A Procedural Method for the Efficient Implementation of Full-Custom VLSI Designs

P. Belk
Flight Computer Systems and Technology Section

N. Hickey
NDH Consulting

An imbedded language system for the layout of VLSI circuits is examined. It is shown that through the judicious use of this system, a large variety of circuits can be designed with circuit density and performance comparable to traditional full-custom design methods, but with design costs more comparable to semi-custom design methods. The high performance of this methodology is attributable to the flexibility of procedural descriptions of VLSI layouts and to a number of automatic and semi-automatic tools within the system.

I. Introduction

Traditionally, full-custom integrated circuit design has been used in those situations where high performance or small size was of critical necessity. The designer has been forced to accept the increased development time and susceptibility to error over the so-called semi-custom approaches. This article is the first in a series of articles which describe a full-custom design methodology that supports many of the structured concepts inherent in the semi-custom approaches, allowing most of the benefits of a full-custom design to be realized while avoiding the penalties normally associated with full-custom design.

In a full-custom design, the designer must specify in great detail the actual cell geometries, cell placement, and cell routing of the chip. In exchange, the designer is able to control the system performance parameters of speed, power consumption, and size. In a semi-custom approach, the designer must select from predefined cell geometries, and the cell placement and routing are constrained by the automatic placement and routing procedures associated with the system. In many designs, there is only a small portion of the chip which is critical to the performance. In others, there are a few unusual functions which are not normally found in standard systems. In still others, the functions exist but are too large or slow or limited to be used in design. Hence, what is needed is a system which will provide the automatic capabilities of the semi-custom approach but which will also provide easy access to the special leaf cells, special placement, and special routing procedures which are needed in a given design.

In addition to the functional differences between this approach and the traditional methods, there is a philosophical one. The apparent goal of many traditional systems is to eliminate the designer from the design cycle. Under the approach described in this article, this is not necessarily considered a

worthwhile goal. Rather, the goal of the system is to maximize the efficiency of the designer by allowing the utilization of automatic tools where appropriate, procedural tools where appropriate, and manual tools where appropriate and to make it easy to determine when each is needed. Furthermore, a single, well-defined interface between all the tools allows the designer to use his intuition and intellect on those design aspects requiring the greatest design power. The interface to the system is based on the premise that the intuition and skill of the designer are valuable resources.

The central element of the layout system described is an imbedded language called Art. Art is compatible with a set of other tools created as part of the system, including editors, plotting programs, and design rule checkers. The methodology described in this article presumes that most of the chip layout task will be accomplished by using Art to perform placement and routing between pre-existing low-level cells. Although this is a very efficient method of implementing a large variety of very complex chip designs, it should be emphasized that there is no element in this system which mandates the use of any feature of an imbedded language for any part of the design. Thus, the designer may choose to use only Art's predefined placement and routing capabilities, to define more advanced or specialized placement and/or routing capabilities in Art, or to bypass Art entirely in favor of implementing the entire design in the graphics editor or other layout tools.

The low-level cells used by Art may be created by other components in the system, or imported. Imported cells may come from existing standard cell libraries, external cell generators, or other types of external design systems. Those generated internally may be manually generated (in the interactive graphics editor) or procedurally generated at any level. Procedural generation of cells is normally limited to composite cells (cells created by combining previously defined cells) and generated cells (such as PLAs and ROMs), but can optionally be used for the creation of special types of cells, such as technology-independent cells, or cells having configurable speed or drive capabilities. Consequently, the system provides for the combination, in a single design, of cells from a variety of different sources, and the designer may determine which sources best satisfy the requirements of the design.

The important design information is stored in a small collection of design databases by Art. Most of this information is contained in a cell database for each cell in the design. When cell information is read or created by the system, it is distilled down to a basic external description of the cell which contains only that information necessary to utilize the cell as a component in other cells. Thus, the size and amount of time necessary to scan the design database are minimized. This feature offers the flexibility to create design methods which process the entire external cell interface while still allowing a relatively short design loop.

Placement and routing of cells within Art is facilitated by a number of automatic routines. These include the "Tiler" program, which produces data-path-like structures, and an automatic channel router, "MidBus," which implements a very general two-layer routing algorithm for random interconnect between rows or columns of cells. Where these automatic tools are inappropriate, the designer may specify the placement and routing through easily written procedures. The designer may utilize any or all of these methods in a given design.

All of the tools are written in and compatible with Art, and thus the designer is able to choose the appropriate tool independently for each part of a layout. Hence, the designer is always able to be as specific as he desires in controlling the action of the system. If none of the existing tools is appropriate to the task at hand, he is free to modify an existing tool, to create an entirely new special-purpose tool, or even to directly specify the design procedurally. The designer is thus not forced to distort a design in an attempt to map an otherwise incompatible design to the capabilities of the system.

This methodology is sufficiently flexible to allow a large number of unique problems to be addressed. The example chip in this article required the generation of regular and irregular rectangular arrays of cells, the hierarchical interconnection of functional blocks, the use of a PLA, and the layout and interconnection of random logic. The designers were able to use the data-path compiler, the channel router, the automatic PLA generator, and various procedural place and route functions to complete the chip. It will be shown that the combination of these techniques provided an efficient, well-structured, and easily maintained design.

Other articles will deal in more detail with the generation of specialized cell generators, including an in-depth examination of the PLA generator and an examination of methods for producing technology-independent generated cells. The use of Art for the efficient generation of procedural place and route, such as its use in the control block, will be described. Also, the interface between this layout system and external simulation and verification tools will be discussed.

II. History of the System

The concept of using an imbedded language approach was first developed at the California Institute of Technology in 1977 with the creation of LAP [1]. As used at Caltech, however, this tool was regarded primarily as a method for creating low-level cells rather than for the assembly of complex higher-level cells [2]. Subsequent work of the Caltech Silicon Struc-

tures Project (SSP) proceeded along a path toward a "silicon compiler." The design system would require the designer to specify connectivity information and, possibly, relative cell placement information, from which the geometry would be created in an automated manner [2]. Examples of this approach include a system for the implementation of data paths, the Bristle-blocks system written by D. Johannansen in 1978 [3], and Earl [2].

The original Art program was developed at JPL in 1980 by John Wawrzynek using many of the concepts present in LAP. This version of Art was written in Pascal and supported only the capability of creating the basic primitive geometrical elements found in Caltech Intermediate Form (CIF) [4]. This original Art suffered from the requirement that a completely new version was needed for each new CMOS or NMOS technology.

In 1982, a completely new version of Art was produced at JPL by Paul Belk and Steve Trimberger. This Art supported some of the concepts of the current system, specifically the use of symbol names, the definition of connection points, and the maintenance of a design database for each cell. This system evolved over the next few years to include a number of semi-automatic routers and cell generators but still suffered from the requirement of a separate version for each technology.

In 1986, Art was completely rewritten under the C language. This version of Art was designed to be technology-independent and to support the generation of technology-independent designs. In the course of producing this version, a number of more sophisticated placement and routing tools were developed. In addition, the dependence on CIF as the geometric database was eliminated; the parser and generator for the geometry files were separated from the other code to allow the substitution of other geometrical formats. Furthermore, the concept of logical names for symbols and ports was extended to include instance names, and the ease of access and completeness of each cell's database were substantially improved.

Since its initial introduction, Art has seen substantial improvements and additions. It is expected that this enhancement will continue because of the ease with which these additions can be made to the basic system.

III. Description of Design

The sample chip to be described in this article is part of a multi-processor signal processing unit. The chip is responsible for controlling access to dual-port RAM, providing sequential RAM access for one of the CPUs, and providing a collection of utility counters and timers. In addition, it supports an autonomous data collection mode during the time that both CPUs are powered down.

The chip was chosen for this article because it provides an effective demonstration of the combination of several disparate design techniques on a single chip. At the highest level, the chip contains a data-path-like structure containing a large number of standard utilities, such as counters, latches, shift registers, and buffers; and a control section containing highly regular structures (including a PLA), irregular structures (clock generators), and completely random logic. It will be shown that, with the exception of lowest-level leaf cells, each containing about 10 simple gates and numbering less than 20, it was possible to efficiently implement all elements of the chip in Art using automatic or semi-automatic methods.

The chip is divided into four functional units (Fig. 1). These units are discussed below.

A. High-Speed Block

The high-speed block contains the cells which control access to the RAM by the two CPUs. Specifically:

(1) a 3-word by 16-bit buffered read FIFO;

(2) a 3-word by 16-bit buffered write FIFO;

(3) a 16-bit starting address latch;

(4) a 16-bit auto-increment address pointer; and

(5) a 16-bit bidirectional tristate buffer between the high-speed block and the low-speed block.

B. Low-Speed Block

The low-speed block contains a number of utility blocks, including timers, address latches and pointers, serial-to-parallel and parallel-to-serial converters, and command state latches. Specifically:

(1) a 16-bit address multiplexer;

(2) two 16-bit latches;

(3) a 16-bit auto-increment address pointer;

(4) 16-bit serial-to-parallel and parallel-to-serial converters;

(5) 6 4-bit to 25-bit timers;

(6) a 5-bit, double-buffered command latch; and

(7) a 16-bit test shift register.

C. Control Block

The control block contains the PLA which controls all other devices on the chip. It also contains circuits for condi-

tioning off-chip inputs and outputs and for producing the two-phase clocks required by other circuits. Specifically:

(1) a 23-word by 42-bit control PLA;

(2) an inverting Schmitt trigger for reset conditioning;

(3) two one-shot circuits for edge detection;

(4) 7 flip-flops for conditioning of input signals;

(5) 3 two-phase, non-overlapping clock generators for producing chip clocks; and

(6) synchronization logic for synchronizing memory requests from the low-speed block with the high-speed clocks.

D. RAM Access Arbiter Block

The RAM access arbiter block contains the random logic necessary to control RAM access between the two CPUs and the chip. It also creates the signals that increment the RAM address pointers.

IV. Description of Methodology

Much of the power and generality of the approach described comes from the fact that the designer may solve a complicated design problem by breaking the problem into a number of separate stages, each of which utilizes a small number of simple and well-understood design techniques. The design is thus reduced to the successive application of these techniques. Since these techniques may be combined in a very flexible manner at each level, it is possible to implement a chip architecture which is quite complicated when viewed in its entirety, by successively breaking the design into architectural blocks, each of which in turn consists of a small number of blocks having a single simple, well-defined interface. Further, the implementation of each technique may be broken into a small number of simple primitives.

Since the methodology described in this article is a layout methodology, no consideration is given to other systems issues. In particular, it must be assumed that the system design has been properly partitioned to allocate reasonable functionality to the chip being implemented and that the system design as a whole has been simulated to verify that the chip, once produced, will operate properly within the system. It will also be necessary at the conclusion of the layout effort to provide for functional layout verification in the form of a layout vs. schematic check or a net list extraction. Layout design rule checking is also important but is usually performed continuously during the design. In the case of the chip described in this article, design rule checking was performed with the internal design rule checker DRC, which is compatible with both Art and the graphics editor.

As with any layout methodology, it is very important to devote sufficient design time at the beginning of the project to the development of an optimal design partitioning scheme. It is extremely important that, to the greatest extent possible, each stage of the design be reduced to a small number (normally less than 10) of self-contained blocks having a simple and well-defined interface. Although it is possible to implement a badly partitioned design using the methodology described, the design will almost certainly be significantly more prone to error and take much longer to implement.

The following sections provide a brief description of the primitives and techniques which were combined to implement the chip discussed. These concepts are a subset of those supported by this methodology but are sufficient to demonstrate its use.

A. Imbedded Languages

The power and flexibility of the design methodology described in this article are attributable to the use of direct procedural methods for specifying the placement and routing of objects based on logical references to size, location, and type attributes associated with those objects. The generation of the layout is thus reduced to a programming task utilizing all the power and flexibility that such an approach implies.

Art consists of the definition of various database structures containing information about the design, a subroutine library for creating and manipulating these structures, routines for converting these structures into the final geometrical information, and a collection of macros to allow easier access to the program functions. Instead of attempting to define a special purpose syntax for the specification of the design, Art is written in and accessed by programs written in C. Thus the designer is able to use all the functionality and power of a standard programming language. Simply stated, Art is a layout system imbedded in C, or more simply an "Imbedded Layout Language."

No assumptions about the nature of the geometrical design rules or the target fabrication technology need be made in Art. Art is very largely technology-independent in the sense that such details as design metric, mask layers, geometrical design rules, and the format of the final layout database are not built into Art. The design metric may be chosen by the designer, since the numbers used in Art may be taken to represent whatever measurement units are necessary. The names and natures of the mask layers are read by Art from a file unique to that technology, as are a set of geometrical design rules, and the layout database is read and written by separate input and output modules and converted to an internal database which is

used by Art. Art tends to assume a certain flexibility in terms of symbol hierarchy, but the system includes filters which may be used to flatten the hierarchy to whatever extent is necessary for the target mask pattern-generation technology. The chip described was produced for a standard 3.0-micron CMOS/Bulk technology and the mask data was transferred to the fabrication house in the Caltech Intermediate Form (CIF).

Art is not limited to use by experienced programmers. Although a skilled programmer may access the primitive functions directly to produce a layout, Art may also be used to generate various placement and/or routing functions which are then accessible to non-programmers. In the sample chip, both approaches were used. The data-path placement program Tiler, the channel router MidBus, and the PLA generator were written in Art by skilled programmers to allow simple non-procedural specification for major portions of the design. These programs may be used by non-programmers to solve a more general set of problems. In the control block, however, much of the placement and routing was performed using a few special purpose subroutines which directly accessed the Art primitive functions.

The capability of defining both general-purpose and special-purpose placement and routing functions within the same system allows for maximum efficiency in the design process. Furthermore, it is often possible to enhance a special-purpose solution to the point where it provides a more effective general purpose solution than the more traditional semi-custom approaches.

B. Symbols and Instances

At a basic level, a symbol is a portion of the design which has been grouped into a single self-contained logical unit by the designer for reasons of convenience. It is defined solely by its geometrical representation if generated external to the system, or by the imbedded language source from which the geometry is eventually created. The geometrical representation used in this system is extended to include abstract attributes including each symbol's name, size, connection points, and alignment points. When the symbol is processed by the system, a "symbol reference database" containing this abstract information is created. This database is associated with a program identifier identical to the symbol name.

When a (child) symbol is used within the definition of a (parent) symbol, only the abstract information contained in the child symbol's reference database is accessible. Thus, within the context of the parent symbol, the child's abstract attributes comprise the complete description of the child symbol.

This use of the child symbol is referred to as an instance. The instance database includes the reference to the child symbol, the physical location of the instance within the parent, and an optional instance name. If an instance name is provided, the location of the instance's connection and alignment points may be easily accessed. If no instance name is provided, this information is not readily available. The instance name, if provided, will appear in the extended geometrical representation of the parent symbol.

C. Ports: Connection and Alignment Points

When combining child symbols within the parent symbol, it is necessary to determine both the correct location for the child (relative to other items within the parent) and the locations on the child to which connections are to be made either by wire connection or by direct cell abutment. These locations are referred to as the child symbol's (or instance's) ports. Each port is described in the child symbol's reference database. This description includes the port's location on the child symbol, its name, its connection layer, and its type. For connection points, the connection layer indicates the material (e.g., metal or poly) of the wire which should connect to it; the type indicates the purpose (e.g., input, output, VDD, etc.) of the connection. If the port is used only for alignment purposes, then the material and type are left blank. All references to a port within the language are made using the instance name and port name.

In order to simplify the action of many of the special purpose interconnect procedures, the designer will often impose a naming standard for the ports. For example, it is common to define the upper right VDD port as "VDD_ur" and the lower left ground port as "GND_11" and to use these as the alignment points. Similarly, on signals which feed through the symbol vertically, the top port is referred to as "name_t" and the bottom port as "name_b."

D. Leaf Cells and Composite Cells

Previously, symbols were defined as a portion of the design grouped into a unit by the designer for convenience. It is useful to distinguish between several categories of symbols based on the symbol's use and internal structure. The simplest type of symbol, containing at most a few geometrical objects and no ports, which has been defined solely to allow easy access to a standard feature of the design technology, is referred to as a *macro symbol*. These symbols serve the same purpose as macros in a programming language. Common examples, used in most designs, are the inter-layer contacts.

More complex symbols, which provide a basic circuit function, are referred to as leaf cells. Leaf cells are the basic atomic building blocks of a structured design. Often these cells are

quite simple internally, consisting of only a small number of gates, but occasionally may be very large atomic structures such as PLAs, RAMs, or ROMs, which, though large, still cannot be readily described in terms of simpler functional units. When the cells are small and simple in function, they are frequently created by hand in an interactive graphics editor. When more complex, it is often useful to create special generators such as PLA or ROM generators to create the cell. Either type of leaf cell may also be easily imported from an external system which provides a library of cells or of cell generators.

Leaf cells approximate the function of standard cells in a standard cell system but differ from standard cells in two important respects. First, they are defined by the designer only as needed for the particular application; and second, they may be optimized, or customized, for the specific environment in which they will be used. Though this customization is not necessary for the design methodology to work, it provides a powerful means of increasing the efficiency of the final design.

Leaf cells are normally viewed as "black boxes" for both layout and function. Hence, in addition to the geometrical information, it is necessary to provide the abstract attributes of each leaf cell. Various methods have been provided for doing this. The graphical editor supports the direct entry of this information. When standard cells are used, it is often sufficient to convert the "footprint" data which is provided with the cell library.

As was described previously, the methodology is based on the synthesis of the final design by combining many symbols into more complicated symbols. These symbols, which are readily considered to be a collection of functional sub-blocks, are referred to as composite cells. The distinction between leaf and composite cells is extremely useful in understanding the application of the design methodology within the context of the generation of these symbols, but when a symbol is used as a child symbol for the generation of a parent symbol, this distinction becomes unimportant. That is, within this design methodology, leaf cells and composite cells may be used interchangeably as child symbols in the generation of higher-level parent symbols.

It is important to realize that a normal design will utilize at least 3 separate levels of source files. The results of each level are then used as if they were leaf cells for input to the next level.

V. Implementation Details

Within the Art methodology, a chip is designed by starting with a top-level architectural view and dividing the chip into a small number of relatively self-contained components having a well-defined interface. These components are then analyzed in a recursive manner to reduce them to simpler components in a similar manner. This process continues until the designer is able to identify an appropriate set of leaf cells with which the design may be implemented.

Once a preliminary leaf cell set has been identified, the designer may modify the chip design in order to minimize the number of leaf cells which must be created for the design. Also, he may determine which, if any, of the leaf cells are already available from external sources. He will also make a preliminary decision on the method to use to generate the leaf cells.

Having defined the design hierarchy and the leaf cell set, the designer must then resynthesize the complete chip. He does this by choosing the appropriate placement and routing algorithms with which to create each component. The designer must also consider the overall chip floor plan when implementing each component so that its size and port locations are well matched to the overall structure. It is not uncommon for a designer to modify the initial split-up of the design at this point in order to simplify the final design. Hence the split-up and synthesis should be viewed as an iterative process.

For the sample chip described in Section III, the top level was divided into components which corresponded to the four functional blocks. Each block required connections to the chip's pads and each block shared many control signals with every other block. Analysis of the interblock connections provided the following:

(1) The high speed and low speed blocks' data busses interconnect;

(2) The low speed block and control block have many common signals;

(3) The high speed and arbiter blocks have many common signals; and

(4) The high speed block has most of the data bus pad connections.

Thus, the most practical layout had the high speed block above the low speed block on the left of a central routing area and the arbiter block above the control block to the right. The data busses for the high speed block were routed from the middle of the high speed block to pins on the left; and across the chip, between the control and arbiter blocks, to additional pads on the right. Details of the layout methods used for the top level of the chip are described in the following section.

A. Top-Level Layout

Once the top-level functional division and floor plan were decided on and each of the four functional blocks generated, it was possible to produce the actual top-level geometry. The generation of this geometry used a number of the techniques which are supported under the Art system. The sequence of operations was as follows:

(1) Initialize the Art system and open output files;

(2) Read in geometry information for the sub-blocks;

(3) Scan each sub-block to obtain a list of all its ports;

(4) Place high speed and low speed (PMC) blocks at final locations, calculate y positions of arbiter and control blocks (x position resolved later);

(5) Generate the signal list and tie points for the central bus;

(6) Allocate bus channels and determine bus width;

(7) Place arbiter and control blocks;

(8) Draw the main power grid;

(9) Draw the middle control bus;

(10) Process the non-bus ports; and

(11) Close the symbol definition and the output file.

The main program which performs the above steps is a fairly compact three pages of code. The resulting layout is shown in Fig. 1.

B. Port List Scanning

One of the more tedious aspects of entering the layout program is the individual description of the types of interconnect that are required for each individual port in the subcells. In the top level of the sample circuit, it was determined that all of the ports on each cell fell into one of a few groups. The actions necessary for each port in the group were easily specified. Since one of the actions supported in the Art system is the ability to scan each port of a symbol, the layout program was able to generate the necessary two lists of port names for each symbol.

All ports which were to be connected to the central data bus were given names starting with the characters "bus_". These characters were followed by the name of the control signal to which it was to be connected plus an optional "_1", "_2", etc. For example, port "bus_phi2" would be connected to control line "phi2", and ports "bus_phi1_1" and "bus_phi1_2" would both be connected to control line "phi1". The control bus connectivity was thus readily available directly from the port names themselves.

Similarly, the VDD, GND, and pad I/O ports were provided with names which indicated the correct connective action. For example, the low speed block supported automatic checking of the interconnect with the high speed block (ports starting with "lp_"), pad connection ports on the bottom of the block ("pd_"), pad connections on the left ("pl_"), and VDD and GND connections. Any port whose name did not start with one of the expected prefixes would cause an error message to be displayed.

At first glance, restricting port names to start with the prefixes above might seem to place an unreasonable burden on the designer. In fact, however, all the port names were generated procedurally in the lower blocks and the final interconnect was verified with a layout vs. schematic tool. Hence, the top-level interconnection was produced with very little effort.

C. MidBus Router

The large numbers of control signals which were interconnected between the four sub-blocks of the chip suggested the use of an automatic channel router. Since no channel router existed in the system at the time this chip was implemented, it was decided to produce one specifically to satisfy the requirements for this chip.

The first issue resolved was the router type. Due to the preexisting power bus routing, it was decided to use a simple channel router utilizing a vertical metal channel with horizontal connections to the sub-block ports made in metal-2. Furthermore, it was decided to support the sharing of multiple signals within a given vertical channel but not to allow a signal to jog between vertical channels. This resulted in a very fast procedure which produced acceptable interconnect routing for this chip.

Using the port list which was generated as described in the previous section, it was a simple exercise to generate the list of signal names and tie points. An initial list of signals was also provided in the source code as a check for any missing signals. Finally, a list of the control signals which were to be connected to the chip's pads was also included.

The generation of the entire control bus was thus reduced to the following steps:

(1) Define control signals with external (i.e., pad) connections;

(2) Add an initial list of expected control signals (for extra check);

(3) Generate a complete list of signals and tie points from port scan;

(4) Allocate vertical channels to signals (and determine bus width);

(5) Place each block at the calculated location; and

(6) Create the geometry for the busses.

Note that the routines utilized in steps 4 and 6 constitute what is considered the MidBus router. It is thus able to produce a bus from any list of signal names and tie points.

D. Sub-Block Implementation

Having defined the interface between the top-level cells, it was necessary to determine the design technique most appropriate for each of the cells. The highly regular structure of the high and low speed blocks, along with the fact that each of these blocks operated on three common data busses and had a minimum of other interconnections (except for control signals), made them ideal candidates for layout using a datapath layout tool. Such tools are generally limited to bus-type structures but often prove to be the most efficient implementations of such structures.

The arbiter block was composed mainly of random logic and had very little internal regularity. Large amounts of random logic are notoriously vulnerable to layout error and often require many design iterations, but it was determined in this case that the required random logic could be reduced to a small number of leaf cells, each containing fewer than 10 simple gates. The cells were then placed in two columns and routed together using the MidBus router (Fig. 2).

The control block also lacked the regularity apparent in the high and low speed blocks but could make use of many of the lower-level primitive subroutines which had been developed as part of Tiler to allow the direct specification of its internal placements and wiring. The PLA in the control block required the generation of a special-purpose CMOS/Bulk PLA generator which accepted assembler-like input describing the state machine and generated a PLA cell. Neither of the special techniques used within the block is within the scope of this article, and the control block is thus best treated as a leaf cell within this context.

E. Tiler Datapath Compiler

Two major sub-blocks of the chip, the high speed and low speed blocks, are easily represented as rectangular arrays of leaf cells. It was determined early in the design cycle that the optimum arrangement is a vertical slice for each functional unit with 3 common data busses routed between each bit. Control logic for each unit is contained in a leaf cell at the top of each column, and the control lines run vertically through each cell.

Placement of the cells in each column is defined by a simple "COLUMN" macro. This macro creates a data structure containing the name of the active cell, the name of the control cell, the active bit positions, and whether the cells should be placed with or without first mirroring them. In addition, a number of data arrays are defined for each column. The "1_name" array provides a list of the ports which must line up vertically inside the column. This information is used by the system to verify the leaf cell design. The "x_name" arrays contain a list of contact points for the contacts to the busses and for those cells which are contact programmable. The "p_name" arrays provide the port assignments for the external control signals. Using the information provided in the structures just described, after placement of each column, the system would automatically define the external control ports, verify the control signal alignment, and place contacts at the appropriate locations for bus connections and configuration.

In addition to the three busses, adjacent columns may connect directly together. Placement of these input and output ports was performed in an *ad hoc* manner and then verified by the interconnect procedures. Figure 3 shows the low-speed block. It should be noted that most sections of the low-speed block have a very high density of active circuitry. Those sections which are empty are due to the fact that some of the elements in this block are less than 16 bits wide and therefore did not require a full bus slice. The high-speed block was composed almost entirely of 16-bit slices and thus is uniformly dense.

F. Probe Point Generator

In any complicated chip, it is useful to provide for direct probe access to verify the chip operation and/or determine the reason for non-operation of the prototype units. On the sample chip, it was determined that the majority of signals of interest were present on the middle control bus. In addition, it was determined that there would be ample space between the control block and the bus arbiter on the right side of the chip, directly adjacent to the middle control bus. This provided an opportunity to create two arrays of probe points, which would provide the capability to sample any signal on the control bus.

The creation of the probe point arrays was accomplished by writing a simple routine in Art which created a metal pad with a large enough cut in the passivation layer to allow the metal to be directly contacted by a normal chip probe. A wire was run from the center of the metal pad to the inside periphery of the probe array, and a port was placed at the end of the wire. To provide for the probing of each signal of interest, a list was

compiled of all signals in the control bus. That list was then scanned in a loop, and a probe point was created for each signal in the list. When this cell was added to the top-level chip, the MidBus router automatically scanned the ports at the periphery of the cell and connected each of the probe points to the appropriate signal on the control bus.

As can be seen in Fig. 4, the addition of the probe point array significantly changed the topography of the chip, especially the width of the MidBus, due to the increased length of most of the signal lines, which had previously been much more local. With the exception of the generation of the routine that created a single probe point in the array (less than one page of code), the entire process was automatic. The generation of the probe point array required a total of approximately 4 man-hours. Further, the difference in the characteristics of the control bus before and after the addition of the probe points provides a reasonable indication of the efficiency of MidBus in use of space.

VI. Summary

In this article, a methodology was presented, the Art methodology, in which an imbedded language is used to manage the complexity of a full-custom VLSI design. It was shown that the use of this methodology significantly reduced the complexity faced by a designer at any stage of a design without appreciably reducing the density or performance of that design. Thus, the Art methodology provides most of the benefits of traditional full-custom integrated-circuit design while substantially reducing the design costs.

Unlike many semi-custom design approaches, the Art methodology does not impose any inflexible constraints on the design itself. Several methods are shown which make the implementation of circuit structures less difficult, and through the application of combinations of these methods, a large variety of circuits may be implemented. Further, the designer is always free to disregard any of the existing methods in favor of developing special-purpose techniques for the implementation of specific structures. Although this development effort may increase the cost of the design, it is shown that many such techniques can be developed quite efficiently within the context of Art, and, more importantly, that such development will not affect the implementation of other parts of the design. Thus, the designer is allowed to make individual cost/benefit trade-offs on whichever portions of the design are considered critical.

Art provides considerable power beyond the management of high-level design complexity. Since Art is essentially technology-independent, it can be used for the development of designs in a wide variety of technologies, allowing rapid adaptation to the new circuit design technologies and techniques. In addition, the independence of the functional aspects of Art from the final representation of the mask geometry allows the adaptation of Art to diverse fabrication technologies and the implementation in Art of more fabrication-specific constructs, such as alignment marks.

The chip discussed in this article provides only one example of the power of the Art methodology. The chip was chosen because it demonstrated several different implementation techniques, but there are additional techniques that may be explored by examination of other types of chips. The chip described, however, provides useful information about the efficiency of the Art methodology. Because it was possible to use the same leaf cell in multiple contexts, it was possible to reduce the number of leaf cells designed to approximately 20. The layout of these cells required approximately 8 man-weeks. After the completion of leaf cell design, the implementation of the entire chip required three weeks for a two-person design team. Thus, a 14-man-week design effort resulted in the implementation of a chip having approximately 12,000 transistors. A small number of errors were detected in the chip by subsequent automatic layout vs. schematic comparison, ranging from errors in top-level interconnection to errors in leaf cell design. These errors required less than a day to correct.

Acknowledgment

The authors wish to acknowledge Tom Bulgerin, Linda Lee, Peter Jones, and Tim Shaw for their contribution to the development of the layout system and the chip used as an example in this article. The authors also gratefully acknowledge Dr. William Whitney for his insight and advice in the development of the topics discussed.

References

[1] C. R. Lang, *LAP User's Manual*, California Institute of Technology Computer Science Department Technical Report No. 3356, 1979.

[2] S. Trimberger, "A Structured Design Methodology and Associated Software Tools," *IEEE Transactions on Circuits and Systems*, vol. CAS-28, no. 7, July 1981.

[3] D. Johannansen, "Bristle Blocks—A Silicon Compiler," in *Proceedings of the 16th Design Automation Conference*, 1979.

[4] C. A. Mead and L. A. Conway, *Introduction to VLSI Systems*, Addison-Wesley, 1980.

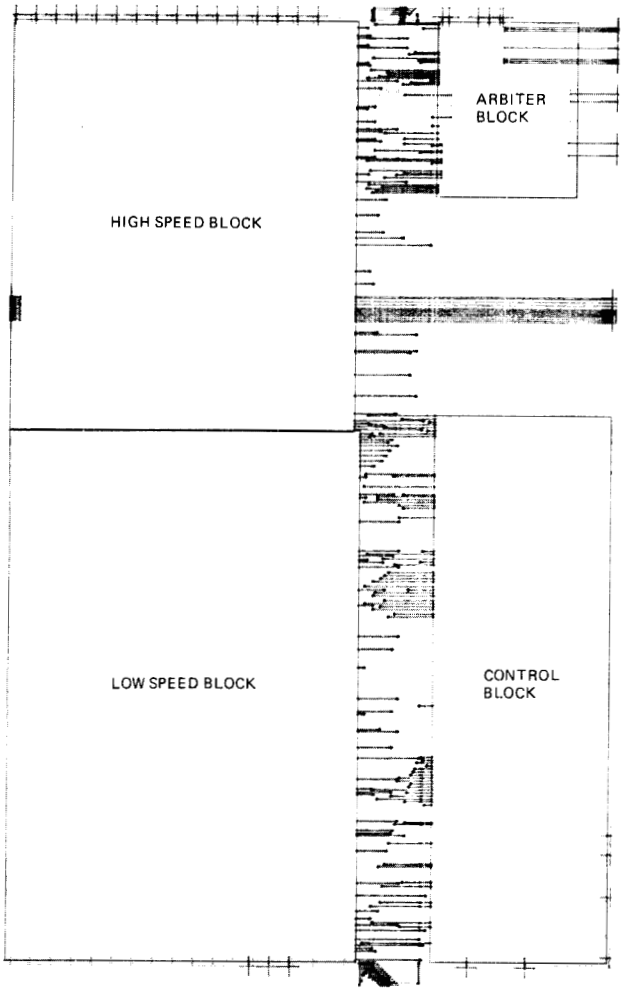

Fig. 1. Top level of sample chip

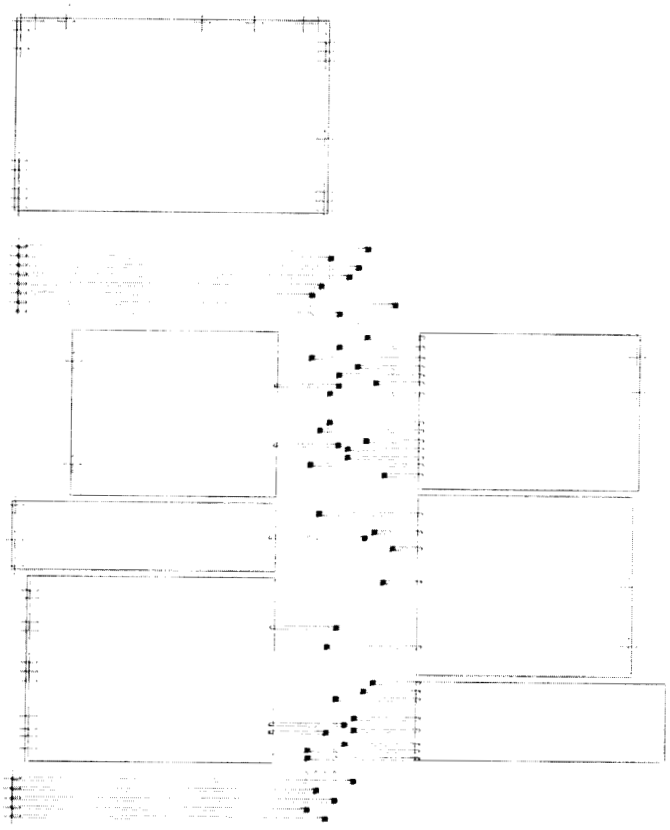

Fig. 2. Arbiter block

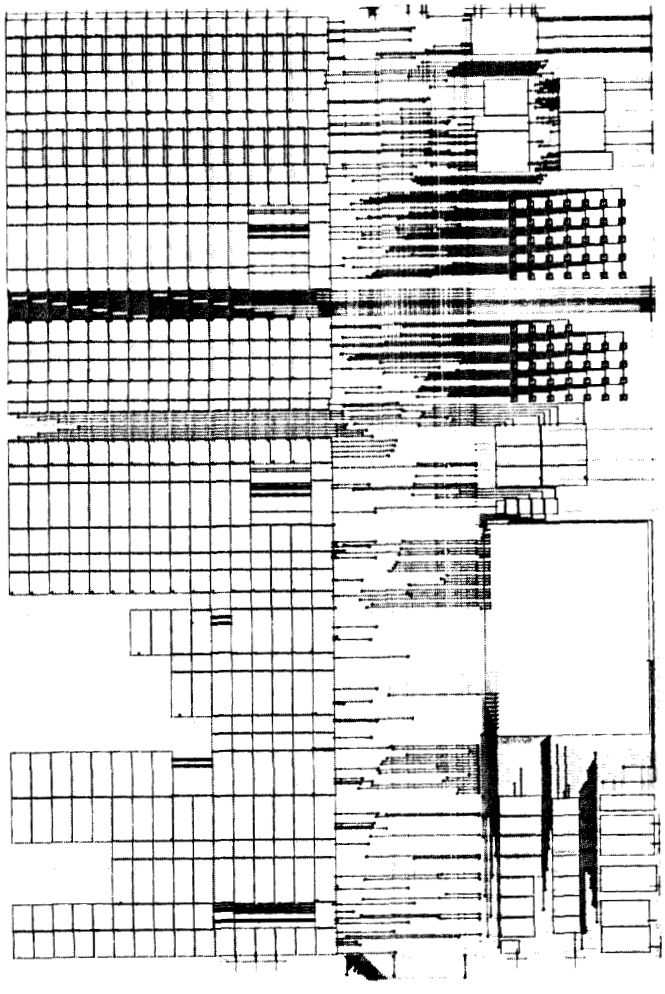

Fig. 3. Low speed block

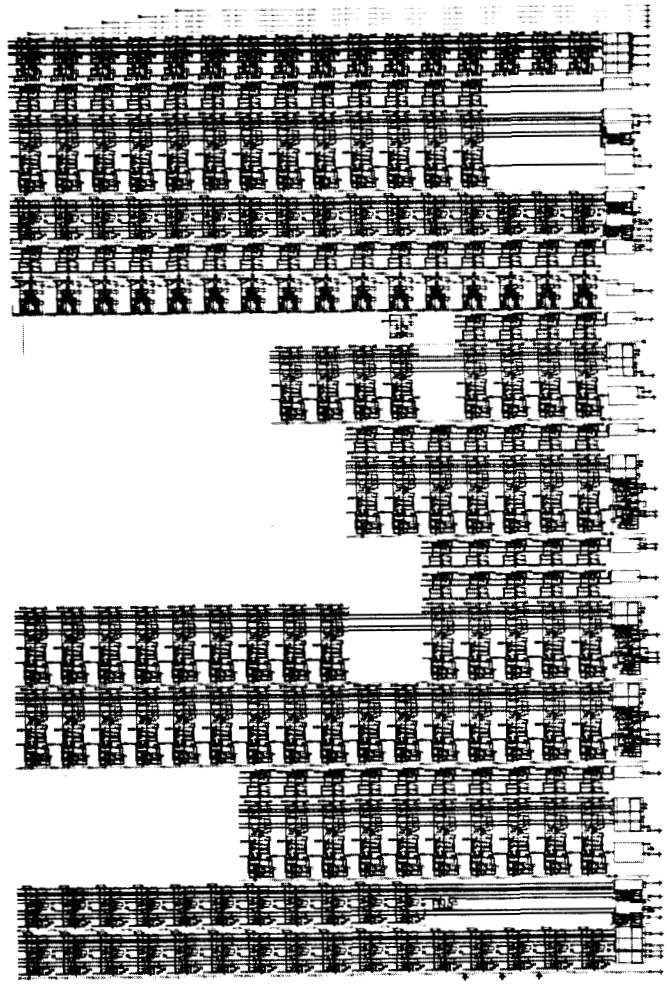

Fig. 4. Top level of sample chip with probe points