is an extension of pass transistor implementations for binary-valued logic.**

1 Introduction

Multiple-valued logic has been a research topic for the past two decades. Two basic motivations for multivalued logic functions are to increase the amount of information conveyed on each interconnect line and to decrease the amount of area required to build logical circuits implemented in VLSI technology [1]. Hurst in his technology status report concluded that practical application of the research was dependent on circuit realizations and that CMOS transmission gates should be exploited as a potential circuit realization [2].

Formal techniques for the realization of Boolean logic functions with pass transistors have been introduced [3] and it will be shown that they can be applied to the design of multiple-valued logic functions. In Section 2, the properties of CMOS pass gates which allow the implementation of multiple-valued logic are presented. Formal logic design techniques are introduced in Section 3 for the realization of pass transistor circuits passing multiple logic levels. Section 4, extends the logic design theory to cover the passing of multivalued variables as well as constants.

2 Properties of Pass Transistors

In the design of combinational pass transistor logic for Boolean functions, p-channel MOSFET's are normally used to pass logic 1's while n-channel MOSFET's are used to pass logic 0's. Both the PMOS and NMOS transistors are used to form a transmission gate when the input is a variable which could assume either Boolean value.

The MOSFET is a voltage controlled current source device [4]. When used as a pass gate, however, the MOSFET can be described as a voltage follower for a limited input voltage range. For a pass gate, the drain and source are input and output terminals and the gate is a control terminal as shown in Figure 1. The equation for the output voltage

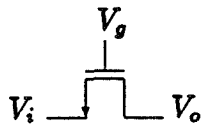


Figure 1: NMOS pass transistor

of an n-channel MOSFET when $V_g = V_{dd}$ is given by

$$V_o = \begin{cases} V_i & for \quad V_i < V_{dd} - V'_{tn} \\ V_{dd} - V'_{tn} & for \quad V_i > V_{dd} - V'_{tn} \end{cases} \tag{1}$$

where the threshold $V'_{tn} = V_{tn0} + \gamma_n(\sqrt{V_{sb} + 2\phi_f} - \sqrt{2\phi_f})$ and V_{dd} is the most positive voltage on the chip.

The equation for the PMOS FET output voltage when $V_g = V_{ss}$ is given by

$$V_o = \begin{cases} V_i & for \quad V_i > |V'_{tp}| \\ |V'_{tp}| & for \quad V_i < |V'_{tp}| \end{cases} \tag{2}$$

where the threshold $V'_{tp} = V_{tp0} - \gamma_p(\sqrt{|V_{sb}| + 2\phi_f} - \sqrt{2\phi_f})$ and V_{ss} is the reference voltage for the chip.

Multivalued logic signals could therefore be passed by networks of PMOS and NMOS transistors. From the DC operating equations given above, PMOS FET's can be used to pass logic level V_x when $V_x > |V'_{tp}|$, while NMOS FET's can be used to pass logic level V_x when $V_x < V_{dd} - V'_{tn}$. The operating voltage ranges for the inputs based on the DC equations therefore overlap between $|V'_{tp}| < V_x < V_{dd} - V'_{tn}$.

The switching speed of FET's used as pass gates is proportional to $V_{gs} - |V'_t|$ [4]. Good switching characteristics could be achieved if the input range were divided such that NMOS FET's were used when $V_i \leq V_{dd}/2$ and PMOS FET's were used when $V_i > V_{dd}/2$. The pass transistor control gates would still be driven by Boolean functions of the highest logic value ($1 = V_{dd}$) and the lowest logic value ($0 = V_{ss}$).

A general multivalued pass network can be depicted as shown in Figure 2. Each P_i is a series of NMOS transistors for $V_i \leq V_{dd}/2$ or a series of PMOS transistors for $V_i > V_{dd}/2$ such that when each transistor is enabled by a Boolean logic 1 (NMOS) or 0 (PMOS) the voltage on the input, V_i, passes to the output, V_o. P_i can be represented as a sum of literals where each literal represents the Boolean variable driving the gate of a series pass transistor. The output voltage V_o can be expressed as a logic (voltage) level being passed to the output,

$$V_o = P_1(V_1) + P_2(V_2) + \cdots + P_n(V_n) \tag{3}$$

Each transistor in P_i can be schematically represented as shown in Figure 3.

The pass transistor network is to be distinguished from the multiple-valued T-gate [5]. The general T-gate has $r + 1$ inputs with one of the inputs being a r-valued control input whose value determines which of the other r (r-valued) inputs is selected for output. The pass transistor itself has one r-valued input and a binary control input that switches the r-valued input to the output of the transistor. A pass transistor network has n r-valued inputs that are switched by m binary variables.

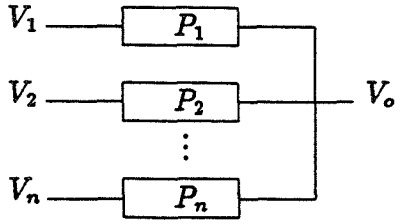


Figure 2: General pass network.

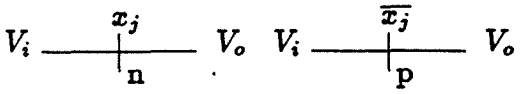


Figure 3: Pass transistor logical representation.

3 Logic Design

3.1 Formal Statement of Problem

The inputs to a general logic circuit are the multivalue logic voltage levels, constants in the set $[v_1, v_2, \ldots v_n]$. These logic levels will be presented to the output by means of a switching network consisting of pass transistors. Each pass transistor is controlled by binary variables $x_1, x_2, \ldots x_m$.

Example 1 *Consider the circuit illustrated in Figure 4.*

The inputs are multivalue logic levels [0,2,1] where $0 = V_{ss}$, $1 = V_{dd}$ and $2 = V_{dd}/2$. The control variables are x_1, x_2 and x_3. A few examples help illustrate the circuit operation. If x_1 and x_3 are both low then the output is logic level 0; if $x_1x_2x_3 = 110$ then the output is logic level 2. The Karnaugh map for the above circuit is shown in Figure 5. Every combination of inputs produces one and only one output value.

3.2 Definitions

The following definitions extend the formalism of pass transistor logic already presented in the literature [3].

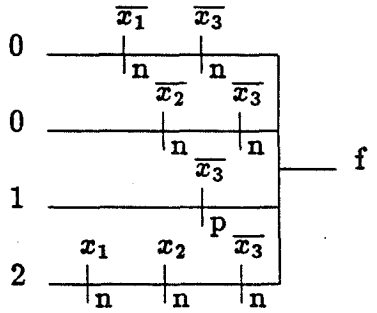


Figure 4: Pass Network Example 1

x_3 \ x_1x_2	00	01	11	10
0	0	0	2	0
1	1	1	1	1

Figure 5: Karnaugh map representation.

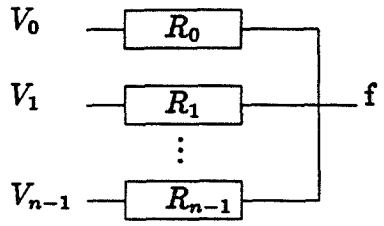


Figure 6: General network.

Definition 1 *A multivalue pass implicant (MPI), $P_i(V_i)$, consists of a product expression P_i and a pass variable V_i.*

When the literals of P_i evaluate to 1, the multivalue logic level V_i is presented to the output of the pass transistor network realizing $P_i(V_i)$; otherwise the output is in the high impedance state.

For Example 1, the multivalue pass implicants are $\overline{x_1}\,\overline{x_3}(0)$, $\overline{x_2}\,\overline{x_3}(0)$, $x_3(1)$, and $x_1x_2\overline{x_3}(2)$. It is clear that a pass implicant can pass only one multivalue logic level. The general model of a network realization is shown in Figure 6. Each R_i represents a set of MPI's that pass logic level V_i. In Example 1, $R_0 = \overline{x_1}\,\overline{x_3}(0) + \overline{x_2}\,\overline{x_3}(0)$, $R_1 = x_3(1)$ and $R_2 = x_1x_2\overline{x_3}(2)$.

Definition 2 *A multivalue MSP expression is sum of multivalue pass implicants that realize a given function.*

The multivalue MSP expression for the above circuit is

$$f = \overline{x_1}\,\overline{x_3}(0) + \overline{x_2}\,\overline{x_3}(0) + x_3(1) + x_1\,x_2\,\overline{x_3}(2) \tag{4}$$

3.3 Design Algorithm

Procedure 1 *The logic design rules for implementing a multivalue pass network can be incorporated into a design procedure.*

1. Place the multivalue logic states on a Karnaugh map.

x_3x_4 \ x_1x_2	00	01	11	10
00	1	1	1	2
01	1	3	3	2
11	2	3	3	0
10	2	1	1	0

Figure 7: Design Example

Pass Variable	MPI
0	$x_1\,\overline{x_2}\,x_3(0)$
1	$\overline{x_1}\,\overline{x_2}\,\overline{x_3}(1) + x_2\,\overline{x_4}(1)$
2	$x_1\,\overline{x_2}\,\overline{x_3}(2) + \overline{x_1}\,\overline{x_2}\,x_3(2)$
3	$x_2\,x_4(3)$

Table 1: Design example MPI.

2. Identify each set of multivalue entries that meet the conditions of a prime implicant; these are the pass implicants.
3. Find an optimal covering of the circuit and form the multivalue MSP.

Example 2 *Consider the example with the Karnaugh map in Figure 7.*

The MPI's for this circuit are shown in Table 1. The function then is

$$\begin{aligned} f = \; & x_1\,\overline{x_2}\,x_3(0) + \overline{x_1}\,\overline{x_2}\,\overline{x_3}(1) + x_2\,\overline{x_4}(1) + \\ & x_1\,\overline{x_2}\,\overline{x_3}(2) + \overline{x_1}\,\overline{x_2}\,x_3(2) + x_2\,x_4(3) \end{aligned} \tag{5}$$

4 Multivalue Switching Functions

A generalization of the problem addressed above is to allow the multivalue logic inputs to the pass network to be variables instead of constants. The formal definition of the problem becomes: the control variables are binary signals $[x_1, x_2, \ldots, x_n]$, $x_i \in [0,1]$, and the pass variables are multivalue variables $[F_1, F_2, \ldots F_m]$, $F_i \in [V_0, V_1, \ldots V_k]$, V_j is a multivalue logic level.

The only change in the design process presented in the previous section is to replace V_i with F_i in the algorithm and to implement the pass network with transmission gates rather than single transistors. Since pass transistor networks using transmission gate structures, that is, both PMOS and NMOS pass transistor arrays, can pass multivalue logic levels, these signals can be variables and not just constants. Definition 1 is modified below while Definition 2 remains unchanged.

Definition 3 *A multivalue pass implicant (MPI), $P_i(F_i)$, consists of a product expression P_i and a pass variable F_i.*

x_3 \ x_1x_2	00	01	11	10
0	F_0	F_0	F_2	F_1
1	F_0	F_0	F_2	F_1

Figure 8: Multiple function switching function.

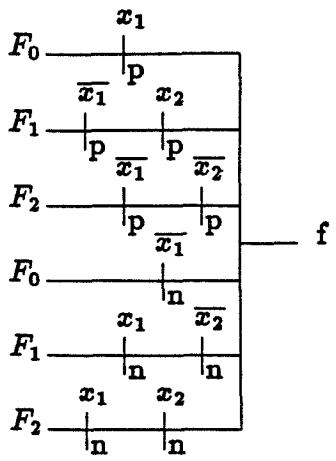


Figure 9: Multivalue switching network.

When the literals of P_i evaluate to 1, the multivalue logic variable F_i is presented to the output of the pass transistor network realizing $P_i(F_i)$; otherwise the output is in the high impedance state.

Example 3 *Consider the example shown in Figure 8 to illustrate definition 9.*

The inputs are multivalue logic variables $[F_0, F_1, F_2]$ and the control variables are x_1, x_2 and x_3. Identifying the MPI's is accomplished in an identical manner as before. Here the MPI's are $\overline{x_1}(F_0), x_1\overline{x_2}(F_1)$ and $x_1x_2(F_2)$.

The pass logic expression is

$$f = \overline{x_1}(F_0) + x_1\overline{x_2}(F_1) + x_1x_2(F_2) \tag{6}$$

The pass logic circuit shown in Figure 9 implements the function with both an NMOS array and a PMOS array.

References

[1] J. Muzio and I. Rosenberg, "Introduction - Multiple-Valued Logic", IEEE Transactions on Computers, Vol. C-35, No. 2, Feb., 1986, pp. 97-98

[2] S. Hurst, "Multiple-Valued Logic – Its Status and Its Future", IEEE Transactions on Computers, Vol. C-33, No. 12, Dec., 1984, pp. 1160-1179

[3] 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

[4] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*, Reading, Mass., Addison-Wesley, 1985

[5] K. C. Smith, "Multiple-Valued Logic: A Tutorial and Appreciation," IEEE Computer, April, 1988, pp. 17-27

This work was supported in part by NASA under Contract NAGW-1406.

Statistical Circuit Design For Yield Improvement In CMOS Circuits

H.J. Kamath, J.E. Purviance, and S.R. Whitaker
Department of Electrical Engineering
University of Idaho,
Department of Electrical Engineering
Moscow, Idaho 83843

Abstract- **This paper addresses the statistical design of CMOS integrated circuits for improved parametric yield. The work uses the Monte Carlo technique of circuit simulation to obtain an unbiased estimation of the yield. A simple graphical analysis tool, the yield factor histogram , is presented. The yield factor histograms are generated by a new computer program called SPICENTER. Using the yield factor histograms, the most sensitive circuit parameters are noted, and their nominal values changed to improve the yield. Two basic CMOS example circuits, one analog and one digital, are chosen and their designs are 'centered' to illustrate the use of the yield factor histograms for statistical circuit design.**

1 Introduction

Manufacturability is a key word in industry today. Due to inherent fluctuations in the thermal, chemical, and optical processes used in the fabrication of any integrated circuit(IC), the yield (which is defined as the ratio of the number of IC's that perform correctly to the total number of IC's manufactured) is usually less than 100 percent. As the very large scale integrated (VLSI) circuits become more complex, and the dimensions of VLSI devices decrease, the circuit performance becomes more sensitive to fluctuations in the manufacturing process. Since the profitability of a manufacturing process is directly related to yield, the need for computer aided methods for yield improvement has risen sharply in recent years.

Generally a CMOS designer takes the manufacturing variation into account by doing a worst-case design. A worst-case design assures that the circuit will meet specifications when all the process parameters are simultaneously their worst possible values. Three problems with worst-case design are apparent:

1. Choosing the worst-case design parameters can be difficult.

2. There is a significant performance tradeoff when a circuit is required to meet specifications at the worst-case parameter values.

3. It is usually very improbable that a circuit will simultaneously encounter all the worst-case parameters during manufacture.

The price that is paid for a worst-case design is a significant performance degradation. For instance, it may be possible to double the clock speed specification of a certain digital IC if the goal for the circuit is an 80% yield, rather than the 100% yield imposed by a worst-case design. It is the premise of this paper that the worst-case design is very conservative and leads to circuits which are under specified. The solution to this problem is to use statistical design methods.

The intent of this paper is to apply the statistical design centering techniques developed for microwave circuits in [15,17] to the design of CMOS integrated circuits. Two example circuits are chosen and design centering is applied to these circuits using the new techniques. A CMOS operational amplifier (op amp) and a chain of five CMOS inverters were chosen as example circuits since they are the basic building blocks of digital and analog VLSI circuits.

2 Design Centering And Yield

During the manufacture of a circuit, the actual component values used in the circuit are not the nominal values determined by the designer, but are values that are near the nominal values. The range of values encountered during manufacture is given by the component tolerance, usually expressed as a percentage of the nominal values. A circuit design that is manufacturable (i.e., it achieves a high yield) must not only perform well at the nominal value of the components but also perform well over the entire tolerance range of the component values. Therefore, if the design goal is high yield it is important to include in the design process the circuit performance over the entire range of component values encountered during manufacture.

The design criterion we wish to maximize in this work is manufacturing yield – the fraction of circuits which meets the performance criteria when the circuit is manufactured. The component value statistics are assumed known. For our example circuits, the performance criteria are rise time (which is a measure of bandwidth) for the operational amplifier and delay time (which is the measure of its speed) for the inverter chain. Circuit yield in general can not be calculated exactly. Therefore, we must be satisfied with an estimate of the yield and a knowledge of its statistics.

Consider a circuit performance function, $G(X)$, which depends on a set of parameter values $X = (x1, x2, ..., xn)$. A strength of this study is that no assumptions are needed as to the form of $G(X)$. The acceptance of a circuit with parameters X can be expressed as follows;

$$G(X) \in \Phi \tag{1}$$

Φ is the region of G in which the circuit meets all its performance specifications. Therefore,

$$G(X) \in \Phi \; impliesX \in R \tag{2}$$

where R is the region of acceptable parameter values in the parameter space.

During manufacture, the values of parameters are not necessarily independent but are varying statistically with a joint probability density function, $p(X)$. Thus, the manufacturing yield, Y, can be described by:

$$Y = \int_R pX\,dX \tag{3}$$

Another useful formulation results from the acceptance function $accept(X)$, such that:

$$accept(x) = \begin{cases} 1, & if G(X) \in \Phi : circuit\ accepted,\ X \in R \\ 0, & if G(X) \in \overline{\Phi} : circuit\ rejected,\ X \in \overline{R} \end{cases}$$

Now Y can be expressed as an expectation with respect to $accept(X)$;

$$Y = E[accept(X)] = \int_{-\infty}^{+\infty} accept(X)\ p(X)\ dX. \quad where\ 0 \leq E[accept(X)] \leq 1$$

Although this is an exact expression for Y, it can not be evaluated in general because an exact expression for $accept(X)$ is not known.

3 Monte Carlo Technique for Yield Estimation

A method commonly used for evaluating integrals in higher dimension is the Monte Carlo technique. The Monte Carlo technique approximates Y by:

$$\hat{Y} = 1/N \sum_{i=1}^{N} accept(X_i) = N_{pass}/N$$

where N_{pass} is the number of circuits which pass the specification test and N is the total number of circuits which are simulated, and the X_i are chosen according to $p(X)$. If we do not know the analytical expression for the acceptability region, R, then a Monte Carlo evaluation of the yield for a particular nominal design (i.e., $X = X_o$) requires N circuit analyses, one for each trial set of parameters X_i. Typically, hundreds of trials are required to obtain a reasonable estimate for Y.

The primary use of Monte Carlo analysis is to determine the yield, given component values and tolerances. Standard software packages include routines which accomplish this. However, in this work we are attempting to reverse the process and determine the component values and tolerances for a specified yield. This problem is not as simple and is not accomplished for CMOS circuits by any commercially available software packages. The yield factor histograms developed in the next section are excellent tools which will help the designer solve this problem.

4 Yield Factor Histograms

We wish to evaluate the yield as a function of each of the parameter values, $(x1, x2, ...xn)$. This is done by developing a yield factor which is given by:

$$Y(xi) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \cdots \int_{-\infty}^{+\infty} accept(X)p(X)dx1, dx2, \cdots, dxi-1, dxi+1, \cdots, dxn$$

This factor essentially averages out the effects of all but the ith component, and then a parametric study can be made to determine the effects of the ith component on this factor, and on the yield. To better calculate the yield factor an unbiased estimator of $Y(xi)$ is;

$$\hat{Y}(xi) = 1/M \sum_{j=1}^{M} acceptXj$$

where the ith component of X is fixed at xi and the others are allowed to vary by their known statistics.

The implementation is further simplified by dividing the acceptable values for $\hat{Y}xi$ into nine equal regions, with the nominal value of the component lying at the center of region five. An approximation to (xi) is developed by performing a Monte Carlo analysis where all parameters are allowed to vary by their known statistics and the sums are evaluated separately depending on the interval in which the value of xi lies. Since the estimate variance goes down as $1/M$, we have found a value of $M = 100$ is satisfactory. As 9 regions are used and approximately 100 evaluations of the circuit per region are adequate, at least 900 Monte Carlo runs (rounded off to 1000)iterations are needed to adequately describe the estimator of yield $(\hat{Y})$.

The graphical display of the yield factor (histogram) provides the important information of the circuit's yield with respect to a given parameter. The program SPICENTER computes and plots these histograms for each of the nine regions of a given circuit parameter. An interpretation of the histograms follows: 1) Design centering step: If the histogram is not symmetric about the selected point, the operating point should be moved to make the curve symmetrical. 2) Tolerancing step: If the histogram shows low yield values at the parameter limits, the acceptable parameter limits can be decreased within permissible costs.

These two steps are iterative. After the tolerances and nominal values are adjusted, the circuit is reanalyzed and the histograms are again plotted. Further adjustments are made if necessary. We have noticed that there is interaction between the two steps, because changing the nominal value can change the slope on the yield factor and vise versa. These steps and a few example histograms are shown in Figures 1 and 2 respectively.

Studying histograms will quickly tell the designer where the nominal value for each component might be placed for a better yield. Figure 2 shows three typical histograms. Histogram (a) is flat across the tolerance range and indicates that the previously chosen

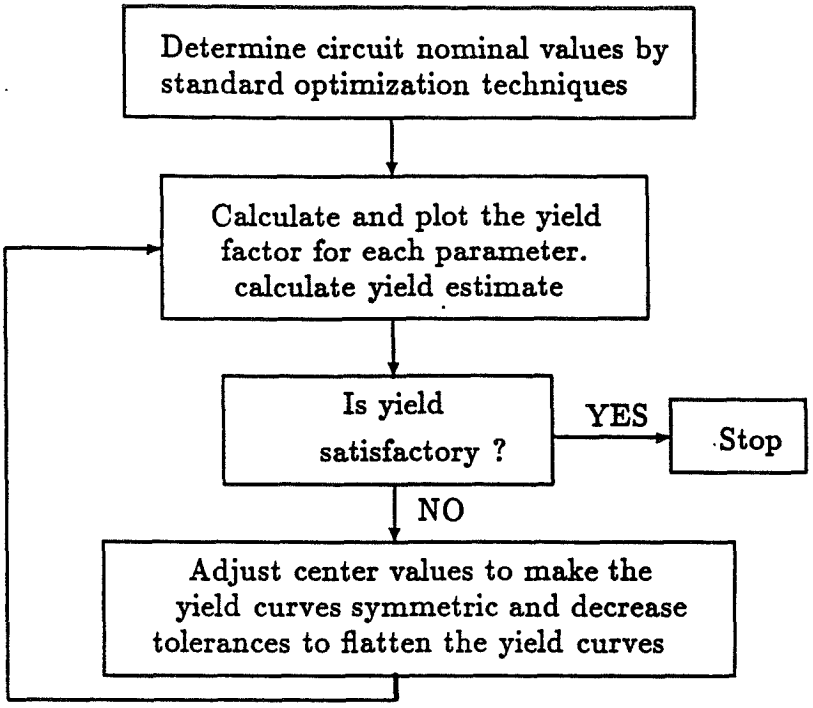


Figure 1: A Typical Approach to Statistical Circuit Design

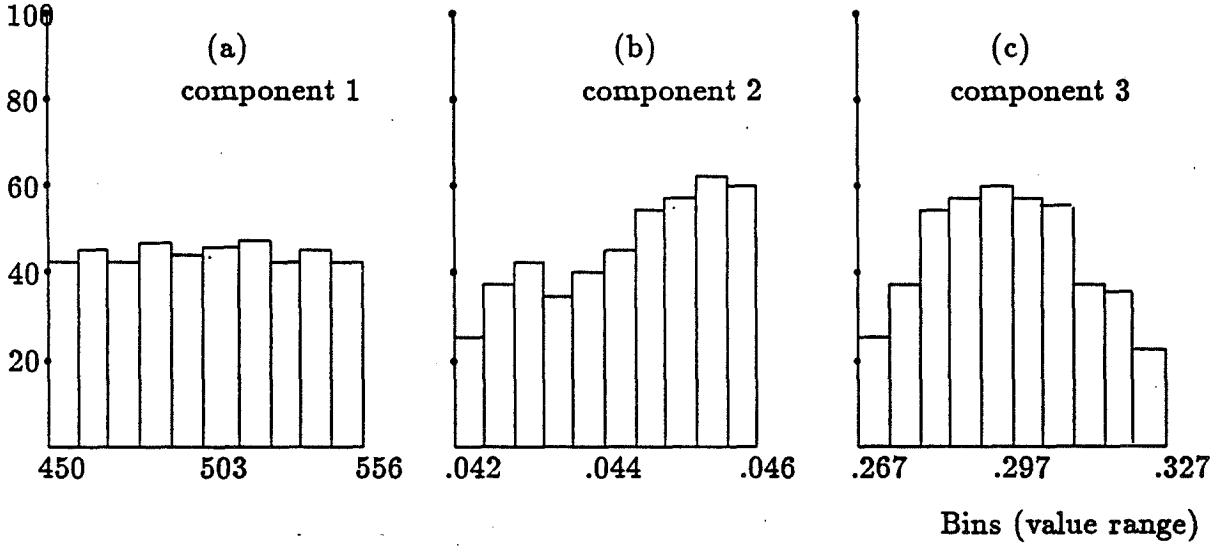


Figure 2: Typical Yield VS Component Value Histograms

C-4.

nominal value and tolerance are correct for that parameter. Histogram (b) indicates that a higher nominal value should be tried for the displayed parameter because the yield is higher for higher values of the parameter. Histogram (c) suggests that the designer should pick a slightly lower nominal value and tighten the tolerance on that parameter provided the cost of doing so is within reasonable limits.

In our work on the design centering problem, we ran through histograms for 12 parameters for the CMOS op amp circuit and for 13 parameters for the CMOS inverter chain circuit and made the required changes as demonstrated earlier with Figures 2 (b)and (c). We then reran the SPICENTER and obtained a new set of histograms. This process was repeated until the last set of simulations achieved the desired yield.

5 Example Circuits

Two example CMOS circuits are chosen to illustrate the use of the Yield Factor Histograms for improving the parametric yield of a circuit. The first example circuit is a CMOS operational amplifier. This is an analog circuit used in many standard analog cells. In both examples, the parameter statistics were assumed to be uniform and independent.

5.1 Operational Amplifier

Figure 3 is a two stage buffered CMOS op amp which is the first of the example circuits chosen for this paper.

5.1.1 Design Centering for Rise Time

The performance criterion chosen for design centering this op amp was the rise time, which is defined as the time taken by the output signal to rise from 10% to 90% of its final steady state value in response to a step voltage applied at the input. For the op amp the rise time is specified to be below 1.75 usec. It is important to note that our efforts in optimizing the design for a better rise time did not affect the other specifications of the circuit. The step input is applied at the noninverting input terminal of the op amp. SPICECENTER uses the transient analysis output (voltage versus time) supplied by SPICE to evaluate the circuit performance.

5.1.2 Yield Factor Histograms

The Figures CMOS OP AMP-SPICENTER(1) through (4) are samples of yield factor histograms generated by SPICENTER after iterations (1) through (4) of 1000 simulations of the circuit. These yield factor histograms depict the sensitivity of the circuit's yield to a few critical components and parameters. The histograms of the other parameters to which the yield is practically insensitive are not included here. The four iterations of SPICENTER are as follows.

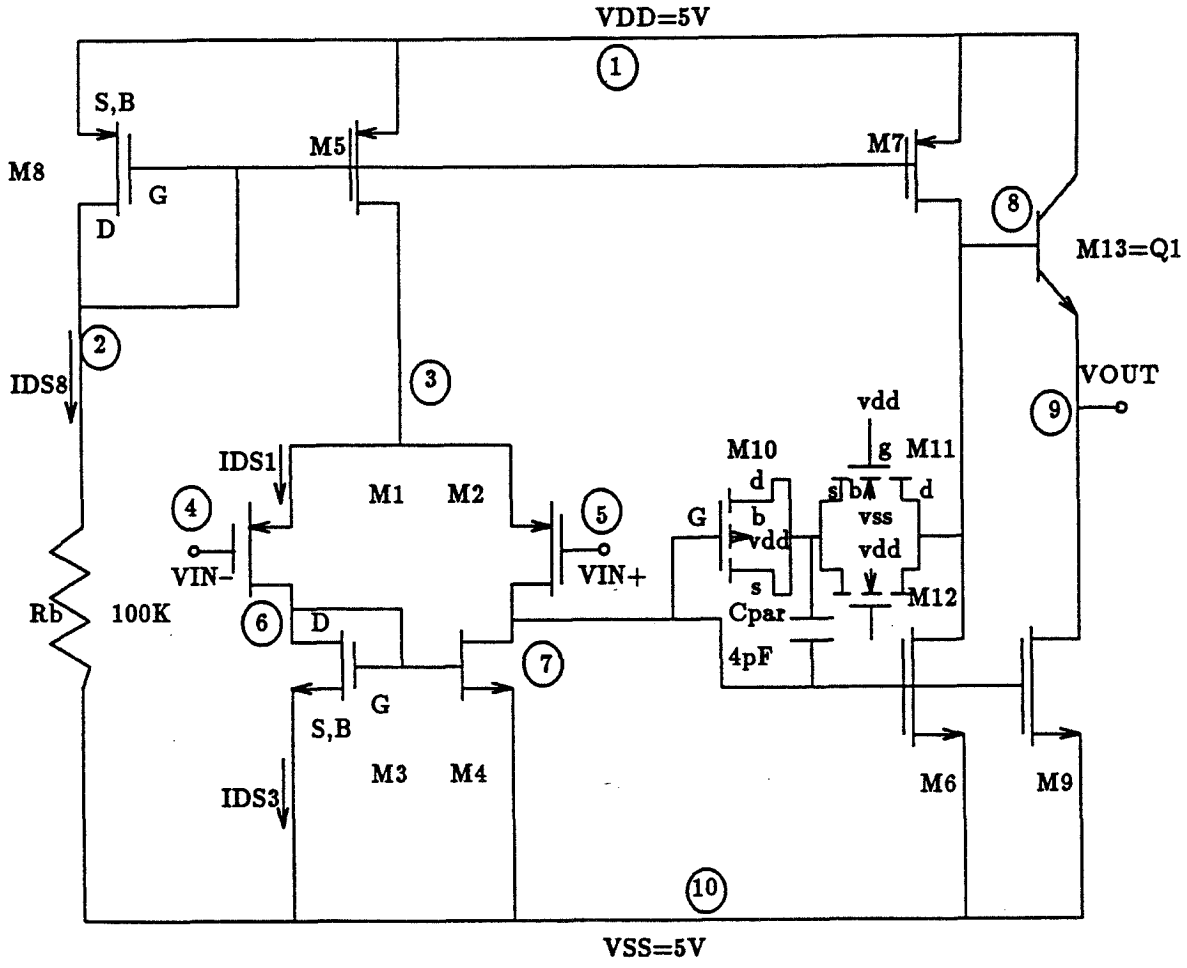


Figure 3: A Low-Power Buffered CMOS Op Amp Example Circuit

The first iteration of SPICENTER at the nominal values of the components and parameters resulted in a average yield of 21.9%.This yield and other information are printed on the right side of the histograms. It was observed that out of the twelve parameters two are very critical to the yield of the circuit. They are component 4, RBIAS (RES2) and component 10, Kp(PMOS).It is also clear from the plots that the yield is higher for lower values of RBIAS which is the bias resistor and higher values of Kp(PMOS) which is the transconductance of the PMOS transistors. (CMOS OP AMP-SPICENTER(1).

For the second iteration of the circuit simulations, the nominal value of RBIAS was changed from 100K ohms to 95K ohms in the circuit file. Its tolerance and other components or parameters were not altered. This resulted in nearly 100% improvement on the yield (21.9% to 43.0%). (CMOS OP AMP SPICENTER(2)).

Based on our interpretation of the histogram of bias resistor from SPICENTER(2), its value was further reduced to 90K ohms and the circuit simulated again. Once again there was a considerable improvement in the average yield of the circuit (43% to 70.6%. (CMOS OP AMP-SPICENTER(3)).

For the fourth and the final run of SPICENTER we modified Kp(increased from 13.5 to $15.525uA/V^2$) fixing the bias resistor at 90K. (CMOS OP AMP-SPICENTER(4)). These changes resulted in a yield of 80.7 %.

5.2 Inverter Chain

The inverter is an important circuit element in both analog and digital CMOS designs. It is the most basic gain stage of all amplifiers and the design of digital systems is virtually impossible without the basic inverter. A chain of inverters(inverter chain) chosen as the second example for this work is mainly used in ring oscillators and memory circuits.

5.2.1 Design Centering for Delay Time

One of the important characteristics of the inverter chain is the propagation delay time (delay time, for short). Delay time has been interpreted as the time taken at the output to attain 90% of its final steady state value from the instant the input is triggered with a step voltage. The transition magnitude on the input of the inverter chain is -5 volts to 5 volts. VDD is 5Volts and VSS is -5 Volts.

Delay time is used as the performance criterion for the inverter chain example circuit. For the inverter chain circuit the delay time is specified to be less than 4.5 nsec. Like the CMOS opamp's rise time, the CMOS inverter chain's delay time is calculated after each simulation of the circuit from the data provided by SPICE in the transient analysis table.

5.3 Histograms

The Figures CMOS Inverter Chain-SPICENTER(1) through (4) shown below are samples of the yield factor histograms generated by SPICENTER. The first run of 1000 simulations of the inverter chain estimated the yield at 38.5%. The histograms also pointed out two critical parameters of the circuit to which the yield is sensitive. They are Kp(NMOS) and

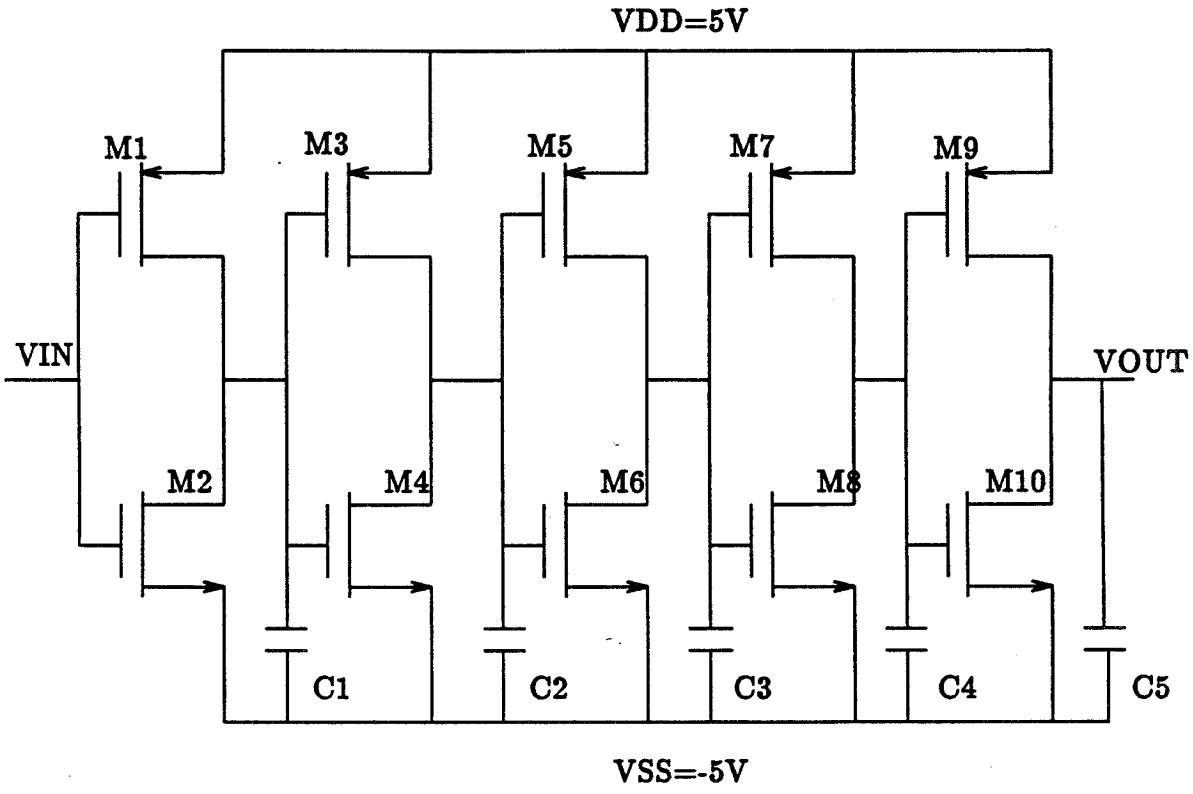


Figure 4: CMOS Inverter Chain Example Circuit

Kp(PMOS) the transconductances of both P and N channel transistors (CMOS Inverter Chain-SPICENTER(1)).

For the second iteration, Kp(NMOS), was changed to $20uA/V^2$ from its nominal value of $17uA/V^2$ without altering the nominal values of the other parameters and components. The resulting yield was 65.7%. (CMOS Inverter Chain SPICENTER (2)).

With Kp(NMOS) restored to $17uA/V^2$, Kp(PMOS) was changed to $9.5uA/V^2$ from its nominal $8uA/V^2$ before the start of the third round of circuit simulations. This time the yield was 56% as shown in CMOS Inverter Chain SPICENTER(3). This indicates that the influence of the transconductance of the N channel MOSFET on the circuit yield is slightly more than that of the P channel MOSFET.

The last simulation was done with the Kp(NMOS) changed to $20uA/V^2$ and Kp(PMOS) changed to $9.5uA/V^2$. The yield from this iteration was the highest at 80.9% (CMOS Inverter Chain-SPICENTER(4)).

6 Conclusions

This paper presents a unified design approach to parametric yield optimization by using statistical design methods recently developed at the University of Idaho [15,17]. The work involved the development and implementation of the computer program SPICENTER which calculates and displays Yield Factor Histograms.These histograms are used by the CMOS IC designer to perform design centering. These methods applied to the OP Amp increased the yield form 21% to 80% and to the inverter chain increased the yield from 38.5% to 80.9%.

Acknowledgements– This work was partly supported by a grant from the Idaho State Board of Education and the NASA SERC at the University of Idaho. A Patent application has been filed by the Idaho Research Foundation covering, among other things, the Yield Factor Histograms and their use in statistical circuit design. The Idaho Research Foundation address is P.O. Box 9645, Moscow,Idaho, 838430178, (208)8838366.

References

[1] . Balaban and J. J. Golembeski, "Statistical Analysis for Practical Circuit Design" ,IEEE Transactions on Circuits and Systems, Vol. CAS22, No.2, February 1975, pp.100–108.

[2] . K. Brayton and S. W. Director, "Computation of Delay Time Sensitivities for use in Time Domain Optimization" IEEE Transactions on Circuits and Systems, Vol. CAS22, No.12,December 1975, pp.910–920.

[3] . W. Director, G. D. Hatchel and L. M. Vidigal, "Computationally Efficient Yield Estimation Procedures Based on Simplical Approximation" IEEE Transactions on Circuits and Systems, Vol. CAS25, No.3, March 1978, pp.121–130.

[4] . W. Bandler and H. L. Abdel-Malek, "Optimal Centering, Tolerancing and Yield Determination", IEEE Transactions on CAS25, No.10, October 1978, pp.853–871.

[5] . Singhal and J. F. Pinel, "Statistical Design Centering and Tolerancing using Parametric Sampling", IEEE Transactions on CAS28, No.7, July 1981, pp.692–702.

[6] . K. Brayton, G. D. Hachtel and A. L. Sangiovanni Vincentelli, "A Survey of Optimization for I.C. Design", IEEE Transactions on CAS69, No.10, October 1981, pp.1334–1362.

[7] . J. Anterich and R. K. Koblitz, "Design Centering by Yield Prediction", IEEE Transactions on CAS29, No.2, February 1982,pp.88–96.

[8] . E. Hocevar, P. Yang, T. N. Trick and B. D. Epler, "Time Domain Sensitivities", IEEE Transactions on CAD4, No.4, October,1985, pp.609–620.

[9] . R. Nassif, A. J. Strojwas and S. W. Director, "A Methodology for Worst Case Analysis of Integrated Circuits", IEEE Transactions on Computer Aided Design, Vol. CAD5, No.1, January 1986, pp.104–112.

[10] . Yang, D. E. Hocevar, P. F. Cox, C. Machala, and P. K. Chatterjee, " An Integrated and Efficient Approach for MOS VLSI Statistical Circuit Design", IEEE Transactions on Computer Aided Design Vol. CAd5, No.1, January 1986, pp.514.

[11] . A. Styblinski and L. J. Opalski "Algorithms and Software Tools for I. C. Yield Optimization based on Fundamental Fabrication Parameters", IEEE Transactions on CAD5, N0.1,January 1986, pp.7989.

[12] . Herr and J. J. Barnes, "Statistical Circuit Simulation Modeling of CMOS VLSI", IEEE Transactions on CAD5, No.1, January,1986, pp.1522.

[13] . D. Matson and L. A. Glasser, "Macromodeling and Optimization of Digital Mos VLSI Circuits", IEEE Transactions on CAD5, No.4, October 1986, pp.659–677.

[14] . Hedensierna and K. O. Jeppson, "CMOS Circuit Speed and Buffer Optimization", IEEE Transactions on CAD6, No.2, March 1987, pp.270–281.

[15] . MacFarland and J. E. Purviance, "Centering and Tolerancing the Components of Microwave Amplifiers", Proceedings of IEEE International Symposium on Microwave Theory and Techniques, June 1987.

[16] . K. Yu, S. M. Kang, I. N. Hajj, and T. N. Trick, " Statistical Performance Modeling and Parametric Yield Estimation of MOS VLSI", IEEE Transactions on CAD6, No.6,November 1987, pp.1013–1022.

[17] . E. Purviance and M. D. Meehan, "A Sensitivity Figure for Yield Improvement", IEEE Transactions on MTT Vol.36, No.2,February 1978, pp.413–417.

[18] . E. Hocevar, P. F. Cox and P. Yang, "Parametric Yield Optimization for MOS Circuit Blocks", IEEE Transactions on CAD7, No.6, June 1988, pp.645–658.

[19] . M. Butler, "Realistic Design Using Large Change Sensitivities and Performance Contours", IEEE Transactions on Circuit Theory, Vol. CT18, NO.1, January 1971, pp.58–66.

[20] aly W, and Director S. W, "Dimension Reduction Procedure for Simplical Approximation Approach to Design Centering", Proceedings of Institute of Electrical Engg, Vol.127, No.6,December 1981.

[21] . E. Allen and D. R. Holberg, "CMOS Analog Circuit Design", 1st edition, Holt, Rinehart, and Winston Inc., New York.

[22] . Vladimirescu and S. Liu, "The Simulation of MOS Integrated Circuits Using SPICE2", Electronic Research Laboratory, College of Engineering, University of California,Berkeley, CA.

[23] . K. Brayton and R. Spence, "Sensitivity and Optimization", New York: Elsevier,

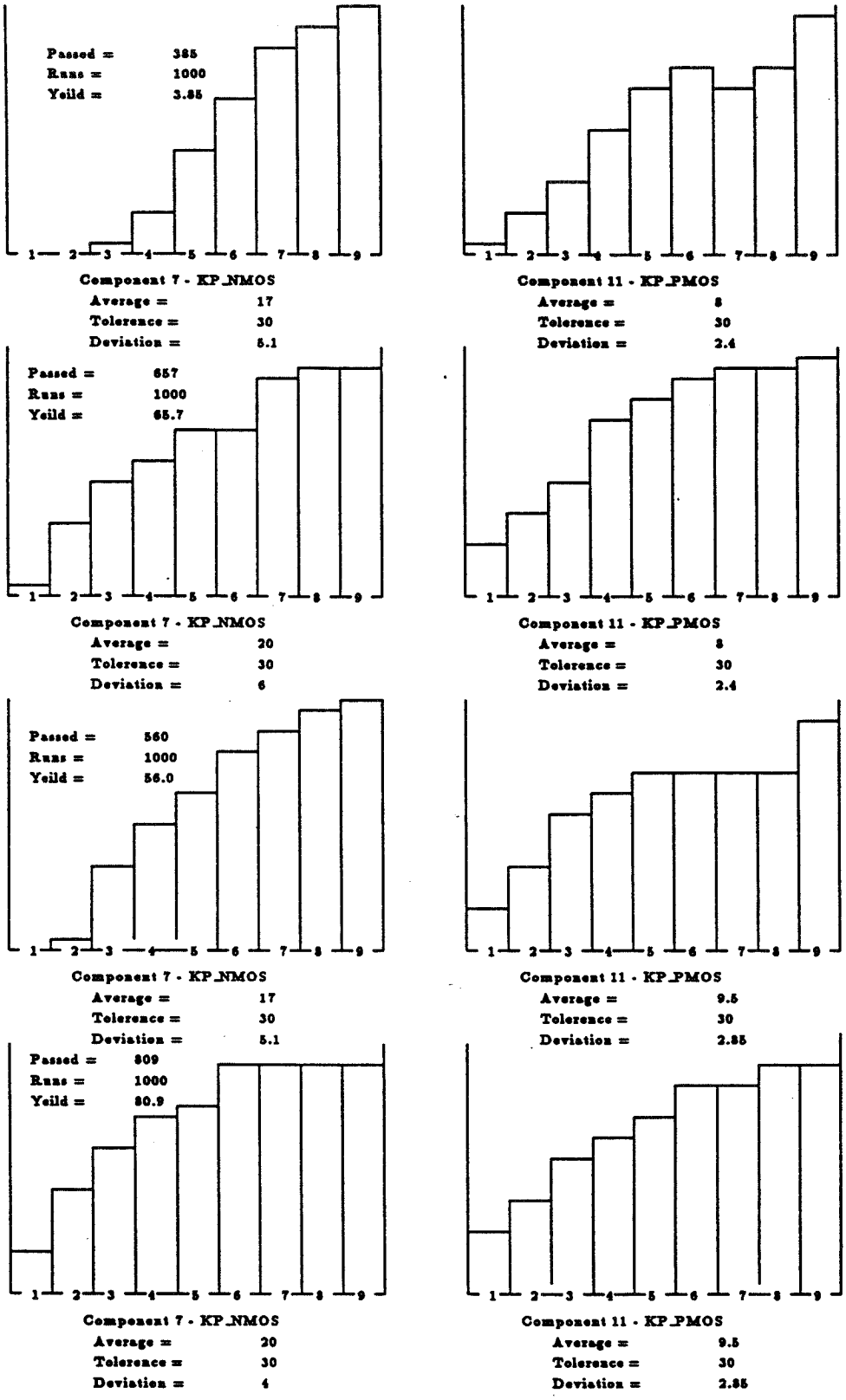


Figure 5: CMOS Inverter Chain

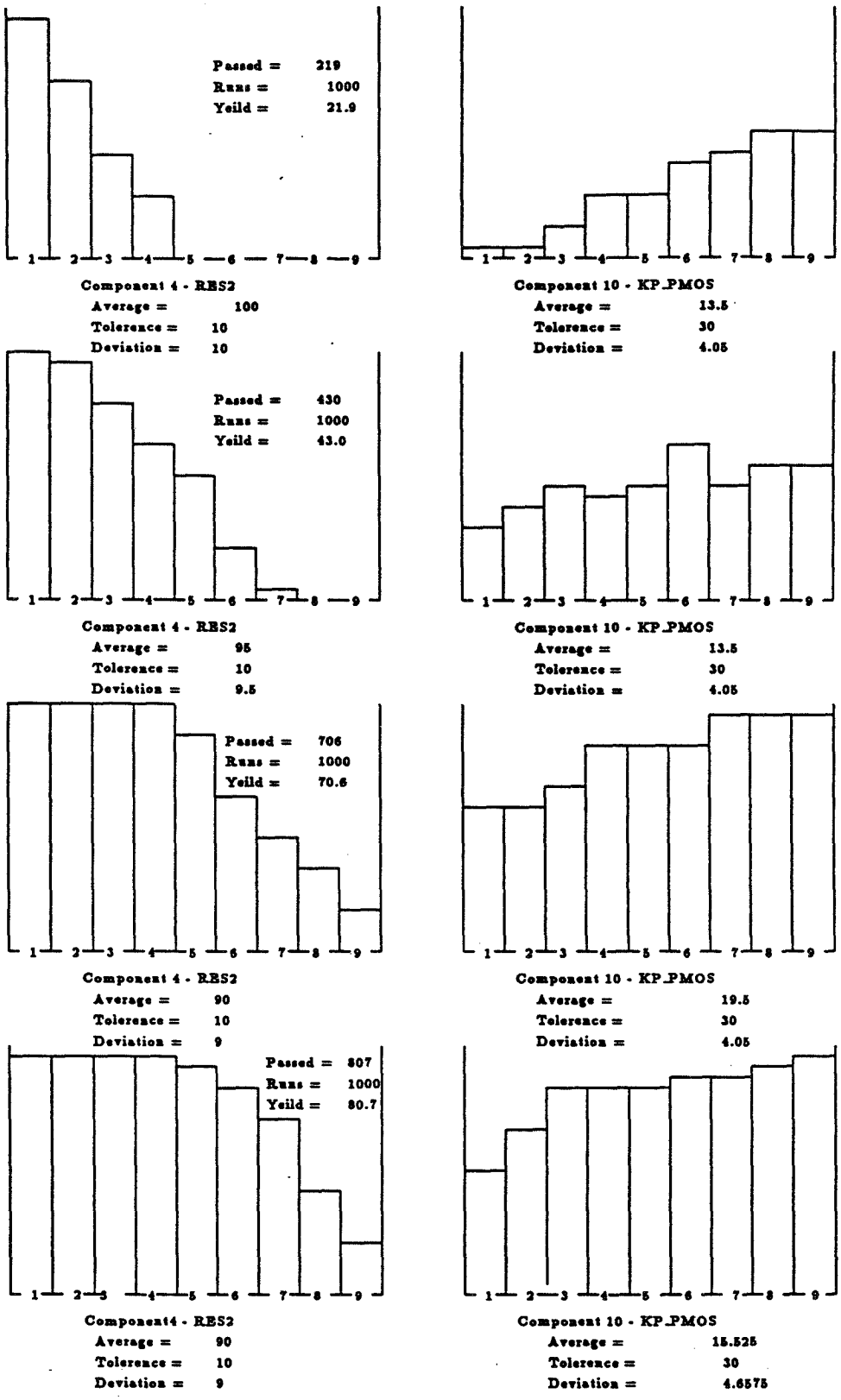


Figure 6: CMOS Op Amp