Cell Matrix Methodologies for Integrated Circuit Design†

Tony M. Carter, Kent F. Smith, Steven R. Jacobs, Richard M. Neff
University of Utah
Department of Computer Science
3190 Merrill Engineering Building
Salt Lake City, Utah 84112

Tony Carter
UUCS-89-004

Abstract

A class of integrated circuit design and implementation methodologies is described. These techniques are unique in that they simultaneously model both function and interconnect using cells. These cells are designed such that cell adjacency normally implies interconnection. The absence of an interconnection is explicitly modeled as a wire break between adjacent cells. These methodologies have the potential to greatly simplify and shorten the design process since some design steps are either eliminated or merged with others. They permit near custom layout density while reducing design time over full custom design by up to thirty times and over gate array design by up to four times.

Introduction

Integrated circuit design is a complex and time consuming task. Originally, no CAD tools were available to support full custom integrated circuit design. Layout was done by manually cutting patterns out of colored ruby-lith sheets that were then photographically reduced to produce the actual masks used in fabricating the integrated circuit. Circuit designs were simple, so design engineers could verify their function simply by manual simulation and circuit extraction.

Integrated circuits soon became far too complex for such labor intensive techniques. Indeed, it became impossible to manage the complexity of the design process without CAD tools. Computer graphics techniques were applied to the problem of integrated circuit layout. Mask data stored in the computer was placed automatically on small glass masks that were only a few times larger than actual devices themselves would be. Logic simulation of gate level circuits and circuit simulation of transistor level circuits permitted designs to be evaluated before actual fabrication, thereby eliminating many costly fabrication cycles before the circuit functioned properly.

Tools were developed to help ensure the correctness of the circuit being designed. Design rules checking helps assure that the fabrication process will not introduce any faults into the circuit (other than material defects). Electrical rules checking helps assure that the circuit design has no fatal electrical flaws. Layout versus schematic checking helps assure that the layout does indeed match the logic design. These checks require extensive use of sophisticated high speed computers and often generate a great deal of error data that must be examined manually by designers in order to fix design problems.

Although these CAD tools significantly shortened the integrated circuit development cycle, they did not structurally alter it. The development of the gate array and standard cell design approaches resulted in major advances in the design process by removing the designer from the detailed world of integrated circuit layout. CAD tools for automatic placement and routing were developed to shorten the design cycles for these approaches. The use of standard cells or gate array metallization patterns coupled with automatic routing tools resulted in a ten-fold reduction in design time. The penalty was about a factor of four in circuit area and loss of control over circuit performance because the automatic place and route programs could not be driven by timing constraints. Circuit performance could not even be reasonably estimated before the entire circuit design was finished since the capacitance of the interconnect between gates (or cells) could not be known until after the routing was completed.

† This work was supported by DARPA under contract number DAAK-84-C-9701.

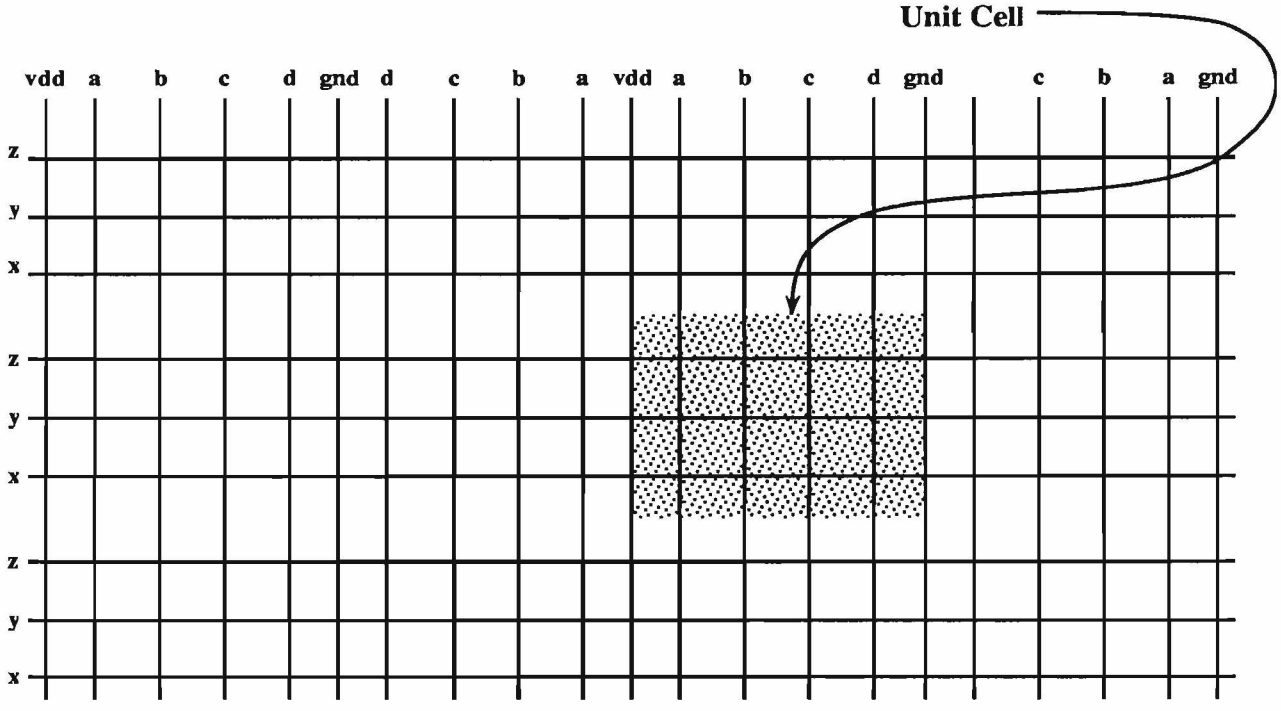

Figure 1: The Cell Matrix "Sea-of-Wires"

Using full-custom design techniques, gate arrays and standard cells, silicon compilers were developed in an attempt to hide even more implementation details from the integrated circuit designer. Integrated circuit specifications for silicon compilers are often in the form of a set of interconnected modules, moving the designer completely away from gates and transistors. Although silicon compilation promises to revolutionize integrated circuit design, it has met with only limited success. Compilations done with gate arrays and standard cells are often too area inefficient while compilations done with full custom techniques involve a great deal of complexity and rarely, if ever, produce truly satisfactory results. In addition, silicon compilers are often technology specific (e.g., CMOS). The techniques used for one technology are not necessarily applicable to other technologies.

As good, useful and successful as all these design methodologies are, there is a better way to design integrated circuits. A class of integrated circuit design techniques known as the *Cell Matrix* methodologies has significant advantages over these other design techniques. For example, circuits can be designed using one of these Cell Matrix techniques (known as Path-Programmable Logic [1, 2, 3]) in approximately one third the time required for gate array design while maintaining excellent transistor densities that are generally within 30% of the densities achievable with full custom techniques. In fact, Path-Programmable Logic (or PPL) can be implemented as a sea-of-gates gate array whose usable transistor density is within a factor of two of full custom density and is twice as dense as other methods for customizing gate arrays. In the following sections, we will explore the fundamentals of Cell Matrix methodologies.

Wiring is the Most Important

As was discovered in the gate array and standard cell approaches to design, interconnecting gates and transistors is the most important aspect of integrated circuit design. Often, the wiring area of an integrated circuit will be in excess of 50%. Area optimizations done on small portions of the chip can significantly increase total routing area. Often the ordering and placement of ports around a circuit module have more impact on overall circuit size than does the size of the circuit module because of simplifications in inter-module routing.

Regular gate array and standard cell approaches provide rows of gates or cells and wiring channels running between these rows. Place and route programs assign gates or cells to locations in these rows and then interconnect them by adding wires in the wiring channels. Sea-of-gates gate arrays cover the entire substrate with a pattern of transistors that can be interconnected by wires. Despite a promise of increased circuit density due to increased maximum gate count, sea-of-gates gate arrays actually achieve no significant improvement in circuit density over standard gate arrays.

If wiring density is at least as important as transistor density (as it nearly always is), then we should consider an approach that gives wiring a chance to be as dense as possible. If wiring takes up more than 50% of the chip area, then reducing the size of a circuit module by that last micron at the expense of wiring can easily increase the overall size and cost of an integrated circuit.

The sea-of-gates approach has a dual --- the sea-of-wires. Instead of thinking of an integrated circuit as being initially covered with a sea of transistors or gates (spread out at predefined intervals) that can be interconnected with wires on N layers, we think of it being covered with up to N sets of wires under which we will place transistors or gates to add function. As seen in figure 1, when N is 2, there are two sets of (orthogonal) wires, one running horizontally (the row wires) and one running vertically (the column wires). These wires are spaced so that a reasonable number of contacts between wires and transistors can be made in an area of a given size. Subsets of the wires in each direction are collected into an area known as a *unit cell.* The size of the unit cell is dictated by two considerations, the number of wires that must pass through in each direction and constraints dictated by how many transistors must be present in the simplest functional unit cells. Sometimes, layout considerations like well boundaries in CMOS will also affect the size and shape of the unit cell.

Some of the wires in at least one direction are generally used to carry power and ground to each unit cell. Although this is not strictly required, power and ground are usually included in each unit cell so that no explicit power and ground routing need be done by the designer. Power and ground wires may or may not be shared between cells in

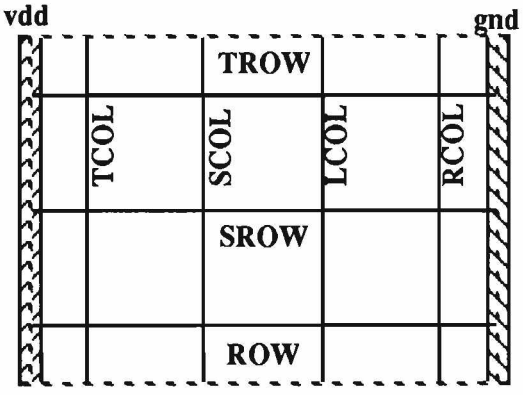

Figure 2: A Rectangular Unit-Cell

adjacent columns and/or rows. If they are shared (as is generally the case since doing so reduces the area overhead for power and ground routing), then alternate columns and/or rows must be mirrored so that the power and ground busses will be correctly completed.

In situations where power distribution problems are especially severe, it is possible to construct a power mesh by running power and ground on two layers of interconnect and interconnecting the two at regular intervals (i.e. in each unit cell) to construct a mesh.

The unit cell, defined by (orthogonal for N=2) subsets of wires, becomes the unit of measurement in designing the integrated circuit. Conceptually, the surface of the integrated circuit has been partitioned into a grid of rectangular unit cell locations, with a predetermined number of wires crossing each edge of the unit cell. In Cell Matrix parlance, these unit-cells that contain only wires are referred to as *blanks*. In the specific case we describe here, power and ground are available in each unit-cell on

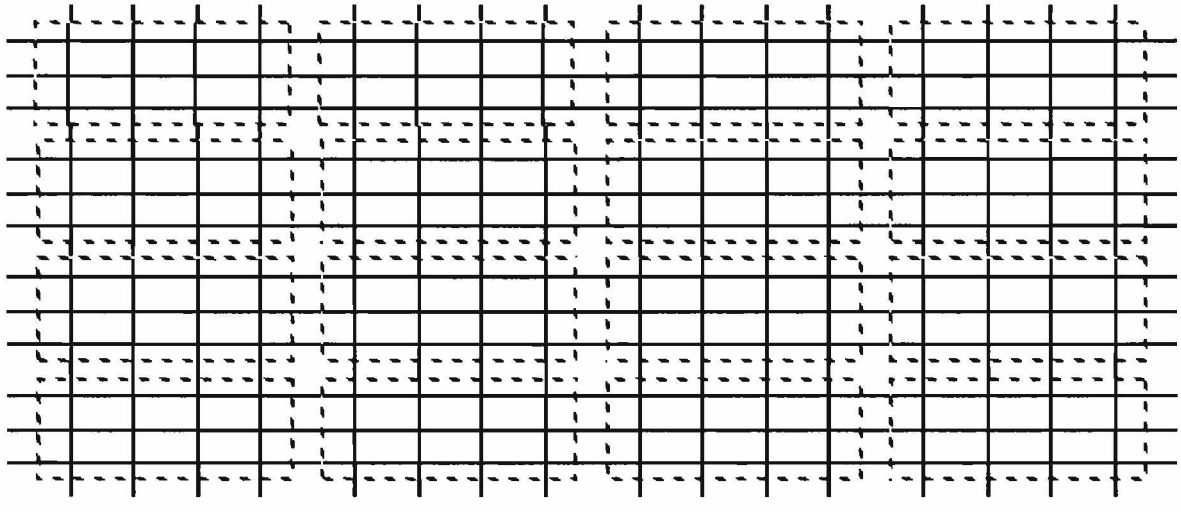

Figure 3: The Initial Cell Matrix Design Plane

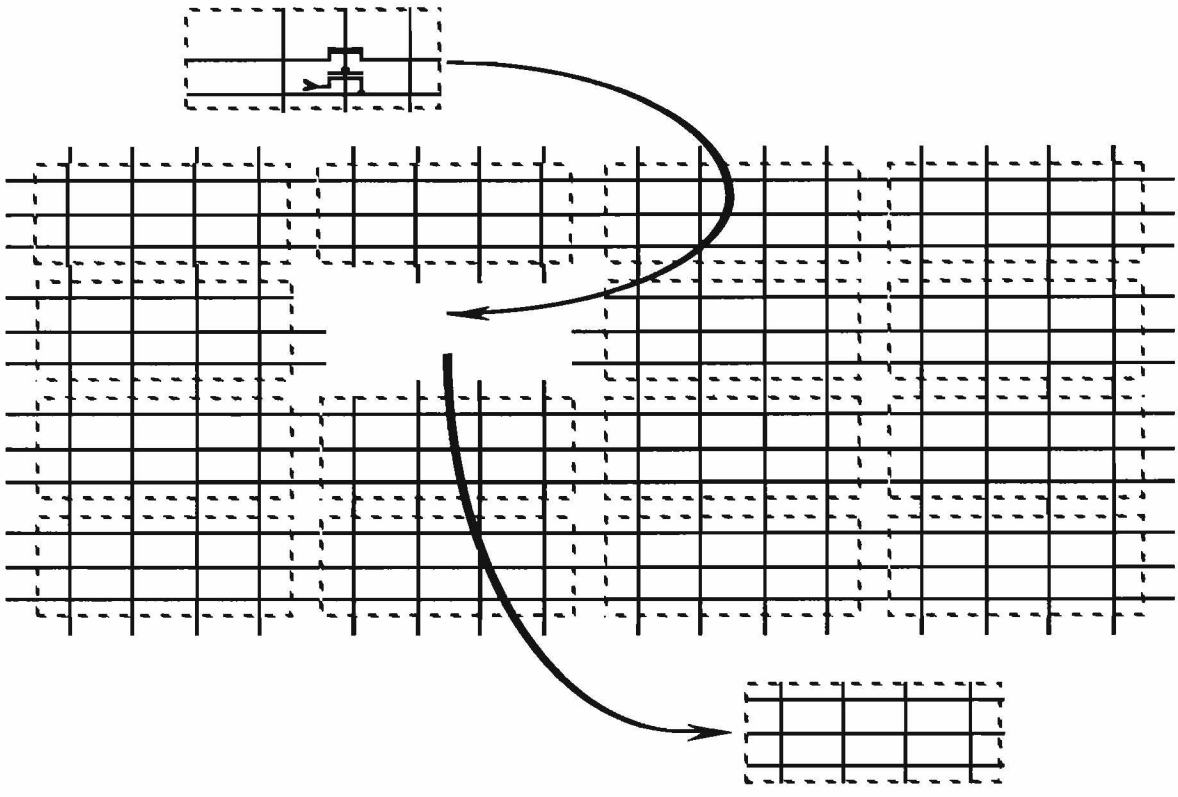

Figure 4: How Designs are Constructed in the Plane

its vertical edges (see figure 2). When power and ground routing are automatically completed by placing cells in the design plane, we may hide any representation of them from the designer as shown in figure 3.

Theoretically, if more than two independent layers of interconnect are available, wires could run in more than two directions. For example, if three layers of wire were available, they could run in three different directions, meeting at 60 degree angles. In this case, the unit cell would be hexagonal. If four levels were available, they could run horizontally, vertically, at 45 and at 135 degrees. The unit cell could not be an octagon however since octagons do not tessellate the design plane. Practically speaking, the rectangular unit cell is the only one that is truly useful today since economical use of mask making equipment either eliminates the use of non–manhattan geometries or restricts non-orthogonal angles to be multiples of 45 degrees. Most of the circuit structures we rely on for building complex systems in integrated circuits map perfectly well onto a rectangular system. There are some applications, however, that would map better onto a hexagonal system. Among these is the array multiplier in which the bits of the multiplier, multiplicand and the carries are distributed in different directions. In all probability, the rectangular unit-cell will remain the only useful and economical one since the complexity and cost of almost all aspects of integrated circuit design and production would increase if any other shape were used.

Cells Large and Small

With the design plane initially tessellated using only *blank* cells as in figure 3, the circuit is then designed by conceptually replacing blank cells with cells that add function (e.g., collections of interconnected transistors, resistors and capacitors) or interconnect. In figure 4, a cell containing two transistors replaces a single blank unit cell in the design plane. Cells form the heart of Cell Matrix methodologies. Every circuit structure, including interconnect, is embodied as a cell. Unit cells cover exactly one grid location. A cell has ports corresponding to the points where signal wires (those not carrying power and ground) cross cell borders. A cell may contain any collection of connected transistors or wires as long as power and ground are uninterrupted and as long as all wires that must communicate with other (possibly adjacent) cells use the port locations defined by the wires in the design plane.

In some cases, like Structured Arithmetic Tiling [4, 5], each cell is a self-contained functional module --- all the transistors required to implement the

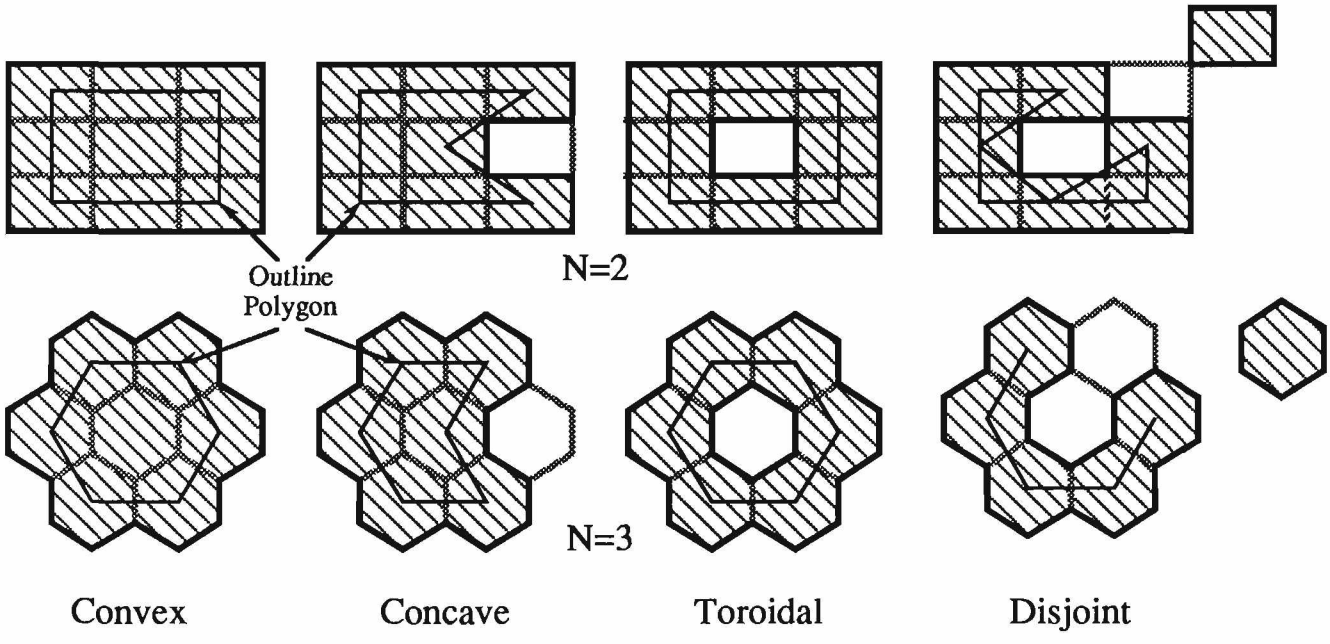

Figure 5: Classes of Cell Matrix Cell Shapes

function are included in a single cell and all ports represent either inputs or outputs of the module. In others, like PPL and its predecessor, the Storage/ Logic Array (or SLA) [6, 7, 8,9,10], many cells are practically useless by themselves and logic gates are often *constructed* by using collections of cells to form a distributed gate. Cells may cover more than one grid location; these are referred to as non-unit cells. In PPL and the SLA, all cells are currently rectangular, including all non-unit-cells. This is not required by the design methodology but is rather a result of a narrower view of Cell Matrix based integrated circuit design. In the NMOS implementation of Structured Arithmetic Tiling [4] many non-unit cells were, for the first time, concave (that is, their outlines were not rectangular). Generalizing, non-unit-cells may come in several classes of shapes: convex, concave, toroidal and disjoint. These classes of cell shapes are shown in figure 5 (rectangular for N=2 and hexagonal for N=3). Convex cells are those covering one or two unit cells and those that cover more than two unit cells where a convex *outline polygon* joins the centers of the outer-most unit-cell locations covered by the non-unit-cell. Concave cells are those that cover more than two unit cells and where the outline polygon is concave. Toroidal cells are those in which at least one unit-cell location within the non-unit-cell is not covered and is completely surrounded by covered unit-cell locations. In this case, two polygons are required to define the shape, one for the external outline and one for the internal outline. A disjoint cell is one that has two or more disjoint sections, each of which may be convex, concave or toroidal.

Although non-convex leaf cells (cells not containing other cells) are seldom required (even the CMOS implementation of Structured Arithmetic Tiling has all rectangular cells), the concept of non-convex cells is very important when one begins to consider hierarchical design. Hierarchical modules seldom have rectangular outline polygons. (We avoid using the term *bounding* polygon since it implies that it contains all of the *layout* whereas the outline polygon joins the centers of the outermost layer of cells and does not enclose all of the layout). There will almost always be a large, unused section within the bounding rectangle of a hierarchical module that might possibly be used to advantage in a neighboring module. In order to permit the best possible use of area, using non-convex cells is required so that unused cell locations within the bounding box of a hierarchical module can be excluded and possibly used either in another module or for interconnect.

One distinct advantage of allowing leaf-cells to be of all four shape classes is that the same mechanisms used to design with leaf-cells can also be used to design with hierarchical cells. Circuit designers and computer-aided design tool developers both benefit because the same user interaction package is used for both leaf level design and hierarchical design. All that is required is that the cell/ design database be designed to accommodate hierarchical cells as well as leaf cells.

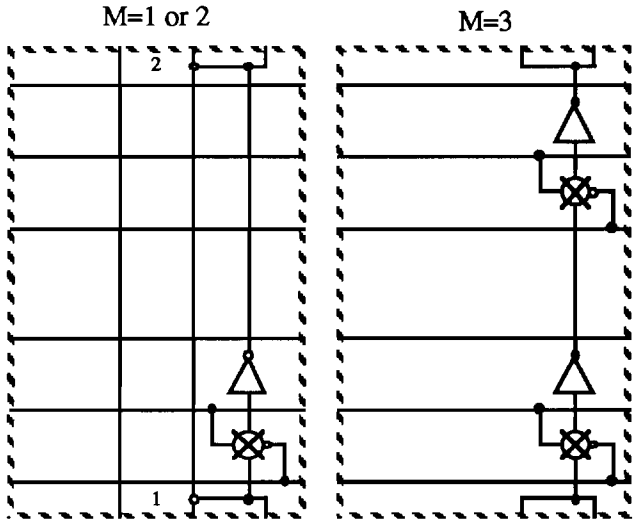

Figure 6: Cell Modifier Types

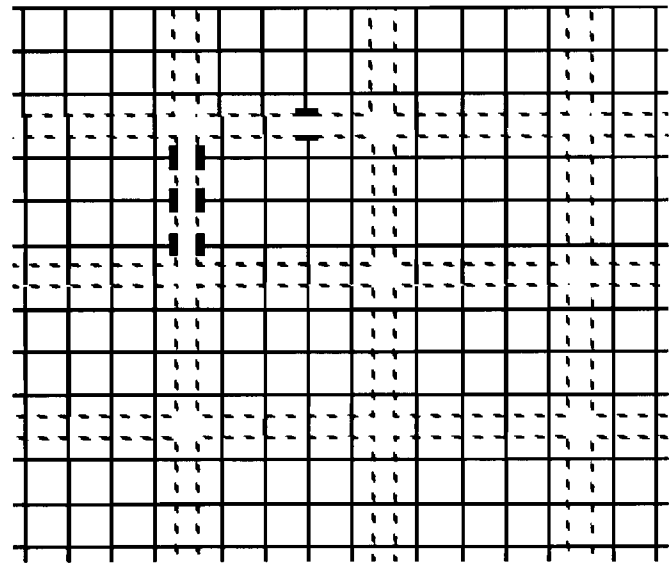

Figure 7: Breaking Wires

Cells With Modifiers

In Cell Matrix methodologies, cells may have *modifiers* that either change the way a cell hooks up to its ports or change its functionality or both. When a modifier is present, it is said to be *asserted*. In the simplest case, modifiers will affect only a cell's ports. In figure 6, we see a cell with three modifiers. Modifier 1 hooks the input of the cell to the next-to-rightmost column wire and modifier 2 hooks the output of the cell to this same wire. In this case, these two modifiers need to be *mutually exclusive*, both of them should not be simultaneously asserted since the result would be a non-functioning circuit. If two or more modifiers can be simultaneously asserted, they are said to be *additive*. This same cell has a third modifier that radically alters its functionality while maintaining its size and aspect ratio.

In the general case, a combination of two or more modifiers being asserted may preclude the use of another. To determine if a modifier may be legally asserted it may be necessary to evaluate a complex function over the assertion status of many modifiers. The mutual exclusivity and additivity of cell modifiers can arise from layout constraints where asserting one modifier makes it impossible to assert another without violating layout design rules. They can also arise from electrical constraints where asserting two modifiers would render a circuit non-functional.

It is best if the ways in which cells can be modified are consistently applied in all of the cells in a cell set to reduce the complexity of the cell set in the mind of the designer. Modifiers are a significant feature of Cell Matrix methodologies in that they permit multiple physical cells to be viewed as small modifications made to some base cell. This feature aids the designer since there are fewer cells to remember.

Breaking Wires

In Cell Matrix methodologies signal wires leave (or enter) a cell at predefined port locations. Initially, the column wires are as long as the circuit is tall and the row wires are as long as the circuit is wide. In other words, wires span the entire width or height of the circuit being designed.

When a cell is placed in the design plane, it is automatically connected to its neighbors on all sides at all port locations. If a connection to a port of a neighboring cell is not desired, then a small piece of wire connecting the two adjacent ports must be removed. The designer performs this task symbolically by indicating that a wire is to be broken. Conceptually, breaking a wire indicates that a port of a cell does not connect to the outside world.

Physically, the signal wires (both row and column) of a cell do not extend all the way to the edge of the cell. Therefore, when two cells are placed adjacent to each other, there are in reality no connections between them. CAD tools must add connections, or small rectangles of metal (or some other material) at the port location, centered on the coincident edges of both cells. In reality, the designer indicates to the CAD tools where connections are not desired, achieving in effect a break in a wire. In general, there are far more connections than breaks; thus, it is easier to indicate only the ends of a wire by inserting two breaks than to trace the path of a wire by inserting many connections.

Wires may therefore be broken between any two adjacent cells (or hierarchical cells). In some cells, a specific port may not be used and the wire intersecting the cell at the location of the port is permanently broken, indicating that it can never be connected to the outside world.

The concept of breaking wires at cell boundaries permits the design plane to be arbitrarily segmented. This enables modules to be easily embedded in the plane and sea-of-wires. It allows a single design structure, the plane, to contain many independent modules that communicate by any number of signals. In the case of PPL and the SLA, the constructed AND and OR gates need extend only as far as necessary to pick up all the required inputs and to drive the required outputs. With PPL and the SLA, logic and layout equivalent to arbitrarily segmented (or folded) PLAs can be easily implemented. Indeed, multiple independent logic modules can be simply embedded in a single PPL or SLA circuit. The same holds true for other Cell Matrix design methodologies, segmentability permits many independent modules to be implemented in the same physical structure.

Useful Symbologies

One of the primary advantages of using Cell Matrix methodologies for integrated circuit design is that there are multiple useful symbologies that can be used to represent each cell in a circuit. Since the design plane is entirely tessellated with cells, there are no areas in which the layout of the circuit cannot be represented abstractly through the use of symbols. Even the interconnect between wiring layers can be represented symbolically as it is achieved by using cells or cells with modifiers.

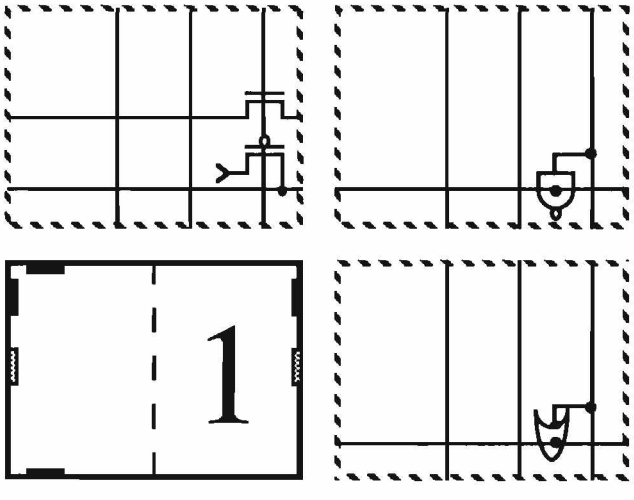

Figure 8: Some Possible Symbologies

A variety of symbols can represent a given cell. The examples we see in this section are taken from Path-Programmable Logic. Figure 8 shows four possible symbolic representations for one cell. Clockwise from the upper left hand corner there is the transistor schematic, a logic schematic with the row wires implementing a NAND gate, a logic schematic with the row wires implementing a NOR gate with true low inputs and finally, the symbol "1" indicating that this cell senses the fact that the column contains a logic 1. The exact function of this cell is explained elsewhere [3] and this figure serves only to illustrate the kinds of symbols that can be used to represent the same physical cell.

The symbol "1" is most effectively used when building modules containing random logic such as a state-machine. In circuits like clock generators whose specification is best viewed as logic gates, either of the logic gate representations would be preferred. Only rarely would the transistor representation be useful, and layout is practically useless from a design standpoint. In fact, it is possible to microprobe Cell Matrix circuits without a composite layout. There are so few different cells that it is possible to recognize them under a microscope and microprobe directly from a symbolic or schematic representation. This would be impossible if not everything were done as a cell.

Cell Matrix Design Rules

In developing a Cell Matrix design methodology, there are only a few rules to follow as well as some guidelines. Most of these have to do with making sure that wires can be broken, that cells abut correctly and that layout design rules are never violated when cells are placed adjacent to one another. First the rules:

1. A cell must cover entire grid locations in the design plane.

2. There must be at least one column wire and one row signal wire in a unit cell.

3. Signal wires must cross cell edges at prespecified locations only.

4. Signal wires extend only to within one-half of the minimum spacing of layout entities on the layer being used. For instance, if first layer metal is being used and the minimum spacing for first layer metal is three microns, then a signal wire

on first layer metal can only come within one and one-half microns of the cell edge.

5. Signal wire connections (which are removed by specifying breaks in the design) must be sized so that they completely fill the gap in a signal wire left between adjacent cells by adhering to rule 4.

6. If physical structures (e.g., power and ground wires) are shared on vertical (horizontal) edges of cells then every other column (row) of cells must be mirrored in X (Y). This imposes a placement restriction on cells that are two or more columns wide (rows high) since sharing of a physical structure would be incorrect if it were placed in the wrong column (row).

Guidelines:

1. Cells should be designed to permit as much wiring flexibility as possible.

2. When possible, all unused signal wires should pass through the cell.

3. Critical physical structures (e.g., power and ground wires) should be carefully considered to see if they can be shared between adjacent columns or rows of cells. For example, in CMOS, wells are shared so that the unusable area around a well boundary is minimized.

4. Cell modifiers should be provided whenever possible because they simplify a cell set considerably and increase its flexibility.

5. Consistency in the way cell modifiers affect all cells in a cell set is desirable since it simplifies the designer's task.

These few simple rules and guidelines permit a wide variety of Cell Matrix methodologies that have great flexibility but can be easily tailored to the design task at hand. Path-Programmable Logic is tailored to the development of random and control logic as well as simple register transfer datapaths. Structured Arithmetic Tiling is tailored to the development of sophisticated arithmetic circuits. We do not believe that one methodology is the answer to all design problems, but we do believe that the design techniques underlying them can be effectively applied to all design problems.

Cell Matrix Design Benefits

There are many substantial benefits of Cell Matrix methodologies in the design of integrated circuits. First and foremost, the designer simultaneously designs both logic and layout. Since each cell has a one-to-one correspondence with a cell layout and all cells cover whole grid locations in the design plane, the symbolic level design of a circuit also shows the size and aspect ratio of the circuit or module being designed. The designer therefore has more control over how well modules fit together than with other methodologies, with the exception of full-custom.

Second, since all design is done with cells and since all cells can be completely characterized with respect to resistance, capacitance and function, speed can be easily estimated at design time. In fact, it is possible to develop a design editor that could indicate to the designer when some predetermined timing constraint had just been violated.

Third, using a character based symbology for cells, design can be done using inexpensive personal computers or even alphanumeric terminals attached to a mainframe or workstation. Expensive hardware to support graphics is not required. The design editor for PPL, known as `tiler`, is completely character based and can use as its display device any inexpensive alphanumeric terminal that has cursor control. Many of the most complex tasks normally performed by high-performance mainframe computers, such as design rules checking or gate placement and routing, are altogether eliminated in Cell Matrix circuit design once the cell library is in place. Some tasks, such as circuit extraction, that are traditionally very time consuming are significantly simplified because of the additional and complete structure imposed in Cell Matrix methodologies.

Fourth, Cell Matrix design gives the designer more flexibility than most other methodologies while limiting the degrees of design freedom to a manageable level.The primary advantage here is derived from the fact that routing and interconnect are both part of Cell Matrix design. The next most important advantage is that cells can be any size and/or shape and contain any collection of transistors or other devices such as capacitors and resistors. This easily permits new design primitives to be added to the leaf cell library in as small an area as possible.

Fifth, design times are drastically reduced over other known methodologies while transistor densities are close to those achieved with full-custom design. The primary basis for this observation is a pair of studies that compared the design times and

densities of circuits designed using PPL with those achievable with other methodologies. The first of these studies [11] was done by a student at the University of Utah who was also an employee of Evans and Sutherland of Salt Lake City. Two full-custom circuits previously designed at Evans and Sutherland using some silicon compilation techniques and some hand-packed layout were re-implemented using PPL. These two circuits were a shaft encoder and a sorted access memory chip.

The second study was done by an independent integrated circuit design company, Rentek. The designers at Rentek have tens of man-years of experience in full-custom design, gate array design and standard cell design of integrated circuits. They were implementing several circuits using both gate arrays and PPL. They compared PPL with these other methodologies based on their own experience and on the quoted densities from several major suppliers of gate arrays and standard cell circuits.

In the E&S comparison, which directly compared full-custom design to PPL design as well as making some assertions about gate-array designs, design times were reduced by a factor of about 10 to 15 and area penalties of PPL vs. full-custom were 38% and -4% respectively on the two circuits. Based on a 4 micron NMOS fabrication process, the results of this study are included below.

E&S Custom to PPL Comparison

Circuit	Time (man-hours)	Density (μ^2/Transistor)
SE Cust.	400	1,263
SE PPL	28	1,777
SAM Cust.	340	2,312
SAM PPL	30	2,210

In the Rentek study, which compared gate array, standard-cell and full custom design techniques to PPL, it was determined that PPL circuits could be designed 27 times faster than full-custom circuits, and 3 to 4 times faster than gate array or standard cell circuits. Area penalties for PPL vs. full-custom were on the order of 30% while PPL was 3 to 4 times as dense as gate array circuits and standard cell circuits. The results of the Rentek study are included below. All design times in Table 2 are in terms of days per 100-gates and all transistor densities are in terms of square microns per transistor. All of the density figures are based on a one and one-half micron CMOS process. The density figures for PPL are empirical while the others are vendor quotes.

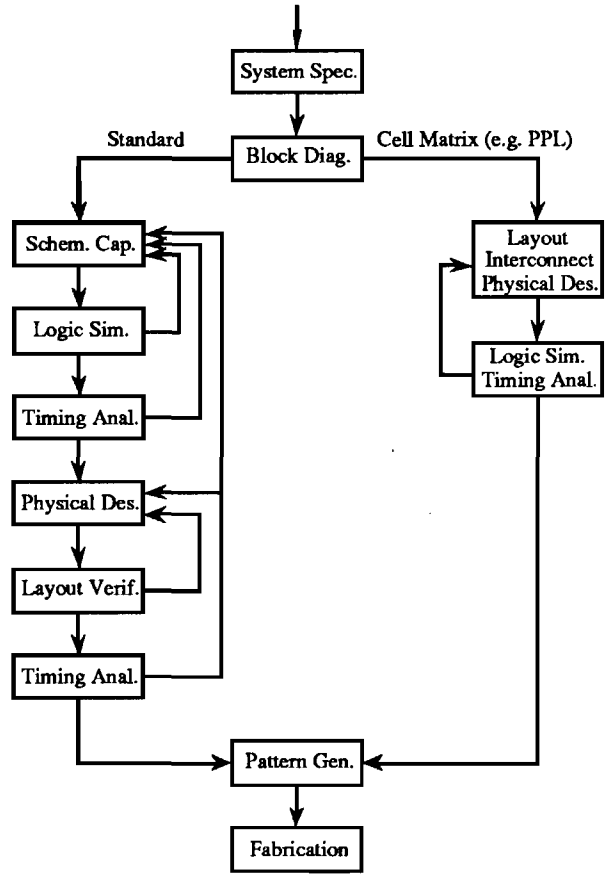

Figure 9: Cell Matrix vs Standard Cell

Rentek PPL Comparison

Methodology	Time (days/100gates)	Density (μ^2/Transistor)
Sea-of-Gates	2.2	1,387.1
Gate Array	2.2	1,271.0
Std. Cell	2.2	851.6
PPL	0.7	419.4
Full Custom	19.0	322.6

Although these two unrelated studies are based on different fabrication processes, there are two things that are clear: PPL circuits can be designed more rapidly than circuits implemented using other techniques, and PPL circuits compare favorably to full-custom circuits with respect to circuit density. Whether or not these comparisons of PPL to other design methodologies can be generally applied to other Cell Matrix design methodologies is questionable. However, the design of a complex arithmetic processor in PPL was compared to a design of the same processor using SAT, another Cell Matrix methodology specifically designed for implementing arithmetic circuits. In this case, the SAT circuit was four times as dense as the PPL circuit [5]. Although the results in this case are inconclu-

sive because the PPL circuit could easily be optimized for area, they do substantiate our notion that Cell Matrix methodologies can achieve near full-custom densities. An SAT circuit module of 15,264 transistors has a density (at 1.5 micron design rules) of 278.5 square microns per transistor, 14% more dense than the Rentek estimate of full-custom density (322.6 square microns per transistor). In yet a third comparison now underway, PPL is being compared against a full-custom layout generated by a silicon compiler and a standard cell design. The circuit being developed is a Hogenauer [12] filter containing nearly 20,000 transistors. The silicon compiler generated circuit has a density of approximately 4,100 square microns per transistor including pads. The standard cell circuit has a density of 3,248 square microns per transistor including pads. The PPL circuit has a density of 952 square microns per transistor including pads. All three circuits were implemented using a 1.25 micron CMOS process. Again, the PPL circuit is four times as dense as circuits produced using these other methods. The PPL circuit was designed directly from the specification as it existed in [13] in one day (the circuit is highly regular). Literally months were spent designing this circuit using both the silicon compiler and standard cells although exact design times are not known.

Why is Cell Matrix design faster than other methodologies? Since logic and layout are done simultaneously, many steps in the normal design cycle are eliminated. Figure 9 shows a graph comparing the design methodology steps for standard cell design and PPL circuit design. In addition, the use of carefully chosen symbologies assist the designer in designing from a much higher level of specification than would otherwise be possible, permitting many design steps to be skipped. In fact, many PPL circuits are designed directly from block diagrams, hence detailed logic diagrams at the gate level are never developed or drawn. In particular, state machines are easily implemented directly from state flow graphs, so detailed logic design is not necessary. The abstraction mechanisms of Cell Matrix methodologies permit the designer to concentrate more fully on problems of system architecture rather than on layout details while giving direct feedback about the impact of the architectural design on area, aspect ratio and speed.

Why is circuit density of Cell Matrix circuits so much better than gate arrays or standard cells and so close to the density achievable with hand-packed full-custom layout? This question is harder to answer. The excellent density of Cell Matrix circuits is at least partly due to careful consideration of wiring before the circuit is actually designed. A considerable amount of the area normally used for routing only can serve the dual purposes of achieving function and providing interconnect.

Also a contributing factor is that design can be done directly from a block level design. This permits structure inherent in the solution to the problem to be kept throughout the design cycle. In other methodologies such as gate arrays or standard cells, any global structure is lost when the netlist of interconnected simple gates is generated. The locality of communication inherent in much of the circuitry cannot be maintained in the gate array or standard cell approaches while it can be in Cell Matrix approaches. The segmentability of the Cell Matrix design plane is also one of the reasons for its area effectiveness. Since modules can be designed to "fit" each other, the designer can frequently eliminate intermodule routing by making module ports align in the design plane.

Given the above arguments, it is not too surprising that Cell Matrix design approaches achieve higher densities than gate arrays or standard cells. What is surprising is that they can achieve densities so close to full-custom densities. Indeed, at times the densities of Cell Matrix circuits are better than those generally achievable with full-custom techniques. The only possible explanation for this is that design time is critical and full-custom layout designers just do not have enough time to find a better, more compact layout.

The benefits of Cell Matrix methodologies for integrated circuit design are significant, particularly design time reduction and high circuit density.

Conclusion

Cell Matrix approaches to integrated circuit design have many advantages over more conventional techniques. Perhaps the most significant disadvantage is that additional education and training is required to understand Cell Matrix design techniques since they are so different from those that are widely used today. They require tools that are not like other VLSI design tools. Tools developed for other methodologies are not applicable to Cell Matrix design with the obvious exception of simulators.

Cell Matrix methodologies are exciting alternatives to the design techniques now generally in use with respect to design time, circuit area efficiency and cost of design hardware and tools. In general, Cell Matrix cell sets are simple enough that they can be reimplemented in a new technology in less than three months, making it possible to move circuit designs between fabrication processes with relative ease and speed. We believe these Cell Matrix design techniques represent a quantum leap in integrated circuit design technology.

References

[1] T. M. Carter, "Path Programmable Logic and the Use of CADDS2/VLSI", *Proceedings of the Fourth Annual International Computervision User Conference,* Computervision Corp., 14-17 Sep. 1982, pp. 523-528.

[2] K. F. Smith, B. E. Nelson, T. M. Carter and A. B. Hayes, "Computer-Aided Design of Integrated Circuits Using Path-Programmable Logic", *IEEE Electro '83 Professional Program Session Record,* Apr. 1983.

[3] B. E. Nelson, D. R. Morrell, C. J. Read and K. F. Smith, "The PPL Integrated Circuit Design Methodology", *Computer Aided Design,* Nov. 1986.

[4] Tony M. Carter, *Structured Arithmetic Tiling of Integrated Circuits,* Ph.D. Dissertation, University of Utah, Department of Computer Science, Dec. 1983.

[5] Tony M. Carter, "Structured Arithmetic Tiling", *Proceedings of the Eighth Symposium on Computer Arithmetic,* May 1987, Como, Italy, pp. 41-48.

[6] S. S. Patil and T. A. Welch, "A Programmable Logic Approach for VLSI", *IEEE Transactions on Computers,* Vol. C-28, No. 9, Sep. 1979, pp. 594-601.

[7] S. S. Patil, "An Asynchronous Logic Array", Project MAC Technical Memo TM-62, MIT, May 1975.

[8] K. F. Smith, *Design of Integrated Circuits with Structured Logic Using the Storage/Logic Array (SLA): Definition and Implementation,* Ph.D. Dissertation, Univ. of Utah, Dept. of Electrical Engineering, Mar. 1982.

[9] Kent F. Smith, Tony M. Carter and Charles E. Hunt, "The Storage/Logic Array", *IEEE Trans. on Electron Devices,* Vol. TED-3 No. 5.

[10] K. F. Smith, T. M. Carter and C. E. Hunt, "The CMOS SLA and SLA Program Structures", *Proceedings of the 1981 CMU Conference on VLSI Systems and Computations,* Carnegie-Mellon Univ., Dept. of Computer Science, published by Computer Science Press, Oct. 1981, pp. 396-407.

[11] P.D. Israelsen and K. F. Smith, "Comparison of the Path Programmable Logic Design Methodology with Other Custom and Semi-custom Approaches",*Proceedings of ICCD 1985,* Oct. 1985, pp. 73-76.

[12] E. B. Hogenauer, "A Class of Digital Filters for Decimation and Interpolation", *Proc. IEEE Int'l Conf on Acoustics, Speech and Signal Processing,* Apr. 1980, pp. 271-274.