VLSI CIRCUIT LAYOUT

Very-large-scale integrated (VLSI) circuits have been recognized as an important area of electrical and computer engineering. With the accelerating complexity of semiconductors shown in Fig. 1 (1), VLSI must address the needs of consumer products, personal computers, workstations, midrange computers, mainframes, and supercomputers.

Among the stages of VLSI design, such as architectural design, functional design, logic design, and circuit design, the physical (or layout) design stage is the back-end process of determining the physical location of circuit components from the circuit net list and interconnecting them inside the VLSI chip by interconnecting a collection of transistors.

Physical design is an art based on the science of establishing interconnections and fulfilling system functions by placing modules in a chip. Continuous advances in the speed and scale of integrated circuits have created ever greater demands for higher density packaging to ensure reduced interconnective delays for improved electrical performance.

In the physical design stage, the circuit representation of each component is converted into a geometric representation. This representation is a set of geometric patterns that perform the intended logical function of the corresponding component. Connections between different components are also expressed as geometric patterns. The geometric representation of a circuit is called a layout. The exact details of the layout also depend on design rules, which are guidelines based on the limitations of the fabrication process and the electrical properties of the fabrication materials. Physical design is a very complex process, and thus it is generally partitioned into smaller subproblems and solved in a hierarchical fashion. At the highest level of hierarchy, for example, in multichip module (MCM) or printed circuit board (PCB) designs, a set of integrated circuits (ICs) are interconnected. Within each IC, a set of modules, such as memory units, arithmetic logic units (ALUs), input-output ports, and random logic are arranged and interconnected. Each module consists of a set of gates.

Because the cost of fabricating a circuit is a function of the circuit area, circuit layout techniques aim to produce layouts with a small area. Also, a smaller area implies fewer defects and interconnects, hence a higher yield and higher performance.

Conventional physical design is becoming a limiting factor in translating semiconductor speed into system performance. In high-end systems such as supercomputers, mainframes, and medical and military electronics, more than 50% of the total system delay usually results from interconnection, and by the year 2000, the share of interconnection delay is expected to rise to 80% (2). In the medical electronics industry, speed and reliability are the main objectives. Moreover, increasing circuit count and density in circuits continue to place demands on high-level physical design.

Physical design in the timing and performance-driven environment differs from classical VLSI design in a number of important ways. Performance is of overriding importance in

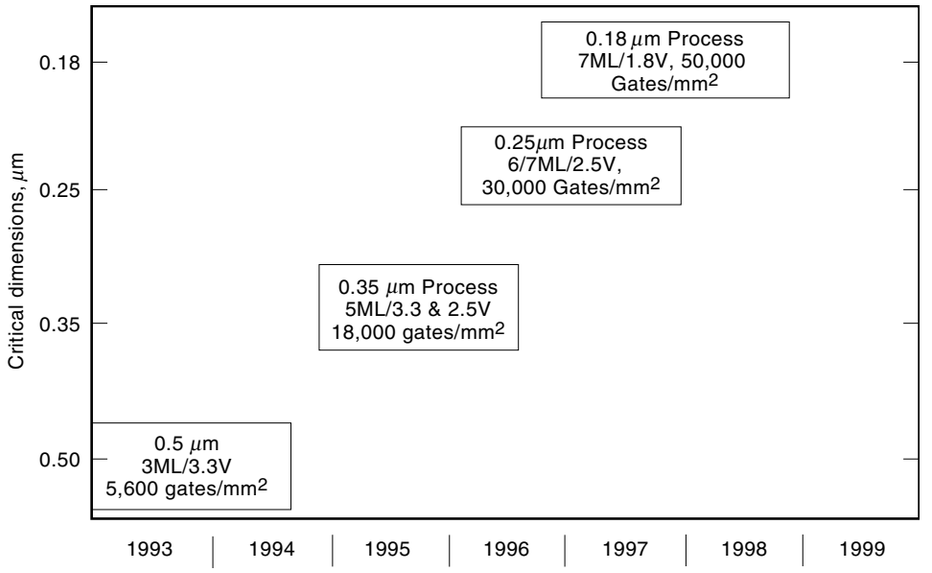


Figure 1. Trends in semiconductor device integration.

"aggressive" design. The need for a systematic approach to physical design and analysis is paramount. This approach must link high-performance tools with timing and integrity parameters. Engineers must be able to exploit these linkages to make early design trade-offs based on such parameters. Thus, physical design algorithms must be driven by performance constraints. This involves careful consideration to minimize signal integrity effects, such as cross talk, reflections, and the effects of crossings, bends, and vias (3–11). The number of layers required continually increases, with a three-dimensional flavor, which is lacking in existing VLSI routing where the number of layers rarely exceeds three.

Because of rapid development in VLSI technology, the average transistor count in a chip has increased enormously. The minimization of power consumption in modern circuits, therefore, is of great importance. In particular, battery operated products, such as portable computers, cellular phones, etc., have come to a point in which minimization of power consumption is among the most crucial issues. On the other hand, it has been shown (12) that circuit activity (which is closely related to power consumption) is an acceptable measure of failure in digital circuits. Transitional density or average switching rate at different sites in a circuit is introduced in Ref. 13 as a model to measure the circuit activity, which can be used to estimate the power consumption in the circuit. Because of the importance of power consumption, there has been a great shift of attention in the logic and layout synthesis from delay and area minimization toward this issue (14–17).

Today's electronics industry requires new automated design tools and design methodologies that allow designers to concurrently design high-performance integrated circuits and high-performance packaging. The purpose of this article is to provide the tool developer and VLSI designer with recent automated design techniques and algorithms in VLSI and MCM methodologies. This article focuses on the VLSI physical design methodology and automation and covers most aspects of up-to-date physical design from partitioning, floor planning, and placement to routings, together with such related areas as special structures for clock, power, and cross talk.

This article explores various high-level physical layout problems in designing high-speed, very large integrated circuits in multilayer environments to minimize interconnect delays, bends, vias, cross talk, clock skew, and, finally, power consumption. Continual advances in the speed and integration scale of integrated circuits have created ever greater demands for high density VLSI to ensure reduced interconnection delays for improved electrical performance. Discontinuities must be controlled to keep the resulting reflections to a minimum. Because high-performance circuits are usually designed aggressively (i.e., most of the nets are considered as critical nets), it is preferable to minimize the number of bends and vias. In high-speed clock design, a number of physical constraints may result in a substantial increase in wiring area especially in smaller routing subregions.

The remainder of this article is organized in the following sections: Layout Methodology, Physical Design Flow, Partitioning, Floor Planning, Placement, Routing, Other Issues in Physical Design Flow, and High-Performance Layout Design with Deep Submicron Technologies (Clock-Tree Synthesis, Cross Talk).

For a comprehensive book on VLSI circuit layout, see Ref. 18.

LAYOUT METHODOLOGY

Design styles are broadly classified as full-custom or semicustom. In a full-custom layout, different blocks of a circuit are placed at any location on a silicon wafer as long as all the blocks do not overlap. On the other hand, in semicustom layout, some parts of a circuit are predesigned and placed at some specific point on the silicon wafer. Selection of layout styles depends on many factors including type of chip, cost,

and time to market. Full-custom layout is a preferred style for mass produced chips because the time required to produce a highly optimized layout is justified. On the other hand, to design an application-specific integrated circuit (ASIC), a semicustom layout style is usually preferred.

The term custom means the least restricted design style. The term gate array is used to describe chips which are personalized by metal levels only. The gate array is the most restricted design style. The term standard cell describes chips which are personalized by all masking levels within the constraints of some design system. The standard cell falls into the middle of this restrictivity spectrum. These chips are sometimes called master image and semicustom. Standard cells have a fixed height but different widths, depending on the functionality of the modules. They are laid out in rows, with routing channels or spaces between rows reserved for laying out the interconnects between the chip components. Standard cells are usually designed so the power and ground interconnects run horizontally through the top and bottom of the cells. When the cells are adjacent to each other, these interconnects form a continuous track in each row. The logic inputs and outputs of the module are available at pins or terminals along the top or bottom edge (or both). They are connected by running interconnects or wires through the routing channels. Connections from one row to another are made through vertical wiring channels at the edges of the chip or by using feedthrough cells, which are standard height cells with a few interconnects running through them vertically. Macro blocks are logic modules not in the standard cell format, usually larger than standard cells, placed at any convenient location on the chip. Here, the circuit consists only of primitive logic gates, such as NAND gates, not only predesigned but prefabricated as a rectangular array, with horizontal and vertical routing channels between gates reserved for interconnects. Then the design of a chip is reduced to designing the layout for the interconnects according to the circuit diagram. Likewise, fabrication of a custom chip requires only the making of steps for interconnect layout. The third chip layout style uses only macro blocks. The blocks are irregular in shape and size and do not fit together in regular rows and columns. Once again, space is left around the modules for wiring. For a detailed description of layout styles, see Ref. 19a.

A VLSI chip designer has several approaches to design. Each of these choices offers the designer trade-offs between chip density and chip design time. A semiconductor wafer is coated with a light-sensitive substance called a photoresist. A mask containing patterns that represent circuits and their interconnections is placed over the wafer, and an ultraviolet light shines through the mask. This light exposes the photoresist on the wafer, and with additional processing, the shapes on the mask are transferred to the wafer. The process is similar to photography, where the mask is the negative and the wafer with photoresist is the light-sensitive photographic paper. The chip design problem is to design a mask with shapes that represent all of the circuits on the chip. Therefore, the complexity of a chip design is measured by the number of shapes that must be drawn to represent a complete chip. This number has historically doubled every two years. There are at least three major reasons for this increase in complexity: decreasing minimum dimensions, larger chip sizes, and increasing process complexity in the vertical dimension. Let us look at each one of these in turn. Because of rapid advances in the state of the art of mask and wafer processing, the minimum dimensions of the shapes on the mask have been shrinking. A minimum dimension is the smallest width of a shape, or the smallest spacing between two adjacent shapes. Because the minimum spacings in a typical process are reduced in both the x and y dimensions, the increase in complexity is a square function, that is, a reduction in minimum dimensions by a factor of 2 results in an increase in complexity by a factor of 4. In the early 1970s, minimum features were on the order of 10 microns. In the early to mid 1980s the minimum features dropped to 2 microns and less. In the 1990s the minimum features shrank to less than one micron.

The second driving force for increased chip complexity is the increasing chip size. As the chip size increases, the number of shapes on a chip increases. The maximum size of a chip is controlled by the ability of the maskmaking tools and by the quality (purity) of the chip manufacturing process. Both of these have been improving, and the result is that chip sizes are increasing. This improvement is projected to continue so that chips in the future will be still larger.

The third driving force for the increase in chip complexity is the complexity of the process in the vertical dimension. Early processes typically had only four masking levels. Today's processes range from 8 to 14 levels, and in the future even more masking levels will be employed. In addition there is exploratory work for placing another layer of circuits on top. This will obviously require even more masking levels resulting in more shapes that must be drawn.

Gate Arrays

A gate array consists of a chip in which a predefined pattern of transistors is fabricated on the chip. These transistors are constructed only in the silicon levels. The customization of the chip occurs only on the metal levels. Because every chip contains the same pattern of transistors independent of its final usage, the chips (wafers) are manufactured in volume, stockpiled, and the metal customizing is placed on the chips at the last moment. Thus, the unique manufacturing processing is only a portion of the total manufacturing time, and the time from design to end product is significantly reduced. In a typical gate array, the manufacturing turnaround time is half that for a chip which has customization built into all of the mask levels.

The layout of a typical gate array is shown in Fig. 2. The chip has three main areas. Around the periphery of the chip

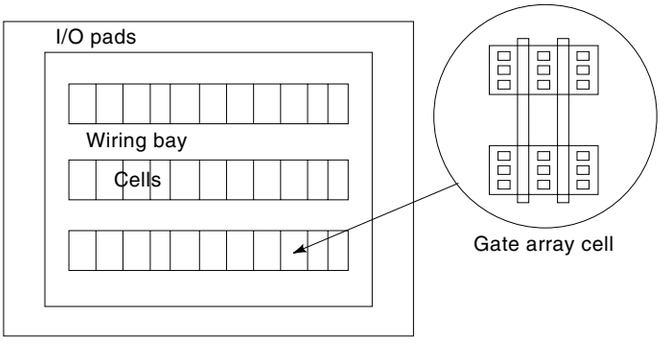


Figure 2. Typical gate array.

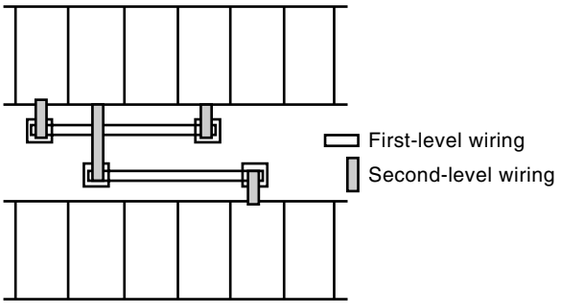


Figure 3. Wiring channels.

is the area reserved for input/output (I/O) circuits and the bonding pads which connect the chip to the next higher level package. Inside the area is the so-called active area that contains the array of transistors which are connected to implement the required logic. This active area is typically divided into entities called cells and wiring bays. Cells are regularly shaped areas that contain the transistors. A circuit is formed by interconnecting the transistors in one or more cells. Between the rows of cells are wiring channels used to connect circuits. In typical gate array, the connections between circuits consist of a series of line segments in a vertical and horizontal direction. These wiring segments lie on the wiring grid. The wiring grid consists of all of the allowable locations for placing line segments used to interconnect circuits. It is usually defined as a series of vertical and horizontal lines spaced as close together as the technological rules allow. These lines do not exist in reality, but they are stored by computer to define permissible line segment locations. If the gate array has two metal levels, the first metal level is typically routed horizontally and the second level metal is routed vertically (Fig. 3). The connection between metal levels is made by a via (a hole in the dielectric that separates the metal levels). These vias can be thought of as wiring segments in a vertical direction. If only one metal level is available, the vertical segments must be implemented on a diffusion or polysilicon mask level.

A typical gate-array cell contains four transistors. The small square shapes are contacts to the transistors which must be connected together to define a specific circuit.

All input and output (I/O) connections of this gate are brought to the cell boundary next to the wiring channels. These circuit I/Os are the only valid connection points to the circuit. They must fall on the exact locations of the vertical and horizontal wiring grid described before. Thus the connections of the transistors to form circuits are independent of the connections between circuits by making all circuit I/Os fall off the wiring grid. This is an important concept in design called hierarchy. Hierarchy is a methodology of doing various parts of a design independent of other portions of the design but under certain constraints. These constraints ensure that the various pieces of the design are merged at the end and that the chip works. This is done by defining standard interfaces which each part of the design must meet. A simple analogy to this is the way automobiles are assembled. Car engines are built in one plant, transmissions are built in another plant, and the car is assembled in still another plant. By designing the interface between the engine and the transmission in a standard format, the engines and transmissions are built independently and a large variety of engine and transmission options are combined in the final car.

The entire chip image described previously is designed ahead of time (before the logic function is implemented on the chip). The image design is usually done in the laboratory of the chip manufacturer by its design engineers. Similarly, these same engineers are usually responsible for the electrical design and the physical design (layout) of the circuits. Doing this portion of the design requires a detailed knowledge of the particular technology, including circuit design and process. This is best done by the manufacturer of the chip. By segmenting the design process in this manner, the logic designers, who implement the logic, do not have to acquire the detailed skills necessary to design the image and circuits. A further advantage is that the chip manufacturer is free to modify the process without concern over distributing that information to a wide variety of designers.

Once the image and circuits are designed, they are converted to digital form for storage in a design system. The chip design process consists of assembling the various circuit configurations on the chip image. Because all of the design requiring detailed technical knowledge is done ahead of time, the process of designing a chip is done by people who do not understand the technological details. In fact, if the design process is fully automated, which many gate array design systems are at this time, the chip design is done by relatively unskilled people. This is described in more detail in this section.

The discussion of gate arrays shows some clear advantages for this approach. There are, however, some drawbacks. Gate arrays shorten and simplify the design process. They do this at the expense of silicon efficiency. A design done as a gate array is larger than a design done by a more manual process. The reason for this is that the gate array image designer must anticipate all of the potential usages of the gate array and incorporate them in the selection of transistors. This makes the transistors larger then if they were designed for each individual application. Also in any typical design, not all of the transistors are used. Another problem with the gate array is that it uses more power than a more customized design. The larger transistors and the inability to tailor the design to the specific application result in higher power usage. The switching speed for gate arrays is usually not as high as that of more customized chips. Performance is usually related to chip size. For any given logic function, the bigger the chip, the slower it runs. Also the inability to custom design transistors for a given application affects performance. In summary, gate arrays offer the designer a trade-off between design time and cost/performance/power of the finished product.

Standard Cells

For the purpose of this section, standard cells are defined as chips which are customized on all mask levels with the exception of pure custom designs. Referring to the design spectrum, standard cells cover the entire spectrum from the gate arrays on one side to the custom design on the other side of the spectrum. This results in a wide variety of design methods. In this introductory section, a simple, restricted design methodology has been chosen. This simple methodology is sufficient to explain the concepts and to contrast standard cells with gate

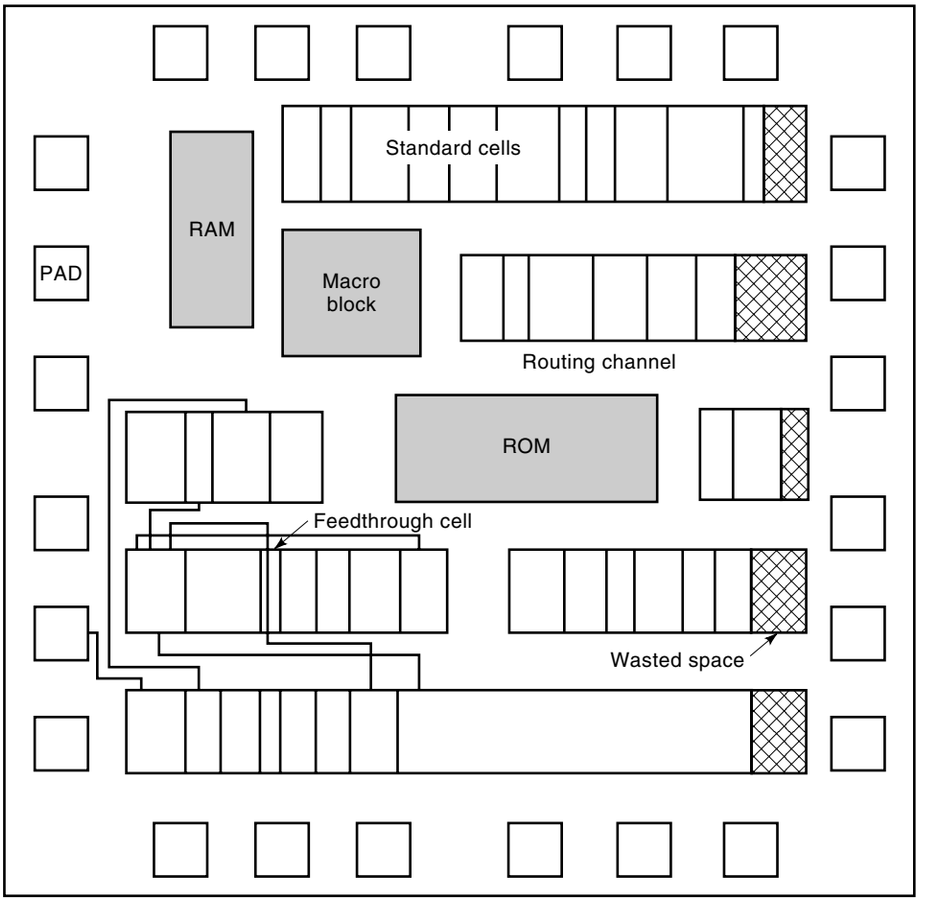


Figure 4. Standard cell layouts with macro blocks.

arrays. The last part of this section deals with variations in the basic approach covered in this section.

At this level of detail, the image looks just like the gate array (Fig. 2). The chip is divided into the same major areas. The outside of the chip contains the I/O circuits and pads. Inside is the so-called active area that contains cells into which circuits are placed and also wiring bays which contain the intercircuit wiring.

For the gate array, the cells contained a predefined pattern of transistors. In a standard cell, the cells are totally blank at the time the image is defined. They are just reserved areas of silicon which will contain circuits. Because these areas are reserved by the design system and the customization of the chip occurs on all mask levels, there are no restrictions on the types of circuit devices that are placed in the cells. This allows much more flexibility in designing standard cell circuits, resulting in a more efficient utilization of silicon. In a standard cell, only the transistors needed to make a circuit are used. In addition the transistors used are tailored to the specific use.

Another feature of standard cells that allows for higher density is the ability to place large complex circuits (called macros) on the image (Fig. 4). These macros occupy all of the image area and also the wiring bay area. They are random access memory (RAM) macros, read only memory (ROM) macros, registers, arithmetic logic units (ALUs), or any other customized piece of logic.

Field-Programmable Gate Arrays

The field-programmable gate array (FPGA) is a new approach to ASIC design that dramatically reduces manufacturing turnaround time and cost (18–19). FPGA designs provide large-scale integration and user programmability. An FPGA consists of horizontal rows of programmable logic blocks which are interconnected by a programmable routing network. In its simplest form, a logic block is a memory block that is programmed to remember the logic table of a function. Given a certain input, the logic block "looks up" the corresponding output from the logic table and sets its output line accordingly. Thus by loading different look-up tables, a logic block is programmed to perform different functions. It is clear that 2^K bits are required in a logic block to represent a K-bit input, 1-bit output combinational logic function. Obviously, logic blocks are feasible only for small values of K. Typically, the value of K is 5 or 6. The value of K is even less for multiple outputs and sequential circuits. The rows of logic blocks are separated by horizontal routing channels. The channels are not simply empty areas in which metal lines are arranged for a specific design. Rather, they contain predefined wiring "segments" of fixed lengths. Each input and output of a logic block is connected to a dedicated vertical segment. Other vertical segments merely pass through the blocks, serving as feedthroughs between channels. Connection between a horizontal segment and a vertical segment is provided through a

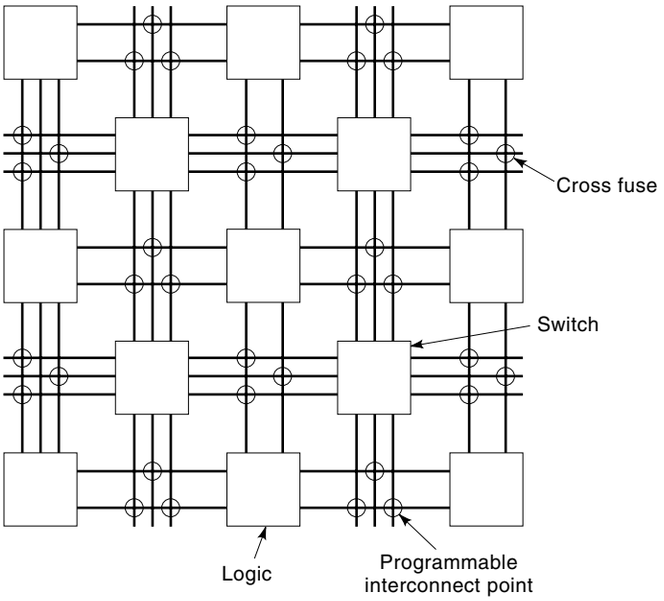


Figure 5. Architecture of an array-style FPGA.

cross fuse. Figure 5 shows the general architecture of an FPGA, which consists of five rows of logic blocks. The cross fuses are shown as circles, whereas antifuses are shown as rectangles.

Because there are no user-specific fabrication steps in an FPGA, the fabrication process is set up cost effectively to produce large quantities of generic (unprogrammed) FPGAs. The customization (programming) of an FPGA is rather simple. Given a circuit, it is decomposed into smaller subcircuits, so that each subcircuit is mapped to a logic block. The interconnections between any two subcircuits are achieved by programming the FPGA interconnects between the corresponding logic blocks. Programming (blowing) one of the fuses (antifuse or cross fuse) provides a low resistance bidirectional connection between two segments. When blown, antifuses connect the two segments to form a longer one. To program a fuse, a high voltage is applied across it. FPGAs have special circuitry to program the fuses. The circuitry consists of the wiring segments and control logic at the periphery of the chip. Fuse addresses are shifted into the fuse programming circuitry serially.

The programmable nature of these FPGAs requires new CAD algorithms to effectively use logic and routing resources. The problems involved in customizing an FPGA are somewhat different from those of other design styles, but many steps are common. For example, the partition problem of FPGAs is different from the partitioning problem in other design styles whereas the placement and the routing are similar to the gate-array approach.

Sea of Gates

The sea of gates is an improved gate array in which the master is filled completely with transistors. The master of the sea of gates has a much higher density of logic implemented on the chip and allows a designer to fabricate complex circuits, such as RAMs. In the absence of routing channels, interconnects have to be completed by routing through gates or by adding more metal or polysilicon interconnection layers. There are problems associated with either solution. The former reduces the gate utilization, and the latter increases the mask count and increases fabrication time and cost.

Comparison of Different Design Styles

The choice of design style depends on the intended functionality of the chip, time to market, and the total number of chips to be manufactured. It is common to use full-custom design style for microprocessors whereas FPGAs are used for a small circuit used in networking. However, there are several chips which have been manufactured by using a mix of design styles. For large circuits, it is common to partition the circuit into several small circuits which are then designed by different teams. Each team may use a different design style or a number of design styles. Another factor complicating the issue of design style is reusability of existing designs. It is a common practice to reuse a complete or partial layout from existing chips for new chips to reduce the cost of a new design.

Design styles are a continuum from very flexible (full-custom) to a rather rigid design style (FPGA) to cater to differing needs. A comparison of layout styles is given in Table 1. For example, it takes longer to fabricate a standard cell; however, the resulting chip operates at a higher speed. As seen from the table, full-custom provides compact layouts for high-performance designs but requires considerable fabrication effort. On the other hand, an FPGA is completely prefabricated and does not require any user-specific fabrication steps. However, FPGAs are used only for small, general purpose designs.

Other Layout Environments

A printed circuit board (PCB) is a multilayer sandwich of routing layers. ICs are packaged into ceramic or plastic carriers and then mounted on a PCB. The current PCB technology offers as many as 30 or more routing layers. Via specifications are also very flexible and vary so that a wide variety of combinations is possible. For example, a set of layers can be connected by a single via called the stacked via. The traditional approach of single chip packages on a PCB has intrinsic limitations in silicon density, system size, and contribution to propagative delay. Thru-hole assemblies gave way to surface-mounted assemblies (SMAs). In an SMA, pins of the device do not go through the board but are rather soldered to the surface of the board and devices are placed on both sides of

Table 1. Comparison Among Various Layout Styles

	Full Custom	Standard Cell	Gate Array	FPGA
Fabrication time	−−	−−	+	++
Packing density	++	+	−	−−
Unit cost in large quantity	++	++	+	−−
Unit cost in small quantity	−−	−−	+	++
Easy design and simulation	−−	−	−	++
Remedy for erroneous design	−−	−−	−	++
Accuracy of timing simulation	−	−	−	+
Chip speed	++	++	+	−

+ desirable
− not desirable

the board. SMAs eliminate the need for large diameter plated-thru-holes, allowing finer pitch packages and increased routing density. SMAs reduce the package footprint and improve performance.

The layout problems for printed circuit boards are similar to those in VLSI design, although printed circuit boards offer more flexibility and a wider variety of technologies. The routing problem is much easier for PCBs because many routing layers are available. The planarity of wires in each layer is a requirement in a PCB as it is a chip. There is little distinction between global routing and detailed routing in the case of circuit boards. In fact, because many layers are available, the routing algorithm must be modified to adapt to this three-dimensional problem. Compaction has no place in PCB layout because of the gridlike domain used to represent layout information.

For more complex VLSI devices with 120 to 196 I/Os, even the surface-mounted approach becomes inefficient and limits system performance. A 132 pin device in a 635 μm pitch carrier requires a 25.4 to 38.1 mm^2 footprint. This represents a four to sixfold density loss and twofold increase in interconnect distances versus a 64 pin device. It has been shown that the interconnect density for current packaging technology is at least one order of magnitude lower than the interconnect density at the chip level. This translates into long interconnection lengths between devices and a corresponding increase in propagation delay. For high-performance systems, the propagation delay is unacceptable, even with surface mounting. A higher performance packaging and interconnection approach is necessary to achieve the performance improvements promised in VLSI technologies. This has led to the development of multichip modules (MCMs).

The key to semiconductor device improvement is the shrinking feature size, that is, the minimum gate or line width on a device. The shrinking feature size provides increased gate density, increased gates per chip, and increased clock rates. These benefits are offset by an increased number of I/Os and increased chip power dissipation. The increased clock rate is directly related to the device feature size. With reduced feature sizes, each on-chip device is smaller and therefore has reduced parasitic effects and allows faster switching. Furthermore, the scaling has reduced on-chip gate distances and therefore interconnect delays.

Much of the improvement in system performance promised by the ever increasing semiconductor device performance, however, has not been realized because of the performance barriers imposed by today's packaging and interconnection technologies. Increasingly more complex and dense semiconductor devices are driving the development of advanced VLSI packaging and interconnection technology to meet increasingly more demanding system performance requirements. The alternative approach to the interconnect and packaging limits of conventional chip carrier/PCB assemblies is to eliminate packaging levels between the chip and PCB. One such approach uses multichip modules. The MCM approach eliminates the single chip package and, instead, mounts and interconnects the chips directly onto a higher density, fine-pitch interconnection substrate. In some MCM technologies, the substrate is simply a silicon wafer on which layers of metal lines are patterned. This substrate provides all of the chip-to-chip interconnections within the MCM. Because the chips are only one-tenth of the area of the packages, they are placed closer together on an MCM. This provides higher density assemblies and shorter and faster interconnects. Figure 6 is a diagram of an MCM package.

It is predicted that multichip modules will have a major impact on all aspects of electronic system design. Multichip module technology offers advantages for all types of electronic assemblies. Mainframes will need to interconnect the high numbers of custom chips needed for the new systems. Cost-performance systems will use the high density interconnect to assemble new chips with a collection of currently available chips to achieve high performance without time-consuming custom design, allowing quick time to market. The significant

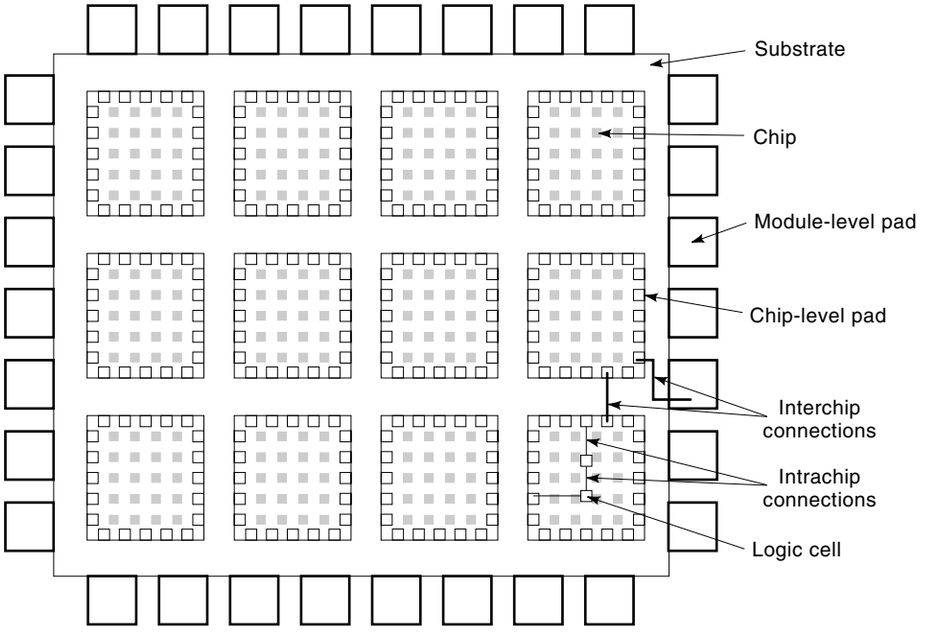


Figure 6. A typical diagram of a multichip module.

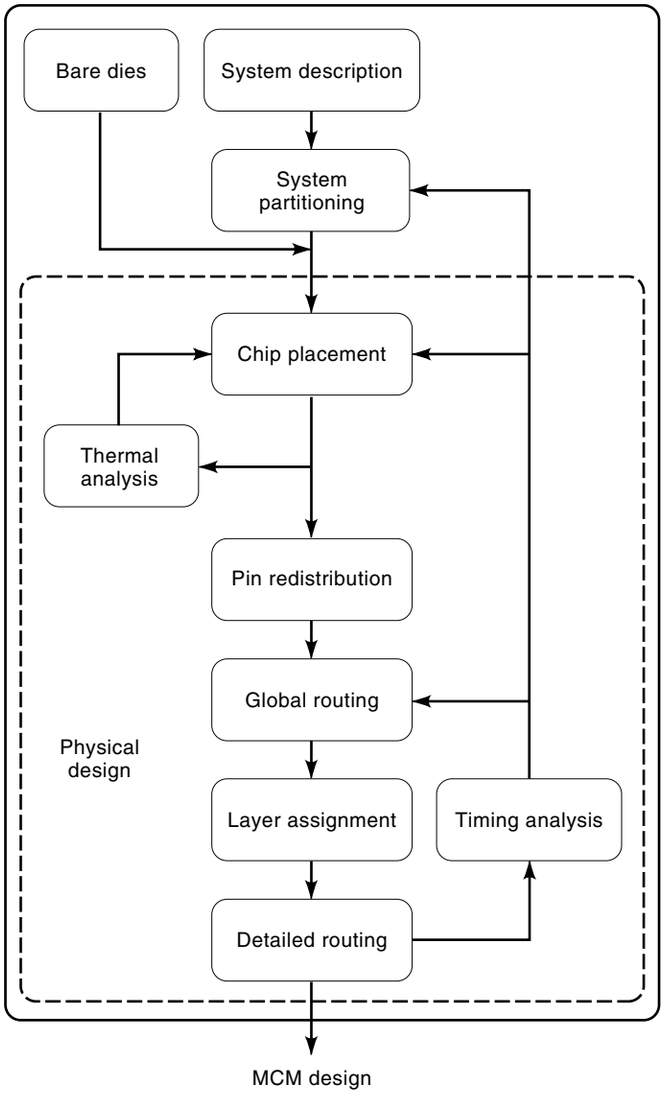


Figure 7. Multichip module design flow.

benefits of multichip modules are reduced size, reduced number of packaging levels, reduced complexity of the interconnection interfaces and clearly cheaper and more efficient assemblies. The multichip revolution in the 1990s will have an impact on electronics as great or greater than surface-mount technology had in the 1980s.

The layout problems in MCMs are essentially performance-driven (Fig. 7). The partitioning problem minimizes the delay in the longest wire. Although placement in MCM is simple compared with VLSI, global routing and detailed routing are complex in MCM because of the large number of layers. The critical issues in routing include the effects of cross talk and delay modeling of long interconnect wires.

MCM packaging technology does not completely remove all of the barriers of IC packaging technology. Wafer-scale integration (WSI) is considered the next major step, bringing with it the removal of a large number of barriers. In WSI, the entire wafer is fabricated with several types of circuits, the circuits are tested, and the defect-free circuits are interconnected to realize the entire system on the wafer. The attractiveness of WSI lies in its promise of greatly reduced cost, high performance, high level of integration, greatly increased reliability, and significant application potential. However there are still major problems with WSI technology, such as redundancy and yield, that are unlikely to be solved in the near future.

PHYSICAL DESIGN FLOW

The design cycle of VLSI chips consists of consecutive steps from high-level synthesis (functional design) to production (packaging). The physical design is the process of transforming a circuit description into the physical layout, which describes the positions of cells and routes for the interconnections between them. The main concern in the physical design of VLSI chips is to find a layout with minimal area and minimal total wire length. Some critical nets have much stricter limitations for the maximal wire length.

Since VLSI is such a complex system, the designer is forced to use a hierarchical approach or top-down approach. In circuit layout, we do placement first, then global and detailed routing. Because of its complexity, the physical design is normally broken into various substeps:

- First, the circuit has to be partitioned to generate some (up to 50) macro cells.
- In the floor-planning stage, the cells have to be placed on the layout surface.
- After placement, the global routing is done. In this step, the "loose" routes for the interconnections between the single modules (macro cells) are determined.
- In detailed routing, the exact routes for the interconnecting wires in the channels between the macro cells have to be computed.
- The last step in the physical design is compacting the layout by compressing it in all dimensions, so that the total area is reduced.

Although we decompose the complex problem into a series of simpler problems to reduce the design complexity, relying on incomplete and abstract models when performing early design decisions may adversely affect the quality of the final solution. Unfortunately, the exact physical attributes of a design are not known, even in the placement step, until the entire design process is carried out. Because design steps are quite expensive, it is not feasible to go through a full design interative step evey time a high-level decision is made.

At the heart of all methodologies, however, are logic entry programs and checking and auditing programs that ensure that the chip has been placed and wired properly according to the logic specification. Figure 8 shows a typical design methodology flowchart.

Each block in the logic list corresponds to a graphical layout of a logic function. The discipline of physical design entails placing, wiring, and checking these physical blocks and then generating the final graphics for manufacture. As described previously, the chip image consists of an array of cells separated by wiring bays into which the physical blocks fall. Automatic placement and wiring programs take advantage of

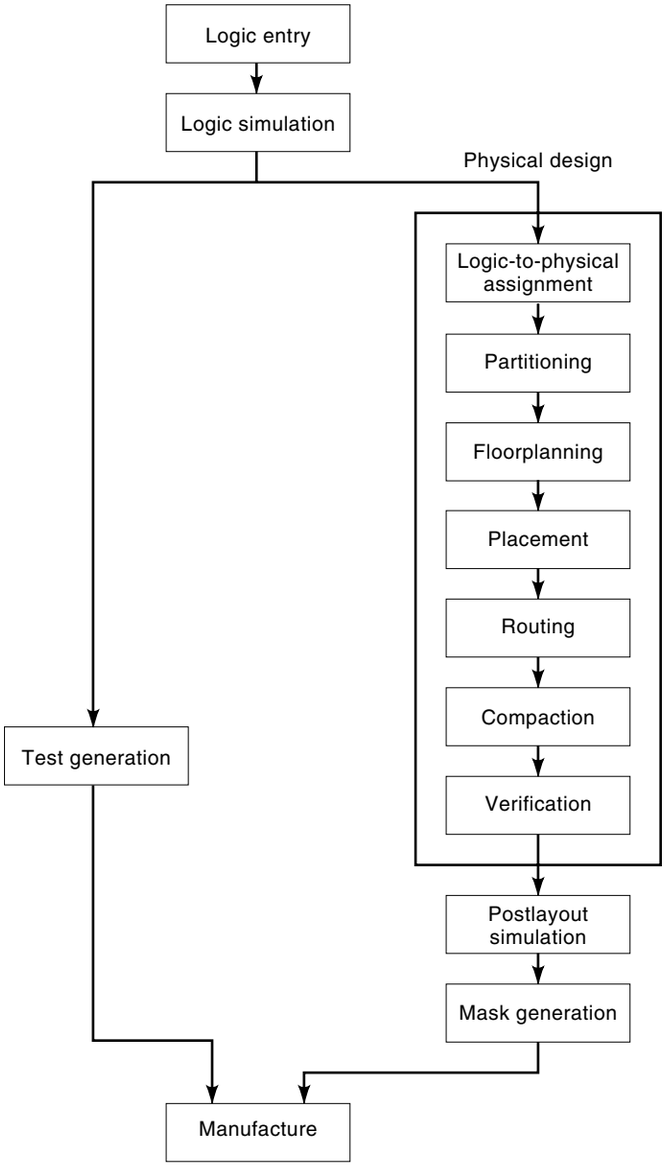


Figure 8. Physical design flow.

this regularity and save much time in the physical design process.

Now we can place and wire the chip. Automatic placement and wiring programs are not strictly necessary for the chip design process. Earlier design systems have relied upon large manual efforts to accomplish this task. Indeed, a manual placement and wiring capability is necessary because often the programs do not do a complete job.

Placement simply consists of assigning a module location to each logic circuit. If the logic library is not all identical in size, then the physical size of each circuit must be known so that the placement does not overlap the circuits. It is all too easy to assign a one-cell circuit to an area already occupied by a multicell circuit. The end goal of placement is to reduce the total wire length on the chip. Preplacement is done if performance of a particular logic path is critical and must be manually controlled. Some standard cell systems offer the option of large array macros, such as RAMs and PLAs. These macros might have to be manually placed, because placing them automatically is not easy.

After placement, global routing decomposes a large routing problem into small, manageable problems for detailed routing. The method first partitions the routing region into a collection of disjoint rectilinear subregions. This decomposition is carried out by finding a "rough" path for each net in order to reduce the chip size, shorten the wirelength, and evenly distribute the congestion over the routing area. Detailed routing follows global routing. The traditional model of detail routing is the two-layer Manhattan model with reserved layers, where horizontal wires are routed on one layer and vertical wires are routed in the other layer. The layout is verified to ensure that the layout meets the system specifications and the fabrication requirements. Design verification consists of design rule checking and circuit extraction.

Once the chip is place and wired, then design rule checking is next. This checks the placement for violations, such as overlapping circuits, circuits falling off the chip, and I/O circuits placed in the internal cell structure, or vice versa. Wiring checks include checks for shorts and opens and checks for wires routed through blockages created by chip power buses. Electrical checking is also done at this point. Bipolar circuits are sensitive to wire widths because they drive currents through global nets. Checking for adequate wire widths should be done here. Excessive numbers of plane to plane vias are checked for, because they add resistance and, hence, delay to the nets. Results from all this checking must be folded back into the placement and wiring, forming another iterative loop in the design process.

The next step in the physical design process is to extract the net lengths from the design to create a more accurate set of delays. Parasitic resistance and capacitance per unit length for the technology are stored in the technology database, and these values are multiplied by the lengths of each net. The delay of each circuit is computed accurately only when this information is known. Then the delays are fed back to the logic designer for a more accurate delay simulation. If delay paths are too long, then the logic must be changed, necessitating a complete restart on the design, or the placement or wiring must be changed. The crude delay estimation done before physical design should try to be accurate enough that this does not happen.

Some design systems offer the capability of doing power optimization. This consists of automatically replacing each logic circuit with a higher power version until certain delay criteria are met. The advantage of doing this is that unnecessary ac or dc power consumption is minimized automatically. The design is started with all logic at the lowest power level. Then, successive iterations of delay calculation, identifying the failing blocks and powering up are done until either all of the nets pass their delay criteria or the maximum power level is reached.

The final step in physical design is converting the placement and wiring information and the graphic designs of the chip power structure and circuit library into a complete graphical description of the chip to send to manufacturing. Shape level layout checking can be done at this point as a final check on the design. If the image and library are designed correctly, then there should never be a layout error at this stage. Extremely large data volumes characteristic of

VLSI have been raising questions about the ability to continue on this chipwide layout checking. One solution to the problem is to use a "shadow" for each circuit when actually doing the checking. The layout of the circuits is checked when the library is designed, and a simplified outline shape is generated which is substituted for the actual graphics before checking. If the gate array approach is chosen, then only the metallization levels need to be checked and sent to manufacturing. Otherwise, all design levels must be checked and sent.

The data volumes of VLSI have forced modern placement programs to approach the chip hierarchically. The task is divided into a coarse of global phase and a fine or detailed phase. The global phase is concerned with partitioning, or dividing the logic into more manageable portions, and the detailed phase makes the exact cell assignment. The global and detailed phases are repeated as many times as necessary to fit each logic partition into the program's available storage area. Partition looks at the circuit's interconnection and decides where a boundary can be drawn with relatively few nets crossing it. Detailed placement then iterates through many physical arrangements based on different random number seeds looking for the smallest total net length.

PARTITIONING

A chip may contain several million transistors. Layout of the entire circuit cannot be handled because of the limitation of memory space and computational power available. Even though fabrication technologies have made great improvements in packing more logic in a smaller area, the complexity of circuits has also been increasing correspondingly. This necessitates breaking a circuit and distributing it across several chips. Thus, the first step in the physical design phase is partitioning.

Partitioning is a complex, discrete, and highly intractable problem which is nondeterministic polynomial (NP)-complete. The nature of the partitioning problem and with the size of the circuit make it difficult to perform an exhaustive search for an optimal solution.

We review the commonly used methods for partitioning. Classic iterative approaches, such as Kernighan–Lin and Fiduccia–Mattheyses begin with some initial solution and try to improve it by making small changes, such as swapping modules between clusters. Iterative improvement has become the industry standard for partitioning because of its simplicity and flexibility. Recently, there have been many significant improvements to the basic FM algorithm.

These are broadly classified into four categories: direct method, group migration method, metric allocation method, and simulated annealing. All these methods, except simulated annealing, suffer from the problem of getting stuck in locally optimal solutions. Simulated annealing (SA) is a probabilistic hill-climbing search technique with the advantage of not being stuck in a local optimum. For this reason, it is widely used.

The foundation of SA in the process of the problem is the central aspect of SA, and it is analogous to the energy of a physical system. The stability of a physical system depends on the energy possessed by it. Lower energy (or low-cost functional value) indicates a more stable state. The SA method corresponds to a solution (or state) of the problem. The process begins at a high temperature T, at which a large number of moves that increase the energy are accepted. The temperature is gradually reduced according to a cooling schedule until it reaches zero or a very low value. As the temperature decreases, the number of accepted moves that increase the cost functional value (or the energy) also decreases.

At low temperatures, the method resembles a greedy algorithm because only moves that reduce the cost functional value are accepted. SA does not get trapped in a local optimum because moves that increase the cost functional value are also accepted which lead to a more exhaustive search of the solution space. Register transfer level (RTL) designs are evaluated with a spreadsheet-like approach and violations for area, power, and pin count constraints are checked. The hierarchical clustering technique is used in partitioning behavioral hardware descriptions in which a similarity measure is computed for all pairs that communicate with one another. Clusters of functions are formed with these similarity measures. However, no designed constraints are considered while forming these clusters.

In addition to these four broad categories of partitioning algorithms, several variations and improvements to partitioning strategies continue to be researched. Network flow techniques have been applied to graph bipartitioning and also to multiway partitioning with area and pin constraints. The maximum-flow, minimum-cut algorithm transforms the minimum-cut problem into a maximum-flow problem (22).

An approach that also serves as a strategy to generate initial partitions is based on eigenvector decomposition. This approach requires transformation of every multiterminal net into two terminal nets which could result in a loss of information needed for a performance-based partitioning. Although this algorithm finds the optimal solution between any pair of nodes in a network, there is no constraint on size of resultant partitions. Another related partitioning technique is that based on spectral partitioning, which relies on finding the eigenvalues and eigenvectors of a Laplacian matrix.

The eigenvectors yield a k-dimensional Euclidean space embedding of a graph which minimizes the weighted sum of squared distance of the vertices. The k smallest eigenvectors provide an embedding of the n vertices of the graphs in a k-dimensional subspace. Then efficient clustering heuristics are used to coerce the points in the embedding into k partitions. Hagen et al. (24) propose the Rent parameter as a quality measure for the underlying partitioning algorithm. The Rent parameter characterizes a power-law relationship between the number of external terminals of a partition in the layout and the number of nodes in the layout. Their results suggest that top-down layout techniques based on spectral ratio-cut partitioning achieve denser layouts than the current methods based on Fiduccia–Mattheyses partitioning. The power of spectral methods lies in their ability to take global information into account during partitioning. Another effective method is using clustering within a two-phase methodology. A clustering on the net list induces a smaller net list. For example, if the average cluster size is 5, a 10000 module net list is clustered to a 2000 module net list. Then iterative improvement is run on the smaller net list to derive an initial solution for the larger solution. Then iterative improvement is run a second time on the larger net list. Bui et al., Hagen and Kahng, and Alpert and Kahng (24) have all shown the effectiveness of the two-phase methodology. This two-phase

approach can be repeated many times to give a multilevel algorithm. Multilevel approaches are very popular in the sparse matrix computational community, with works such as Hendrickson and Leland and Kumar and Karypis (23). To apply it to circuit net lists as opposed to graphs, extension of the work of Kumar and Karypis is presented in Ref. 24. There are three steps in the algorithm: first, the coarsening algorithm, then, the initial partitioning algorithm, and, finally, the refinement algorithm.

FLOOR PLANNING

In the floor-planning phase, the macro cells have to be positioned on the layout surface so that no blocks overlap and there is enough space left to complete the interconnections. The input for the floor planning is a set of modules, a list of terminals (pins for interconnections) for each module, and a net list, which describes the terminals to be connected. At this stage, good estimates for the area of the single macro cells are available, but their exact dimensions still vary in a wide range. Consider, for example, a register file module consisting of 64 registers. These alternatives are described by shape functions. A shape function is a list of feasible height/width combinations for the layout of a single macro cell. The result of the floor-planning phase is the sized floor plan, which describes the position of the cells in the layout and the chosen implementations for the flexible cells.

In this stage, the relative positions of the modules to be laid out are determined. Timing, power, and area estimates are the factors guiding the relative placement. Floor planning is used to verify the feasibility of integrating a design onto chip without performing the detailed layout and design of all of the blocks and functions. If the control logic is implemented with standard cells, then the number of rows used for the modules is not necessarily fixed. Many rows produce a long, skinny block. Few rows produce a short, fat block. As other examples, folding and partitioning of a PLA is used to modify the aspect ratio of the module, or the number of bits used for row and column decoding in a RAM or ROM module also modifies their aspect ratio.

Since floor planning is done very early in the design process, only estimates of the area requirements are given for each module. Recently, the introduction of simulated annealing algorithms has made it possible to develop algorithms where the optimization is carried out with all the degrees of freedom mentioned previously. A system developed at the IBM T. J. Watson Research Center and the TimberWolf package developed at Berkeley use the simulated annealing algorithm to produce a floor plan that gives the relative positions of the modules and also aspect ratios and pin positions.

Automatic floor planning is more and more important as automatic module generators become available which accept pin positions and aspect ratios of the blocks as constraints or parts of the cost functions. Typically, floor plans consists of the following two steps: first, the topology, that is, the relative positions of the modules, is determined. At this point, the chip is viewed as a rectangle and the modules are the (basic) rectangles whose relative positions are fixed. Next, we consider the area optimization problem, that is, we determine a set of implementations (one for each module) so that the total area of the chip is minimized. The topology of a floor plan is obtained by recursively using circuit partitioning techniques. A *partition* divides a given circuit into k parts so that (1) the sizes of the k parts are as close as possible and (2) the number of nets connecting the k parts is minimized. If $k = 2$, a recursive bipartition generates a slicing floor plan. A floor plan is slicing if it is a basic rectangle or there is a line segment (called a slice) that partitions the enclosing rectangle into two slicing floor plans. A slicing floor plan is represented by a slicing tree. Each leaf node of the slicing tree corresponds to a basic rectangle, each nonleaf node of the slicing tree corresponds to a basic rectangle, and each nonleaf node of the slicing tree corresponds to a slice.

There are many different approaches to the floor-planning problem. Wimer et al. (25) describe a branch and bound approach for the floor-plan sizing problem, that is, finding an optimal combination of all possible layout alternatives for all modules after placement. Although their algorithm finds the best solution for this problem, it is very time-consuming, especially for real problem instances. Cohoon et al. (26) implemented a genetic algorithm for the entire floor-planning problem. Their algorithm uses estimates for the required routing space to ensure completing the interconnections. Another more often used heuristic solution method for placement is simulated annealing (27,28).

When the area of the floor plan is considered, the problem of choosing each module for the implementation that optimizes a given evaluation function is called the floor plan area optimization problem (29).

A floor plan consists of an enveloping rectangle partitioned into a nonoverlapping basic rectangles (or modules). For every basic rectangle, a set of implementations is given, which have a rectangular shape characterized by a width w and a height h. The relative positions of the basic rectangles are specified by the floor-plan tree. The leaves are the basic rectangles, the root is the enveloping rectangle, and the internal nodes are the *composite rectangles.* Each of the composite rectangles is divided into k parts in a hierarchical floor plan of order k. If $k = 2$ (slicing floor plan), a vertical or horizontal line is used to partition the rectangle. If $k = 5$, a right or left wheel is obtained. The general case of composite blocks which cannot be partitioned in two or five rectangles is dealt with by allowing them to be composed of *L-shaped* blocks. Once the implementation for each block has been chosen, the sizes of the composite rectangles are determined by traversing up the floor-plan tree. When the root is reached, the area of the enveloping rectangle is computed. The goal of the floor plan area optimization problem is to find the implementation for each basic rectangle, so that the minimum area enveloping rectangle is obtained. The problem has been proved to be NP-complete in the general case, although it is reduced to a problem solvable in polynomial time in the case of slicing floor plans.

Several solutions have been proposed for this problem:

- a branch and bound algorithm able to produce the exact solution when the size of the problem is not too large;
- a complete algorithm which is based on a bottom-up traversal of the floor-plan tree and produces the optimum solution with medium-size problems;
- an approximation algorithm which deals with the greatest problem sizes with acceptable CPU time and memory requirements;

- an alternative algorithm which is based on the replacement of *superblocks* by equivalent basic blocks; and
- a recent algorithm particularly effective for floor plans, which are approximately slicing, and reduces to a polynomial algorithm for slicing floor plans.
- Recently, a genetic algorithm was proposed to deal with general floor plans (29).

PLACEMENT

The placement problem is defined as follows (Fig. 9). Given an electrical circuit consisting of modules with predefined input and output terminals and interconnected in a predefined way, construct a layout indicating the positions of the modules so that the estimated wire length and layout area are minimized. The inputs to the problem are the module description, consisting of the shapes, sizes, and terminal locations, and the net list, describing the interconnections between the terminals of the modules. The output is a list of x and y coordinates for all modules. The main objectives of a placement algorithm are minimizing the total chip area and the total estimated wire length for all of the nets. We need to optimize chip area to fit more functionality into a given chip area. We need to minimize wire length to reduce the capacitive delays of longer nets and speed up the chip operation. These goals are closely related to each other for standard cell and gate-array designs, because the total chip area is approximately equal to the area of the modules plus the area occupied by the interconnects. Hence, minimizing the wire length is approximately equivalent to minimizing the chip area. In the macro design style, the irregularly sized macros do not always fit together, and some space is wasted. This plays a major role in determining the total chip area, and we have a tradeoff between minimizing area and minimizing the wire length. In some cases, secondary performance measures, such as the preferential minimization of the wire length of a few critical nets, are also needed, at the cost of increased total wire length. Module placement is an NP-complete problem and, therefore, cannot be solved exactly in polynomial time. Trying to get an exact solution by evaluating every possible placement to determine the best one would take time proportional to the factorial of the number of modules. This method therefore, is impossible to use for circuits with any reasonable number of modules. A heuristic algorithm must be used to search through a large number of candidate placement configurations efficiently. The quality of the placement obtained depends on the heuristic used. At best, we can hope to find a good placement with wire length quite close to the minimum, with no guarantee of achieving the absolute minimum.

Classification of Placement Algorithm

Placement algorithms are divided into two major classes: constructive placement and iterative improvement. In constructive placement, a method is used to build up a placement from scratch; in iterative improvement, algorithms start with an initial placement and repeatedly modify it in search of a cost reduction. If a modification results in a reduction in cost, the modification is accepted; otherwise it is rejected. In typical constructive placement, a seed module is selected and placed in the chip layout area. Then other modules are selected, one at a time, in order of their connectivity to the modules placed (most densely connected first) and are placed at a vacant location close to the placed modules, so that the wire length is minimized. Such algorithms are generally very fast, but typi-

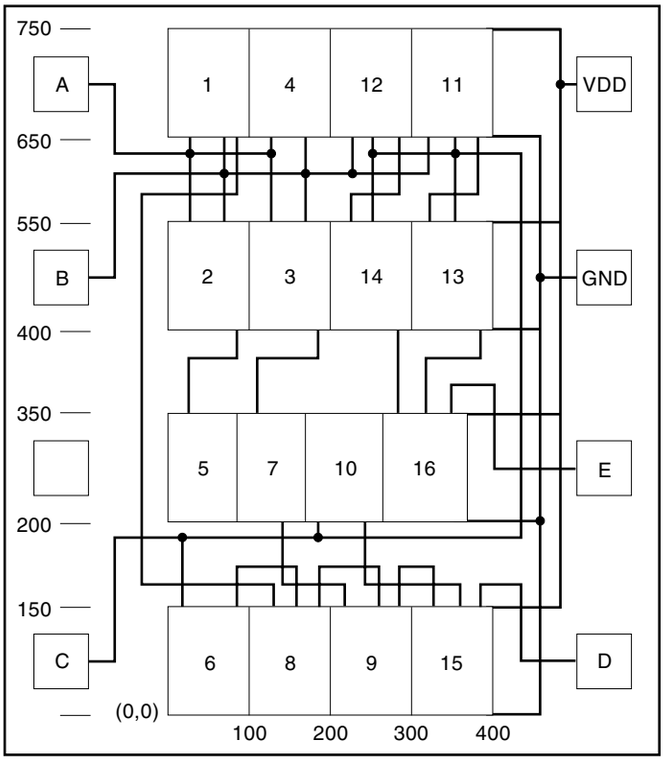


Figure 9. An example of placement.

cally result in poor layouts. These algorithms are now used for generating an initial placement for iterative improvement algorithms. The main reason for their use is their speed. They take a negligible amount of computation time compared to iterative improvement algorithms and provide a good starting point for them. More recent constructive placement algorithms, such as numerical optimization techniques (30,31), placement by partitioning (32), integer programming formulation (33), and simulated annealing (34) yield better layouts but require significantly more CPU time. Iterative improvement algorithms typically produce good placements but require enormous amounts of computation time. The simplest iterative improvement strategy interchanges randomly selected pairs of modules and accepts the interchange if it results in a reduction in cost (35). The algorithm is terminated when there is no further improvement during a given large number of trials. An improvement over this algorithm is repeated iterative improvement in which the iterative improvement process is repeated several times with different initial configurations in the hope of obtaining a good configuration in one of the trials. Currently popular iterative improvement algorithms include simulated annealing, the genetic algorithm, and some force-directed placement techniques, which are discussed in detail in the following sections. Other possible classifications for placement algorithms are deterministic algorithms and probabilistic algorithms. Algorithms that function on the basis of fixed connectivity rules or formulas or determine the placement by solving simultaneous equations are deterministic and always produce the same result for a particular placement problem. Probabilistic algorithms, on the other hand, work by randomly examining configurations and produce a different result each time they are run. Constructive algorithms are usually deterministic, whereas iterative improvement algorithms are usually probabilistic. Simulated annealing (34) is probably the most well-developed method available for module placement today. It is very time-consuming but yields excellent results. It is an excellent heuristic for solving any combinatorial optimization problem, such as the graph coloring (36), partitioning, routing (37,39), floor planning (28), or placement (39,40). The basic procedure in simulated annealing is to accept all moves that result in reduced cost. Moves that result in a cost increase are accepted with a probability that decreases with the increase in cost. A parameter T, called the temperature, controls the acceptance probability of the moves increasing cost. Higher values of T cause more such moves to be accepted. In most implementations of this algorithm, the acceptance probability is given by $\exp(-\Delta C / T)$, where ΔC is the cost increase. In the beginning, the temperature is set to a very high value so that most of the moves are accepted. Then the temperature is gradually decreased so that the moves increasing cost have less chance of being accepted. Ultimately, the temperature is reduced to a very low value so that only moves reducing cost are accepted, and the algorithm converges to a low cost configuration.

Wire Length Estimates

The placement process is followed by routing, that is, determining the physical layout of the interconnects through the available space. Finding an optimal routing given a placement is also a NP-complete problem. Many algorithms work by iteratively improving the placement and, at each step, estimating the wire length of an intermediate configuration. It is not feasible to route each intermediate configuration to determine how good it is. Instead we estimate the wire length. To make a good estimate of the wire length, we should consider the way in which routing is actually done by routing tools. Almost all automatic routing tools use Manhattan geometry, that is, only horizontal and vertical lines are used to connect any two points. Further, two layers are used. Only horizontal lines are allowed in one layer and only vertical lines in the other. The shortest route for connecting a set of pins together is a Steiner tree [Fig. 10(a)]. In this method, a wire branches at any point along its length. This method is usually not used by routers, because of the complexity of computing both the optimum branching point and the resulting optimum route from the branching point to the pins. Instead, minimum spanning tree connections and chain connections are the most commonly used connection techniques. For algorithms that compute the Steiner tree, see (41–45). Source-to-sink connections [Fig. 10(b)] called the Steiner distance-preserving tree

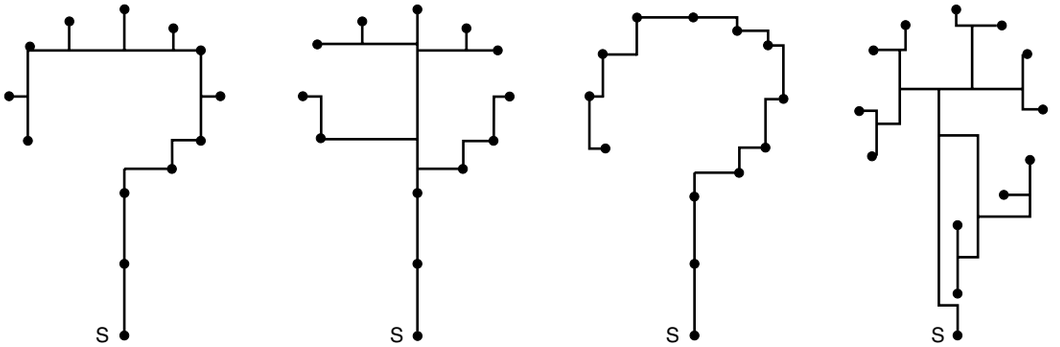


Figure 10. Some wiring topologies.

(46–49), where the output of a module is connected to all of the inputs by the shortest path, are the simplest to implement. They result, however, in excessive interconnect length and significant wiring congestion. Hence, this type of connection is seldom used for estimation. Chain connections [Fig. 10(c)] do not allow any branching at all. Each pin is simply connected to the next one in the form of a chain. These connections are even simpler to implement than the spanning tree connection, but they result in slightly longer interconnects.

An efficient and commonly used method for estimating wire length is the semiperimeter method. The wire length is approximated by half the perimeter of the smallest bounding rectangle enclosing all of the pins. For Manhattan wiring, this method gives the exact wire length for all two-terminal and three-terminal nets, provided that the routing does not overshoot the bounding rectangle. For nets with more pins and more zigzag connections, the semiperimeter wire length is generally less than the actual wire length. Besides, this method provides the best estimate for the most efficient wiring scheme, the Steiner tree. Some of the algorithms use the euclidean wire length or squared euclidean wire length. The squared wire length is used to save the time required for floating-point computations as compared with integral processing. Optimization of the squared wire length ensures that the euclidean wire length is optimized.

For a comprehensive overview of placement and routing algorithms see (50). Over the years, a wide variety of placement algorithms have been developed. Sharookar and Mazumder (51) provide a survey of various placement techniques. Several genetic placement algorithms have been presented (26,40,52,53). The classical work on layout generation with a genetic algorithm (GA) was done by Cohoon et al. (40). They encode a placement by a Polish notation of a binary slicing tree, thus having a chromosome represented by a string. They use different recombinatory operators, which work either directly on this string or take the tree structure into consideration by decoding the chromosome. Chan et al. (54) represent a placement in a bit-matrix. The layout area is divided into discrete regions, and each row in the matrix describes the orientation and the position of a single cell. Recombination is done by mixing two matrices. During the optimization, incorrect placements with overlapping cells are allowed and are handled by adding a penalty term when computing the fitness. A GA for macro cell placement is described where the genotype representation is a binary tree (40). In contrast to the approach of Cohoon et al., this tree does not directly characterize a placement, but it is generated by decoding the tree in a traversal by a given order. The operators directly work on the tree structure.

ROUTING

The routing stage, where the interconnections are laid out on the chip, is then broken into two stages because of complexity: global and detailed routing. In global routing sometimes called channel assignment, the 'loose' routes (which channel the interconnections go through) for the nets are determined. To compute global routing, the routing space is represented as a graph. The edges of this graph represent the routing regions and are weighted with the corresponding capacities. Global routing is described by a list of routing regions for each net of the circuit, and none of the capacities of any routing region is exceeded.

After global routing is done, the number of nets routed through each routing region is known. In the detailed routing phase, the exact physical routes for the wires inside routing regions have to be determined. This is done stepwise, that is, one channel at a time is routed in a predefined order.

Channel Router

If the region to be routed contains pins only on two sides, then effective detailed routing tools called *channel routers* are used. If the routing region has pins on four sides, then a switch box route is used. The shortest path problem is stated as follows: Given a graph G, each arc connecting v_i and v_j has an associated length d_{ij}. Find the shortest path between two vertices in the graph. Here, the length of the path is defined as the sum of the lengths of arcs in the path. This problem was solved by Dijkstra (55). A straightforward implementation of Dijkstra's algorithm needs $O(n^2)$ time. Moore and Lee (56) suggest that breadth-first search be used to find the shortest path. This is known as the maze router. This algorithm is guaranteed to find a path with minimum wire length. Another type of algorithm, called the line-expansion algorithm, was developed to minimize the number of vias in the path and to reduce the memory storage and increase speed (57). The Lee–Moore grid-expansion algorithm (20) and many of its variations (19) have been widely used.

The channel routing problem is stated as follows: Given two parallel horizontal lines at distance $m + 1$ units apart (i.e., there are m tracks between the two horizontal lines), there are numbers 0, 1, 2, . . ., n written on the two horizontal lines. The problem is to connect all nets using two layers and to minimize the required number of tracks m. Channel routing is widely used in automatic layout design because of its high packing density. Given the wiring list of a channel routing problem, we define the maximum density as the maximum intersections of a vertical column with the horizontal net segments over all columns, assuming that each net occupies the routing of any channel with a number of tracks equal to or approaching the maximum density. Thus the maximum density is used to estimate the required channel width, which is crucial in chip planning. The problem is NP-complete (58,59). For over two decades, many researchers have tried to solve various channel routing problems. Several channel routers were developed to generate near optimal solutions: an optimal solution for the channel assignment problem (60), a dogleg channel router (61), greedy channel router (62), hierarchical channel router (63), three- or multilayer channel routing (64–67), 45° channel routing (68), gridless channel routing (69,70), parallel algorithm (71), segmented channel routing for FPGA's (72–75), constrained channel routing for analog and mixed signal circuits (76,77), crosstalk-minimum channel router (78,79). See (80,81) for a survey of other previous channel routing problems and their algorithms.

Global Routing

The problem of global routing is very much like a traffic problem. The pins are the origins and destinations of traffic. The wires connecting the pins are the traffic, and the channels are the streets. If there are more wires than the number of tracks

in a given channel, some of the wires have to be rerouted just like rerouting traffic. For the real traffic problem, all drivers wants to go to their destinations in the quickest way and may try different routes every day. Finally each driver selects the best route possible and the traffic pattern is stabilized. Intuitively, we can do the same for the routing problem. In global routing the usual approach was to route one net at a time sequentially until all nets are connected. To connect one net, we could use the maze runner for example. If some nets cannot be routed at the end, these nets must be routed manually. Obviously, the success of this kind of routing depends on the order in which we route the nets, and there is no systematic way of rerouting the nets. Ting and Tien (82a) used the following approach for gate-array designs:

- First, route every net as if it were the first net to be routed, that is, pay no attention to conditions at boundary overflows.
- After all nets have been routed, identify the boundaries that overflowed and the amount of overflow.
- Identify the nets that use these overflow boundaries and form a bipartite graph with one part of the vertices representing the nets and the other part of the vertices representing the overflowed boundaries. An edge connects a net to a boundary if the net uses the boundary. The bipartite graph shows the supply-demand situation among boundaries and nets, and a subset of nets is selected for rerouting. The criterion of selection is "greedy."

An interesting probabilistic approach called *simulated annealing* is due to Vecchi and Kirkpatrick (37). Basically, they first route the nets randomly. Then they change the routing pattern if the new pattern decreases the objective function:

$$F = \sum_{v=1} m_v^2$$

where m_v is the number of wires in the vth arc. Because this objective function is convex, a lower value of the objective function indicates that the wires are more uniformly distributed over the whole area.

A novel approach is to change the pattern even if the objective function increases. This happens with a small probability, say 1/100, and changes the pattern is the objective function decreases with probability, say 99/100.

As previously mentioned, many ingenious heuristic algorithms have been proposed. However, the fundamental question is Is there an algorithm which can be proved mathematically? (Here, we discount the algorithms, such as backtrack or branch-and-bound, which are classified as implicit enumerations). The amazing answer is "yes" using a linear programming approach. The algorithm was invented in 1949 by G. B. Dantzig. A serious reader should consult books on linear programming.

Recently, as VLSI chips become more tightly packed, multilayer routing with more than two layers is necessary. Several approaches have been proposed for global wire routing. One simple approach is to incorporate net weights into the global routing process by assigning high weights to critical nets. The other method is to assign net priorities. Critical nets are routed before less critical nets, so that their paths are much more direct than the others. A rerouting technique is used to solve the timing problem by first routing all nets to minimize some global objectives (such as total wire length or overall timing), and then rerouting the critical nets, net-by-net, based on net priorities.

A performance-driven global routing algorithm for custom chip design is presented in Ref. 82b. Interconnective delays are modeled, directly included, and incrementally updated during the routing process. The objective of routing is to maximize the minimum remaining delay slack. By having the maximum allowable delay from signal source to each sink terminal of the signal (which is obtained from the timing analysis) as a set of constraints during the routing of a signal net, the route (which is based on the A-search technique) uses the remaining net delay slack as the primary parameter for guiding connective path searching. It is shown that when interconnection resistance is comparable to the output driver resistance, minimizing the total net length is not always equivalent to minimizing the delay for a multiterminal net. Sometimes reducing delays at some critical sink terminals is achieved by increasing delays at some less critical terminals. Because the delay at each sink terminal of a signal depends on how the interconnective tree is constructed.

There are various approaches to the routing problem in two-dimensional arrays based on hierarchical wiring (82,83), sequential methods (84–87), simulated annealing (34,37), linear programming (88,89), multicommodity flow (90,91) and flat approaches (92,93). Global routing is NP-complete even in the case of one-bend routing of two-terminal nets (92).

Based on a binary-cut tree, a custom chip routing algorithm using linear assignment together with hierarchical net decomposition is proposed in Ref. 94. The algorithm is based on a top-down hierarchical scheme. At each level of hierarchy, the current routing problem is partitioned into two subproblems by assigning pins to channels on a cut line (or bisector). To find an optimal pin assignment for a cost function, a linear assignment minimizes the total summation of the costs subject to the capacity constraints of the channels (or holes). Then, to determine the path of nets inside each subregion, interface (or pseudo) pins are created on the cut lines and nets are broken into several parts that are processed independently. The process of cutting and linear assignment continues until all the nets are connected or all the boundaries are included in the cuts. Note that this approach produces many bends. With signals in the gigahertz range, the electrical characteristics of the packages require treating the signal lines as transmission lines. On most conventional interconnecting substrates, transmission line delay is linearly proportional to the distance traveled, but with multilayer thin films, delay becomes proportional to the distance squared. Because of the high series resistance of thin conductive lines and the high capacitance of these lines to ground caused by thin dielectrics. In practice, transmission lines are not perfectly uniform, that is, in the package level, significant reflections are generated from capacitive and inductive discontinuities along the transmission lines. Moreover, in a multilayer ceramic substrate of MCM, wires at different levels do not have exactly the same impedance. Such mismatches of line impedance cause reflections from the junction points, such as vias and bends. In general, xy plane-pair routing techniques used for multilayer routing yield many vias. If we minimize the number of segments per net (i.e., the number of bends), the number of vias is reduced proportionally. Therefore, propaga-

tive delays associated with discontinuities (e.g., see Refs. 5, 95–97) are minimized by careful design.

We call the process of assigning pseudoterminals to cutlines in each level of hierarchy *terminal propagation.* Terminal propagation depth (denoted as δ_i) for a net i denotes the number of levels of top-down hierarchy through which a net's pseudoterminals are allowed to propagate. In this way, the depth of terminal propagation controls the number of bends.

Moat Router

The final stage in detailed routing typically is routing the connections between the I/O pads and the core circuits. The area between the core and the pads is called the moat. Moat routing consists of nets whose pins lie on either or both the inside perimeter of the pad frame and the outside perimeter of the core circuit area. The moat between the pads and the core is divided into a number of concentric tracks, similar to channel routing except each track forms a circle rather than a line segment. The moat routing problem is to minimize the number of tracks required to connect all nets in the moat. The pad assignment that avoids vertical constraints and minimizes the total wire length between pads and terminals on the core is effectively solved by the linear assignment algorithm.

A similar problem is the case when all the points lie on a core frame periphery. The problem is to find the smallest enclosing rectangle with the least area among all feasible layouts. Gonzalez and Lee (98) presented a linear-time optimal algorithm for the case when all nets have exactly two terminals. They (98a) also presented a polynomial time approximation algorithm for multi terminal nets that generates a layout with an area at most 1.6 times the area of an optimal layout. Also note that a similar problem, minimum-congestion routing around a rectangle is polynomial time solvable for two-pin nets (99), and the problem of routing two-terminal nets around two equal-width rectangles to minimize the total area was presented by Gonzalez and Lee (100) with an $O(n \log n)$ time two-approximation.

The density of channel route is the maximum number of intervals that intersect any vertical line, that is, the size of the maximum clique in the corresponding interval graph. Clearly the density in a channel is a lower bound on the number of tracks required to route through the given channel. In fact, interval graphs are *perfect graphs,* meaning that the maximum clique size is equal to the minimum number of colors required to color the graph. Thus, the optimal number of tracks is precisely equal to the density. The left-edge algorithm (60) computes an optimal channel routing solution.

A circular-arc graph is similar to an interval graph except that the vertices correspond to arcs on a circle rather than intervals on a line. Unlike interval graphs, a circular-arc graph is not necessarily equal to its density. Thus, coloring the circular-arc graph is NP-complete.

It is proved by Tucker (101) that $\psi(G) \leq 2\omega(G)$. The theorem results in a two approximation algorithm for coloring a circular graph G. Find a maximum clique C in G in $O(n \log n)$ time (102), and color it using $|C|$ colors. Color the interval graph $G - C$ optimally using the left-edge algorithm, which requires at most $|C|$ colors in $O(n \log n)$ time (60,103). Thus, a coloring using at most $2|C|$ colors is computed, and the optimal coloring requires at least $|C|$ colors. The time complexity of the algorithm is $O(n \log n)$ time.

One may suspect that moat routing is analogous to coloring an circular-arc graph. Then, one may easily prove that moat routing under the restricted model is NP-complete, but the complexity for the case containing only two-terminal nets in not known. Furthermore, moat routing has a special property that the circular moat creates the possibility of many different routing paths, independent of the assignment of nets to tracks. Specifically, a net with m pins can be routed in m different ways, each corresponding to a complete track with the span between two adjacent pins removed. Because nets in a moat can be routed in multiple ways, the notion of density applied to channel routing is not directly generalizable to moat routing. Thus, given restricted moat routing, we should resort to an approximation algorithm based on finding the associated circular-arc graphs. An approximation algorithm called an iterated maximum independent set heuristic is proposed by Ganley and Cohoon (104). The idea is to construct a circular-arc graph G' (we call it an extended circular-arc graph) considering all different routing paths as additional arcs and iteratively peeling off a maximum independent set (MIS) from G until no more MIS remains. The time complexity of the heuristic is $O[n(nm)^2]$ time.

Over-The-Cell Router

The conventional channel is defined as a rectangular area with terminal rows along its top and bottom. The main role of the channel router is to connect the given terminals according to the net list with minimum area. In two-layer standard cell design methodology, the M1 layer is typically used for connections internal to the cell, and the M2 layer is available for routing over the cell.

Recently, the concept of over-the-cell (OTC) routing was introduced to minimize layout area. In OTC routing, the cell layout area and the channel area between two cell rows are used as a routing resource. Some heuristics have been developed to achieve 20–35% reduction in channel height compared with those for non-OTC channel routers. A recent algorithm yields as much as 65% reduction in the channel height with a triple-layer OTC router. Recently, Kim and Kang (105) present a new channel routing algorithm that uses three metal layers in the OTC area. Metal layers can be dynamically located for horizontal and vertical connections in OTC area even for the case where cyclic constraint exists. The algorithm relocates as many critical net segments as possible for the OTC area and, thus, the real channel space.

High-Performance MCM Router

With the accelerating complexity of semiconductors shown in Fig. 1 (1), packaging must address the needs of consumer products, personal computers, workstations, midrange computers, mainframes, and supercomputers. Electronic packaging is an art based on the science of establishing interconnections and fulfilling system functions. Continuous advances in the speed and integrative scale of integrated circuits have created ever greater demands for higher density packaging to ensure reduced interconnective delays for improved electrical performance.

Packaging is becoming a limiting factor in translating semiconductor speed into system performance. Moreover, increasing circuit count and density in circuits have been continuing to place further demands on packaging. To minimize

the delay, chips must be placed close together. Thus, the multichip module (MCM) (106) technology has been introduced to improve system performance significantly by eliminating the entire level of interconnection. MCM is a packaging technique that places several semiconductor chips, interconnected in a high-density substrate, into a single package. This innovation led to major advances in interconnective density at the chip level of packaging. Compared with single-chip packages or surface-mounted packages, MCMs reduce circuit board area by 5 to 10 times and improve system performance by 20% or more.

Therefore, MCM is used in a large percentage of today's mainframes to replace the individual packages. The size of MCMs varies widely (107): 10–150 ICs, 40–1000 I/Os per IC, 1,000–10,000 nets, where the low end is ceramic/wire-bond, the high end is thin film/flip chip (maximum linear dimension is now up to 4–6 inches for thin-film MCMs, up to around 8.5 inches for ceramic MCMs, and up to 18 inches for laminated MCMs).

Physical design in the MCM environment differs from classical VLSI design in a number of important ways. Performance is of overriding importance in MCM design, because that is the primary reason for choosing MCMs over conventional single-chip packages. Thus, MCM physical design algorithms must be driven by performance constraints. This involves careful consideration of minimizing transmission line effects, such as crosstalk, reflections, and the effects of crossings, bends and vias, as opposed to simple lumped-capacitor approximations commonly used in single-chip packages. Another important difference is that some types of MCMs, such as ceramic and laminated MCMs, require several tens of wiring layers. For example, the ceramic MCM designed by IBM uses over 60 layers of signal, power, and ground wiring (106). This imbues the MCM routing problem with a three-dimensional flavor, which is lacking in existing VLSI routing where the number of layers rarely exceeds three.

It can be argued that printed circuit board (PCB) wiring, which has been studied by many researchers over the past two decades, also handles comparable numbers of layers. However, MCM high-density substrates have many unique features, such as blind, buried, stacked and segmented vias, which are not commonly considered in PCB routing algorithms. Also, the need for simultaneous consideration of performance issues, which are rarely brought up in PCB wiring algorithms, provides stimulus for new and innovative design approaches (3–7,9–11).

Several wiring layers are required to route the large number of interchip connections. Thus, the problem is three-dimensional. A multilayer routing strategy for high performance MCMs is to route all nets optimizing routing performance and to satisfy various design constraints (e.g., minimizing coupling between vias and signal lines and minimizing discontinuities, such as vias and bends).

There are several new and interesting MCM routers for this new packaging technique (3,6,7,9,10). Automatic layout of silicon-on-silicon hybrid packages developed at Xerox PARC is presented in Ref. 3. Placement determines the relative positions of the ICs automatically or interactively and organizes the routing areas into channels. The hybrid routing uses a topological model that reduces the complexity of hybrid routing by abstracting the geometrical information. Computation of geometry is deferred until needed. Global routing attempts to find a minimum Steiner tree based on symmetric expansion from all pins.

OTHER ISSUES IN PHYSICAL DESIGN FLOW

Compaction is simply the task of compressing the layout in all directions so that the total area is reduced. By making the chip smaller, wire lengths are reduced which, in turn, reduces the signal delay between circuit components. At the same time, a smaller area means that more chips are produced on a wafer which in turn reduces the manufacturing cost. The expense of computing time, however, mandates that extensive compaction is used only when large quantities of IC's are produced. Compaction must ensure that no rules regarding the design and fabrication process are violated during the process.

Design verification encompasses many different techniques to guarantee the specification of a VLSI logic chip. The types of analysis done on VLSI logic chips include functional simulation, delay calculation, delay simulation and timing analysis. A chip designer may choose to do all of the above analysis or only a subset. All simulation routines require a model which describes the function of the logic chip. This model is created by interconnecting functions available in the design system library using a standardized descriptive language. Each circuit in the library has a logical model coded by the circuit designer and provided to the chip designer via the technology database described in the last section.

The types of functions in a gate array design system library include primitive functions (i.e., NAND, NOR, AND, and OR) and complex function (i.e., AO, OA, AOI, and OAI). The standard cell library includes all the functions in the gate array library with the addition of macros. A typical list of macros includes RAMs, PLAs, ALU, and register macros. The logical models for simple circuit functions are coded by Boolean primitives. The logical models for macros are coded as flat (Boolean) models using the primitives for test generation, but behaviorals are coded for functional simulation. Behaviorals are programs which interact with the simulator during the actual simulation and reduce the overall CPU time when simulating an entire chip.

Once the logical model of the chip is coded, the designer must code patterns for exercising the logic. The first step is defining all the primary inputs and outputs of the chip and organizing them into groups which form the data flow. Each group of inputs or outputs is given a variable name, and that name is used to set the input patterns and monitor the results at the output. The input pattern modes available in a typical simulator include binary, hexadecimal, and decimal. A well-structured input language offers DO loops for repetitive operations and a WAVE facility for clock pulse generation. After functional simulation is completed, the next step in design verification is delay calculation. Delay calculation is run before placement and wiring by estimating the amount of capacitance per fan-out. An average capacitance value for a single fan-out is hard-coded into the delay program. The delays are obtained by two different techniques. For the simple books in the library (NOR, NAND, etc.), the device parameters are stored in the technology rules. The parameters include device width and length and input capacitance. These parameters

are used as input to a device model which uses a numerical integrative scheme to compute the book delays.

This scheme proves too costly and inefficient for large macros, however. These functions have precalculated delays which are stored in tables of delay versus capacitance. The delay calculator either interpolates or extrapolates the delay from the points given in the delay tables. The macro function (i.e., RAMs, PLAs, etc.) delays are divided into two parts: the skin delay and the body delay. The skin delays are for the input and output blocks of the macro and depend on input transition and output capacitance. These delays are precalculated and stored in the database. The body delay depends on a particular path in the macro and is embedded in the behavioral.

Delay calculation is typically run after the logic is placed and wire on the chip. Actual wiring lengths for poly and metal wires are used to compute the wiring capacitance. Each wiring level has a precalculated capacitance value per unit length which is stored in the technology rules. Certain assumptions are made with about the amount of wire crossings and the amount of parallel and fringe capacitance. In a design system, it is too costly to compute the exact capacitance for each net because the positional relationship to other wires. The input gate capacitances are stored in the rules and are added to the wire capacitance to compute the total net capacitance. The capacitance values are passed to the delay calculator, and the delays are computed for each book and stored in a file. Delays computed by the delay calculator are typically three-sigma worst case.

HIGH-PERFORMANCE LAYOUT DESIGN WITH SUBMICRON TECHNOLOGIES

IC designers moving to deep submicron technologies face big challenges. Initial design projects have experienced unexpectedly long design cycles, a larger than expected number of design iterations, problems getting chips to operate at target clock speeds and surprises with die size late in the design cycle. The effects of deep submicron geometries, high clock speeds, and soaring gate counts all create new design problems that are not addressed by existing tools and methodologies. The limitations of available tools and methodologies are clear. Logic designers, who once needed to know little about the physical implementation of their devices to successfully complete their designs, now must have access to key physical design information early in the design process. Without this information, timing delays, routability, and power dissipation are not accurately predicted, and the logic designer has no way of knowing if basic design constraints, such as functionality, cost, power, and speed are being met. The result is often big surprises late in the design cycle. Gate delay has decreased by about 30%, but interconnect delays have not been reduced as quickly. For 0.5 micron technologies, the solution lies in enabling early design improvement by providing logic designers with insight into physical implementation realities without forcing designers to do detailed placement and routing.

Because information about the actual physical implementation of a design was not necessary to make accurate predictions, logic and physical design were highly decoupled. Conventionally, only a single iteration between the logic and the physical design phases was necessary to successfully complete a design. For deep submicron designs, this model does not hold true. Delay, routability, size, and power dissipation are no longer predicted accurately unless information about the physical implementation is available to the logic designer. Interconnect, for example, now plays such a dominant role in predicting delay and power dissipation that it cannot be loosely estimated.

Timing, routability, size, and power problems are not discovered until after detailed placement and routing. Multiple, lengthy iterations ensue between the logic and physical designer to repair the problems. The end result is that iterations between logic and physical designers grow from 1–3 iterations for 0.8 micron designs, to more than 10 iterations for 0.6 micron and smaller feature size designs.

Only by developing a floor plan of a design and precisely predicting the placement of cells and the routing of nets are interconnect effects accurately forecast during the logic design phase. Because a floor plan provides a high-level abstraction of the eventual physical design implementation, it is created and modified quickly. As a result, multiple iterations with synthesis and timing analysis tools are completed quickly, and timing, size, and power constraint violations are resolved. Because the logic designer has access to key physical design information, problems are solved during the logic design phase, where they are least expensive to fix. Because the final logic design takes into account the physical implementation of the design, a single placement and routing cycle is achieved and design cycle times are significantly reduced.

Increased interconnect resistance affects the load seen by a driving gate and pin-to-pin interconnect delay. The increase in coupling and interlayer capacitance caused by interconnects running closer together and the heightened use of multilayer interconnect technologies must also be considered. Failure to consider these submicron effects causes significant predictive errors. After full placement and routing, most physical design groups accurately predict delays if they have taken into account the submicron effects discussed previously. But during logic design, tools and methodologies in common use today often predict delays with errors in excess of 100–200, and the design fails.

Determining the routability of a design is critical to predicting both size and performance. Today, relatively simple calculations based on total cell area and number of nets are typically used to estimate design size early in the design process. The estimates do not consider routability. Given the mounting routability challenges facing deep submicron design projects, predictions based on these calculations are extremely inaccurate and cause major surprises late in the design process.

Design routability has become a big problem for a number of reasons. One important factor is the continued explosion in design complexity. There are more cells and nets in a design to place and route optimally, and the likelihood of considerable routability problems is increased. A second factor is the use of top-down design methodologies, specifically synthesis tools. They have increased the average number of connections per net by 50–100 throughout the chip. A third factor is the increased number of macro blocks or mega cells with a high number of input and output pins, increasing congestion and more challenging routing problems. Finally, the use of larger and larger buses causes special problems for routers. De-

pending on the design, one or more of these issues needs to be considered to accurately determine the routability of a design.

Fixing routability problems is also extremely difficult. Because most place and route tools operate flat, it is difficult to isolate routability problems and fix them incrementally. In addition, because place and route tools typically immediately flatten design hierarchy, it is impossible for them to take advantage of the highly structured major portions of a logic hierarchy. As a result, resolving routability issues requires many iterations and time.

Clock-Tree Synthesis

In a synchronous VLSI design, which carries the heaviest load and switches at high frequency, clock distribution is a major source of power dissipation. Also, circuit speed and chip area have been an important consideration, and the delay on the longest path (phase delay) through combinatorial logic and the maximum skew among the synchronizing components should be minimized. There has been active research in the area of high-performance and low-power clock routing. This chapter gives an overview of a number of combinatorial aspects and highlights the state of the art of currently active research in clock-tree synthesis.

Reduction in power consumption without sacrificing processing speed is an increasingly important objective in integrated circuit design nowadays. One reason is the longer battery life required for the growing class of personal computing devices, such as portable desktops and wireless communication equipment. Another reason is the dramatic decrease in chip size and increase in transistor count and clock rate for integrated circuits. Power is reduced at various design levels, including circuit, logic, register-transfer, behavior, and system levels. Clock is the fastest and most heavily loaded net in a digital system. Power dissipation of the clock net contributes a large fraction of the total power consumption. The clocking circuitry in an adaptive equalizer consumes 33% of the total power. In a microprocessor, 18% of the total power is consumed by clocking (108) because the clock frequency is typically several times higher than other signals, such as data and control. There are clock signals which synchronite elements. Both clock and data signals are delayed by the combinational blocks delays introduced by wires.

For a circuit to function correctly, clock pulses must arrive nearly simultaneously at the clock pins of all clocked components. Performance of a digital system is measured by its cycle time. Shorter cycle time means higher performance. At the layout model, performance of a system is affected by two factors, signal propagative time and clock skew. The difference in arrival times between a single pulse arriving at two different clocked components is referred to as clock skew which must be within a certain tolerance. Using advanced routing tools to minimize total wire length is helpful in reducing resistance of wires. But in high frequency applications, clock skew and phase delay should be considered to attain a desirable chip performance. It is said that the clock skew must be less than 5% of the critical path delay time to build a high-performance system. Timing critical nets with a high fan-out requires several layers of buffering to drive all the leaf cells.

With the scaling of device technology and die size, interconnective delay now contributes up to 70% of the clock cycle in dense, high-performance circuits. The earlier works on timing-driven routing problem are in (109). A minimum rectilinear Steiner tree (MRST) approach for interconnecting the terminals of a clock net is not necessarily the best in various applications. To reduce phase delays and supply sufficient driving current, several levels of buffers are added to create a so-called multistaged clock tree [refer to Fig. 11(b)]. To optimize the buffer placement inside each group and between groups, we simply apply the variations of H-tree (called hierarchical matching tree (110). Then, we generate the rectilinear clock net topology to minimize the clock skew, Elmore delay (111), and wire length, simultaneously.

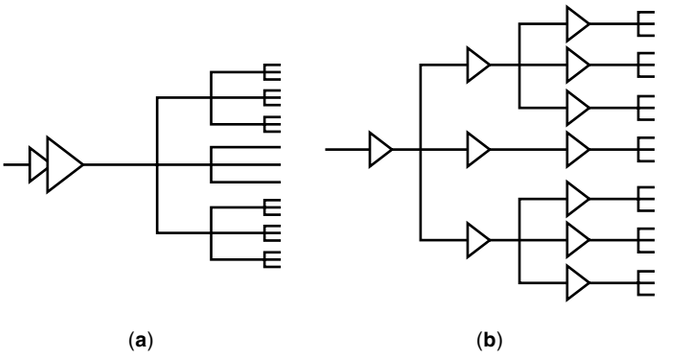


Figure 11. Clock network topologies: (a) Single driver scheme; (b) Distributed buffers scheme.

Because of its fidelity to SPICE-computed delay, Elmore delay is a good performance objective for constructing high-performance routing trees. Figure 12 shows an instance of Elmore delay estimation for a hierarchical distributed buffer tree.

Because interconnect resistance and capacitance are usually proportional to the edge length, we see that the delay has a quadratic relationship to the length of the source-sink path, suggesting a min-radius criterion. However, the C_j term implies that Elmore delay is also linear in the total edge length of the tree which lies outside the $n_0 - n_i$ path, suggesting a min-cost criterion.

The relative size of the driver resistance heavily influences the optimal routing topology (ORT). If r_d decreases, the ORT

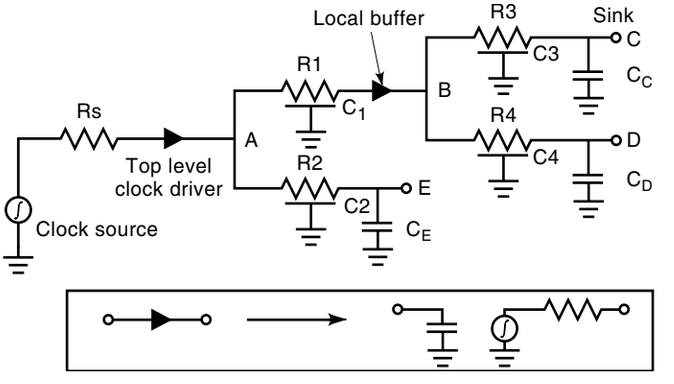


Figure 12. An instance of Elmore delay estimation for a hierarchical, distributed, buffer tree. For example, $d(A) = \beta R_s(C_1 + C_2 + C_3 + C_4 + C_C + C_D + C_E)$, $d(B) = d(A) + \alpha R_1 C_1 + \beta R_1(C_3 + C_4 + C_C + C_D)$.

tends to a star topology. Elmore delay implies that the number of Steiner points in the source-sink path should be minimized and the Steiner points "shifted" toward n_0 (i.e., branches off the source-sink path should occur as close to the source as possible).

It is well understood that reducing the wirelength reduces the delay sensitivities at the leaf nodes because small process variations in the wire length result in smaller changes in the overall delays and, consequently, smaller skew. Then the objective is to make the delay sensitivities at the leaf nodes small enough (by shortening wires) so that the upper bound on the skew is acceptable without elongatting wires excessively. It is important to point out that the wirelengths are optimized in a bottom-up manner from the leaf-nodes to properly consider the possible changes in the upstream sensitivities.

The change in the Elmore delay to any node n downstream of branch 1 is given by $\Delta T_{D_n} = \Delta R_1(C_{d_1} - C_1)$. We see that the skew of binary tree-like clock nets is extremely sensitive to changes in the wire lengths of the branches closest to the root of the tree. Small changes in the lengths of such branches, therefore, have a large effect on skew.

Delay, power, skew, area, and sensitivity to process variations are most important concerns in current clock-tree design. As with other signal integrity issues, clock layout problems force designers to evaluate a set of complicated, opposing alternatives, such as area, delay, skew sensitivity, and power consumption.

In the global routing phase, route shapes for all nets are determined on a two-dimensional grid, to minimize the maximum routing density and minimize the interconnect delays in the nets. In VLSI routing, the primary objective is to minimize wire length. This is equivalent to minimizing delay when the lumped-capacitor model is used. However, when the Elmore or second-order model is used, minimum wire length does not necessarily imply minimum interconnect delay. Consequently, conventional minimum Steiner tree algorithms developed for VLSI routing are inadequate for MCMs.

Performance-driven, tree-generating algorithms are currently a topic of considerable research. Using a first-order delay model, Cong et al. (112) show that the delay of a net consists of three terms. The first term is proportional to the total wire length of the net. The second term is proportional to the sum of the path lengths from the source to each of the sink vertices in the tree. The third term is related quadratically to the path lengths. Based on this analysis, they introduce the concept of an A-tree, whose interesting property is that the second term is always minimized and the first and third terms are closely related, so that minimization of one leads to minimization of the others. They propose a near-optimal algorithm for A-tree construction and find experimentally that the algorithm reduces delays up to 43% compared with a very good Steiner tree algorithm.

A different approach based on a second-order delay model is proposed in Ref. 113. In this approach, the tree for a net is generated constructively by adding one sink at a time. During the ith step, a point k is found in the current tree, so that, when a minimum-length path from the current sink s_i to k is introduced, the second-order delays to sinks $s_1 \ldots s_i$ are minimized. The point k is found efficiently by performing a "trial" connection from s_i to a set of points in the current tree and incrementally updating the admittance and coefficients to compute the delays. Although the algorithm is greedy, it finds trees with significantly smaller delays (up to 50% less) than a sophisticated Steiner tree algorithm (114).

In the PowerPC clock design methodologies (115), the first stage of clock design is during synthesis. The synthesis performs load balancing and duplication of clock buffers to minimize clock skew. The objective of load balancing is to balance capacitive and resistive loadings among the groups and the number of pins among the group. Capacitive loading is approximated by the Elmore sum of clocked cells' input loading capacitances. Resistive loading of each group is approximated by the half perimeter of the smallest bounding box enclosing all of the clocked pins in the group. The purpose of balancing the number of clocked pins is to distribute the wiring congestion over the routing area. All three parameters are incorporated into a cost function to be optimized. Here clock buffers are inserted based on net fan-out and pin capacitances. The second stage is a physical clock design. First, clustering of regenerator (circuits that split the master clock into phases) and estimation of skews inside circuit blocks is performed. The second stage of physical clock design is optimizing the wire widths of the main trunks of the clock tree. The final stage of physical clock design is final buffer selection in the clock network.

Two different clock distribution shemes are used in PowerPC designs: an H-tree clock distribution network [Fig. 13(a)], and a multilevel buffered clock grid [Fig. 13(b)]. The H-tree style designs are used for designs targeted for lower power and desktop markets, whereas the multilevel buffered grid design style has been used for high-performance processors with larger clock distribution area.

There are many works related to high-performance clock physical network designs (116,117). In this article, we explore a number of important contributions to clock synthesis, and we use the following notations:

- T: A clock tree with a driver w_0 at the root (clock source, N_0) and a set of s sinks $\{N_1, N_2, \ldots, N_s\}$.
- S: a set of t Steiner points $\{S_1, S_2, \ldots, S_t\}$ whose locations are to be determined.
- dist(N_i, N_j): Manhattan distance between N_i and N_j.
- w_i: ith wire segment or buffer.
- X: $X = (x_0, x_1, \ldots, x_{n+m})$, where n is the number of wire segments and m is the number of buffers, a wire- and buffer-sizing solution.
- r_i, c_i: resistance and capacitance of w_i, respectively.
- U_i, L_i: upper bound and lower bound of the size of w_i.
- x_i, ℓ_i: size and length of w_i.
- P_i: all wires and buffers on the path from the source to sinks N_i.
- T_i: All wires and buffers in the subtree of T rooted at w_i.
- $Ans(w_i$ to the nearest upstream buffer or the root.
- $Dec(w_i)$: all wires, buffers, or sinks on the path from w_i to the nearest downstream buffers or sinks.
- R_i: upstream resistance of w_i, $R_i = \sum_{w_i \in Ans(w_i)} r_j$.
- C_i: downstream capacitance of w_i, $C_i = \sum_{w_j \in Dec(w_i)} (C_j + c_j) + \sum_{N_j \in Dec(w_i)} c'_j$, where c'_j is the capacitance of sink N_j.
- D_i: Elmore signal delay (Fig. 12) at sink N_i, $D_i = \sum_{w_j \in P_i, wire} r_j(C_j + c_j/2) + \sum_{w_j \in P_i, buffer} r_j C_j$.

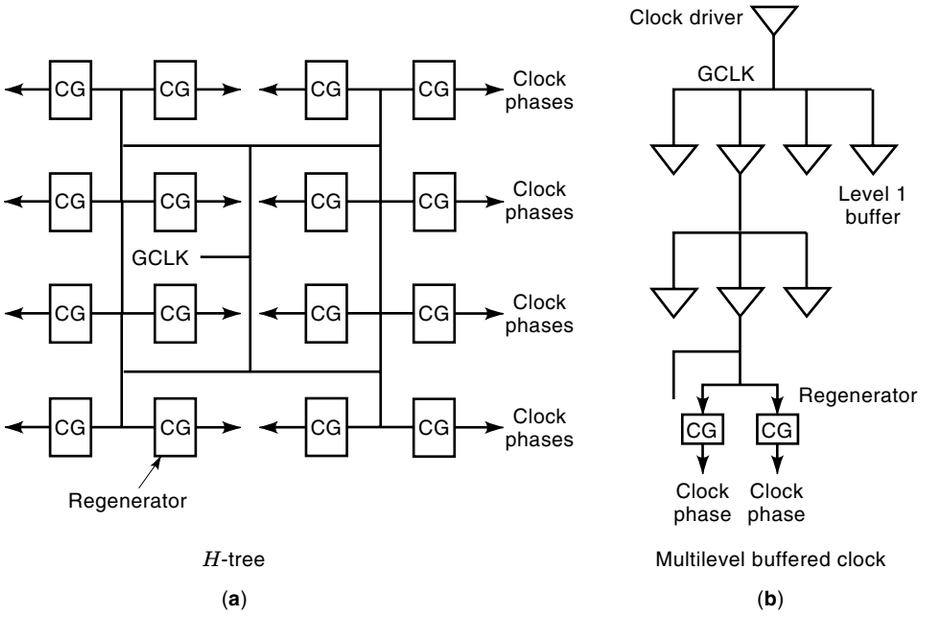


Figure 13. Typical PowerPC clocking schemes.

S: clock skew defined as the maximum difference in the delays from the clock source to clock sinks, $S = \max_{i,j}|D_i - D_j|$.

A: area of a clock tree: $A = \sum_{i=1}^{n} \ell_i + \sum_{i=n+1}^{n+m} x_i$.

P: power consumption proportional to nCV_{dd}^2, where nC is the sum of capacitance times transitions (switching activity) needed to compute an operation and V_{dd} is the supply voltage.

The signal net N is to be embedded in an underlying graph $G = (V, E)$ with $N \in V$. The graph G is associated with Hanan's grid (41) and has variable edge costs. Each edge $(i, j) \in E$ has a cost $d(i, j)$ equal to the routing cost between node $i = (x_i, y_i)$ and node $j = (x_j, y_j)$, that is, the rectilinear distance between the two points $(|x_i - x_j| + |y_i - y_j|)$. The nodes in V correspond to the interconnection point (called Hanan's point) in Hanan's grid. Hanan's grid is generated by drawing a horizontal straight line and a vertical straight line crossing each point in N. The cost of T is defined as $cost(T) = \sum_{(i,j)\in T} d(i, j)$. The Steiner tree (118) is a routing tree T in G that spans N. Hanan's point denoted as H can be as large as n^2. We set $H = H - N$. A Steiner tree for a set N contains at most $(n - 2)$ other points of set $S \in H$ called Steiner points on the plane. The Minimum Rectilinear Steiner Tree (MRST) problem, given a set of N of n points in the plane, is to determine a set S of Steiner points so that the tree cost over $N \cup S$ has a minimum rectilinear cost. The problems are known to be NP-hard for a long time (119).

Figure 10 shows a set of alternative Steiner trees, each of whose objective is different. In this section, we exhibit a number of results on constrained Steiner trees for clock networks.

Hierarchical, Recursive, Matching Tree. Let us briefly describe the algorithm proposed by Kahng, Cong, and Robins (120). Given a set N of randomly distributed clock pins and a distinguished pin called a source, they first match the closest pairs using minimum edge-weighted matching (MEWM). A balance point is computed by finding the point p along the straight line connecting the roots of the two subtrees so that the difference in path lengths from p to any two leaves in the combined tree is minimum. Then, another MEWM is performed on generated balanced points. In this manner, a height-(= log n) balanced binary clock tree is constructed by recursive geometric matching in a bottom-up manner. We call the generated tree the rectilinear hierarchical matching tree (RHMT). The RHMT produces a near-optimal solution in terms of total wiring cost and clock skew when clock pins are evenly distributed over chip area.

Bounded-Skew Steiner Tree: The Deferred-Merge Embedding Approach. Based on the hierarchical matching tree, the deferred-merge embedding (DME) algorithm (121) improves the existing method and consists of two stages. In a bottom-up phase, a tree of merging segments is constructed that represents loci of possible placements of internal nodes (or Steiner points) in a zero-skew tree, and, in a top-down phase, a tree is embedded determining exact locations for the internal nodes in T. For node v with children a and b, the merging region of v corresponds to the set of all locations at which subtrees T_a and T_b are joined to v with minimum wiring cost while still maintaining the skew bound B.

In general, to ensure correct clock operation under a required clock period P, the allowable clock skews between two adjacent flip-flops i and j are (1) to avoid zero-clocking with negative skews, $D_i \le D_j$:

$$D_j - D_i \le MIN(D_{\text{logic}}) + D_{ff} - D_{\text{hold}}$$

and (2) to avoid zero-clocking with positive skew, $D_i \ge D_j$:

$$D_i - D_j \le P - (D_{ff} + MAX(D_{\text{logic}}) + D_{\text{setup}})$$

where D_i and D_j are clock arrival times, $MAX(D_{\text{logic}})$ and $MIN(D_{\text{logic}})$ denote the longest and shortest path delays of the combinational block between two FFs, and D_{ff} is the delay in a FF.

However, the formulation of the previous algorithm does not address the skew constraints. In practice, only minimizing the skew is not an actual design requirement. We should allow some tolerable (or sometimes useful for lower power) skew with which the system functions correctly. Bounded-skew clock routing under the Elmore delay model is presented in Ref. 122 and its problem is as follows:

Minimum-Cost, Bounded-Skew Routing Tree Problem: Given a set of $N = (N_1, N_2, \ldots, N_s)$ of sinks in a plane and a skew bound B, find a routing topology G and a minimum-cost clock tree T that satisfies $D_{\max} - D_{\min} \leq B$.

The section proves several key properties of the deferred-merge embedding regions.

Minimal Steiner Trees with Bounded Path Length. Let R be the length of the direct path from the source to the farthest sink and ϵ be a nonnegative, user-specified parameter. The shortest path between u and v in graph G is $\text{dist}_G(u, v)$. The shortest path between u and v in tree T is $\text{dist}_T(u, v)$. The radius of node $v \in G$ is $\text{radius}_T(v)$ (i.e., $\text{maxdist}_T(u, v)$, $\forall u \in V$). The method of (123) constructs a bounded path length minimal Steiner tree (BMST) with radius at most $(1 + \epsilon) \cdot R$ by using an analog of the classical Kruskal's MST construction. The tree cost is empirically at most 1.19 of that of an optimal BMST.

Given the routing graph $G(V, E)$ in Manhattan space, find a minimum cost routing tree with $\text{radius}(S) \leq (1 + \epsilon) \cdot R$.

A heuristic algorithm for the problem is as follows: The Kruskal algorithm adds an edge (u, v) in G to MST, or equivalently, merges two partial trees t_u and t_v by the edge (u, v) if (u, v) is the least weight edge among the available edges and the merged tree satisfies the path length bound $(1 + \epsilon) \cdot R$ from the farthest sink. Negative-sum exchange is defined as a sequence of T-exchanges where the sum of weights of exchange is negative. An exact algorithm starts from any solution tree, finds the negative-sum exchange, converts the solution tree to a new solution tree by exchanging edges, and iterates until no more possible exchanges(s) are found. Let us denote the search tree as τ. Each node in τ represents a spanning tree. The root of τ is the initial solution. A child node is generated by a T-exchange from its parent node. Note that one reaches any spanning tree from the initial tree by a series of at most $(V - 1)$ T-exchanges. Because the number of possible T-exchanges in a tree T is $O(EV)$, a node in τ has $O(EV)$ children. So τ has $O(E^n V^n)$ nodes where n is the depth of τ.

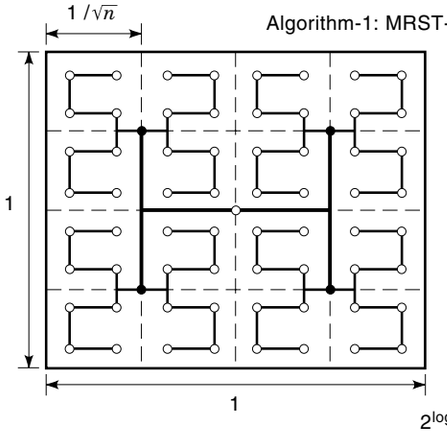
Trade-off between Cost and Skew. We denote by $L(T)$ the lower bound on wire length (cost or weight) of a worst-case tree T in the unit square.

Kahng, Cong and Robins (120) showed that, for random sets of terminals chosen from a uniform distribution in the unit square, the total wire length of rectilinear hierarchical matching tree (RHMT) is, on average, within a *constant* factor of the total wire length of the optimal Steiner tree. The constant is at most two (124).

Then, the bound is used for establishing bounds on wire length and clock-skew of the combined approach of MRST and RHMT (124,110) (refer to Fig. 14).

Theorem. The total weight of a worst-case MRST-RHMT or RHMT-MRST in the unit square is at most twice the total weight of a worst-case MRST.

Many systems uses clock trees with a buffer at each internal node and short daisy chains at the clustered leaves. By properly adjusting the size of the cluster at the leaves, one can tradeoff among skew and wire length. Minimizing the wirelength or the number of buffers also leads to power savings.



Algorithm-1: MRST-RHMT construction

Input: a set of clock pins N

Output: A clock tree topology

Step 1. Construct a RHMT.

Step 2. Reconstruct a set of subtrees of RHMT at Level k using the minimum rectilinear Steiner subtrees (MRSS).

$$L[\text{MRST-RHMT}(k)] = \sum_{i=1}^{2^{\log n - k}} L[\text{MRST}(k)] + \sum_{i=k+1}^{\log n - 1} L[\text{MRST}(i)]$$

$$= \sum_{i=1}^{2^{\log n - k}} (2^k - 1)\sqrt{n} + \frac{1}{2} \sum_{i=k+1}^{\log n - 1} \sqrt{n}/2^i$$

Figure 14. MRST-RHMT construction.

Minimum-Cost, Rectilinear, Distance-Preserving Tree. The shortest distance between the source and a given sink i in G is denoted as r_i. The shortest distance between the source and a given sink i in T is denoted as c_i.

The problem of a minimum-cost, rectilinear, Steiner distance-preserving tree (MRDPT) (46) seeks a minimum-cost tree with a special property such as $c_i = r_i$, for every sink i (refer to Fig. 5).

It is known that the minimum-cost Steiner distance-preserving tree (125) (or min-cost, shortest path Steiner tree (117) in general graphs is NP-hard (126). The complexity of the problem in a planar graph has not been known. See Ref. 117 for the history.

A typical approach for finding MRDPT as in (125) is as follows:

1. Partition the plane into quadrants Q_0, Q_1, Q_2, and Q_3. The partitioning of the plane divides the terminals into a one-quadrant MRDPT problem (1Q MRDPT) as depicted in Fig. 1(c).
2. Solve the 1Q MRDPT problem for Q_0, Q_1, Q_2, and Q_3, independently, obtaining an MRDPT $T(Q_0)$, $T(Q_1)$, $T(Q_2)$, and $T(Q_3)$.
3. Merge the solutions for each quadrant, thus obtaining the MRDPT T.

Cho (127) was concerned with finding $T(Q_0)$ whose sinks are in the quadrant Q_0 (i.e., right, upper corner of the plane). Given the solutions for each quadrants, the MRDPT T is found in polynomial time (125). Thus, the underlying graph of the 1Q MRDPT (we call it just MRDPT) is called *flow graph* $G = [A, V = (H \cup N)]$ so that there is a directed arc in A from i to j in V if $r_j \leq r_i$, $x_j \leq x_i$ and $y_j \leq y_i$, that is, every arc is oriented toward the source s. Thus, arcs are embedded using some monotone or "staircase" path between the source to any sink.

A tree is directed in-tree rooted at node s if the unique path in the tree from any node to node s is a directed path. Observe that every node in the directed in-tree has outdegree 1. Note that all paths from a sink to the source have the same length in G. Thus the problem of MRDPT is to find a directed in-tree from G with minimum cost. An exact algorithm based on min-cost flow transformation is given in (127).

Minimum-Cost, Bounded-Skew, Bounded-Delay Clock Tree. The wire- and buffer-sizing problem is defined as follows:

The Clock-Tree Wire- and Buffer-Sizing Problem
Given: A clock tree T with source N_0 and sinks $\{N_1, N_2, \ldots, N_s\}$, wire segments $\{w_1, w_2, \ldots, w_n\}$, buffers $\{w_0, w_{n+1}, w_{n+2}, \ldots, w_{n+m}\}$, upper bounds $\{U_0, U_1, \ldots, U_{n+m}\}$, and lower bounds $\{L_0, L_1, \ldots, L_{n+m}\}$.
Objective: Find an X that minimizes $\max_{1 \leq i \leq s} D_I$, S, P, A, and/or Δ.

An approach by Chen et al. (128) is based on Lagrangian relaxation so that the delay constraints are relaxed into the objective function by introducing Lagrange multipliers λ_i and δ_i, where λ_i and δ_i are the Lagrange multipliers associated with the delay constraint $D_i(X) \leq D_{\max}$ and $D_i(X) \geq D_{\min}$, respectively. The constraints are relaxed, scaled with Lagrangian multipliers into its objective function, and then the subproblems resulting from dynamically adjusting the Lagrangian multiplier are iteratively solved. Then,

$$\begin{aligned}&\text{Minimize } \alpha D_{\max} + \beta P + \gamma A + \delta(D_{\max} - D_{\min}) \\ &\quad + \textstyle\sum_{i=1}^{s} \lambda_i [D_i(X) - D_{\max}] + \sum_{i=1}^{s} \delta_i [D_{\min} - D_i(X)], \\ &\text{subject to } L_i \leq x_i \leq U_i,\ 0 \leq i \leq n+m,\ D_{\max} \geq 0,\ D_{\min} \geq 0\end{aligned}$$

Note that $D_{\max}$ is a variable introduced to minimize maximum delay and $D_{\min}$ is introduced to minimize clock skew. Chen et al. (128) presented an algorithm for simultaneously optimizing the previous objectives by sizing wires and buffers in clock trees. The algorithm, based on the Lagrangian relaxation method, effectively minimizes delay, power and area, simultaneously, with low skew and sensitivity.

However, the formulation of the previous algorithm does not address the skew constraints. In practice, minimizing only the skew is not an actual design requirement. We should allow some tolerable (or sometimes useful for lower power) skew with which the system functions correctly. Bounded-skew clock routing is presented in Ref. 122. This method, however, considers only the skew bound and does not control the maximum source-sink delay. Long wires require more buffers and cause slower rise and fall times. More buffers and slower switching result in higher power dissipation. A power optimizing clock routing algorithm with bounded skew and bounded maximum source-sink delay under the Elmore delay model is presented in Ref. 129. In this algorithm, delays are controlled by buffer sizing rather than by controlling the wire lengths, the clock tree is an equal, source-sink, path length Steiner-tree regardless of skew bounds (it is a zero-skew tree under the linear model), and finally the routing cost becomes large when non-zero skew is required.

Thus, Oh et al. (130) proposed allowing the user to specify different delay bounds for each individual sink, which lead to a further reduction of the routing cost. In case of clock routing, the required signal arrival times among sinks are made to differ. In addition, if the combinational delay between two FFs violates the short path delay constraint, common practice is to insert delay elements on the short path or increase the wire length. However, one cannot arbitrarily increase the length a wire because it may violate the required arrival times of other sinks. These observations motivated development of a method for controlling the path lengths so that any delays lie beween given upper and lower bounds. Variables of the proposed mathematical programming formulation are the edge lengths of the trees. The following formulation leads to a simple linear programming problem under the linear delay model which is solved optimally in polynomial time.

$$\begin{aligned}&\text{Minimize } \sum_{i=1}^{n} \ell_i, \\ &\text{subject to } \textstyle\sum_{w_i \in \text{path}(N_i, N_j)} \ell_i \geq \text{dist}(N_i, N_j), \\ &\forall \text{ sinks } N_i, N_j \text{ (Steiner point constraints)} \\ &D_{\min} \leq D_i \leq D_{\max} \forall \text{ sink } N_i \text{ (delay constraints).}\end{aligned}$$

Once the edge lengths are determined, the positions of Steiner points are determined from geometric considerations.

The method for placing Steiner points is similar to the DME algorithm. In the DME algorithm, the feasible regions for Steiner points and the edge lengths are found in a bottom-up fashion, and then Steiner points are placed in the feasible regions in a top-down fashion. The proposed method is different in that edge lengths are predetermined and the feasible regions are rectangular regions instead of simple line segments.

In the Elmore delay model, the delay equation is quadratic with respect to ℓ_i's. Because the Elmore delay function is quadratic and the sum of the quadratic terms is positive (i.e., the function is posynomial in ℓ_i.), the delay function is convex. The feasible set defined by a convex function with both lower and upper bounds, however, is not a convex set. Some edges may be given higher weights to account for wirability concerns, blockage, type of metals used, cross talk or switching activities. However, the approach splits the original problem into two subproblems solved separately and does not consider the interaction between two stages. Thus, we may need a new approach for determining the bounded-skew and bounded-delay Steiner tree in a "single" step.

Bounded-Radius, Weighted Steiner Tree. With progress in VLSI fabrication technology, interconnective delay has become increasingly significant in determining circuit speed. Recently, it has been reported that interconnective delay contributes from 50% to 70% of the clock cycle in the design of dense, high-performance circuits (131,132). Thus, with submicron device dimensions and up to a million transistors integrated on a single microprocessor, on-chip and chip-to-chip interconnections play a major role in determining the performance of digital systems.

Because of this trend, performance-driven layout design has received increased attention in the past several years. Most of the work in this area has been on the timing-driven placement problem, where a number of methods have been developed for placing blocks or cells in timing-critical paths close together, for example, see Refs. 131–135.

Although such techniques have been developed for timing-driven placement, only limited progress has been reported for the timing-driven interconnective problem. In Ref. 109, net priorities are determined on the basis of static timing analysis. Nets with high priorities are processed earlier using fewer feedthroughs. In Ref. 136, a hierarchical approach to timing-driven routing was outlined. A timing-driven global router based on the A^* heuristic search algorithm was proposed for building-block design. However, these results do not provide a general formulation of the timing-driven global routing problem. Moreover, their solutions are not flexible enough to provide a tradeoff between interconnective delay and routing "cost." Cong et al. (137) proposed a bounded-radius, minimum-routing tree and gave experimental results on the tradeoff between interconnective delay and routing "cost."

Three types of Steiner trees have been studied before in connection with global routing: min-max weight Steiner trees (84), minimum-length Steiner trees (20,38), and minimum-weight Steiner trees (138). A minimum-length Steiner tree is not appropriate, for it goes through "critical regions" including the modules. Indeed such a path needs to be modified so that it does not go through any modules. After the modification (shown by an arrow in Fig. 15) the path still goes through "critical regions" (e.g., the region with weight 9 in Fig. 15). After the modification, we may even violate the optimality of its length. A min-max Steiner tree is also not suitable, for it is excessively long. (Heuristics have been introduced to obtain shorter min-max Steiner trees in Ref. 84). A minimum-weight Steiner tree is a Steiner tree with minimum total weight, where an edge thereof with length ℓ in a region with weight w has total weight ℓw (138). A minimum-weight Steiner tree deals with length and density, simultaneously. A minimum-weight Steiner tree, however, does not consider the interconnective delay (e.g., from P_1 to P_3 in Fig. 15). We employ the notion of bounded-radius weighted Steiner trees (BRWRST) to trade off between interconnective delay and routing weight.

Balanced-Mesh Clock Routing. Clock routing using a balanced-mesh routing with circuit partitioning (Fig. 16) was

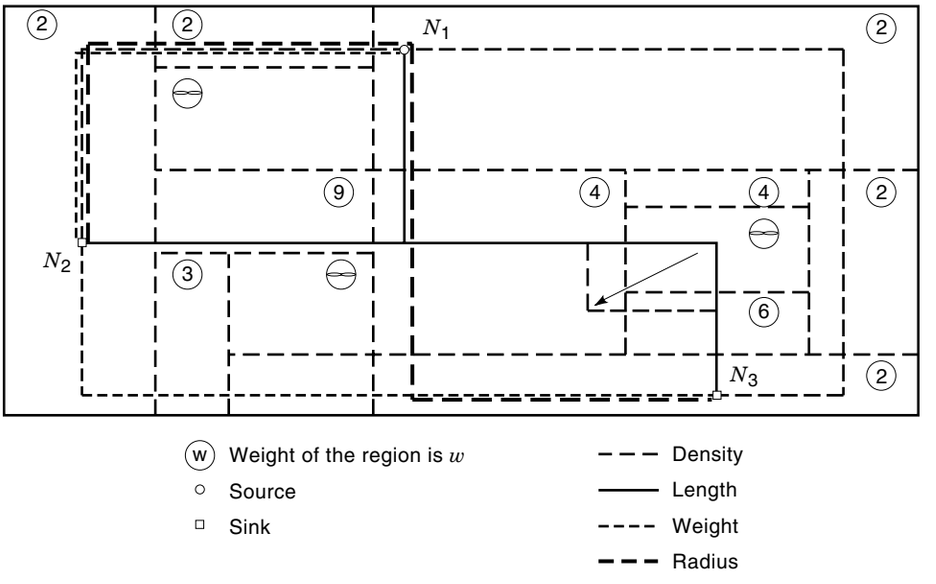


Figure 15. Four types of Steiner trees.

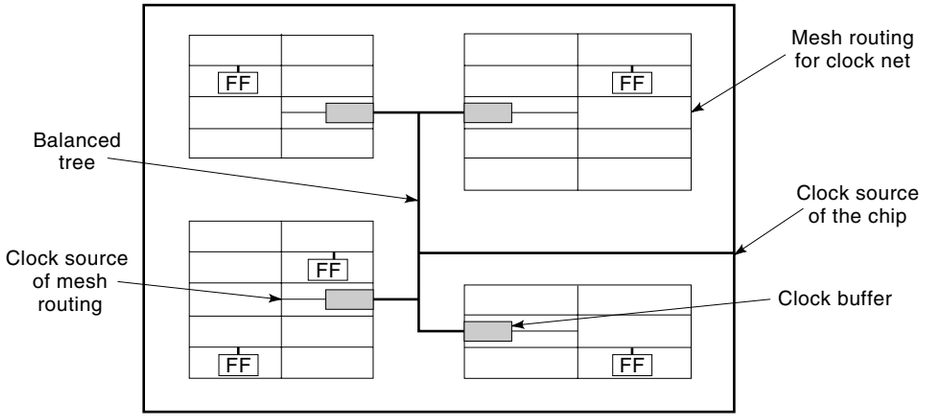


Figure 16. Balanced-mesh clock routing in cell-array designs.

proposed (21,139). The circuit is partitioned into subblocks called mesh-routing regions (MRs) in which clock skew is suppressed below a constant by mesh routing. Then the net from the clock source to each MR is routed as a balanced tree. Using the technique in the design of MPEG2-encoder LSI, a skew of 210ps was achieved.

The balanced-tree method (BTM) achieves very low skew, but it increass area and delay time by making the skew unnecessarily small. This is especially crucial in the design of chips having many FF's (e.g., MPEG2 LSI's).

The fixed-mesh method (FMM) generates a clock net in a fixed mesh driven by a large buffer with wire sizing. FMM has been applied to the design of a DEC Alpha chip (140). The entire chip is covered by a big mesh of interconnect metal that drives all the FF's. Although it achieves clock skew of less than 300ps for 0.75 μm technology, the power dissipated by the clock is almost 40% of the total power dissipation of the chip because the FMM overestimates the skew, which increases the number of interconnects requiring a large buffer. However, a fixed mesh is easy to route and at most one routing track is required in each channel. Taking advantage of both of BTM and FMM, Sato et al. (139) developed a practical clock routing method called the balanced-mesh method (BMM), in which the circuit is partitioned into some subblocks and the clock net in each subblock is routed as a mesh. Each mesh is driven by a relatively small clock buffer placed at its center row and these buffers are routed from the clock source by a balanced or a minimum-delay tree. The circuit is partitioned so that each subblocks's skew and the clock-signal delay time are bounded under given allowances, based on the relationship of the clock skew, delay time, and FF density in a chip. A subblock region that ensures skew is called a mesh-routing region (MR). In MR partitioning, the circuit is partitioned into MRs so that they satisfy the MR constraints and then a clock buffer, whose size depends on the number of FFs and area, is selected in each MR. In the placement stage, the FF's cells and its buffers are placed within each MR to which they belong to. The routing is classified into two types: intra-MR and inter-MR. Inter-MR routing is the mesh routing in each MR. Inter-MR routing is the minimum-delay time routing from the clock source to all MR's.

The MR-partitioning problem. Minimize $\sum_{i,j} C(i,j)$ subject to MR constraints,

where $C(i, j)$ is the number of nets connecting MR_i and MR_j.

Activity-Driven Clock Design. The activity patterns are obtained from Fig. 17. The tree also contains the possible activity patterns of its internal nodes. These activity patterns are derived assuming that a gate is placed one very node of the clock tree and they represent the times when these gates must allow propagation of the clock signal. A gate at the root of any clock tree must be active during a time period if any of

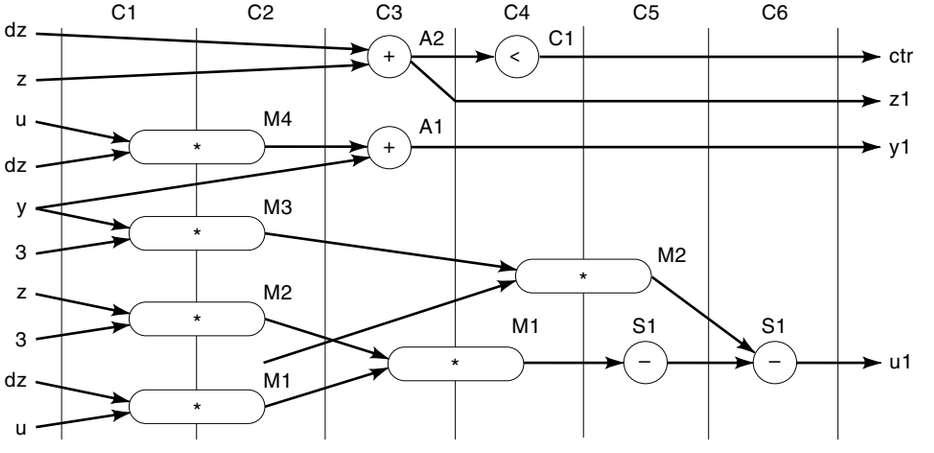


Figure 17. High-level design transformation of differential equation to control data-flow graph.

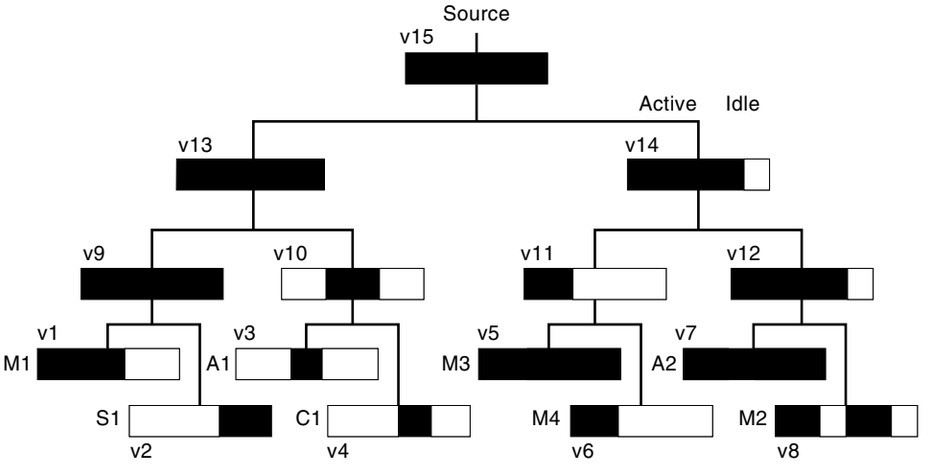


Figure 18. Differential equation example: A clock-tree circuit for the modules of the differential equation circuit.

its sinks is also active. Thus, the activity of the internal nodes is obtained by ORing the activity patterns of the sinks that belong to the corresponding subtree. Using Fig. 17, consider the activity pattern which contains modules A2 and M2 as sinks (001000 + 110110 = 111110). The tree in Fig. 18 has a total of 40 idle time periods. This is improved to 52 idle periods by closely placing modules of similar activity. The second task is to locate the clock gates, given a clock-tree topology, so that the total power is minimized. The gate insertion problem requires detailed information about the parasitic capacitances of the clock tree and the control lines of the gates. Hence, we model the clock-tree topology with an H-tree construction. First we define an activity pattern for an element i of the system, $U(i) = \{a_{ij} \| j = 1, \ldots, u, a_{ij} \in \{0, 1\}\}$, with u time periods and activity a_{ij}. Let element i consume total active circuit power $P_A(i)$ during periods when the circuits are active, and thus the clock must be supplied for proper function. Also let $P_I(i)$ denote the total inactive or idle circuit power, when clock supply is unnecessary. The power consumed by the clock supply is negligible. The power consumed by clocked element i is $P(U(i), i) = \sum_{j=1}^{u} [a_{ij}P_A(i) + (1 - a_{ij})P_I(i)]$. Define the function $t[U(i)]$, which measures changes in the activity pattern and the power consumed by the control signals of the clock gates. The total power of the clock tree is obtained from the sum of the power contributions defined previously.

Activity-Driven Clock Tree Construction

Let the activity pattern of a clock tree node be obtained by ORing the patterns of its sinks. Construct a tree $T(V, E)$ on a set of sinks N so that the weighted sum of nodes activities $A(T) = \sum_{v_i \in V} \{c_i t[U_T(i)] + \sum_{j=1}^{u} k_{ij}a_{ij}\}$ in the resulting tree is minimized. The weights k_{ij} and c_i are derived from the power contributions defined previously.

The algorithm proposed by Téllez et al. (141) is based on recursive weighted matching, where the matching weight is the value of the objective function of the resulting subtree.

Cross Talk

Rapid growth of multimedia and communication systems demands the use of both analog/digital mixed signal ICs and deep submicron (below 0.6 μm) CMOS technologies. The higher density and improved electrical performance of such technologies are needed in these systems.

In the mixed analog-digital layout design and deep submicron CMOS technologies, automated synthesis of interconnections during the early placement stage of the design cycle is emerging as a most promising approach. Current placement level design models do not capture important physical design effects, such as cross talk, power, and timing, simultaneously, which is the first order of factors in chip performance.

Because of the continually decreasing distances between components and the simultaneous increase of operating frequencies, the noise induced from signal wires physically too close to each other, called cross talk, becomes stronger. Increased cross talk holds for coupling via the interconnects, and also for cross talk via the substrate (142). Cross talk contributes as much as 50% to 75% to interconnective delay as the width of the wire and the space between wires is reduced (2). The problem is particularly important in high-frequency integrated circuits. A critical area in the layout is an area in which spot defects are centered and a malfunction in the respective critical circuit arises. Both cross talk and spot defects occur frequently in a channel and are avoided by rearranging the wires and vias.

Cross-talk noise should be considered because cross talk between long wires increases delay (because of larger effective line capacitance) and also degrades signal integrity and causes logic faults.

Excessive local congestion gives rise to future routing difficulty and also increases the potential cross-talk noise in high-speed signal lines. Furthermore, it increases power dissipation due to coupling capacitance. Cross talk is minimized by ensuring that wires carrying high-activity signals are placed sufficiently far from other wires. Moreover, for high-performance circuit routing, intersections of wires cause the use of more vias which, in turn, require more routing resources (because of the large via pitch), lower manufacturing yield, and cause noise problems (because of the mismatched characteristic impedance between wires and vias)(143).

The problem of cross talk is addressed typically after the placement step. The next step in physical design is to assign every global route in the layout environment to a plane pair, called *layer assignment,* so that the capacity constraints are

satisfied on all plane pairs and the number of plane pairs is minimized. A layer assignment algorithm to reduce cross talk presented in Refs. 9, 144–145 maximizes the layer separation between interfering nets, so as to reduce both intralayer and interlayer cross talk. There are several works related to cross-talk minimum routings. The main goal of the MCM router developed in Refs. 79, 146–148 is to route all the nets with a minimum number of layers and reduce the cross talk by separating high-frequency wires with a bound over the number of vias used in routing.

In the following, we present cross-talk minimization techniques during placement, global routing and channel routing phases.

The placement model in this paper targets MCMs. A given input is a set of rectangular chips of the same size with pins fixed within each block and a specification of n nets, including timing constraints on nets. Each output solution specifies an absolute position of each chip. The problem is stated as follows: Given a set of chips C and a set of chip sites S, find a mapping ϕ: $C \rightarrow S$, so as to minimize the cross talk, crossings, and total wire length needed for routing and to ensure routability of the design in a minimum number of routing layers. Early estimation of wirability during placement is important, but net topology is difficult to estimate at the placement stage.

Conventionally, a cost metric based on wirelength plus congestion increases the wirability. However, in our formulation, we do not consider the congestion measure, explicitly. We observed by experiments that congestion minimization is done automatically and we perform cross talk and crossing minimization simultaneously, because it distributes wires evenly over the MCM substrate. Note that minimizing the number of crossings reduces the wire length, whereas minimizing the cross talk does not always do so. Next, we introduce a new interference measure based on cross talk and crossings.

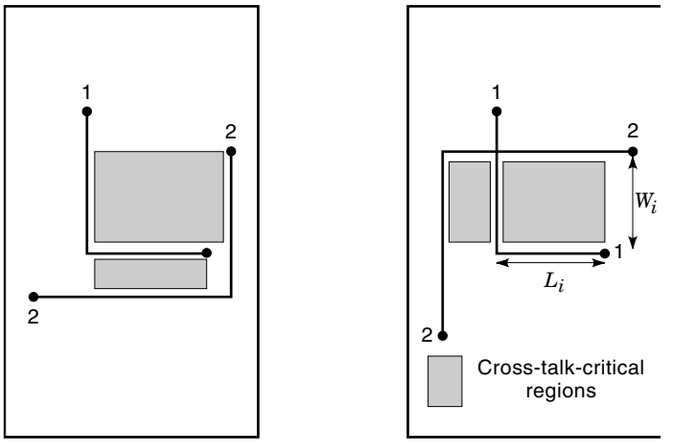


Figure 19. Cross talk and crossing. Observe that minimizing cross talk introduces more crossings.

Net Topology and Graph Generation. Multiterminal nets have many possible routing topologies, such as daisy chain, Steiner tree, star and A-tree (128,149). However, it is impractical to consider all configurations of a large fan-out net because the number of net topologies as a function of the number of a large fan-out receivers increases rapidly. Raghavan, Cohoon, and Sahni (150) demonstrated a polynomial time solution ($O(n^2)$ time) for a one-layer routing problem called *single-bend wirability problem,* for two-terminal nets, which is the problem of determining whether there exists a planar routing with at most one bend per net. The problem can be reduced to the 2-satisfiability problem. However, allowing multiple terminals renders the single-bend wirability problem NP-complete (151). The formulation, however cannot be directly applied to solve our problem that considers multiple constraints on wires. The bounding box measure (of wiring interference) for placement without taking net topologies into account completely is not sufficient. For example, a simple measure which satisfies this property adds an edge between two vertices if the bounding boxes of corresponding nets intersect. If the bounding boxes of two nets intersect in a highly congested region, the routability is more severely affected than if they intersect in a region with very few nets. Thus, we consider, for two-terminal net i, two possible one-bend global routes, $i(\tau_1)$ and $i(\tau_2)$. It is desirable that the multiterminal nets are routed within the smallest bounding box enclosing the terminals belonging to the nets, and with their favorite topologies as mentioned previously. For example, one restricts one to a specific routing pattern for a multiterminal net with a mincost Steiner tree having minimum wire length, minimum bends, and minimum stubs. For nets with large terminals, a mincost Steiner heuristic is used. A stub or branch in a tree introduces extra delay and/or ringing in the received signal waveform (113). Evidently, the topology estimate from a placement in this way is poorer than the estimate from global routing, but it is a necessary compromise for a strong coupling between the placement and global routing. Based on these facts, given a placement, we create a new interference graph $G = (V, E)$, where $|V| = 2n$ and $|E| \leq 2n(n - 1)$ (in the case of two-terminal nets, refer to Fig. 19), to formulate the interference relationship between nets, so that each node in V represents a net and a weight on an edge in E represents a net-pair cross talk and crossing measured as below.

Cross-Talk Measure. A popular approach used in the past to model the dependency of performance functions on parasitics is net classification. Nets are classified according to the type of signal they carry (stable, large swing, sensitive to noise, etc.). A bus of several sensitive nets running parallel to each other with correlated signals might inject considerable noise into a single net. The *cross-talk-critical region* is defined as a region enclosed by two wire segments of net i and net j so that their coupling distance $d(i, j)$ is less than or equal to a small constant. The value depends on device technology. For example, using ac device technology on an MCM-L layer, δ = 1 cm (146). The shaded regions in Fig. 19 correspond to the set of cross-talk-critical regions induced by the given global routes of the two nets. The cross talk between two nets $i(\tau_p)$ and $j(\tau_q)$, denoted as $\mu[i(\tau_p), j(\tau_q)]$, is estimated as proportional to the maximum length for which two nets run in parallel and

is inversely proportional to the minimum separation between the parallel wires:

$$\mu[i(\tau_p), j(\tau_q)] = \sum_{k \in K(i(\tau_q), j(\tau_q))} (\ell_k / d_k)$$

where $K[i(\tau_p), j(\tau_q)]$ is the set of cross-talk-critical regions between two nets i with topology τ_p and j with topology τ_q. An interference graph is established for the net-pairwise cross-talk value which is an edge-weight of the graph in $O(n^2)$ time. Then, noise tolerance T_i for net i with topology τ with respect to the cross-talk measure μ is approximated as $T_{i(\tau)} = M_{i(\tau)} - \sum_{\forall j} \mu[i(\tau), j]$, where $M_{i(\tau)}$ is the maximum allowable coupled noise for net i with topology τ and j is the cross-talk critically adjacent net with respect to net i.

We aim to identify the placement which maximizes the minimum noise tolerance for all nets. We say that the placement satisfies the *noise-tolerance condition.*

BIBLIOGRAPHY

1. E. E. Davidson and G. A. Katopis, Package electrical design, in E. R. Tummala and E. J. Rynaszewski (eds.), *Microelectronics Packaging Handbook,* New York: Van Nostrand Reinhold, 1989, Chap. 3.
2. H. B. Bakoglu, *Circuits, Interconnections, and Packaging for VLSI,* Reading, MA: Addison-Wesley, 1990, pp. 81–112.
3. B. Preas, M. Pedram, and D. Curry, Automatic layout of silicon-on-silicon hybrid packages. In *Design Automation Conference,* 1989, pp. 394–399.
4. A. Hanafusa, Y. Yamashita, and M. Yasuda, Three-dimensional routing for multilayer ceramic printed circuit boards. In *Int. Conf. Computer-Aided Des.,* IEEE, November 1990, pp. 386–389.
5. W. Wei-Ming Dai, Performance driven layout of thin-film substrates for multichip modules, in *Proc. Multichip Module Workshop,* IEEE, 1990, pp. 114–121.
6. W. Wei-Ming Dai, Topological routing in SURF: Generating a rubber-band sketch. In *Proc. IEEE Des. Automation Conf.,* IEEE, 1991, pp. 39–48.
7. J. M. Ho et al., Layer assignment for multi-chip modules. *IEEE Trans. Comput. Aided Des.,* **CAD-9**: 1272–1277, 1990.
8. J. D. Cho, K. F. Liao, and M. Sarrafzadeh, Multilayer Routing Algorithm for High Performance MCMs. In *Proc. Fifth Annu. IEEE Int. ASIC Conf. Exhibit,* 1992, pp. 226–229.
9. J. D. Cho et al., Crosstalk minimum layer assignment. In *Proc. IEEE Custom Integr. Circuits Conf.,* San Diego, CA, 1993, pp. 29.7.1–29.7.4.
10. J. D. Cho and M. Sarrafzadeh, The pin redistribution problem in multichip modules. In *Proc. Fourth Annu. IEEE Int. ASIC Conf. Exhibit,* IEEE, September 1991. pp. p9-2.1–p9-2.4.
11. M. Sriram and S. M. Kang, Detailed layer assignment for MCM routing. In *Int. Conf. Computer-Aided Des.,* IEEE, 1992, pp. 386–389.
12. R. Iyer, D. Rossetti, and H. Hsueh, Measurement and modeling of computer reliability as affected by system activity, *ACM Trans. Comput. Syst.,* **4** (3): 214–237, 1986.
13. F. Najm, Transition density: A new measure of activity in digital circuits, *IEEE Trans. Comput. Aided Des.,* **12** (2): 310–323, 1992.
14. B. Lin and H. DeMan, Low-power driven technology mapping under timing constraints. *Int. Conf. Comput. Des.,* IEEE, 1993, pp. 421–427.
15. V. Tiwari, P. Ashar, and S. Malik, Technology mapping for low power. In *Des. Automation Conf.,* ACM/IEEE, 1993, pp. 74–79.
16. C. Tsui, M. Pedram, and A. M. Despain, Technology decomposition and mapping targeting low power dissipation. In *Des. Automation Conf.,* ACM/IEEE, 1993, pp. 68–73.
17. H. Vaishnav and M. Pedram, A performance driven placement algorithm for low power designs. In *EURO-DAC,* 1993.
18. M. Sarratzadeh and C. K. Wong, *An Introduction to VLSI Physical Design,* New York: McGraw–Hill, 1996.
19. E. S. Kuh and T. Outsuki, Recent advances in VLSI layout, *Proc. IEEE,* 1990, pp. 250–251.

19a. S. Muroga, *VLSI System Design,* New York: Wiley, 1982.

20. C. Y. Lee, An algorithm for path connection and its application. *IRE Trans. Electronic Comput.,* **EC-10**: 346–365, 1961.
21. Y. Y. Lee and J. D. Cho, A new VLSI clock layout synthesis system. In *Technical Report, Sung Kyun Kwan University,* **46** (2): p. 891–903, 1995.
22. H. Liu and D. F. Wong, Network flow based multi-way partitioning with area and pin constraints. In *Proc. 1997 IEEE/ACM Int. Symp. Physical Des.,* 1997, pp. 12–17.
23. G. Karypis and V. Kumar, Unstructured graph partitioning and sparse matrix ordering. In *Technical Report,* CS Dept., University of Minnesota, 1995.
24. C. Alpert, L. Hagen, and A. Kahng, A hybrid multilevel genetic approach for circuit partitioning, in *Proc. IEEE Asia Pacific Conf. Circuits Syst.,* November 1996.
25. S. Wimer, I. Koren, and I. Cederbaum, Optimal aspect ratios of building blocks in VLSI, *IEEE Trans. Comput. Aided Des.,* **8**: 139–145, 1989.
26. J. P. Cohoon, Distributed genetic algorithms for the floorplan design problem, *IEEE Trans. Comput. Aided Des.,* **10**: 483–492, 1991.
27. R. M. Kling and P. Banerjee, Optimization simulated evolution with applications to standard cell placement. In *Des. Automation Conf.,* 1990, pp. 20–25.
28. D. F. Wong, H. W. Leong, and C. L. Liu, *Simulated Annealing for VLSI Design,* Norwell, MA: Kluwer Academic, 1988.
29. M. Rebaudengo and M. S. Reorda, GALLO: A genetic algorithm for floorplan area optimization, *IEEE Trans. Comput. Aided Des.,* **15**: 1996.
30. J. M. Kleinhans et al., GORDIAN: VLSI placement by quadratic programming and slicing optimization, *IEEE Trans. Comput. Aided Des.,* **10**: 356–365, 1991.
31. C. Alpert et al., Faster minimization of linear wirelength for global placement. In *Proc. 1997 IEEE/ACM Int. Symp. Physical Des.,* 1997, pp. 4–11.
32. D. Huang and A. B. Kahng, Partitioning-based standard-cell global placement with an exact objective. In *Proc. 1997 IEEE/ACM Int. Symp. Physical Des.,* 1997, pp. 18–25.
33. T. Lengauer and M. Lugering, Integer programming formulation of global routing and placement problems. In M. Sarrafzadeh and D. T. Lee (eds.), Special volume on *Algorithm Aspects of VLSI Layout,* Singapore: World Scientific, 1993.
34. S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, Optimization by simulated annealing, *Science,* **220**: 671–680, 1983.
35. D. G. Schweikert, A two-dimensional placement algorithm for the layout of electrical circuits. In *Proc. IEEE Des. Automation Conf.,* IEEE/ACM, 1976, pp. 408–416.
36. D. S. Johnson et al., Optimization by simulated annealing: An experimental evaluation, Part II (graph coloring and number partitioning), *Oper. Res.,* **39**: 378–406, 1991.
37. M. P. Vecchi and S. Kirkpatrick, Global wiring by simulated annealing, *IEEE Trans. Comput. Aided Des.,* **CAD-2**: 215–222, 1983.

38. K.-W. Lee and C. Sechen, A new global router for row-based layout, *Int. Conf. Computer-Aided Des.,* IEEE, November 1988, pp. 180–183.
39. C. Sechen, *VLSI Placement and Global Routing Using Simulated Annealing,* Deventer, The Netherlands; Kluwer, B. V., 1988.
40. H. Esbensen and P. Mazumder, SAGA–A unification of genetic algorithm with simulated annealing and its application to macrocell placement, *Proc. 7th Int. Conf. VLSI Des.,* 1994, pp. 211–214.
41. M. Hanan, On Steiner's problem with rectilinear distance, *SIAM J. Appl. Math.,* **14** (2): 255–265, 1966.
42. A. V. Aho, M. R. Garey, and F. K. Hwang, Rectilinear steiner trees: efficient special-case algorithm, *Networks,* **7**: 35–58, 1977.
43. J. M. Ho, G. Vijayan, and C. K. Wong, New algorithm for the rectilinear Steiner Tree Problem. *IEEE Trans. Comput. Aided Des.,* **9**: 185–193, February 1990.
44. F. K. Hwang and D. S. Richards, Steiner tree problems, manuscript, 1989.
45. A. Z. Zelikovsky, P. Berman, and M. Karpinski, *Improved Approximation Bounds for the Rectilinear Steiner Tree Problem,* Technical Report Report No. 85108-CS, Institut fur Informatik, Universitat Bonn, 1994.
46. S. K. Rao et al., The rectilinear Steiner arborescence problem, *Algorithmica,* **7** (2–3): 277–288, 1992.
47. G. E. Téllez and M. Sarrafzadeh, On rectilinear distance-preserving trees, *Int. Symp. Circuits Syst.,* IEEE, 1995, Vol. 1, pp. 163–166.
48. J. Cong, A. Kahng, and K.-S. Leung, Efficient heuristics for the minimum shortest path Steiner Arborescence problem with applications to VLSI physical design. In *Proc. 1997 IEEE/ACM Int. Symp. Physical Des.,* 1997, pp. 88–95.
49. J. D. Cho, A min-cost flow based min-cost rectilinear Steiner distance-preserving tree construction. In *Proc. 1997 IEEE/ACM Int. Symp. Physical Des.,* 1997, pp. 82–87.
50. S. Chattopadhyay, D. Bouldin, and P. Dehkordi, An overview of placement and routing algorithms & multi-chip nodules, in J.D. Cho and P. D. Fyazon (eds.), *High Performance Design Automation for MCM and Packages,* Singapore: World Scientific, 1996, pp. 3–23.
51. K. Shahookar and P. Mazumder, VLSI cell placement techniques, *ACM Computing Surveys,* **23** (2): 143–220, 1991.
52. K. Shahookar and P. Mazumder, A genetic approach to standard cell placement using meta-genetic parameter optimization, *IEEE Trans. Comput. Aided Des.,* **9**: 500–511, 1990.
53. R. Vemuri, Genetic algorithms for partitioning placement and layer assignment for multichip modules, PhD Thesis, University of Cincinnati, Cincinnati, 1994.
54. H. Chan, P. Mazumder, and K. Shahookar, Macro-cell and module placement by genetic adaptive search with bitmap-represented chromosome, *Integration,* **12**: 49–77, 1991.
55. E. W. Dijkstra, A note on two problems in connection with graphs, *Numerische Mathematik,* **1**: 269–271, 1959.
56. E. F. Moore, Shortest Path Through a Maze. In *Annals of Computation Laboratory,* Cambridge, MA: Harvard University Press, 1959, pp. 285–292.
57. D. W. Hightower, A solution to line routing problems on the continuous plane, *6th Des. Automation Workshop,* IEEE, 1969, pp. 1–24.
58. T. G. Szymanski, Dogleg channel routing is NP-complete, *IEEE Trans. Comput. Aided Des.,* **CAD-4**: 31–41, 1985.
59. M. Sarrafzadeh, Channel-routing problem in the knock-knee mode is NP-complete, *IEEE Trans. Comput. Aided Des.,* **6**: 503–506, 1987.
60. D. T. Lee, U. I. Gupta, and J. Y. Leung, An optimal solution for the channel assignment problem, *IEEE Trans. Comput.,* **28**: 807–810, 1979.
61. D. N. Deutsch, A dogleg channel router. In *Des. Automation Conf.,* IEEE/ACM, 1976, pp. 425–433.
62. R. L. Rivest and C. M. Fiduccia, A greedy channel router, *Des. Automation Conf.,* IEEE/ACM, 1982, pp. 418–424.
63. M. Burstein and R. Pelavin, Hierarchical channel router, *Integration,* **1,** 1983. (Also *Proc. 20th Des. Automation Conf.,* 1983.)
64. F. P. Preparata and W. Lipski, Jr., Optimal three-layer channel routing, *IEEE Trans. Comput. Aided Des.,* **C-33**: 427–437, 1984.
65. M. L. Brady and D. J. Brown, Optimal multilayer channel routing with overlap. In *4th MIT Conf. Advanced Res. VLSI,* Cambridge, MA: MIT Press, 1986, pp. 281–296.
66. R. J. Enbody and H. C. Du, Near optimal n-layer channel routing. *Des. Automation Conf.,* IEEE/ACM, 1986, pp. 708–714.
67. J. Cong, D. F. Wong, and C. L. Liu, A new approach to the three- or four-layer channel routing, *IEEE Trans. Comput. Aided Des.,* **7**: 1094–1104, 1988.
68. K. Chaudhary and P. Robinson, Channel routing by sorting. *IEEE Trans. Comput. Aided Des.,* **10**: 754–760, 1991.
69. H. Chen and E. Kuh, A variable-width gridless channel router. In *International Conference on Computer-Aided Design,* IEEE/ACM, 1985, pp. 304–306.
70. A. Sangiovanni-Vincentelli, M. Santomauro, and J. Reed, A new gridless channel router: Yet another channel router the second (YACR-II), *Int. Conf. Computer-Aided Des.,* 1984.
71. N. Funabiki and Y. Takefuji, A Parallel algorithm for channel routing problems, *IEEE Trans. Comput. Aided Des.,* **11**: 464–474, 1992.
72. J. Greene et al., Segmented channel routing, *Des. Automation Conf.,* IEEE/ACM, 1990, pp. 567–572.
73. K. Zhu and D. F. Wong, On channel segmentation design for row-based FPGA's, *Int. Conf. Computer-Aided Des.,* 1992, pp. 26–29.
74. M. Pedram, B. Nobandegani, and B. Preas, Design and analysis of segmented routing channels for row-based FPGA's, *IEEE Trans. Comput. Aided Des.,* **13**: 1470–1479, 1994.
75. V. Roychowdhury, J. Greene, and A. El Gamal, Segmented channel routing, *IEEE Trans. Comput. Aided Des.,* 79–95, 1993.
76. J.–C. Jeen, R. S. Gyurcsik, and W.–T. Liu, A two layer chemical routing algorithm for mixed analog and digital signal nets. In *IEEE Custom Integrated Circuits Conf.,* 1988, pp. 11.5.1–11.5.4.
77. U. Choudhury and A. Sangiovanni–Vincentelli, Constrained-based channel routing for analog and mixed analog/digital circuits, *Int. Conf. Computer-Aided Des.,* 1990, pp. 198–201.
78. J. D. Cho and M. S. Chang, LEXA: A left-edge based crosstalk-minimum k-colour permutation in VHV channels. manuscript, July 1996.
79. S. Thakur, K.–Y. Chao, and D. F. Wong, An optimal layer assignment algorithm for minimizing crosstalk for three layer VHV channel routing. To appear in *VLSI DESIGN,* an international *J. Custom-Chip Design, Simulation, and Testing,* Jun-Dong Cho (Guest Editor), 1995.
80. T. Leighton, A survey of problem and results for channel routing, in *A.W.O.C.,* 1987.
81. R. L. Rivest, A. E. Baratz, and G. Miller, Provably good channel routing algorithms. In H. T. Kung, R. Sproull and G. Steele (eds.), *VLSI Systems and Computations,* Computer Science Press, Rockville, MD, 1981, pp. 178–185.
82. M. Burstein and R. Pelavin, Hierarchical wire routing, *IEEE Trans. Comput. Aided Des.,* **CAD-2**: 223–234, 1983.

82a. B. S. Ting and B. N. Tien, Routing techniques for gate arrays, *IEEE Trans. Comput. Aided Des.,* **CAD-2**: 301–312, 1983.

82b. A. E. Dunlop et al., Chip layout optimization using critical path weighting. In *Design Automation Conf.,* 1984, pp. 133–136.

83. W. K. Luk et al., A hierarchical global wiring algorithm for custom chip design, *IEEE Trans. Comput. Aided Des.,* **CAD-6**: 518–533, 1987.

84. C. Chiang, M. Sarrafzadeh, and C. K. Wong, Global routing based on Steiner min-max trees, *IEEE Trans. Comput. Aided Des.,* **9**: 1315–1325, 1990.

85. J. T. Li and M. Marek–Sadowska, Global routing for gate arrays. *IEEE Trans. Comput. Aided Des.,* **CAD-3**: 298–307, 1984.

86. K. F. Liao, M. Sarrafzadeh, and C. K. Wong, Single-layer global routing, *Proc. 4th Annu. IEEE Int. ASIC Conf. Exhibit,* IEEE, September 1991, pp. p14-4.1–p14-4.4.

87. R. Nair, A simple yet effective technique for global wiring, *IEEE Trans. Comput. Aided Des.,* **CAD-6**: 165–172, 1987.

88. G. Meixner and U. Lauther, A new global router based on a flow model and linear assignment, *Int. Conf. Computer-Aided Des.,* IEEE, November 1990, pp. 44–47.

89. Y. Nishizaki, M. Igusa, and A. Sangiovanni-Vincentelli, Mercury: A new approach to macro-cell global routing, *Proc. VLSI, Germany,* 1989, pp. 411–419.

90. R. C. Carden IV and C. K. Cheng, A global router using an efficient approximate multicommodity multiterminal flow algorithm. In *Des. Automation Conf.,* IEEE/ACM, 1991, pp. 316–321.

91. E. Shargowitz and J. Keel, A global router based on multicommodity flow model, *INTEGRATION: VLSI J.,* **5**: 3–16, 1987.

92. R. M. Karp et al., Global wire routing in two-dimensional arrays, *Algorithmica,* **2** (1): 113–129, 1987.

93. M. Sarrafzadeh and D. Zhou, Global routing of short nets in two-dimensional arrays, *Int. J. Comput. Aided VLSI Des.,* **2** (2): 197–211, 1990.

94. M. Marek–Sadowska, Route planner for custom chip design, *Int. Conf. Computer-Aided Des.,* IEEE, November 1986, pp. 246–249.

95. L.-T. Hwang et al., Thin-film pulse propagation analysis using frequency techniques, *IEEE Trans. CHMT,* **14**: 1991.

96. L.–T. Hwang and I. Turlik, Calculation of voltage drops in the vias of a multichip package, *MCNC Technical Reports,* Technical Report Series TR90-41, 1990.

97. L.–T. Hwang and I. Turlik, The skin effect in thin-film interconnections for ULSI/VLSI Packages. *MCNC Technical Reports,* Technical Report Series TR91-13, 1991.

98. T. F. Gonzalez and S. L. Lee, A linear time algorithm for optimal wiring around a rectangle, *J. ACM,* **35** (4): 810–832, 1988.

98a. T. F. Gonzalez and S. L. Lee, A 1.6 approximation algorithm for routing multiterminal nets, *SIAM J. Comput.,* **16** (4): 669–704, 1987.

99. A. Frank et al., Algorithm for routing around a rectangle, *Discrete Appl. Math.,* **40**: 363–378, 1992.

100. T. F. Gonzalez and S. Lee, Routing Around Two Rectangles to Minimize the Layout Area. In M. Sarrafzadeh and D. T. Lee, eds., *Algorithmic Aspects of VLSI Layout,* Singapore: World Scientific, 1993, pp. 365–397.

101. A. C. Tucker, Structure theorem for some circular-arc graphs, *Discrete Math.* **7**: 167–195, 1974.

102. D. T. Lee, U. I. Gupta, and J. Y. Leung, Efficient algorithms for interval graphs and circular arc graphs, *Networks,* **12**: 459–467, 1982.

103. A. Hashimoto and J. Stevens, Wire routing by optimizing channel assignment within large apertures, *Proc. 8th Design Automation Workshop, Atlantic City, NJ,* June 1971, pp. 155–169.

104. J. L. Ganley and J. P. Cohoon, Provably good moat routing, Manuscript, 1996.

105. J. Kim and S. M. Kang, A New Triple-Layer OTC Channel Router. *IEEE Trans. Comput. Aided Des.,* 1996.

106. A. J. Blodgett, Microelectronic packaging, *Sci. Amer.,* pp. 86–96, July 1983.

107. P. D. Franzon, Private communication. Electronic MCM Clearing House in North Carolina State University, 1992.

108. R. Bechade, R. Flaker, and B. Kauffmann et al., A 32b 66 mhz 1.8W microprocessor. In *IEEE Int. Solid-State Circuit Conf.,* 1994, pp. 208–209.

109. A. E. Dunlop et al., Chip layout optimization using critical path weighting, *Des. Automation Conf.,* IEEE/ACM, 1984, pp. 133–136.

110. J. D. Cho and M. Sarrafzadeh, Buffer distribution algorithm for high-performance clock optimization. *IEEE Trans. VLSI Syst.,* **3**: 84–98, 1995.

111. W. C. Elmore, The transient response of damped linear networks with particular regard to wideband amplifiers, *J. Appl. Phys.,* **19** (1): 55–63, 1948.

112. J. Cong, K.-S. Leung, and D. Zhou, Performance-driven interconnect design based on distributed RC delay model. In *UCLA Computer Science Department Technical Report CSD-920043,* October 1992.

113. M. Sriram and S. M. Kang, Performance driven MCM routing using a second order RLC tree delay model, *Proc. IEEE Int. Conf. Wafer Scale Integration,* San Francisco, January 1993.

114. M. Sarrafzadeh and C. K. Wong, Hierarchical Steiner tree construction in uniform orientations, *IEEE Trans. Comput. Aided Des.,* **11**: 1095–1103, 1992.

115. S. Ganguly and S. Hojat, Clock distribution design and verification for power PC microprocessor, *Int. Conf. Computer-Aided Des.,* 1995, p. Issues in Clock Designs.

116. E. G. Friedman, *Clock Distribution Networks in VLSI Circuits and Systems,* IEEE, 1995.

117. A. B. Kahng and G. Robins, *On Optimal Interconnections for VLSI.* Norwell, MA: Kluwer Academic Publishers, 1995.

118. P. Winter, Steiner problem in networks: A Survey, *Networks,* **17**: 129–167, 1987.

119. M. R. Garey and D. S. Johnson, The rectilinear Steiner tree problem is NP-complete, *SIAM J. Appl. Math.,* **32** (4): 826–834, 1977.

120. A. Kahng, J. Cong, and G. Robins, High-performance clock routing based on recursive geometric matching, *Des. Automation Conf.,* IEEE/ACM, 1991, pp. 322–327.

121. J. Cong and C. K. Koh, Minimum-cost bounded-skew clock routing, *Int. Symp. Circuits Syst.,* 1995, pp. 215–218.

122. J. Cong et al., Bounded-skew clock and Steiner routing under Elmore delay. In *Int. Conf. Computer-Aided Des.,* p. Issues in Clock Designs, 1995.

123. I. Pyo, J. Oh, and M. Pedram, Constructing minimal spanning/Steiner Trees with bounded path length, *Eur. Des. Test Conf.,* 1996, pp. 244–248.

124. M. A. B. Jackson, A. Srinivasan, and E. S. Kuh, Clock routing for high-performance ICs, *Des. Automation Conf.,* IEEE/ACM, 1990, pp. 573–579.

125. G. Tellez and M. Sarrafzadeh, On Rectilinear Distance-Preserving Trees, manuscript to appear in *VLSI DESIGN,* the special issue in *High Performance Design Automation for VLSI Interconnections,* May 1995.

126. H. A. Choi and A. H. Esfahanian, On complexity of a message-routing strategy for multicomputer systems. In *16th Int. Workshop Graph-Theoretic Concepts Comput. Sci., Germany,* pp. 170–181, 1990.

127. J. D. Cho, Min-cost flow based minimum-cost rectilinear Steiner distance-preserving tree, 1996.

128. C. P. Chen, Y. W. Chang, and D. F. Wong, Fast performance-driven optimization for buffered clock trees based on Lagrangian relaxation. In *Des. Automation Conf.,* 1996, pp. 405–408.

129. J. G. Xi and W. W.–M. Dai, Buffer insertion and sizing under process variations for low power clock distribution, *Des. Automation Conf.,* 1995, pp. 491–496.

130. J. Oh, I. Pyo, and M. Pedram, Constructing lower and upper bounded delay routing trees using linear programming, *Des. Automation Conf.,* 1996, pp. 401–404.

131. W. E. Donath et al., Timing driven placement using complete path delays, *Des. Automation Conf.,* IEEE/ACM, 1990, pp. 84–89.

132. S. Sutanthavibul and E. Shragowitz, An adaptive timing-driven layout for high speed VLSI. In *Des. Automation Conf.,* IEEE/ACM, 1990, pp. 90–95.

133. P. S. Hauge, R. Nair, and E. J. Yoffa, Circuit placement for predictable performance, *Int. Conf. Computer-Aided Des.,* IEEE, 1987, pp. 88–91.

134. E. S. Kuh et al., Timing driven layout. In *VLSI Logic Synthesis Des.,* 1991, pp. 263–270.

135. I. Lin and D. H. C. Du, Performance-driven constructive placement. *Des. Automation Conf.,* IEEE/ACM, 1990, pp. 103–106.

136. M. A. B. Jackson, E. S. Kuh, and M. Marek–Sadowska, Timing-driven routing for building block layout, *Int. Symp. Circuits Syst.,* IEEE, 1987, pp. 518–519.

137. J. Cong et al., Provably good performance-driven global routing, *IEEE Trans. Comput. Aided Des.,* **11**: 739–752, 1992.

138. C. Chiang, M. Sarrafzadeh, and C. K. Wong, A global router with simultaneous length and density minimization, manuscript, June 1991.

139. H. Sato, A. Onozawa, and H. Matsuda, A Balanced-mesh clock routing technique using circuit partitioning, *Eur. Des. Test Conf.,* 1996, pp. 237–243.

140. D. Dobberpuhl et al., A200-Mhz 64-b dual-issue CMOS microprocessor, *IEEE J. Solid-State Circuits,* 1555–1567, 1992.

141. G. E. Téllez, A. Farrahi, and M. Sarrafzadeh, Activity-driven clock design for low-power circuits, *Int. Conf. Computer-Aided Des.,* IEEE/ACM, November 1995.

142. N. P. van der Meijs, T. Smedes, and A. J. Genderen, Extraction of circuit models for substrate cross-talk, *Int. Conf. Computer-Aided Des.,* 1995, pp. 199–206.

143. Cadence Design Systems, *A vision for multichip Module Design in the Nineties.* Tech. Rep. Cadence Design Systems Inc., Santa Clara, CA, 1993.

144. K. Y. Chao and D. F. Wong, Layer assignment for high-performance multi-chip modules, in J. D. Cho and P. D. Franzon (eds.), *High Performance Design Automation for MCM and Packages,* Singapore: World Scientific, 1996, pp. 61–79.

145. J. M. Ho et al., Layer assignment for multi-chip modules, *IEEE Trans. Comput. Aided Des.,* **CAD-9**: 1272–1277, 1990.

146. H. H. Chen and C. K. Wong, 63-layer TCM wiring with three-dimensional crosstalk constraints, in J. D. Cho and P. D. Franzon (eds.), *High Performance Design Automation for MCM and Packages,* Singapore: World Scientific, 1996, pp. 81–92.

147. G. Devaraj, Distributed placement and crosstalk driven router for multichip modules, *M.S. Thesis, University of Cincinnati,* Cincinnati, 1994.

148. K. Chaudhary, A. Onozawa, and E. Kuh, A spacing algorithm for performance enhancement and crosstalk reduction. In *Int. Conf. Computer-Aided Des.,* 1993, pp. 697–702.

149. A. Vittal and M. Marek–Sadowska, Minimal delay interconnect design using alphabetic trees, *Des. Automation Conf.,* 1994, pp. 392–394.

150. R. Raghavan, J. Cohoon, and S. Sahni, Single bend wiring, *J. Algorithms,* **7** (2): 232–257, 1986.

151. H. C. Yen, On multiterminal single bend wirability. *IEEE Trans. Comput. Aided Des.,* **13**: 822–826, 1994.

JUN DONG CHO
Sung Kyun Kwan University

MAJID SARRAFZADEH
Northwestern University

VLSI ISOLATION.

See ISOLATION.