Chapter 11

Clock and Power Routing

Specialized algorithms are required for clock and power nets due to strict specifications for routing such nets. It has been noted that it is better to develop specialized routers for these nets rather than over-complicate the general router. In the worst case, these special nets can be hand-routed. Currently, in many microprocessors, both of these nets are manually routed and optimized. However, as chip frequency moves into the multiple gigahertz range, the clock skew budget will become smaller and smaller and it will be not be possible to design and route clock without the help of sophisticated and accurate clock routing tools. Similarly, due to large amounts of power that needs to be provided to microprocessors, power nets must be very accurately designed and simulated to predict the power availability in different parts of the chip. As a result, power routing and analysis will increasingly depend on CAD tools.

In synchronous systems, chip performance is directly proportional to its clock frequency. Clock nets need to be routed with great precision, since the actual length of the path of a net from its entry point to its terminals determines the maximum clock frequency on which a chip may operate. A clock router needs to take several factors into account, including the resistance and capacitance of the metal layers, the noise and cross talk in wires, and the type of load to be driven. In addition, the clock signal must arrive simultaneously at all functional units with little or no waveform distortion. Another important issue related to clock nets is buffering, which is necessary to control skew, delay and wave distortion. However, buffering not only increases the transistor count, it also significantly impacts the power consumption of the chip. In some cases, clock can consume as much as 25% of the total power and occupy 5-10% of the chip area. Typically, a fixed buffered clock distribution network is used at the chip level. At a block level, a local clock routing scheme ensures minimal skew and delay. The scheme used in each block can differ, depending on the design style used in the block. The clock routing problem has significant impact on overall chip design. Clock frequencies are increasing quite rapidly. Note that current microprocessors can operate at 500 Mhz to 650 Mhz. It is expected that 1.5 - 2.0 Ghz microprocessors will be available within two to three years (See Chapter 3 for SIA roadmap).

Compared to clock routing, power and ground routing is relatively simple. However, due to the large amount of current that these nets carry, power and ground lines are wide. Concerns such as current density and the total area consumed make it necessary to develop special routers for power and ground nets. In some microprocessor chips, power and ground lines use up almost an entire metal layer. Power and ground lines are also used to shield some signal lines. This is done by routing a signal between two power (and/or ground) lines. This reduces the cross-capacitance between the signal line and its adjacent signal lines. As chip design moves into low voltages, power and ground routing will become a even harder design challenge. In this chapter, we will discuss the problems associated with clock, power and ground routing and present the basic routing algorithms for these special nets.

11.1 Clock Routing

Within most VLSI circuits, data transfer between functional elements is synchronized by a single control signal, the processing clock. The clock synchronization is one of the most critical considerations in designing high-performance VLSI circuits. In the case of microprocessor design, the clock frequency f (in MHz) directly determines the performance or the MIPS (Million Instructions Per Second) of the microprocessor.

$$\text{MIPS} = f \times \text{NIPC}$$

In this equation, NIPC denotes for Number of Instructions issued Per Cycle. NIPC depends on the architecture of the processor, RISC versus CISC, and the compilers used for the system. Most modern microprocessors are capable of multiple issue, and some can issue as many as five instructions per cycle. Consider a processor, which has a clock frequency of 200 MHz and can execute two instructions per clock cycle, thus giving it a 400 MIPS rating. If the clock frequency of the processor can be increased to 400 MHz, 800 MIPS performance can be obtained. In I/O and memory buses, the clock frequency determines the rate of data transmission. The data transmission rate is determined by the product of the clock frequency and the bus width. Thus, it is desirable to design the circuit with the fastest possible clock. However, increasing the clock frequency of a chip is a complicated affair.

The clock signal is generated external to the chip and provided to the chip through the clock entry point or the clock pin. Each functional unit which needs the clock is interconnected to the clock entry point by the clock net. Each functional unit computes and waits for the clock signal to pass its results to another unit before the next processing cycle. The clock controls the flow of information within the system. Ideally, the clock must arrive at all functional units at precisely the same time. In this way, all tasks may start at the same time and data can be transferred from one unit to another in an optimum manner. In reality, the clock signals do not arrive at all functional

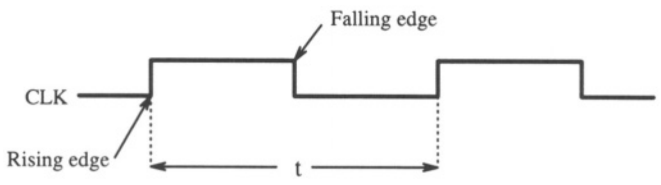


Figure 11.1: The Clock Period.

units simultaneously. The maximum difference in the arrival time of a clock at two different components is called *clock skew*. Clock skew forces the designer to be conservative and use a large time period between clock pulses, that is, lower clock frequency. The designer uses the clock period which allows for logical completion of the task as well as some extra time to allow for deviations in clock arrival times. If the designer can be provided a guarantee that the maximum deviation of the clock arrival time is small, then faster clocks can be used. The smaller the deviation, the faster the clock. Thus, controlling the deviation of signal arrival time is the key to improving circuit performance.

In the following sections, we will study the basics of clock design in a digital system. We will present several algorithms that have been proposed for solving various problems associated with clock nets. We will restrict ourselves to single clock systems, and briefly mention the multiple clock systems.

11.1.1 Clocking Schemes

The clock is a simple pulsating signal alternating between 0 and 1. The clock period is defined as the time taken by the clock signal to complete one cycle (from one rising edge to the other rising edge). Clock frequency is given as

$$f = \frac{1}{t}$$

where t is the clock period which is shown in Figure 11.1.

Digital systems use a number of clocking schemes including single-phase with latches or edge-triggered flip-flops and double-phase clocking with one or two latches. The most common latch is a *D-latch* which is an storage element. It has data D and clock CLK inputs and a data output of Q. While CLK is high, Q follows D, changing whenever D changes. While CLK is low, Q remains constant, holding the last value of D (Figure 11.2(a)). An *edge-triggered D flip-flop* has the same inputs and outputs as the D latch, but Q changes only on the rising or falling edge of the CLK (Figure 11.2(b)).

In single phase clocking with latches, the latch opens when the clock goes high; data is accepted continuously while the clock is high; and the latch closes when the clock goes down. Single phase clocking schemes are not commonly used because of their complicated timing requirements, but some high-end VLSI

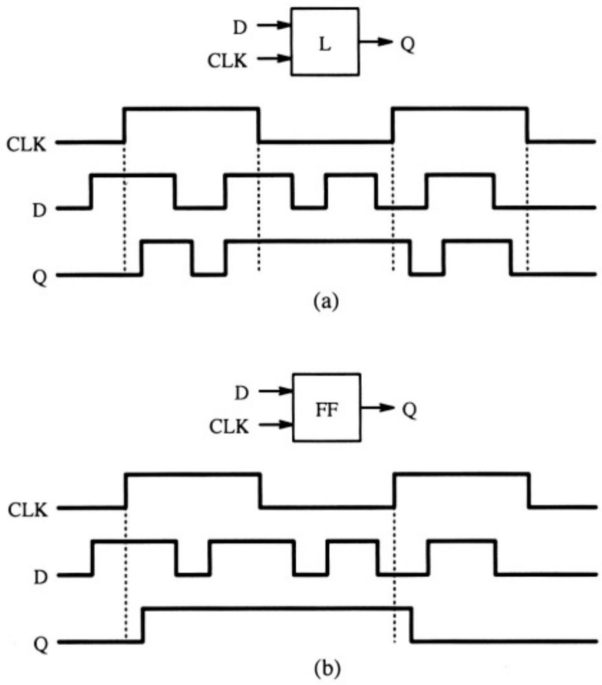


Figure 11.2: Input and output waveforms for (a) D-latch and (b) Edge-triggered D-flip-flop.

designs still use this scheme. The double-phase scheme uses two latches; one is called the *master* and the other the *slave.* The data is first captured by the master latch and then passed on to the slave latch.

The design of a clock system, as shown in Figure 11.3, must satisfy several timing constraints as explained below. When a clock signal arrives at a sequential register, it triggers the data from one sequential register set to the next through a logic unit. This unit performs manipulations of data in an appropriate functional manner. For simplicity and without losing generality, we will assume that the clocking scheme is edge-triggered.

The minimum cycle time must satisfy :

$$t_c > t_f + t_l + t_s + t_{skew}.$$

Where the flip-flop delay t_f is the time from the clock edge that captures the data to the time that the data is available at the output of the flip-flop, time t_l is the maximum delay through any logic block between two flip-flops, and setup time, t_s, is the amount of time the inputs of a flip-flop should be stable prior to the clock edge. Finally, t_{skew} is the worst-case skew between the clock signals, and the maximum amount of time the clock of the receiving flip-flop can precede the clock of the sending flip-flop.

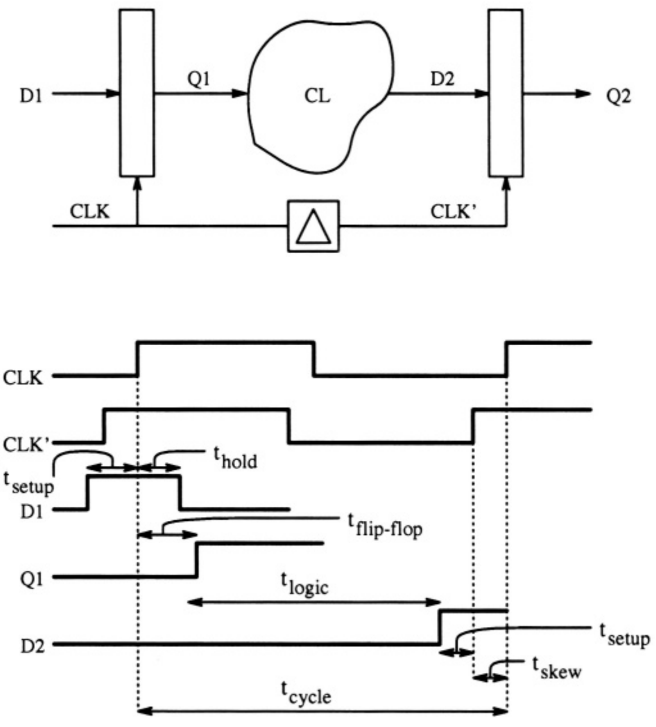


Figure 11.3: The structure of a clocking system.

Another constraint is the hold time, t_{hold}, which is the amount of time the input must stay stable after the clock edge to guarantee capturing the correct data. To guarantee that the data is captured, the clock width must be greater than the hold time:

$$t_{hold} < t_{clk-width}.$$

As a general rule, most systems cannot tolerate a clock skew of more than 10% of the system clock period. As a result, all clock lines must have equal lengths from clock entry point to a component, in order to minimize or eliminate clock skew. It is obvious that in the absence of a proper clock distribution strategy, different clock lines can have greatly varying lengths. This variation in length leads to large skews and delays. Skew is also introduced by the variations in the delay of clock buffers throughout the system because of the process-dependent transistor and capacitive loading. Skew causes uncertainty in the arrival of the clock signal at a given functional unit. If it can be guaranteed that the clock signal always arrives at a given storage element a predetermined amount of time earlier than it arrives at another storage element, design techniques can be employed to compensate for such pre-arrival of clock signal. But the nature of the clock skew is such that the designer does not know which storage elements will receive the clock early and which storage element will receive it late. There are two reasons for this uncertainty. First, the logic design is usually done before the chips are laid out, so the relative positions of storage elements with respect to the clock buffers are not known to the designer. Second, the random variations in the clock buffer delays, which are due to fabrication process dependent device parameter variations.

There are three key steps in designing high performance circuits. The first step in making a design operate at a high clock frequency is to employ a fast circuit family. With faster circuits, a given amount of logical functions can be performed in a shorter time. The second step is to provide a fast storage element (latch, flip-flop, register) and an efficient clocking scheme. The third step is to construct a clock distribution scheme with a small skew. As circuits become faster and cycle time is reduced, the actual maximum skew time allowed is reduced. While selection of faster circuits elements is a logic design decision, reducing clock skew and efficient clock distribution is within the realm of physical design. In the following, we consider the factors that influence design of efficient clock distribution schemes.

11.1.2 Design Considerations for the Clocking System

Clock signal is global in nature and therefore clock lines have to be very long. The delay caused by long wires is due to their capacitance and resistance. Long wires have large capacitances and limit the performance of the system. At low levels of integration, gate capacitance is much greater as compared to the interconnect capacitance and therefore need not be considered. For high level of integration, however, the gate capacitance is much smaller as compared to the interconnect capacitance and as a result, interconnect capacitance must be taken into account when clock wires are routed. For example, in 7 μm nMOS technology, the gate capacitance is equal to capacitance of 1 mm of wire. Assuming 5 mm side dies, few nets are 1 mm long. On the other hand, in 0.7 μm CMOS technology, the gate capacitance is equal to only 0.1 mm of wire. Thus gate capacitance is very small as compared to the capacitance of the long clock line, which may have to traverse as much as 25 mm. In addition to large capacitive loads, long wires also have large resistances. The resistance of a conductor is inversely proportional to its cross-sectional area. As chips are scaled down, the resistance per unit length becomes a major concern. The delay caused by the combined effect of resistance and capacitance is called the *RC* delay, which increases as the square of the scaling factor. In a given technology, *RC* delay cannot be reduced by making the wire wider. Although, *R* is reduced, but correspondingly *C* is increased. One effective way of reducing *RC* delay is the use of buffers (repeaters), which also help to preserve the clock waveform. If *RC* delay of a clock line is 4×4 units, then dividing the line in four equal segments and inserting buffers, the total *RC* constant is reduced to $1\times1 + 1\times1 + 1\times1 + 1\times1 = 4$. In this way capacitance is not carried over and that is how buffers help in reducing delay. The buffers, however, have internal delays, which must be taken into account when computing the total delay. In addition, buffers consume area and power. Despite these disadvantages, clock buffers play a key factor in the overall layout of high performance designs. In some processors, clock buffers may occupy as much as 5% of total area and may consume a significant amount of power. The problem of buffer insertion has significant attention and good algorithms are now known for both uniform and non-uniform lines [DFW84, WS92].

Buffers could be used in two different ways in the clock tree. One way is to use a big centralized buffer, whereas the other is to use distribute buffers in the branches of the tree. Figure 11.4 (a), and (b) illustrate both buffering mechanisms.

In case of distributed buffer, it is important to use identical drivers so that delay introduced by all the buffers is equal in all branches. In addition, it is important to equalize the load so that every driver sees the same capacitive load. The clock skew may still be there due to the mismatches among the drivers because of the device parameter variations across the chip. Using the identical layout for all the drivers and placing them next to each other and in the same orientation on the chip reduces the driver delay mismatch. Placing them in the same orientation guarantees that all are affected similarly by orientation dependence of the fabrication processing steps.

From the skew minimization point of view, the large centralized buffer is better than the distributed buffers. However, the area and power consideration are among other criteria that drive selection of the buffering mechanism.

In addition to *RC* delay, if the lines are sufficiently long or operate on high frequencies, then inductance also becomes important and clock lines behave like transmission lines, thereby changing the delay model. Transmission line behavior becomes significant when the rise time t_r of a signal is less than or comparable to the transmission line time-of-flight delay t_f. The rise time is defined as the time required for the signal to move from 10% to 90% of its final value. The time of flight is expressed as

$$t_f = \frac{l}{v}$$

where l is the line length, and v is the propagation speed. The rise time of a signal is determined by two factors: the rate at which the clock driver is turned on and the ratio of the driver source resistance to line impedance. In present CMOS systems, transmission line properties are significant only at the module and board levels; bipolar circuits require transmission line analysis at the chip carrier level and beyond; GaAs technology requires transmission line analysis even for on-chip interconnections.

11.1.2.1 Delay Calculation for Clock Trees

The exact computation of the *RC* delay of a clock tree is quite difficult. It is, however, not very difficult to approximate the delay. We will use a simple method for delay calculation for *RC* tree networks using the Elmore delay model [LM84b]. We follow the discussion presented by Tsay [Tsa91]. We will compute the delay for both buffered and unbuffered clock trees.

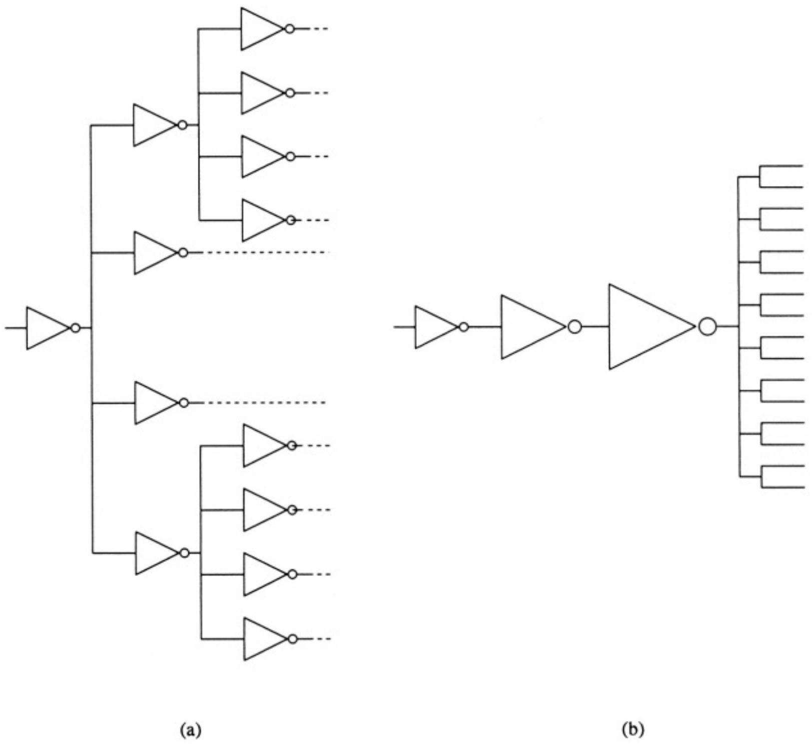


Figure 11.4: Clock buffering mechanisms.

Let T be an *RC* tree, c_i be the node capacitance and r_i be the resistance of edge i. The edge between node i and its parent is referred to as edge i. Note $r_0 = 0$, since root (node 0) has no parent. Let *IS*(i) be the set of immediate successors of node i, that is, *IS*(i) is a set of nodes adjacent to node i and does not contain its parent. Let T_i denote the subtree formed by node i and its successors.

For an unbuffered tree, the total capacitance of a subtree can be defined recursively as:

$$C_i = c_i + \sum_{j \in IS(i)} C_j$$

Let *N*(*i,j*) be the set of nodes between nodes i and j, including j but excluding i. The time delay of the clock signal from root (node 0) to a node i is given by:

$$t(0,i) = \sum_{j \in N(0,i)} r_j C_j$$

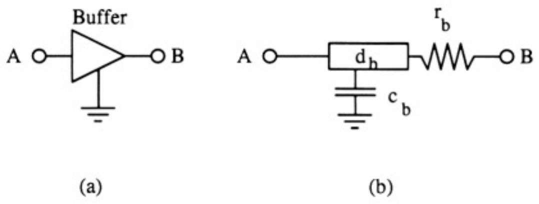


Figure 11.5: A clock buffer and its model.

The time delay from any node i to one of its successors j can be computed as:

$$t(i,j) = \sum_{k \in N(i,j)} r_k C_k$$

It is easy to see that for intermediate node k between i and j, the delay is given by:

$$t(k,j) = t(k,i) + t(i,j) \tag{11.1}$$

Thus time delay for an unbuffered tree can be computed in linear time using a depth first search.

For buffered trees, there are several different equivalent circuit models for the buffer as shown in Figure 11.5. Let d_b denote internal delay of the buffer, r_b denote its output driving resistance, and c_b denote its input capacitance. The only difference between a buffered *RC* tree and a unbuffered *RC* tree is the branch delay d_i, which accounts for buffer delay. The capacitance for a buffered *RC* tree is given by:

$$C_i = \begin{cases} c_i & \text{if node } i \text{ is a buffer input node} \\ c_i + \sum_{j \in IS(i)} C_j & \text{otherwise} \end{cases}$$

Similarly delay between node i and node j can be computed using:

$$t(i,j) = \sum_{k \in N(i,j)} (r_k C_k + d_k)$$

There are several ways of modeling *RC* trees, some of them are shown in Figure 11.6. More widely used model is the π-model as shown in Figure 11.6(b). Using π-model, one branch is modeled as shown in Figure 11.7. From Eq. (11.1), and by lumping the delay, we can compute the delay of a node i as

$$t(i,j) = d_i + r_k C_k + t(k,j) \tag{11.2}$$

Where k is a immediate successor of i and j is the leaf node. Our delay model is now complete as it specifies all the resistances, capacitances and delays, so that we can compute the delay from root to leaf.

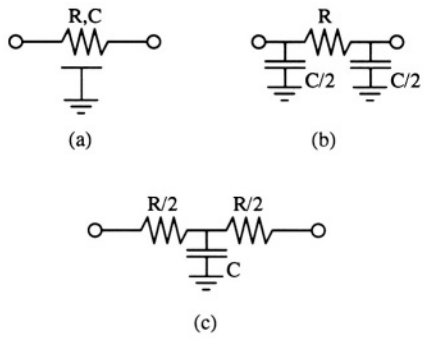


Figure 11.6: (a) A distributed RC line (b) The equivalent π-model. (c) The equivalent T-model.

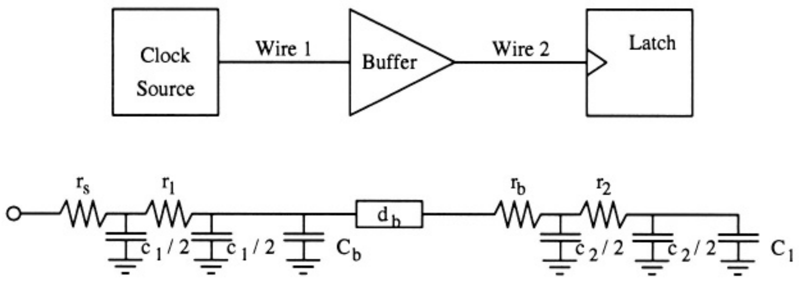


Figure 11.7: A Buffered RC tree and its Equivalent model.

11.1.3 Problem Formulation

Given the routing plane and a set of points $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ lying within the plane and clock entry point P_0 on the boundary of the plane. We refer to points by their indices. Let $t(i, j)$ refer to the delay between points i and j, then the Clock Routing Problem(CRP) is to interconnect each $P_i \in \mathcal{P}$ such that:

$$\max_{i \in \mathcal{P}} t(0, i)$$

$$\max_{(i,j) \in \mathcal{P}} \mid t(0, i) - t(0, j) \mid$$

are both minimized.

Additional objective functions such as minimization of total wire length, protection from noise and coupling may also be defined. The clock routing problem has traditionally been studied to minimize skew.

It is important to see that CRP is not a steiner tree problem for global routing of high performance circuits, since the interconnection distance between two clock terminals is of no significance in CRP. The clock routing problem is critical in high performance circuits. In other circuits, the clock is simply routed along with rest of the nets, and the router is given a maximum routing length so that it may route any segment of the clock.

11.1.3.1 Design Style Specific Problems

The clock routing changes significantly in different design styles. The problem is well studied for full-custom and gate array design styles, but no special model has been developed for standard cell designs.

1. **Full Custom:** The clock routing problem in full custom style depends on the availability of a routing layer for clocks. If a dedicated layer, free of obstacles, is available for routing, the clock routing problem in full custom design is exactly the same as CRP. If obstacles are present, however, we refer to that problem as the *Building Block Clock Routing Problem*(BBCRP).

   Given the routing plane and a set of rectangles $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$ lying within the plane and each rectangle R_i has its clock terminal P_i on its boundary, and the clock entry point P_0 on the boundary of the plane.

   Then the BBCRP is to interconnect each $P_i \in \mathcal{P}$ to P_0 so that wires do not intersect with any rectangles and both skew and delay are minimized.

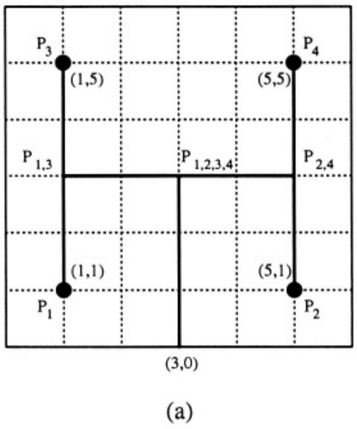
   In microprocessors, a chip level fixed buffered clock distribution is used to distribute the clock signals to different blocks. Then the problem described above can be used to locally distribute the clock.

2. **Standard Cell:** The clock routing problem in standard cell designs is somewhat easier than full-custom in some aspects, since clock lines have to be routed in channels and feedthroughs. Conventional methods do not work in standard cell design since clock terminals are neither uniformly distributed (as in full-custom), nor are they symmetric in nature (as in gate array).

3. **Gate Array:** Gate arrays are symmetrically arranged in a plane and allow the clock to be routed in a symmetric manner as well. The algorithms for clock routing in such symmetric structures have been well studied and well analyzed.

11.1.4 Clock Routing Algorithms

The skew can be minimized by distributing the clock signal in such a way that the interconnections carrying the clock signal to functional sub-blocks are equal in length. A perfect synchronization between the clock signals can be achieved by delaying the signals equally before they arrive at the sub-blocks. Note that we do not discuss buffered clock routing algorithms. As stated above, the problems and their corresponding algorithms should be viewed as local clock routing algorithms. In microprocessors, these algorithms can be used at a block



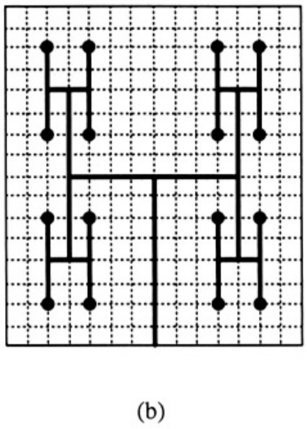


Figure 11.8: (a) H-tree over 4 points (b) H-tree over 16 points.

level. In ASICs, due to lower operating frequencies, these algorithms can also be used for chip level clock. However, even in that case, some buffering has to be used. In the following we will discuss, skew minimization algorithms. Minimization of clock skew has been studied by a number of researchers in recent years. In the following, we review several clock routing algorithms.

11.1.4.1 H-tree Based Algorithm

Consider a special case of CRP, where all the clock terminals are arranged in a symmetric fashion, as is the case in gate arrays. The clock routing in such cases can be accomplished with zero skew using the *H-tree* algorithm. Let us explain the algorithm with the help of a small example shown in Figure 11.8(a). Consider the case with four points, $P_1 = (1,1), P_2 = (5,1), P_3 = (1,5)$ and $P_4 = (5,5)$, in a routing plane with l = *6,w* = 6. The clock entry point is at $P_0 = (3,0)$. In the H-tree algorithm, P_1 is connected to P_3 and P_2 is connected to P_4 by vertical segments. Let $P_{13} = (1,3)$ and $P_{24} = (5,3)$ be the two middle points of these vertical segments. These middle points are also called the *tapping points*. P_{13} and P_{24} are connected by a horizontal segment, whose middle point is $P_{1234} = (3,3)$. Finally, clock entry point (P_0) is connected to P_{1234} by a vertical segment. It can be seen that all points are exactly 7 units from the point P_0, hence skew is zero. Since the longest rectilinear distance between any two points(P_0 and P_3) is seven units, this routing is minimum delay routing as well. Thus, the routing shown in Figure 11.8(a) provides clock signals to all clock points with zero skew and minimum delay.

This method can be easily generalized to n points, where n is a power of 4. The basic 4 point H-structure is duplicated in a recursive fashion. An H-tree with 16 terminals is shown in Figure 11.8(b). *H-tree* constructions have been used extensively for clock routing in regular systolic arrays [FK82, DFW84].

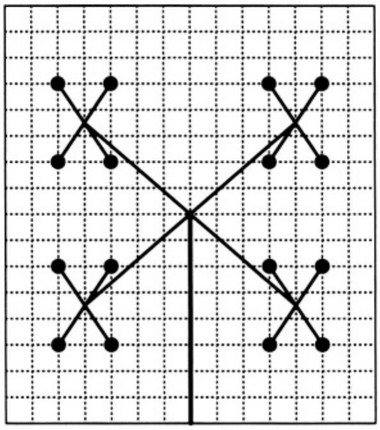


Figure 11.9: X-tree over 16 points.

If the routing is not restricted to being rectilinear, an alternate tree structure with smaller delay may be used.This tree structure, called the X-Tree, ensures that skew is zero (see Figure 11.9). However, X-trees are undesirable, since they may cause cross talk due to close proximity of wires. H-tree clock lines do not produce corners sharper than 90°, and no two clock lines in an H-tree are ever in close proximity as a result cross talk is significantly less in H-tree as compared to X-tree.

The H-tree algorithm is applicable for very special structures. In general, clock terminals are randomly arranged all over the chip surface and require much more general algorithms.

11.1.4.2 The MMM Algorithm

Jackson, Srinivasan and Kuh [JSK90] presented a clock routing algorithm called Method of Means and Medians(MMM) for the CRP. The MMM Algorithm follows a strategy very similar to the H-tree algorithm. The MMM algorithm recursively partitions a circuit into two equal parts, and then connects the center of the mass of the whole circuit to the centers of mass of the two sub-circuits.

The algorithm is simple and yields good results. Let L_x be the list of points $\mathcal{P}$ sorted according to their x-coordinate. Let P_x be the median in L_x. Assign points in list to the left of P_x to $\mathcal{P}_L$. Assign the remaining points to $\mathcal{P}_R$. Due to the geometric nature of the problem, we may consider the partition of the point set as the partitioning of a region. Thus $\mathcal{P}_L$ and $\mathcal{P}_R$ partition the original region by x-median into two sub-regions with an approximately equal number of points in each sub-region. Similarly, $\mathcal{P}_B$ and $\mathcal{P}_T$ represent the division of $\mathcal{P}$ into two sets about the y-median.

The basic algorithm first splits $\mathcal{P}$ into two sets(arbitrarily in the x or y direction.). Assume that a split of $\mathcal{P}$ into $\mathcal{P}_L$ and $\mathcal{P}_R$ is selected. Then, the algorithm routes from the center of the mass of P to each of the center of mass of $\mathcal{P}_L$ and $\mathcal{P}_R$ respectively. The regions $\mathcal{P}_L$ and $\mathcal{P}_R$ are then recursively split in the y direction (the direction opposite to the previous one). Thus, splits between x and y are introduced on the set of points recursively until there is only one point in each sub-region. An example of this algorithm is shown in Figure 11.10.

Notice that basic algorithm discussed above ignores the blockages and produces a non-rectilinear tree. It is also possible that some wires may intersect with each other. In the second phase, each wire in the tree can be converted so that it only consists of rectilinear segments and avoids blockages and other nets.

11.1.4.3 Geometric Matching based Algorithm

Another binary tree based routing scheme is presented by Kahng, Cong and Robins [KCR93]. In this approach, clock routing is achieved by constructing binary tree using recursive *Geometric matching.* We call this algorithm *Geometric Matching Algorithm*(GMA). Unlike MMM algorithm which is a top down algorithm, GMA works bottom up. Let us start by defining the geometric matching.

Given a set $\mathcal{P}$ of n points, a geometric matching on $\mathcal{P}$ is a set of $\frac{n}{2}$ line segments whose endpoints are in $\mathcal{P}$, with no two line segments sharing the endpoint. Each line segment in the matching defines an *edge.* The cost of a geometric matching is the sum of the lengths of its edges.

To construct a tree by recursive matching, a forest of n isolated nodes is considered, each of which is a tree with the clock entry point being the node itself. The minimum-cost matching on these points yields $\frac{n}{2}$ segments, each of which defines a subtree with two nodes. As pointed out earlier, the center point of each segment will be called the tapping point and if the clock signal is provided at the tapping point, then the signal will arrive at the two endpoints of the segment with zero skew. The set of tapping points serves as the set of points for the next iteration of the algorithm. In general, the matching operation will pair up the clock entry points (i.e., roots) of all the trees in the current forest. At each level, the algorithm chooses the root of the newly merged tree to be the tapping point which minimizes the path length skew to the leaves of the two subtrees. Figure 11.11 shows GMA algorithm running on 8-point set.

When subtrees T_1 and T_2 are merged into a higher level subtree T_{12}, the optimal entry point may not be equidistant from the entry point of T_1 and T_2. Intuitively, balancing requires *sliding* the tapping point along the "bar of the H". However, it might not always be possible to obtain perfectly balanced path lengths in this manner. Therefore, *H-flipping* scheme is used: for each edge e H structure formed by the three edges of T_{12} is replaced by the H structure T'_{12} over the same four points which minimizes path length skew, and further minimizes tree cost.

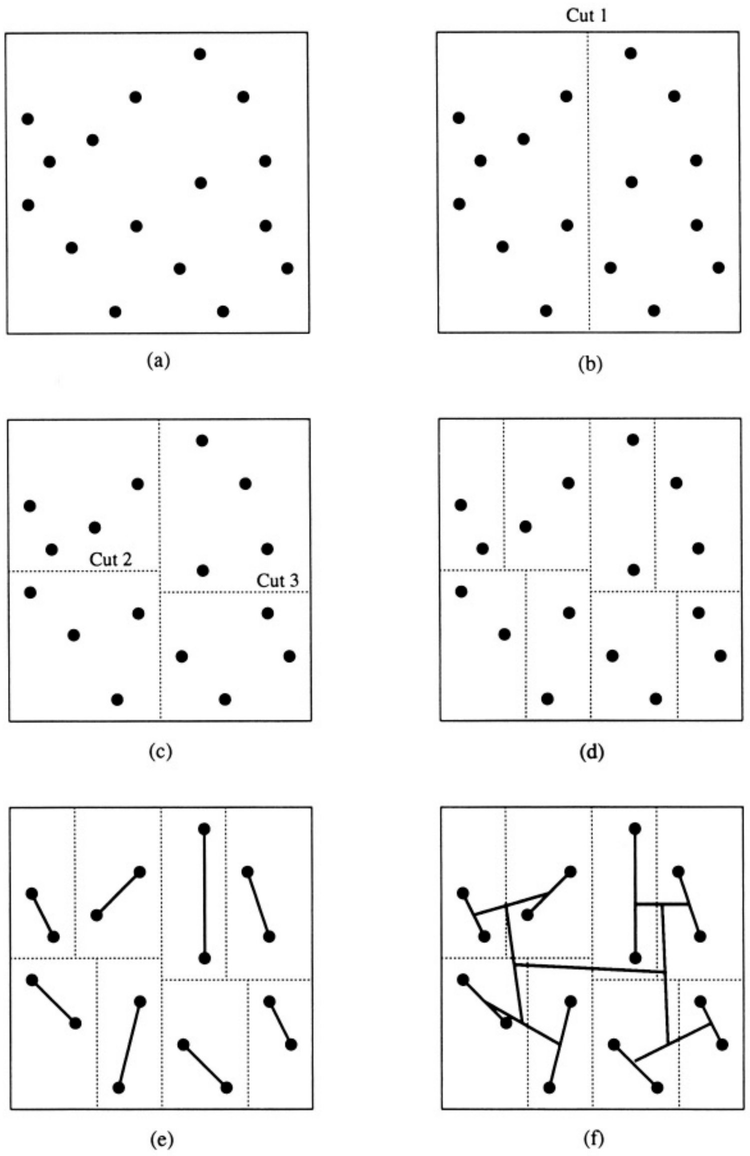


Figure 11.10: Routing by MMM Algorithm.

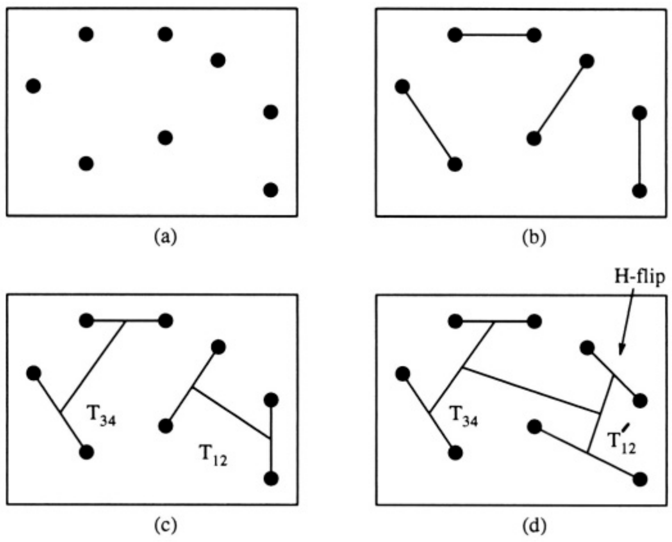


Figure 11.11: GMA Algorithm running on 8-point set.

As shown in Figure 11.11(c), two subtrees T_{34} and T_{12} are obtained, however, it is not possible to connect tapping points of T_{12} and T_{34}. Therefore, T_{12} is H-flipped to obtain T'_{12}. Finally T'_{12} is merged with T_{34} as shown in Figure 11.11(d).

Since the algorithm is based on geometric matching, its time complexity depends on the matching subroutine. The fastest known algorithms for general matching are $O(n^3)$. By taking advantage of planar geometry, the algorithmic complexity can be reduced to $O(n^{2.5} \log n)$.

11.1.4.4 Weighted Center Algorithm

The geometric matching algorithm is not applicable to Building Block Layout Problem (BBCRP) since it assumes that a complete layer is available for routing. Sherwani and Wu presented a new clock routing algorithm [SW91] called the *Weighted Center Algorithm* (WCA) for the BBCRP. In WCA, a weighted Clock Distribution Graph (CDG) for the problem is created. The vertices of CDG are the clock terminals, while the edges represent the steiner paths which may be used to connect two terminals. The weights of all the edges are obtained by the *RC* time delay calculation. The CDG is a complete graph, as it is always possible to connect two points in a BBCRP problem. The weight of the edge (u, v) is computed using a shortest path algorithm to find the (u, v) path followed by delay calculation for that path.

The WCA is greedy in nature and the basic idea of the algorithm is as follows: Using the clock distribution graph, the algorithm first finds the edge

(u, v) with the minimum weight (minimum delay), replace u and v with another vertex w which lies on their weighted center (tapping point). The CDG is updated to reflect new edge costs. Using this new CDG, the algorithm repeats this process recursively, until all the clock terminals are joined into one global weighted center. This global weighted center is designated as the clock signal entry point. Building up the clock distribution in this way, the clock skew between different clock terminals can be held to minimum. As the clock tree is built by using smallest edges first (just like the spanning tree algorithm), therefore the total clock tree wire length is minimized as compared to other clock distribution schemes. An example of clock routing by WCA is shown in Figure 11.12. WCA algorithm can be easily extended to multiple layers by including delays in via in calculation of path delays.

11.1.4.5 Exact Zero Skew Algorithm

Tsay [Tsa91] presented an algorithm for creating a clock tree with exact zero skew. The algorithm assumes that pairing of points has been done, and concerns itself with finding the tapping point very accurately, based on capacitive loading of the clock terminals as well as the delay in the sub-trees.

The zero skew algorithm is a recursive and bottom-up in nature. Assume two sub-trees T_1 and T_2 as shown in Figure 11.13. This algorithm computes a tapping point as discussed below.

In order to balance skew in both sub-trees, using Eq. (11.2), we have:

$$r_1(\frac{c_1}{2} + C_1) + t_1 = r_2(\frac{c_2}{2} + C_2) + t_2 \tag{11.3}$$

where t_i refers to the delay between node i and one of the leaves. Note that the delay would be the same for all leaves. Assuming that the total wire length between two trees is l, then the length of wire from the tapping point to the root of T_1 is equal to $x \times l$ (see Figure 11.13). Similarly, the wire length from tapping point to root of T_2 is given by $(1 - x) \times l$. Let α be resistance per unit length and β be the capacitance per unit length of wire. Then, $r_1 = \alpha x l$, $r_2 = \alpha(1 - x)l$, $c_1 = \beta x l$, and $c_2 = \beta(1 - x)l$. Solving equation 11.3 with these parameters we get

$$x = \frac{(t_2 - t_1 + \alpha l(C_2 + \frac{\beta l}{2})}{\alpha l(\beta l + C_1 + C_2)}$$

If $0 \leq x \leq 1$, then tapping point is on the line segment joining two trees. On the other hand, if $x < 0$ or if $x > 1$ then tapping point is not on the line segment and wire elongation is needed. This is done by *snaking* a short segment of wire which in essence allows the tapping point to fall on the wire. The actual length of the *snake* can be easily determined in the following manner: Let us assume that $x < 0$ and let length of the elongated wire is l'. Then its resistance is $\alpha l'$ and its capacitance is $\beta l'$. In order to balance the skew

$$t_1 = t_2 + \alpha l'(c_2 + \frac{\beta l'}{2})$$

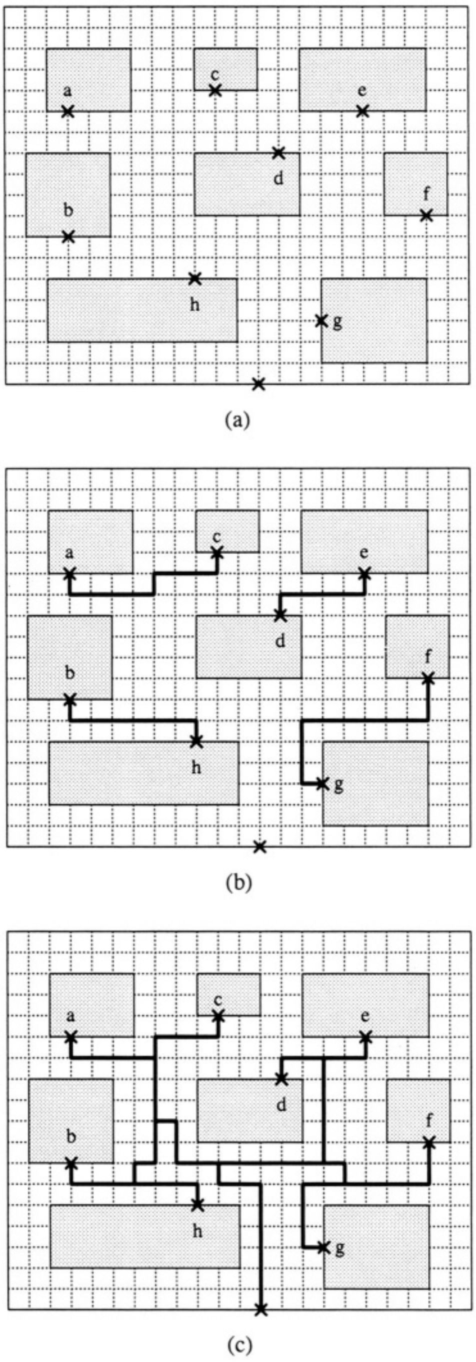


Figure 11.12: An example clock routing by weighted center algorithm.

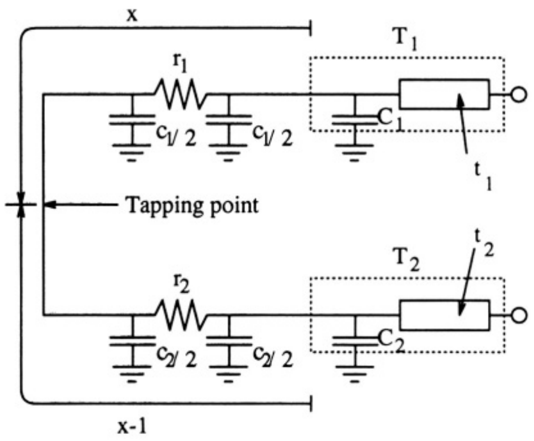


Figure 11.13: Merging of two trees.

and therefore l' is given by

$$l' = \frac{[\sqrt{(\alpha C - 2)^2 + 2\alpha\beta(t_1 - t_2)}] - \alpha C_2}{\alpha\beta}$$

Similarly, we can determine l' if $x > 1$. If l' is too long then additional buffer or capacitive terminators must be used to balance the skew.

We explain the algorithm with the help of eight pin example shown in Figure 11.14. The capacitive loading of each pin is shown in the figure. The capacitances shown are measured in farads (F) for the ease of calculation. However, the practical values of capacitances are usually in fifo farads (fF). According to the algorithm, first the tapping point M_1 is calculated for P_1 and P_2. The calculated location of M_1 is (3,21.52), which balances the delay of the path between P_1 and P_2 at 1.96 ns. The capacitance of M_1 is the sum of the capacitance at P_1, P_2 and the capacitance of the wire joining P_1 and P_2, i.e., C=8+3+(0.2x 8)=12.6 F. The tapping point M_2 for pairs P_3 and P_4 is calculated in the same manner. M_2 is calculated at (7,15) and its load capacitance is calculated to be 26.8 F. The delay from tapping point to both pins P_3 and P_4 is same, i.e., 3.99 ns. Similarly, tapping points for P_5 , P_6 and P_7, P_8 are calculated to be at (25,31) and (30,26) with load capacitances of 5 F and 30 F, respectively. At this point, we have four subtrees rooted at M_1, M_2, M_3 and M_4, such that M_1 and M_2 are in one pair and M_3 and M_4 in another. Following the same algorithm, we calculate the locations of tapping point M_6 at (7,17.97) with 41.50 F load capacitance. While calculating tapping point for M_3 and M_4, we find that $x = -0.175(< 0)$. The wire connecting M_3 and M_4, therefore, needs to be elongated. The length of elongation (snaking) for the case $x < 0$, l' is calculated to be 18.28. Therefore, 8.28 is the actual elongation (as shown in Figure 11.14). In this case the tapping point M_5 coincides with

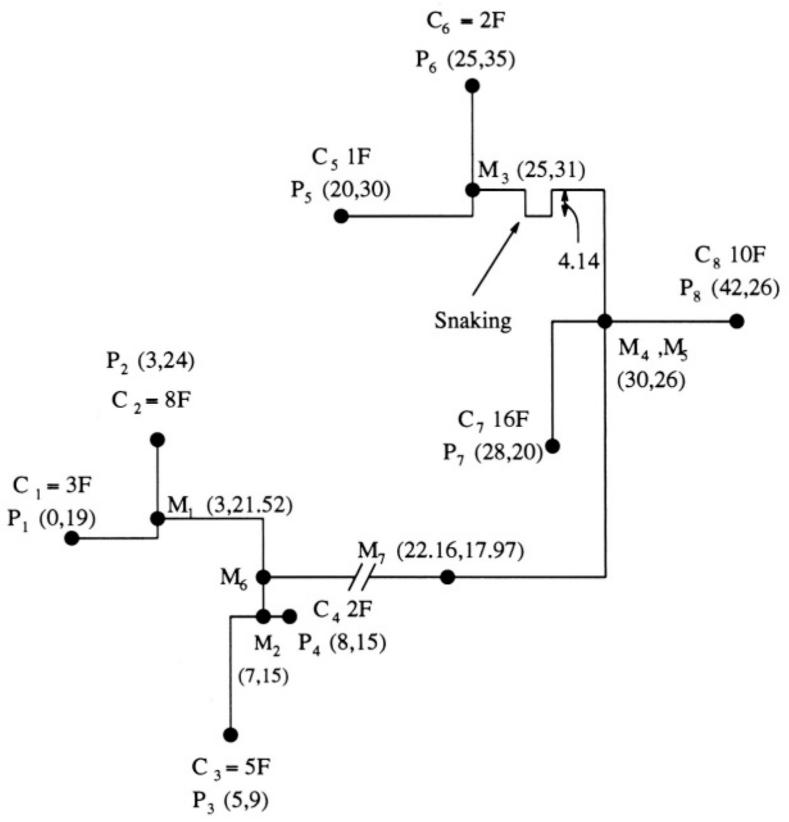


Figure 11.14: Zero skew clock routing.

M_4. The last step is to connect M_6 and M_5 to get final tapping point, which is calculated to be at (22.16,17.97). The final solution is shown in Figure 11.14. Note that the practical values of α and β are $3\ m\Omega$ and 0.02 fF, respectively. The chip width and height units are both in $1/10\ \mu$m.

As discussed above, this algorithm assumes pairing of points and only computes tapping points to construct the clock tree. Pairing of points can be done by using MMM or GMA if the entire layer is available. If obstacles are present, then WCA may be used to find point pairs.

11.1.4.6 DME Algorithm

Three independent groups [(Boese and Kahng), (Chao, Hsu and Ho), (Edahiro)] independently proposed the Deferred Merge Embedding (DME) method in [BK92, CHH92, Eda91]. DME is a linear - time algorithm which optimally embeds any given topology in the Manhattan plane, i.e. with exact zero skew and minimum total wire length. A generic DME is a two phase; bottom up and

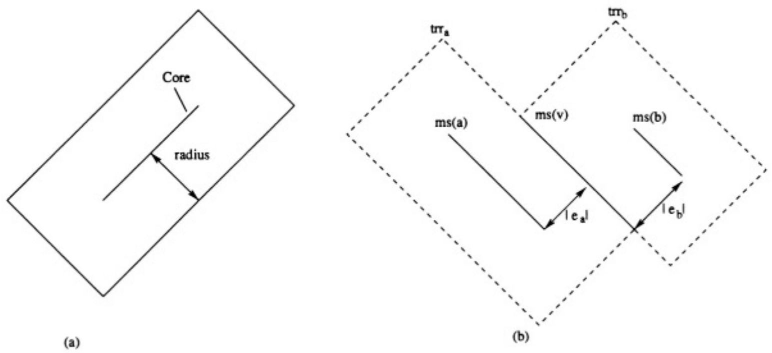


Figure 11.15: (a) Example of TRR (b) Construction of merging Segment ms(v)

top down process. The bottom up phase constructs a tree of *merging segments* which represent the loci of possible placements of nodes in the tree. The top down embedding phase determines the exact locations for internal nodes.

Before defining the DME formally, let us review definitions of terms commonly used in this section. A *manhattan arc* is defined to be a line segment, possibly of zero length, with slope +1 or -1; in other words a Manhattan arc is a line segment tilted at 45 deg. from the wiring directions. The collection of points within a fixed distance of manhattan arc is called a *tilted rectangular region* or TRR whose boundary is composed of manhattan arcs, (see Figure 11.15(a)). The manhattan arc at the center of the TRR is called its core. The *radius* of a TRR is the distance between its core and its boundary. Note that a manhattan arc is itself a TRR with radius 0. A merging segment at an internal node is a set of all placements which merge the TRRs of the child nodes with minimum wire cost.

A formal recursive definition of $ms(v)$, the merging segment of node $v \in G$, is as follows. If v is a sink s_i, then $ms(v) = \{s_i\}$ (note that this single point is a manhattan arc). If v is an internal node, then $ms(v)$ is a set of all placements $l(v)$, which merge TS_a and TS_b with minimum wire cost, that is, all points within distance $|\ e_a\ |$ of $ms(a)$ and within distance $|\ e_b\ |$ of $ms(b)$. If $ms(a)$ and $ms(b)$ are both manhattan arcs, then $ms(v) = trr_a \cap trr_b$ is obtained by intersecting two TRRs, trr_a with core $ms(a)$ and radius $|\ e_a\ |$, and trr_b with core $ms(b)$ and radius $|\ e_b\ |$. (See Figure 11.15(b)) If $ms(a)$ and $ms(b)$ are both Manhattan arcs, then $ms(v)$ is also a Manhattan arc [BK92]. Since the merging segment $ms(s_i)$ for each sink s_i is a single point and this a manhattan arc, by induction all merging segments are Manhattan arcs.

In the bottom up phase, each node $v \in G$ is associated with a merging segment which represents a set of possible placements of v in a minimum-cost ZST. The merging segment of a node depends on the merging segments of its two children, hence the bottom-up processing order. More precisely, let a and b be the children of node v in G, and let TS_a and TS_b denote the subtrees

```
**Algorithm** Build_Tree_of_Segments($G, S$)
**Input:** Topology $G$; set of sink locations $S$
**Output:**Tree of merging segments $TS$ containing $ms(v)$
    for each node $v$ in $G$, and edge length $| e_v |$for each $v \neq s_0$
**for** each node $v$ in $G$ (bottom-up order)
    **if** $v$ is a sink node,
        $ms(v) \leftarrow l(v)$
    **else**
        Let $a$ and $b$ be the children of $v$
        Calculate_Edge_Lengths($| e_a |, | e_b |$)
        Create TRRs $trr_a$ and $trr_b$ as follows:
            core($trr_a$) $\leftarrow ms(a)$
            radius($trr_a$) $\leftarrow | e_a |$
            core($trr_b$) $\leftarrow ms(b)$
            radius($trr_b$) $\leftarrow | e_b |$
        $ms(v) \leftarrow trr_a \cap trr_b$
```

Figure 11.16: Bottom Up Phase: Construction of Tree of Merging Segments TS

of merging segments rooted at a and b, respectively. We seek placements of v which allow TS_a and TS_b to be merged with minimum added wire while preserving zero skew. This means that we want to minimize $| e_a | + | e_b |$ in T, while balancing delays from l(v) to all leaves in the subtree rooted at v. The values of $| e_a |$ and $| e_b |$ which achieve this property are unique. They are computed and stored for use in the top-down embedding phase of DME. The details of the bottom up phase are given in 11.16.

Given the tree of merging segments corresponding to G, the top-down phase chooses exact embeddings of internal nodes in the ZST. For node v in topology G, (i) if v is the root node, then DME selects any point in $ms(v)$ to be $l(v)$; or if v is an internal node other than the root, DME chooses $l(v)$ to be any point in $ms(v)$ that is at distance $| ev |$ or less from the placement of v's parent p (the merging segment $ms(p)$ was constructed such that $d(ms(v), ms(p)) \leq | ev |$, so there must exist some $l(v)$ satisfying this condition). In case (ii), $l(v)$ can be any point in the intersection of $ms(v)$ and the square TRR trr_p which has radius $| e_v |$ and core $l(p)$. The details of the top down phase are given in 11.17.

DME requires an input topology, as a result, several authors have proposed topology constructions that yield low-cost routing solutions when DME is applied.

```
Algorithm Find_Exact_Placements($TS$))
Input: Tree of segments $TS$ containing $ms(v)$
    and value of $| e_v |$ for each node $v$ in $G$
Output: $ZST\ T(S)$
for each internal node $v$ in $G$ (top down order)
    if $v$ is the root
        Choose any $l(v) \in ms(v)$
    else
        Let $p$ be the parent node of $v$
        Construct $trr_p$ as follows:
            core($trr_p$) $\leftarrow l(p)$
            radius($trr_p$) $\leftarrow | e_v |$
            Choose any $l(v) \in ms(v) \cap trr_p$
```

Figure 11.17: Top Down Phase : Construction of ZST by embedding internal nodes of G within TS

11.1.5 Skew and Delay Reduction by Pin Assignment

In [WS91], clock routing is done at pin assignment phase of the layout. If clock routing is considered at the floorplanning stage of the layout, then some flexibility in location of the clock terminals is allowed. During layout several iterative steps in placement and routing phases are allowed. During these redesign cycles, circuit layout is iteratively improved and design is made 'more' rigid. This allows successive re-positioning of clock terminals of functional block. By appropriately locating the clock terminals total clock skew and delay can be reduced significantly. Movable Clock Terminal Routing Problem (MCTRP) is a clock routing problem in which the clock terminals of the functional blocks in floorplan can be moved along the block boundaries.

MCTRP basically consists of two subproblems. The first subproblem is to find the best location for clock terminal of each functional element to minimize the clock delay. The second subproblem is to find a clock routing such that the clock signals can reach all the terminals with equal time delay. The first subproblem is shown to be NP-complete [WS91], and a greedy heuristic algorithm is presented. The second subproblem of interconnecting points to obtain a minimum skew can be solved by using any algorithm discussed earlier for BBCRP.

11.1.6 Multiple Clock Routing

Large VLSI systems may use multiple clocks because the existence of multiple clock phases gives an extra degree of freedom to the timing characteristics of the synchronizing circuits. The multiple clock routing problem is, however, more complex because of two types of skew: the *intra clock skew* within a clock and the *inter clock skew* among multiple clocks. Thus, for high performance circuits, it is necessary to develop a routing algorithm for multiple clocks which minimizes the delay as well as both types of skews. An additional problem of routing two phase clock on a single layer is crossing of two clock signals. This problem is resolved by the use of 'low resistance' *crossunders.*

Let us consider a system with k clocks $\phi_1, \phi_2, \ldots, \phi_k$. Let us also assume that there are n blocks each requiring one clock input from each clock. Let $\mathcal{P}$ be the set of n clock terminals. Let $P_{ij} \in \mathcal{P}$ denote the terminal of clock ϕ_i at block j. Let t_{ij} be the arrival time of clock signal at P_{ij}. For any clock ϕ_i, intra clock skew ρ_i is defined as,

$$\rho_i = \max\{t_{ij}\} - \min\{t_{ij}\} \quad j = \{1, 2, \ldots, n\}$$

For a block j, the inter-clock skew is defined as,

$$\chi_j = \max\{t_{ij}\} - \min\{t_{ij}\} \quad i = \{1, 2, \ldots, k\}$$

For the system, the cross skew is defined as,

$$\chi^* = \max\{t_{ij}\} - \min\{t_{ij}\} \quad i = \{1, 2, \ldots, k\}, j = \{1, 2, \ldots, n\}$$

Thus the objective of multiple clock routing system is not only to minimize σ_j for each clock ϕ_j, but also to minimize χ_i between each set of clocks and χ^*, the cross skew, of multiple clock system. However, this task is complicated due to intersection between different clock trees. When two clock trees intersect, crossunder may be used to pass one signal under the other signal. Crossunders should be minimized, subject to the constraint that the number of crossunders should be equalized for multiple clocks, in order to equalize the signal delays.

In [KHS92], Khan, Hossain, and Sherwani proposed zero skew routing for two clocks. The basic idea is to build the two trees independently. In the first phase points are paired up and crossunders are assigned to allow two trees to be routed in a planar fashion. This phase attempts to minimize the cross-skew by alternating the order in which the crossunders are used. In this way the number of crossunders are balanced on each path of both trees. In the second phase, algorithm eliminates intra clock skew in both trees independently, taking crossunders into account.

11.2 Power and Ground Routing

In VLSI design, almost all the blocks need power supply and need to be connected to ground as well. The power and ground nets are usually laid out entirely on the metal layer(s) of the chip due to smaller resistivity of metal as compared to poly. Since, contacts(vias) also significantly add to the parasitics, it is also advisable to utilize a planar single-layer implementation of these nets. It should be noted that the area requirements for power and ground nets depend on the voltage drop, current density and other constraints. In case of normal signal nets, the current they carry is very small. Hence, they can be routed with minimum-width wires. Thus minimizing the total wire length also ensures minimizing the area needed to route them. The same is not true for power and ground nets.

Routing of power(VDD) and ground(GND) nets consists of two main tasks: (i) construction of interconnection topology, and (ii) determination of the widths of the various segments of topologies. In recent years, most of the research and development efforts have been focused on the topological routing of the power and ground signals.

For a given placement of arbitrary rectangular blocks on a chip, the problem of routing power and ground nets is to find two non-intersecting interconnection trees, each for VDD and GND. The width of trees at any point must be proportional to the amount of current being drawn by the points in that subtree. We assume that each block has an entry point for VDD and GND. In standard cell designs VDD and GND are routed by using inter-weaved combs (as discussed in chapter 1). In fact, VDD and GND are already laid out in the cells and simply connected on one side with GND and VDD on the other. In gate arrays, VDD and GND routing is similar to standard cell and is usually laid out on the master, and not subject to customization.

A simple scheme that is often used for power and ground using two layers of metal is a grid structure. Several rows of horizontal (M5) wires for both power and ground run parallel to each other. The vertical wires run in M4 and connect the horizontal wires. In this way, two grids are formed. All the blocks simply connect to the nearest power and ground wire. This scheme is shown in Figure 11.18. The blocks and connections to blocks are not shown for sake of clarity.

Syed and El Gamal [SG82] proved the necessary and sufficient conditions for a planar routing of power/ground nets using single pads. Two nets can be routed on a single layer without crossover only if there exists a cut for each block in the chip that separates the terminals of one net from the terminals of the other net. The nets are grown as interdigitated trees. Applying simple traffic rules to the free channels between modules prevent the two trees from crossing. An example routing is shown in Figure 11.19.

Another approach is proposed by Moulton [Mou83]. The basic idea of the proposed algorithm is to partition the chip surface into a VDD region and a GND region and then to route each net within the appropriate region. It is easy to see that if all modules are visited once while keeping the VDD to one side and the GND on the another side, a cycle can be drawn which will connect all modules to the VDD and GND pads. Thus a Hamiltonian cycle is drawn, and the algorithm allows this cycle to determine the layout of the trees. Tree traversal determines how much current might flow through each wire of the tree. The maximum current of a wire ending in a terminal is the maximum current of the terminal. The maximum current of other wires is the sum of its children's maximum currents. After every wires maximum current is known, multiplying it by a design-rule constant gives every wires minimum width. Both the Hamiltonian cycle and Steiner tree operations are, however, computationally very expensive.

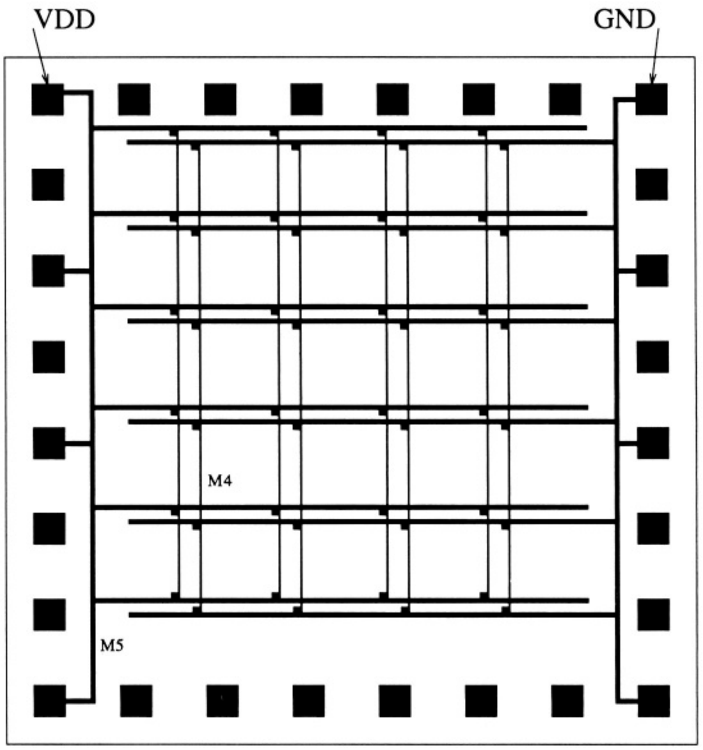


Figure 11.18: Power and Ground Routing using Grids.

The algorithm proposed by Rothermel and Mlynski [RM81] tends to route nets interdigitated. It extends one net from the left edge of the chip, and the other from the right. This routing order of the connecting points is determined by the horizontal distances of connecting points from the edge of the chip. Calculation of nets is accomplished by a combined Lee and Line Search algorithm. At first only points of the left net which lie in the left half of the chip are routed. Then those points of the right net which lie in the right half of the chip are routed. This process uses a fast line search algorithm similar to Hightower's algorithm [Hig80]. Next, all other points of the two sets are routed by Lee's algorithm [Rub74], which takes into account obstacles created by already routed net segments.

In [HF87], Haruyama and Fussell propose a method for routing non-crossing VDD and GND trees on a layer which tries to minimize the chip area devoted to power routing under metal migration and voltage drop constraints. The metal migration has to be prevented by using a wide enough metal wire. In addition, the voltage drop has to be kept small, because a large voltage drop between a pad and a module decreases switching speed and noise margin. The algorithm

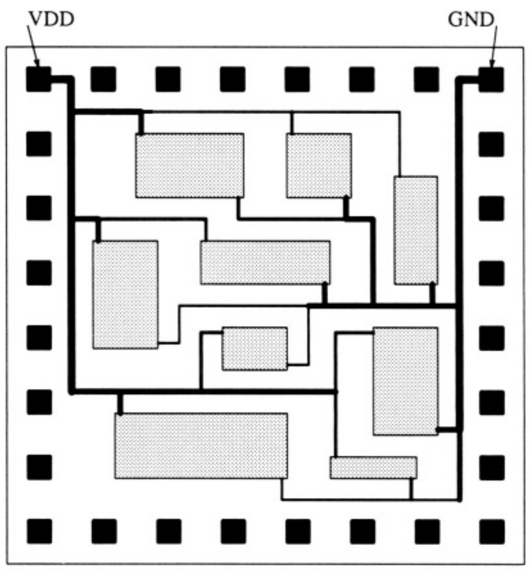


Figure 11.19: Power and Ground Routing using interdigitated Trees.

also takes the width of channels into consideration so that if a channel is too congested to allow a wire to pass through it, the wire avoids the congested channel and chooses other channels. The goal is to grow the VDD and the GND trees by connecting modules, one by one, to trees under construction. Modules are sorted by their power consumption. First, the pins of the most power-consuming module are connected to the pads. Subsequent modules are routed in decreasing order of power consumption. This is a greedy approach based on the notion that it is better for more power-consuming modules to have shorter paths, since more power-consuming modules need wider wires in order to be supplied with more current. At earlier stages in the routing, paths can generally be shorter, since they are blocked by fewer wires already routed. A smaller area is thus occupied by a wire. Pins of the second module (and later considered modules) are connected to non-root vertices of the net (or possibly to an unconnected pad when there is more than one VDD or GND pad). The constructed net is a tree whose leaves are pins of modules and whose root is a pad. When there is more than one VDD or GND pad, the algorithm may create a forest of multiple trees. The wire area becomes even smaller than when there is only one VDD pad and one GND pad because the search can find a shorter path to a power source. This multiple pad method eases the current load of each pad.

Routing of power and ground nets is often given first priority, because the power and ground wires are usually laid out entirely on a metal layer(s) due to its low resistivity, as described above. Signal nets may share the metal layer(s) with power and ground, but they change layers whenever a power or ground wire is encountered.

11.3 Summary

Clock routing is one of the factors which determines the throughput of any chip. In advanced VLSI systems, clock skew caused by interconnection delay, if not controlled, can lead to significant performance degradation. Ideally, the clock skew should be less than 5-10 percent of the clock period. Several clock routing algorithms have been proposed, and it is possible to route a clock very accurately with exactly zero skew if a complete layer is available. Much research remains to be done for clock routing problems with obstacles on the routing layer. Multiple clock routing is another area that promises to be a focus of attention as more and more designs use multiple clocks. Some radical design methodologies have been presented (asynchronous self timed systems), which do away with system level clock. Instead, the flow of information from unit to unit is based on hand shaking protocols and time stamping of the data. However, this approach presents considerable design difficulties. Clock signal serves as a convenient sequence and timing reference and it would be difficult to design circuits with such sequencing.

Power and ground routing needs special attention because of wire widths. Power and ground wires carry large amounts of current and as a result wider wires are used. The width cannot be uniform since current requirement is not uniform over the chip surface. As a result, wires must be carefully sized to allow proper current flow. Too thin wires lead to low currents, while too wide wires may lead to wastage of area.

Currently, Aluminum (Al) is the metal of choice for long interconnect lines, such as clock, power and ground lines. Al has low resistivity, good adherence to silicon and silicon oxide, it is easy to bond, pattern and deposit. Furthermore, Al is low-cost, readily available and easy to purify. Despite these qualities, Al suffers from a variety of problems, such as, electro-migration and contact failures. Au, Cu and Ag all have resistivities lower than that of Al. However, replacing Al with any of them will require a major effort because none of them are as compatible with integrated circuit processing as Al.

In future, superconductivity and optical interconnect offer alternative to aluminum wires for clock routing. In particular, optical interconnect allows fast (speed of light), reliable (no metal migration problems), noise-free and easy clock distribution. It is possible to distribute a light signal to all the functional units with zero skew and no delay. However, both superconductor and optical interconnect are still topics of research and are currently not practical.

11.4 Exercises

1. Generate an instance of CRP by randomly placing n points on a plane. Choose (randomly) one point on the boundary of the plane as the clock

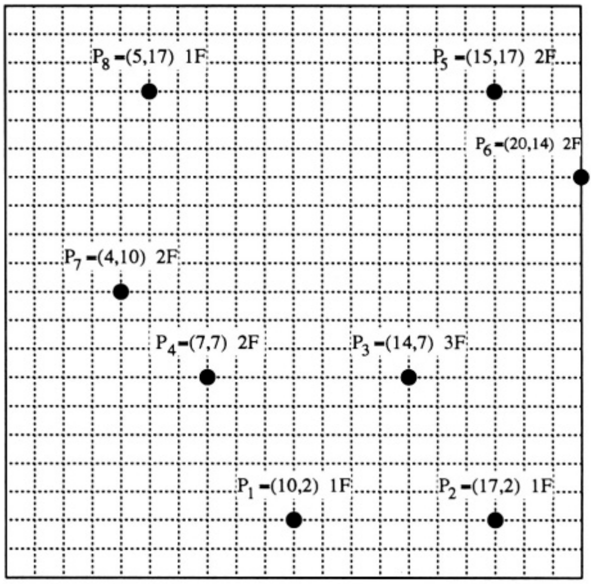


Figure 11.20: Sample clock points.

entry point. Implement MMM algorithm and test it on the instance generated.

2. Implement Geometric Matching algorithm for point set developed in exercise above. Compare the results with that of MMM Algorithm, in terms of skew and total wire length.

3. Generate an instance of BBCRP by randomly placing n rectangles on a plane such that none of the rectangles intersect. Randomly choose a point on the boundary of each rectangle as its clock terminal. Also choose (randomly) one point on the boundary of the plane as a clock entry point. Implement Weighted center algorithm and test it on the generated problem. Compare the total wire length results for weighted center algorithm and geometric algorithm.

† 4. It is possible to combine geometric matching and weighted center algorithms. The basic idea is to use the clock distribution graph to identify the paths, and use geometric matching to pair up the points. Modify the weighted center algorithm to use geometric matching.

5. Consider the 8 point instance given in Figure 11.20. Find the routing with exact zero skew. Assume $\alpha = 0.1\ \Omega$ and $\beta = 0.2\ F$.

6. For an instance of CRP, randomly assign load capacitances of each point between 1 F and 20 F. Assume $\alpha = 0.1\ \Omega$ and $\beta = 0.2\ F$. Implement the

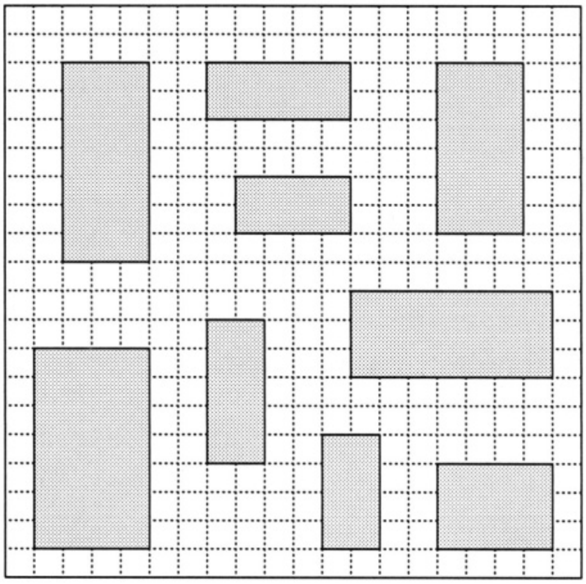


Figure 11.21: Instance of MCTRP.

Exact zero skew algorithm and test it on instance generated above. Use GMA for point pairing.

7. For a given instance, compute the number of times snaking is required in Exact zero skew algorithm if MMM algorithm is used for pairing up the points instead of GMA.

8. Consider the following instance of MCTRP given in Figure 11.21. Find clock entry point for each block such that minimum skew algorithm can route the clock net with minimum wire length. The clock entry point of chip can be placed anywhere on the boundary.

9. Consider the points given in Figure 11.22. Find the optimal clock entry point for this chip.

† 10. Develop an algorithm which finds the optimal clock entry point of the chip for any instance of BBCRP. Can this problem be solved in polynomial time ?

† 11. We define the following restricted *Standard Cell Clock Routing Problem*(SCCRP): Given a $l \times w$ grid representing a channel and L clock terminals on top and bottom, and clock entry point on the right side of the channel(Figure 11.23). Find a routing with minimum delay and zero skew. More precisely, the points are located on (0,0), (2,0), ..., $(l,0)$, $(0,w)$, $(2,w)$, ..., (l,w) and the clock entry point is located at $(l,\frac{w}{2})$.

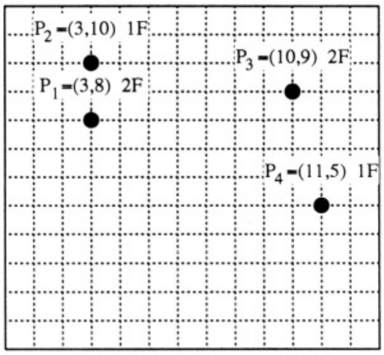


Figure 11.22: Finding clock entry point.

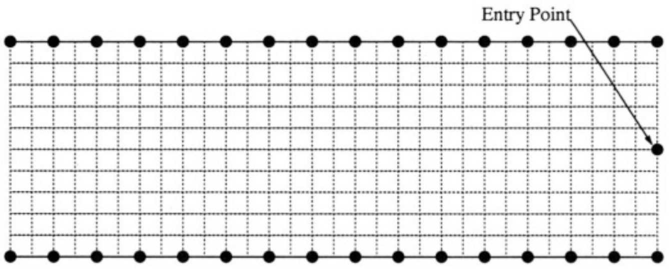


Figure 11.23: Standard cell clock routing problem.

† 12. Given only one layer for clock routing, prove that there exists a routing for SCCRP with zero skew if $w = 2\log(\frac{l+1}{2}) + 1$ tracks are allowed in the channel. Prove that the maximum delay in such routing is no more than $l + w$.

† 13. Given two layers, prove that there exists a routing for SCCRP with zero skew if $w = \log(\frac{l+1}{2}) + 1$ tracks are allowed in the channel. Assume that vias are ideal, i.e., they do not cause any additional delay.

† 14. Given two layers, develop a routing for SCCRP so that the path length and the number of vias is equal from clock entry point to each terminal.

Bibliographic Notes:
Bakoglu [Bak90] presents an excellent coverage of parameters involved in interconnect. Details of delay computation may also be found there. Another algorithm for the clock routing problem of building block design has been presented by Ramanathan and Shin [RS89]. The problem of power and ground routing has been extensively studied and several related problems have also

been investigated. In [HSVW90a], Ho, Sarrafzadeh, Vijayan and Wong discuss the problem of minimizing the number of power pads, in order to guarantee the existence of a planar routing of multiple power nets. They also show that the general pad minimization problem is NP-complete. They derive a general lower bound and present a heuristic for the general problem. They also present optimal algorithms for some special cases. In [LG87], Lursinsap and Gajski, consider the problem of power routing in a top-down design approach. In this approach, a layout is decomposed into cells connected by abutment. The active cells contain transistors and interconnections, while passive cells are routing cells. They consider power routing of all active cells so that the total wire length is minimized, and present an optimal power routing algorithm for this special problem. In [XK86], Xiong and Kuh present an algorithm which grows both the VDD (from one side) and the GND trees (from the other side) simultaneously using a plane sweep algorithm. In [Eda94], Edahiro presents a bucket algorithm for zero skew routing with linear time complexity on the average. In [CS93], Cho and Sarrafzadeh introduce a new approach for optimizing clock tree. In [EL96], a clock buffer placement algorithm is proposed.