Chapter 13

Physical Design Automation of FPGAs

Despite advances in VLSI design automation, the time-to-market for even an ASIC chip is unacceptable for many applications. The key problem is the time taken due to fabrication of chips, and therefore there is a need to find new technologies, which minimize the fabrication time. Gate Arrays use less time in fabrication as compared to full-custom chips, since only routing layers are fabricated on top of pre-fabricated wafer. However, fabrication time for gate-arrays is still unacceptable for several applications. In order to reduce time to fabricate interconnects, programmable devices have been introduced, which allow users to program the devices as well as the interconnect. In this way all custom fabrication steps are eliminated.

Programmable Logic Devices (PLDs) are devices that can be programmed by the user to implement a logic function. These devices offer short turnaround time and as a result they are becoming increasingly important for systems as well as system prototypes. In addition, they have a low manufacturing cost and are fully testable. One such device which is gaining more popularity is *Field Programmable Gate Arrays (FPGAs)* .

As discussed in Chapter 1, FPGA is a new approach to ASIC design that can dramatically reduce manufacturing turn around time and cost. In its simplest form, an FPGA consists of a regular array of programmable logic blocks interconnected by a programmable routing network. A programmable logic block is a RAM and can be programmed by the user to act as a small logic module. Given a circuit, user can program the programmable logic module using an FPGA programming tool. The key advantage of FPGAs is re-programmability. The RAM nature of the FPGAs allows for in-circuit flexibility that is most useful when the specifications are likely to change in the final application. In some applications such as remote sensors, it is necessary to make system updates via software. In FPGA, a data channel is provided, which allows easy transfer of the new logic function and reprogramming the FPGA.

The physical design automation of FPGAs involves mainly three steps which

A	B	C	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Table 13.1: Truth table for $f = \bar{A}BC + A\bar{B}\bar{C}$.

include partitioning, placement and routing. Partitioning problem in FPGAs is significantly different from the partitioning problems in other design styles. This problem mainly depends on the architecture in which the circuit has to be implemented. Placement problem in FPGAs is very similar to the gate array placement problem. The routing problem in FPGAs is to find a connection path and program the appropriate interconnection points. In this chapter, we discuss the architecture of FPGAs, their physical design cycle, and algorithms used for partitioning and routing problems in FPGAs.

In order to gain a better perspective of the physical design problems related to FPGAs, we start with a description of FPGA architectures.

13.1 FPGA Technologies

An FPGA architecture mainly consists of two parts: the logic blocks, and the routing network. A logic block has a fixed number of inputs and one output. A wide range of functions can be implemented using a logic block. Given a circuit to be implemented using FPGAs, it is first decomposed into smaller sub-circuits such that each of the sub-circuit can be implemented using a single logic block. There are two types of logic blocks. The first type is based on Look-Up Tables (LUTs), while second type is based on multiplexers.

1. **Look-up table based logic blocks:** A LUT based logic block is just a segment of RAM. A function can be implemented by simply loading its LUT into the logic block at power up. If function $f = \bar{A}BC + A\bar{B}\bar{C}$ needs to be implemented, then its truth table (shown in Table 13.1) is loaded into the logic block. In this way, on receiving a certain set of inputs, the logic blocks simply 'look up' the appropriate output and set the output line accordingly. Because of the reconfigurable nature of the LUT based logic blocks, they are also called the *Configurable Logic Blocks (CLBs).*

   It is clear that $2^{I_{max}}$ bits are required in a logic block to represent a I_{max}-bit input, 1-bit output combinational logic function. Obviously, logic

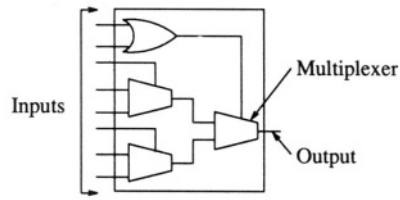

Figure 13.1: A multiplexer based logic block.

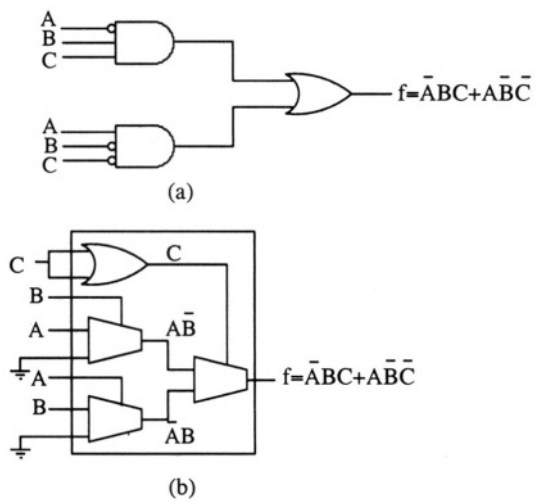

Figure 13.2: A logic function mapped to a multiplexer based logic block.

blocks are only feasible for small values of $I_{\max}$. Typically, the value of $I_{\max}$ is 5 or 6. For multiple output and sequential circuits the value of $I_{\max}$ is even less.

2. **Multiplexer based logic blocks:** Typically a multiplexer based logic block consist of three 2-to-l multiplexers and one two-input OR gate as shown in Figure 13.1. The number of inputs is eight. The circuit within the logic block can be used to implement a wide range of functions. One such function, shown in Figure 13.2(a) can be mapped to a logic block as shown in Figure 13.2(b). Thus, the programming of multiplexer based logic block is achieved by routing different inputs into the block.

There are two models of routing network: the segmented and the non-segmented.

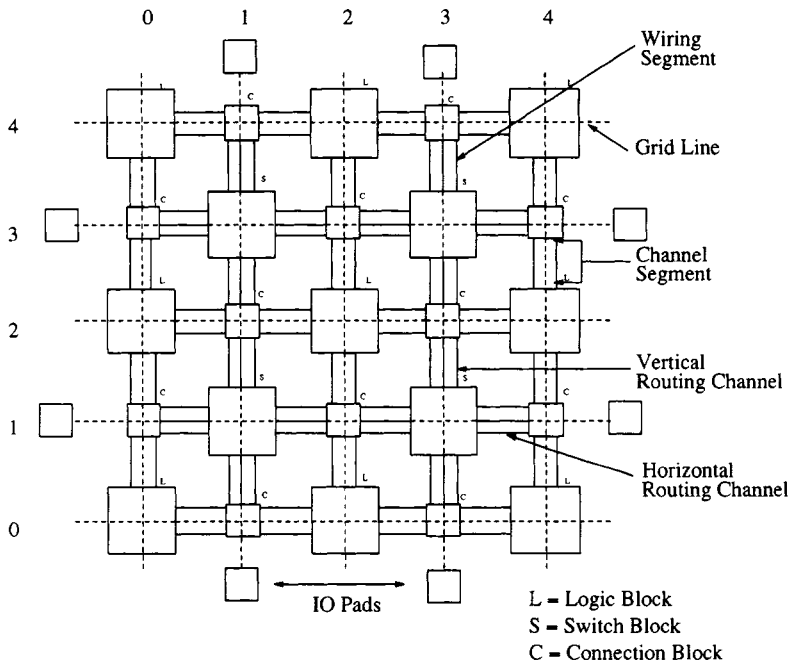

Figure 13.3: Non-segmented interconnect.

1. **Non-segmented model:** A typical non-segmented model is shown in Figure 13.3. The non-segmented model is set up as a regular grid of five horizontal and five vertical metal lines passing between switch blocks (S). The switch blocks are rectangular switch boxes. They are used to connect the wiring segments in one channel segment to those in another. Depending on the topology of the S block, each wiring segment on one side of S may be switchable to either all or some fraction of wiring segments on each side of the S block. The fewer the wiring segments a wiring segment can be switched to, the harder the FPGA is to route. Figure 13.6 and Figure 13.5 are two of the switch box architectures used by Xilinx for their 4000XC and 2000XC series. In Fig 13.5 a predefined set of programmable connections based on some probability and statistical data is used to obtain an efficient and economical switch box routing architecture. On the other hand, Figure 13.6 shows a more versatile and efficient routing architecture, but far more expensive to implement.

   In addition to the switch blocks, there are the connection blocks (C) that are used to connect the logic block pins to the routing channels. Depending on the topology, each L block pin may be switchable to either all or some fraction of wiring segments that pass through the C block. Again, the fewer the wiring segments a pin can be switched to, the harder

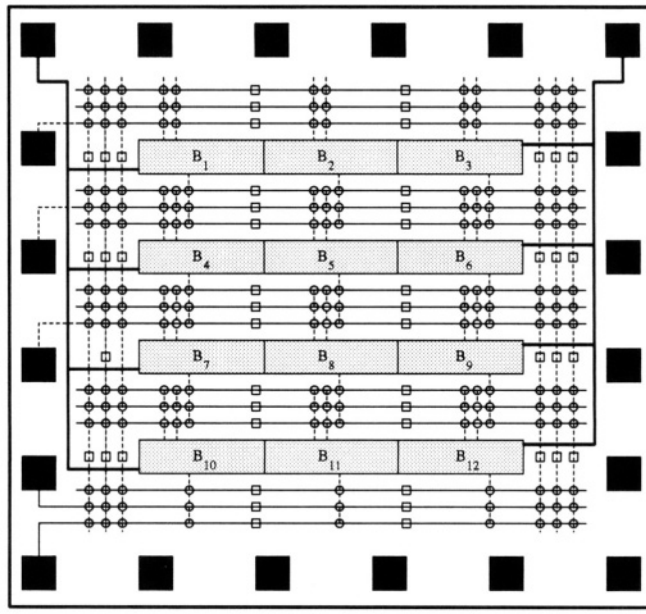

Figure 13.4: Segmented interconnect.

the FPGA is to route.

2. **Segmented model:** In segmented model, the tracks in the channels contain predefined wiring segments of same or different lengths. Other wiring segments pass through the channels vertically. Each input and output of a logic block is connected to a dedicated vertical segment. As a result, there are no vertical constraints. There are additional global vertical lines which provide connections between different channels. Connection between two horizontal segments is provided through an *antifuse,* whereas the connection between a horizontal segment and a vertical segment is provided through a *cross fuse* (see Figure 13.4). Programming (blowing) one of these fuses provides a low resistance bidirectional connection between two segments. When blown, antifuses connect the two segments to form a longer one. In order 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. When the objective is to fabricate reconfigurable routing network re-programmable switches can be used instead of fuses.

   The segmented model is *uniform* if the segments in all tracks have same length and the antifuses in different tracks in a channel are aligned in columns.

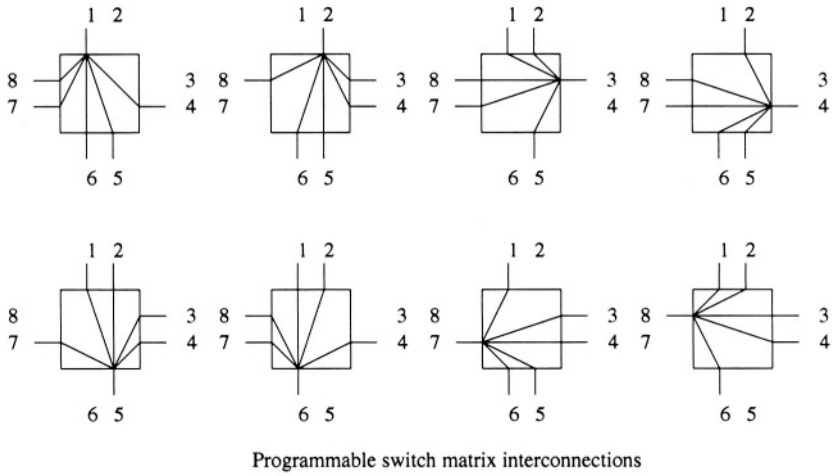

Figure 13.5: XC2000 family switch box architecture

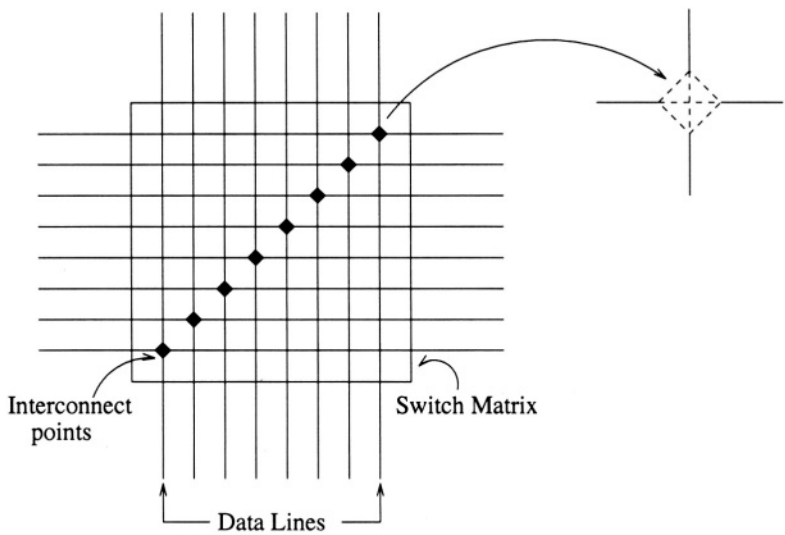

Figure 13.6: XC4000 family switch box architecture

The segmented model normally has advantage over the non-segmented model in terms of utilization of routing resources. In the non-segmented model only one segment of one net can be routed on a track. Whereas, in the segmented model, the segments of several nets can be assigned to a track as long as no two net segments are assigned to the same track segment.

The total number of programmable switches in the segmented model is higher as compared to the number of switches in the non-segmented model. The delay of a net is directly proportional to the number of programmable switches used to route that net. The number of programmable switches used to route a net is higher in segmented model as compared to non-segmented model. As a result, the non-segmented model is preferred over the segmented model when the performance is the primary objective.

13.2 Physical Design Cycle for FPGAs

The physical design cycle for FPGAs consists of the following steps:

1. **Partitioning:** The circuit to be mapped onto the FPGA has to be partitioned into smaller sub-circuits, such that each sub-circuit can be mapped to a programmable logic block. Unlike the partitioning in other design styles, there are no constraints on the size of a partition. However, there are constraints on the inputs and outputs of a partition. This is due to the unique architecture of FPGAs.

2. **Placement:** In this step of the design cycle, the sub-circuits which are formed in the partitioning phase are allocated physical locations on the FPGA, i.e., the logic block on the FPGA is programmed to behave like the sub-circuit that is mapped to it. This placement must be carried out in a manner that the routers can complete the interconnections. This is very critical as the routing resources of the FPGA are limited. The placement algorithms for general gate arrays are normally used for the placement in FPGAs, and therefore, will not be discussed in this chapter.

3. **Routing:** In this phase, all the sub-circuits which have been programmed on the FPGA blocks are interconnected by blowing the fuses between the routing segments to achieve the interconnections.

Figure 13.7 shows complete physical design cycle of FPGAs. System design is available as a directed graph which is partitioned in second step. Placement involves mapping of sub-circuits onto CLBs. Shaded rectangles represent CLBs which have been programmed. Final step is routing of channels.

13.3 Partitioning

A I_{max}-input, 1-output LUT based logic block is powerful than a I_{max}-input, 1-output multiplexer based logic block, as the former can implement all the

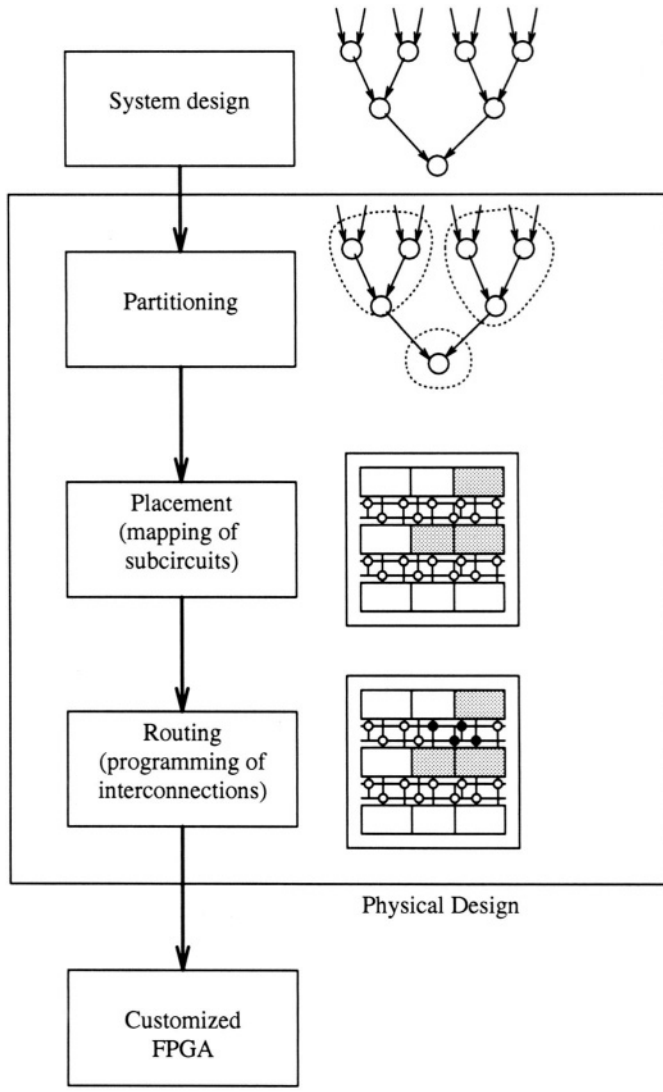

Figure 13.7: Physical design cycle of FPGA.

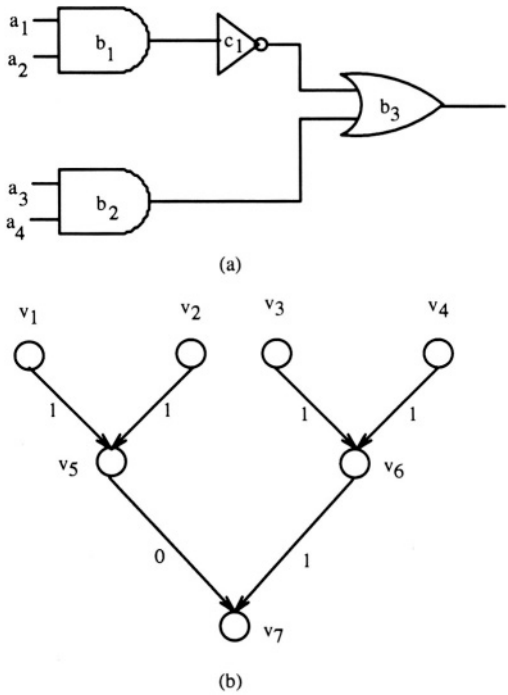

Figure 13.8: A boolean circuit and its DAG representation.

logic functions of $I_{\max}$ inputs, whereas the capabilities of later are limited by the circuitry inside the block. As a result the LUT based logic blocks are more popular than the multiplexer based logic blocks. Therefore, in this section we discuss a partitioning problem only for the LUT based logic blocks.

The circuit to be mapped onto the FPGA has to be partitioned into smaller sub-circuits such that each sub-circuit can be implemented using the logic blocks. In order to achieve this, we model the boolean network (N) consisting of AND, OR and NOT gates using directed acyclic graph (DAG) as follows: Let N be a set of inputs $a_1, a_2, \ldots, a_l$, B be a set of AND and OR gates $b_1, b_2, \ldots, b_m$ and C be a set of NOT gates $c_1, c_2, \ldots, c_n$. The corresponding DAG is defined as $G = (V_a \cup V_b, E)$, where $V_a = \{v_1, v_2, \ldots, v_l\}$ such that $v_i \in V_a$ represents an input a_i to the network, and $V_b = \{v_{l+1}, v_{l+2}, \ldots, v_{l+m}\}$ such that $v_j \in V_b$ represents an AND or OR gate g_{j-l}. An edge $e_{ij} = (v_i, v_j) \in E$, directed from v_i to v_j, represents a connection from the output of g_i to an input of g_j, either through a NOT gate or direct. If connection between g_i and g_j is through a NOT gate, then a weight 0 is associated with the edge e_{ij}, otherwise the weight of edge $e_{ij} \in E$ is 1. A logic network is shown in Figure 13.8(a) and corresponding DAG is shown in Figure 13.8(b).

The nodes in V_a are also referred as the *input nodes.* A node $v_i \in V$ is a

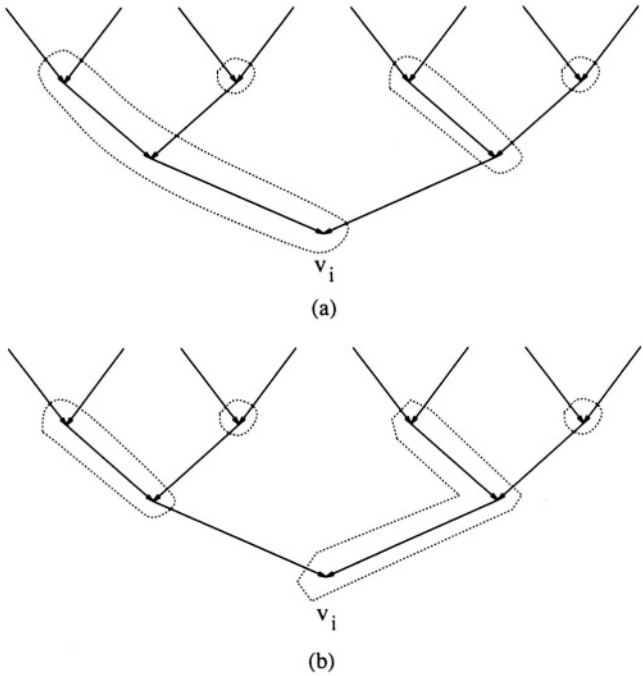

Figure 13.9: Utilization division.

fan-in node of $v_j \in V$ if there is a directed edge from node v_i to node v_j. An input node does not have a fan-in node.

Graph $G_i = (V_i, E_i)$ is a *subgraph* of a graph $G = (V, E)$, if $V_i \subseteq V$ and $E_i = \{e_{ij} | e_{ij} = (v_i, v_j), v_i \in V_i, v_j \in V_i\}$. The *indegree* $I(G_i)$ of a directed subgraph G_i is defined as the number of edges coming into G_i, while $O(G_i)$, the *outdegree* of G_i is defined as the number of edges coming out of G_i. More precisely,

$$I(G_i) = |\{(v_i, v_j) | (v_i, v_j) \in E, v_i \notin V_i, v_j \in V_i\}|$$

$$O(G_i) = |\{(v_i, v_j) | (v_i, v_j) \in E, v_i \in V_i, v_j \notin V_i\}|$$

The partitioning problem can be formally stated as follows: Given a directed acyclic graph G, maximum number of output terminals of a logic block denoted as $O_{\max}$ and maximum number of input terminals of a logic block denoted as $I_{\max}$, partition G into minimum number of vertex sets $V_1, V_2, \ldots, V_k$ such that subgraphs $G_1, G_2, \ldots, G_k$ satisfy the constraints

$$I(G_i) \leq I_{\max}, 1 \leq i \leq k$$

$$O(G_i) \leq O_{\max}, 1 \leq i \leq k$$

The partitioning problem is also referred as the mapping problem as it maps sub-circuits to logic blocks. Note that this partitioning problem is significantly different than the partitioning problem considered in Chapter 5. In particular, the number of vertices assigned to any subgraph is not important. The important parameter is the number of edges coming in or going out of a subgraph.

In [FRC90], Francis, Rose and Chung presented a dynamic programming algorithm for partitioning the DAG for which the fan-in of each node does not exceed $I_{\max}$. The following terms need to be defined in order to explain the algorithm. A *mapping* of a node v_i, in a tree T, is a circuit of $I_{\max}$-input LUTs which implements the sub-tree of T that is rooted at v_i and extends to the leaf nodes of T. The *cost* of a mapping is the number of LUTs needed to implement that mapping. The *root* lookup table of a mapping of the node v_i has as its single output the boolean function of the node v_i. The *utilization* of a lookup table is the number of inputs U, out of the K inputs that are actually used in a circuit. If $v_{f1}, v_{f2}, \ldots, v_{fp}$ are the fan-in nodes of a node v_i, then the root lookup table of mapping of v_i includes all the fan-in edges of v_i and some sub-tree S_i rooted at each fan-in node v_{fi}. (see Figure 13.9(a)). The term *utilization division* is introduced to denote the distribution of the inputs to the root lookup table among these subtrees. If u_i is the number of leaf nodes in the sub-tree S_i then the set $\mathcal{U} = \{u_1, u_2, \ldots, u_p\}$ specifies the utilization division of the root lookup table. There may be many possible utilization divisions of the root lookup table of a mapping of a node. Figure 13.9 shows the utilization {3,1} for the node v_i, whereas, Figure 13.9(b) shows the utilization {1,3} for the same node v_i.

Let MinMap(v_i, U) be the optimal mapping of node v_i with a root utilization of U. For each leaf node v_i, MinMap(v_i, U) is set to 0 for all values of U. Assuming that MinMap(v_j, U) is computed for each fan-in node $v_j = v_{f1}, v_{f2}, \ldots, v_{fp}$ of an internal node v_i and for all $U = 1$ to $I_{\max}$, MinMap(v_i, U) can be computed for $U = 1$ to $I_{\max}$ as discussed below. In order to compute MinMap(v_i, U), compute MinMap(v_i, U) for each utilization division $\mathcal{U}$ of U by combining v_i with the mappings of fan-in nodes of v_i. MinMap(v_i, U) is simply the minimum cost MinMap(v_i, U) computed over all utilization divisions of U. MinMap(v_i, U) for all $U = 1, \ldots, I_{\max}$ is computed while visiting the node v_i in a post-order traversal of the tree. This ensures the condition that the mappings for all fan-in nodes of v_i are already computed.

In the cases when a node in DAG has a fan-in greater than $I_{\max}$, a node decomposition phase has to be carried out before applying the above algorithm. The output of the node decomposition is a functionally equivalent DAG in which all the nodes have fan-in less than $I_{\max}$. Figure 13.10(b) shows a DAG obtained after decomposition of node v_1 in Figure 13.10(a).

13.4 Routing

After all the sub-circuits have been mapped to logic blocks, these sub-circuits are interconnected by blowing the fuses in the routing channels. Routing of

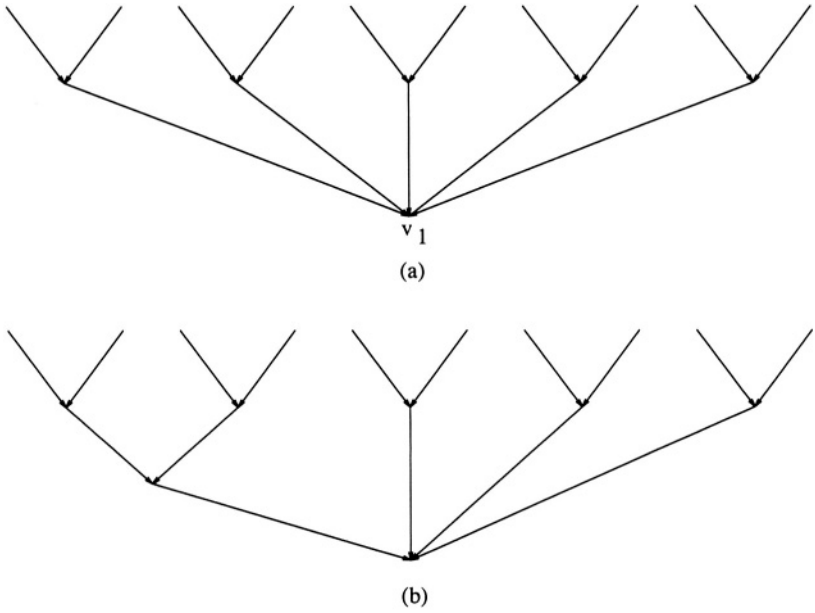

Figure 13.10: Node decomposition.

FPGAs is different from the routing of general blocks because of the segmented nature of channels. In the following sections, we discuss FPGA routing for different models.

13.4.1 Routing Algorithm for the Non-Segmented Model

In this section, we discuss the algorithm presented by Brown, Rose and Vranesic [BRV92]. The routing is completed in two steps.

1. **Global routing:** Global routing in FPGAs can be done by using a global router for standard cell designs. In general, such a global router divides the multi-terminal nets into two terminal nets and routes them with minimum distance path. While doing so it also tries to balance the densities by distributing the connections among the channels. The global route defines a coarse route for each connection by assigning it a sequence of channel segments. Figure 13.11(a) shows a sequence of channel segments that a global route might choose to connect some pin of logic block at grid location 4,1 to another at 0,1. The global route is also called as a course grid graph. Note that the coarse grid graph gives a path between two L nodes through a sequence of S and C nodes.

2. **Detailed routing:** Given a course grid graph $G = (V, E)$ for a two terminal net, the objective of the detailed router is to choose specific

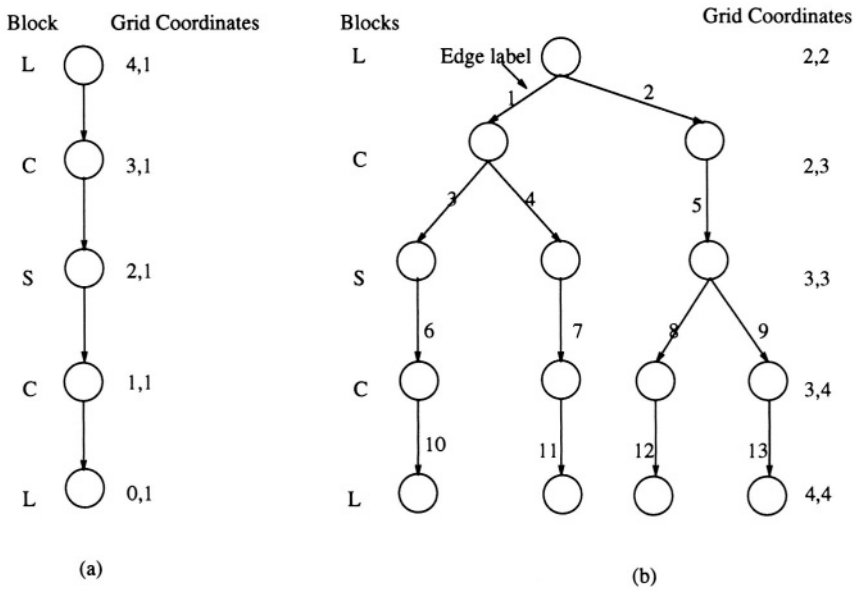

Figure 13.11: (a) Coarse grid graph and (b) Expanded graph.

Algorithm GRAPH-EXPANSION(G)
begin
  $G_D = G$;
  while (DFS-COMPLETE(G_D)==FALSE) **do**
    v_i=CURRENT-DFS-VISIT(G_D);
    l_i=WIRE-SEGMENT(v_i, G_D);
    if (NODE-TYPE(v_i)=C OR NODE-TYPE(v_i)=S) **then**
      v_j=SUCCESSOR(G_D, v_i);
      T_j=SUBTREE(G_D, v_j);
    if (NODE-TYPE(v_i)=C) **then**
      for (each wiring segment l in $F_C(v_i, v_j, l)$) **do**
        T=DUPLICATE(T_j);
        CONNECT(v_i, v_j, T, l);
      DELETE(G_D, T_j);
    if (NODE-TYPE(v_i)=S) **then**
      for (each wiring segment l in $F_S(v_i, v_j, l, v_k)$) **do**
        T=DUPLICATE(T_j);
        CONNECT(v_i, v_j, T, l);
      DELETE(G_D, T_j);
end.

Figure 13.12: Algorithm GRAPH-EXPANSION.

wiring segments in each channel segment assigned during global routing. This is achieved in two steps:

(a) **Expansion of coarse grid graph:** In this step, a coarse grid graph is expanded to record a subset of possible ways of implementing the connection. The expansion is carried out while spanning the graph in depth first search manner. The formal description of algorithm is shown in Figure 13.12. Function DFS-COMPLETE(D) returns TRUE if all the nodes of D are visited during the depth-first-search. Function CURRENT-DFS-VISIT(G_D) returns the node being visited during DFS. Function WIRE-SEGMENT(v_i, G_D) returns a wire segment that connects v_i to its predecessor. Function SUCCESSOR(v_i) returns the successor of v_i in G_D. Function SUBTREE(G_D, v_j) returns the subtree of G_D rooted at v_j. Function NODE-TYPE(v_i) returns 'C' if the node v_i represents a C block, it returns 'S' if v_i represents a S block, it returns an 'L' otherwise. If v_j is a C node, v_k is its successor of v_j and a wire segment l is used to connect v_i to v_j then the function $\mathrm{F}_C(v_i, v_j, l)$ returns a set of wiring segments that can be used to connect v_j to v_k. Similarly, if v_j is an S node, v_k is its successor of v_j and a wire segment l is used to connect v_i to v_j then the function $\mathrm{F}_S(v_i, v_j, l, v_k)$ returns the a set of wiring segments that can be used to connect v_j to v_k. Function DUPLICATE(T) returns a copy of tree T. Procedure CONNECT(v_i, v_j, T, l) connects the node v_i to the node v_j in T by a directed edge from v_j to T and labels the connecting edge by l. Procedure DELETE(G, T) deletes the subtree T from G. Let $G_D = (V_D, E_D)$ denote the graph obtained after expansion of $G = (V, E)$. Figure 13.11(b) the graph obtained by expanding the coarse graph in Figure 13.11 (a).

(b) **Connection formation:** The expanded graph $G_D = (V_D, E_D)$ contains a number of alternative paths. In this step, all these paths are enumerated, their cost is computed and the minimum cost path is selected to implement the connection. Cost of a path is the summation of the cost of edges in that path. The cost of an edge consists of two parts: $c_f(e)$ and $c_t(e)$. $c_f(e)$ accounts for the competition between different nets for the same wiring segments, and $c_t(e)$ reflects the routing delay associated with the routing segment.

13.4.2 Routing Algorithms for the Segmented Model

In this section, we discuss a basic routing algorithm for segmented model. This is followed by a discussion on a new segmented model called *staggered segmented model* and an associated router.

```
Algorithm SEG-ROUTER (I, T, A)
    input: I, T;
    output: A;
begin
    for i = 1 to n do
        for j = 1 to m do
            s=GET-SEGMENT(j, LEFT(I_i));
            if OCCUPIED(s) =FALSE then
                A[i]=j;
                MARK-OCCUPIED(I_i,T_j);
end.
```

Figure 13.13: Algorithm SEG-ROUTER.

13.4.2.1 Basic Algorithm

In this section, we discuss an algorithm presented by Green, Roychowdhury, Kaptanoglu and Gamal for routing in segmented model [GRKG93]. The input to the routing problem is a set of intervals $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$, a set of tracks $\mathcal{T} = \{T_1, T_2, \ldots, T_m\}$. Each track $T_i \in \mathcal{T}$ extend from column 1 to column *N,* and is divided into a set of contiguous segments separated by switches. These switches are placed between two successive segments.

For each interval $I_i \in \mathcal{I}$, we define $\mathrm{LEFT}(I_i)$ and $\mathrm{RIGHT}(I_i)$ to be the leftmost and the rightmost column in which the interval is present. $1 \leq LEFT(I_i) \leq RIGHT(I_i) \leq N$. We assume that the intervals in $\mathcal{I}$ are sorted on their left edges, i.e., $\mathrm{LEFT}(I_i) < \mathrm{LEFT}(I_j)$ for all $i < j$.

If an interval I_i is *assigned* to a track T_j, then the segments in track T_j that are present in the columns spanned by the interval are considered *occupied.* More precisely, a segment s in track T_j is occupied by an interval I_i if $\mathrm{RIGHT}(s) < \mathrm{RIGHT}(I_i)$ and $\mathrm{LEFT}(s) < \mathrm{LEFT}(I_i)$.

A *routing* of $\mathcal{I}$ consists of an assignment of each interval $I_i \in \mathcal{I}$ to a track such that no segment is occupied by more than one connection.

The routing in the segmented model can be achieved using the algorithm SEG-ROUTER presented in Figure 13.13. The algorithm SEG-ROUTER is a modified left-edge algorithm. The input to the algorithm is the set of intervals $\mathcal{I}$, the set of tracks $\mathcal{T}$, whereas the output is an array A, such that *A[i]* gives the number of the track on which the interval I_i is routed. In algorithm SEG-ROUTER, function GET-SEGMENT(j, c) returns a segment s on track T_j such that column c is in the span of s. Function OCCUPIED(s) returns TRUE if the segment s is occupied, it returns FALSE otherwise. Procedure MARK-OCCUPIED(I_i,T_j) marks all the segments on tracks T_j that are occupied by I_i. Figure 13.14(b) shows a routing in a uniform segmented channel, generated by Algorithm SEG-ROUTER. Figure 13.14(a) shows a routing in a non-segmented channel, generated by the left edge algorithm. Note that less number of tracks

are used in uniform segmented channel as compared to the non-segmented channel.

13.4.2.2 Routing Algorithm for Staggered Model

The segmented model can be improved in several ways: Figure 13.14(c) shows that if the antifuses are staggered the routing of the channel in Figure 13.14(a) can be completed in 3 tracks. Figure 13.14(d) shows that if the antifuses are staggered and if different track segments have different lengths then the routing of the channel in Figure 13.14(a) can be completed in 2 tracks. In this section, we discuss a staggered segmentation model and its routing algorithm presented by Burman, Kamalanathan and Sherwani [BKS92].

In this model, a channel is partitioned into several regions. Each region is characterized by the segment length. The tracks in each region have equal length segments separated by staggered placement of antifuse switches. There are three parameters with respect to the new model: number of regions (p), number of tracks (t), length of segment in each region (l). Determination of these three parameters is an important step in this segmentation scheme. These parameters can be determined by a detailed empirical analysis on several standard benchmarks. A detailed analysis and determination of these parameters can be found in [BKS92]. If the length of segments in all the regions is same then the model is called as the *uniform staggered model* otherwise it is called *non-uniform staggered model.* Note that the model in Figure 13.14(c) is uniform staggered model, whereas the one in Figure 13.14(d) is non-uniform staggered model.

Algorithm SEG-ROUTER can be used for routing in the staggered models. In a uniform segmented model the delay of a net is same irrespective of the routing track. Whereas, in the staggered models the delay of a net is dependent on the the routing track as the number of antifuses in the path of a net in different tracks may be different. The algorithm SEG-ROUTER is not suitable for the high performance routing as it does not consider the delay of a net. In the following, we discuss the algorithm FSCR for the high performance routing in staggered model [BKS92]. The key feature of this routing algorithm is the assignments of the nets to the appropriate tracks by delay computation and delay matching techniques. It should be noted that, for minimum delay routing, it is not sufficient to just minimize delay based on the antifuse elements, but also capacitance effects due to the unused portion of the segments spanned by a net segment (also called as hang-over wires) and the unprogrammed switches must be considered [BKS92].

The algorithm starts routing the longest nets first. This ensures that the delay due to the longest net is minimized, which is a prerequisite for the high performance routing systems. For each net, it finds out a track on which the net can be routed with minimum delay. The original algorithm has three phases, *region selection, track selection* and the *region reselection* [BKS92]. In Figure 13.15, we present its simplified version. In algorithm FSCR, the function OK-TO-ASSIGN(I_i, T_j) returns TRUE if all the segments spanned by the

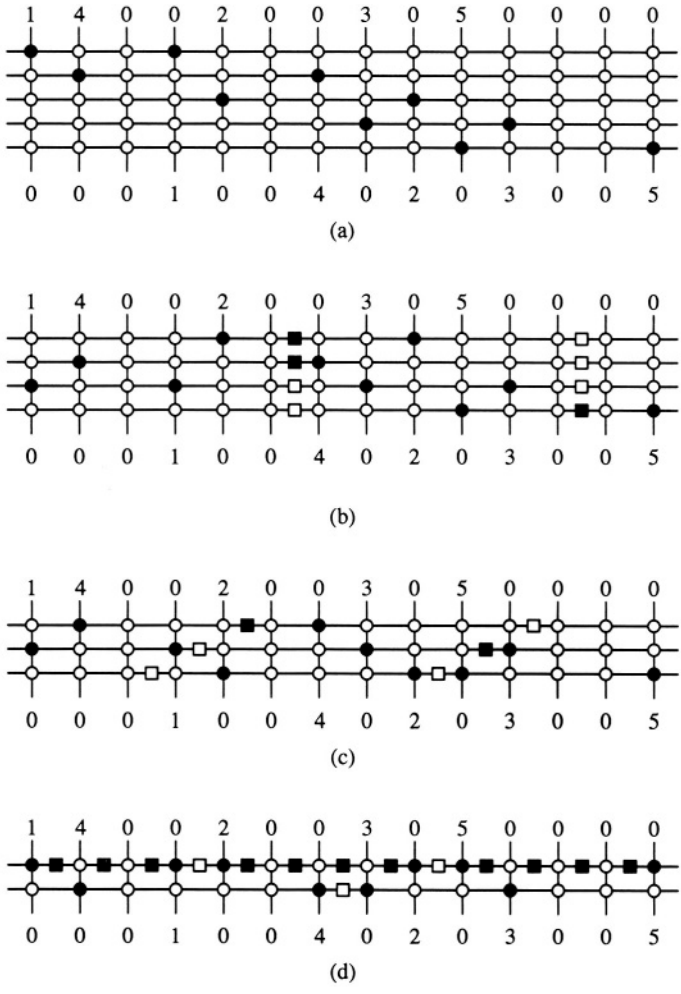

Figure 13.14: Example of (a) Non-segmentation model, (b) uniform segmentation model , (c) uniform staggered segmentation model, and (d) non-uniform staggered segmentation model.

```
Algorithm FSCR(I, T, A)
    input: I, T;
    output: A;
begin
    for i = 1 to n do
        selected-track = 0;
        minimum-delay = ∞;
        for j = 1 to m do
            if (OK-TO-ASSIGN(I_i, T_j) =TRUE) then
                current-delay = COMPUTE-DELAY(I_i, T_j);
                if ( minimum-delay > current-delay ) then
                    minimum-delay = current-delay;
                    selected-track = j;
        if (selected-track ≠ 0) then
            A[i]=j;
            MARK-OCCUPIED(I_i,T_j);
        else exit; (* Routing not possible *)
end.
```

Figure 13.15: Algorithm FSCR

interval I_i on track T_j are unoccupied. Function COMPUTE-DELAY(I_i, T_j) computes the delay in the interval I_i if I_i is routed on the track T_j. Function MARK-OCCUPIED(I_i,T_j) is same as that used in algorithm SEG-ROUTER.

13.5 Summary

FPGAs are being used as a new approach to ASIC design which offers dramatic reduction in manufacturing turnaround time and cost. The physical design cycle of an FPGA consists of three steps, partitioning, placement and routing. The FPGA partitioning problem is different from the conventional area partitioning problem in the sense that it depends on the architecture in which the circuit has to be implemented. Placement problem is equivalent to the general gate array placement problem. However, because of the segmented nature of the FPGA channels, the routing considerations are quite different. In high performance FPGA designs, the number of antifuse elements along with unused tracks and antifuses must be given due considerations as part of the routing phase. A significant amount of research in the direction of physical design automation has to be done in order to fully utilize the potential of FPGAs.

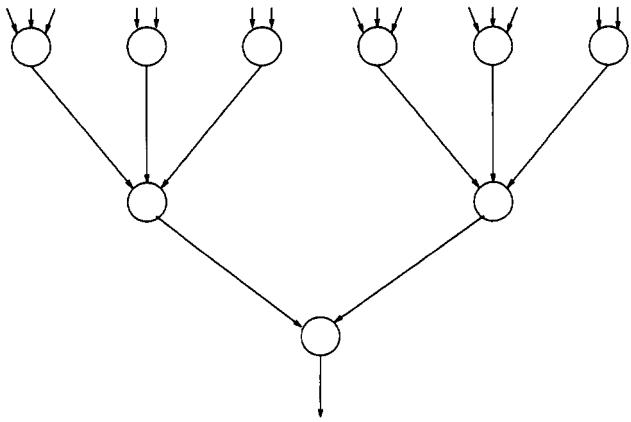

Figure 13.16:

13.6 Exercises

1. Given the graph in Figure 13.16, find the minimum number of configurable logic blocks with five inputs and one output ($O_{\max} = 1, I_{\max} = 5$), required to partition the circuit.

2. Given a k-array tree of height h, find minimum number of CLBs required with $I_{\max}$ inputs and $O_{\max}$ outputs.

3. Develop a bin-packing algorithm for partitioning a set of functional blocks into minimum number of CLBs.

4. Given a set of tracks, number of antifuse elements for the new segmentation model, formulate the mixed integer program to minimize the number of antifuse elements. While formulating the problem, consider the delay caused by the antifuse elements and the hang-over wires.

† 5. Several factors play a key role in improving the utilization of *channel resources* in an FPGA. Three such factors have been discussed in this chapter. Discuss what other factors may be considered important for designing a channel segmentation model for high performance applications.

‡ 6. Suggest an efficient channel segmentation model for a three layer routing in FPGAs.

7. Develop a channel routing algorithm for three layer routing model in FPGA.

8. Consider the function $f = ABC + \bar{A}\bar{B}\bar{C} + \bar{A}B + A\bar{B} + \bar{A}C + A\bar{B}C$. Partition the circuit corresponding to f such that

(a) It can be mapped to a minimum number of CLBs. Show the mapping.

(b) It can be mapped to a minimum number of logic modules (each module has three 2-to-l MUX and a OR gate). Show the mapping.

† 9. In a CLB all 2^5 combinations have to be stored. Suggest a method which stores only those entries which generate either a 0 or a 1 output, whichever is greater without loss of functionality ?

10. Can the size and number of MUX inside a logic block be increased arbitrarily ? What can be the maximum size of the MUX inside the logic block ?

Bibliographic Notes

In *mis-pga* (old) from Murgai, Nishizaki, Shenoy, Brayton, Vincentelli, decomposition is performed using Roth-Karp method and kernel extraction. The reduction method used in this algorithm is computationally expensive. Hill [Hil91] presented a CAD system for the design of Field Programmable Gate Arrays. New FPGA architecture was developed and the use of FPGAs from the user point of view. Another technology mapper, *Hydra* has been described by Filo, Yang, Mailhot and Micheli [FYMM91]. The approach is similar to mis-pga and performs a disjoint decomposition followed by a node minimization phase. The main difference is that both of these phases are driven by the fact that the Xilinx CLB may realize two outputs also. *Xmap and Amap,* developed by Karplus [Kar91a, Kar91b] constructs an if-then-else dag as decomposition of the function and uses a covering procedure to map it to CLBs. In *mis-pga* (new) by Murgai, Shenoy, Brayton and Sagiovanni-Vincentelli [MSBSV91a], a combinatorial circuit has been described in terms of Boolean equations to realize it using the minimum number of basic blocks of the target Table Lookup architecture. Murgai, Shenoy, Brayton and Sagiovanni-Vincentelli [MSBSV91b] presented delay optimization for programmable gate arrays. The main considerations in this paper are the number of levels in the circuit and the wiring delay. A two phase approach was given. The first phase involves delay optimization during logic synthesis before placement, and the second uses logic resynthesis during a timing-driven placement technique. Nam-Sung Woo [Woo91] presented a heuristic method for the reduction and packing. This is based on the notion of edge visibility and use of global information. The packing method is based on the degree of the node common input. Ercolani and Micheli [EM91] presented a technology mapper for electrically programmable gate arrays. This is based on matching algorithm that determines whether a portion of a combinational logic circuit can be implemented by personalizing a module. The benefits include, an increased efficiency in technology mapping, as well as portability to different types of electrically programmable gate arrays. In [CD93] Cong, and Ding present a study of the area and depth trade-off in LUT based FPGA technology mapping to obtain an area minimized mapping solution. In [FW97] a new integrated synthesis and partitioning method for multiple-FPGA applications is presented. In [CHL97] Technology mapping algorithms for minimizing power

consumption in FPGA design are studied. In [SRB97] Macro Block Based FPGA floorplanning has been discussed. [CL97] presents a new recursive bi-partitiong algorithm targetted for a hierarchical field programmable system. In [LW97], a new performance and Routability-Driven Router for symmetrical array based FPGA's is presented. A variation of gate array called LPGA (Laser Programmable Gate Array) is a high performance gate array fabricated by laser micro-machining system allows development of one-day laser prototypes and two months high volume production. The base wafers are fabricated with all interconnection metal layers and a proprietary technique is used to selectively remove specific metalization points to personalize the arrays. Disconnecting the excess metal links follows an automated cut-list program, generated per specific design.