Chapter 4
Heterogeneous Architectures Exploration Environments

During past few years, the advancement in process technology has resulted in a great increase in the capacity of FPGAs; the devices which were once small have now become large and are used to implement complete designs. Increase in the capacity of FPGAs has allowed their transition from devices that once contained only homogeneous blocks to the devices that now contain a mixture of blocks ranging from soft blocks (e.g. Configurable Logic Blocks) to hard-blocks like multipliers, adders, RAMs etc. The use of hard-blocks in FPGAs has resulted in an improved overall efficiency and now they are used for large and complex applications.

This chapter presents a new exploration environment for tree-based heterogeneous FPGA architecture. This environment is based on the environment discussed in previous chapter. The environment of the previous chapter is modified so that an architecture description mechanism allows to define various architectural parameters including definition of new heterogeneous blocks, the level where they are located and their arity (i.e. number of blocks per cluster). Once the architecture is defined, a software flow partitions and routes the target netlist on the architecture. The partitioning and routing tools are modified to incorporate a mixture of heterogeneous blocks in the architecture. A mesh-based heterogeneous exploration environment, initially presented in [92], is also explored and enhanced in this chapter. This environment is an extended version of homogeneous exploration environment of mesh-based architecture presented in previous chapter. Different floor-planning techniques are explored for mesh-based architecture using different sets of benchmarks and results of those benchmarks are compared with results of tree-based architecture.

4.1 Introduction and Previous Work

During recent past, embedded hard-blocks (HBs) in FPGAs (i.e. heterogenous FPGAs) have become increasingly popular due to their ability to implement complex applications more efficiently as compared to homogeneous FPGAs. The work

U. Farooq et al., *Tree-Based Heterogeneous FPGA Architectures*,
DOI: 10.1007/978-1-4614-3594-5_4,

in [123] shows that the use of embedded memory in FPGA improves its density and performance. Authors in [19] have incorporated floating point multiply-add units in the FPGA and have reported significant area and speed improvements over homogeneous FPGAs. In [58] a virtual embedded block (VEB) methodology is proposed that predicts the effects of embedded blocks in commercial FPGA devices; and it has shown that the use of embedded blocks causes an improvement in area and speed efficiencies. Also authors in [52] and [118], suggest the use of embedded blocks in FPGAs for better performance regarding complex scientific applications. The work in [72] shows that the use of HBs in FPGAs reduces the gap between ASIC and FPGA in terms of area, speed and power consumption. Some of the commercial FPGA vendors like Xilinx [126] and Altera [13] are also using HBs (e.g. multipliers, memories and DSP blocks) in their architectures.

Almost all the work cited above considers mesh-based FPGAs as the reference architecture where HBs are placed in fixed columns; these columns of HBs are interspersed evenly among columns of configurable logic blocks (CLBs). The main advantage of mesh-based, fixed-column heterogeneous FPGA lies in its simple and compact layout generation. However, the column-based floor-planning of FPGA architectures limits each column to support only one type of HB. Due to this limitation, the architecture is bound to have at least one separate column for each type of HB even if the application or a group of applications that is being mapped on it uses only one block of that particular type. This can eventually result in the loss of precious logic and routing resources. This loss can become even more severe with the increase in number of types of blocks that are required to be supported by the architecture.

Although, significant amount of research has already been done regarding mesh-based heterogeneous FPGA architectures; no work has been done yet in this domain for tree-based heterogeneous FPGA architectures. Contrary to mesh-based architectures where logic and routing resources are arranged in an island style, tree-based architecture is a hierarchical architecture where logic and routing resources are arranged in a multilevel clustered structure. So, in this chapter we present a new exploration environment for tree-based heterogeneous FPGA architectures. Different techniques are explored to optimize the use of logic and routing resources of the architecture. Further, in this chapter, an exploration environment for mesh-based heterogeneous FPGA architecture is described [92]. The environment for mesh-based architecture is jointly developed by authors of this book and a previous student of our research team [94]. Contrary to existing environments of mesh-based architecture that use a fixed floor-planning technique, this environment automatically optimizes the floor-planning of hard-blocks. Also, unlike previous research [72, 123] that mainly compares mesh-based heterogeneous FPGA architectures with their homogeneous counterparts, this chapter presents a detailed comparison between different architectural techniques of heterogeneous mesh-based and tree-based architectures.

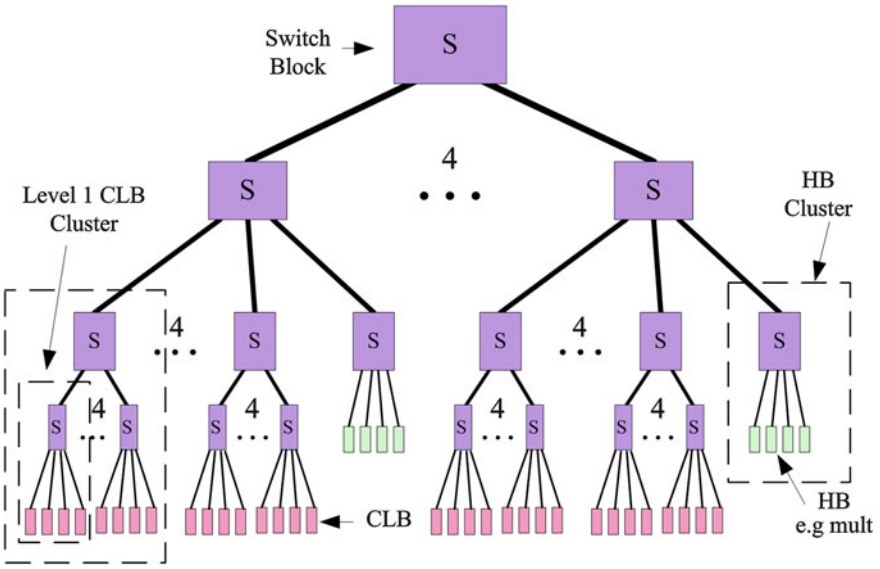

Fig. 4.1 Generalized tree-based heterogeneous FPGA architecture

4.2 Reference Heterogeneous FPGA Architectures

This section gives basic overview of the two heterogeneous FPGA architectures. These FPGA architectures are based on the architectures that are described in Chap. 3 and modifications are made in them so they can support a mixture of heterogenous blocks.

4.2.1 Heterogeneous Tree-Based FPGA Architecture

A tree-based heterogeneous architecture [45] is a hierarchical architecture having unidirectional interconnect. In tree-based heterogeneous architecture CLBs, I/Os and HBs are partitioned into a multilevel clustered structure where each cluster contains sub clusters and switch blocks allow to connect external signals to sub-clusters. Figure 4.1 shows generalized example of a four-level, arity-4, tree-based architecture. The arity of the architecture is basically defined as the number of CLBs in the base-cluster of the architecture and it is respected as we move towards the top of the hierarchy. However, in a heterogeneous architecture we may have a mixed arity. For example in Fig. 4.1, first two levels have arity 4, third level has arity 5 and fourth level has arity 4. But in order to keep the convention, we refer it as arity 4 architecture because its base-cluster contains four CLBs. In a heterogenous tree-based architecture, CLBs are placed at the bottom of hierarchy whereas HBs can be

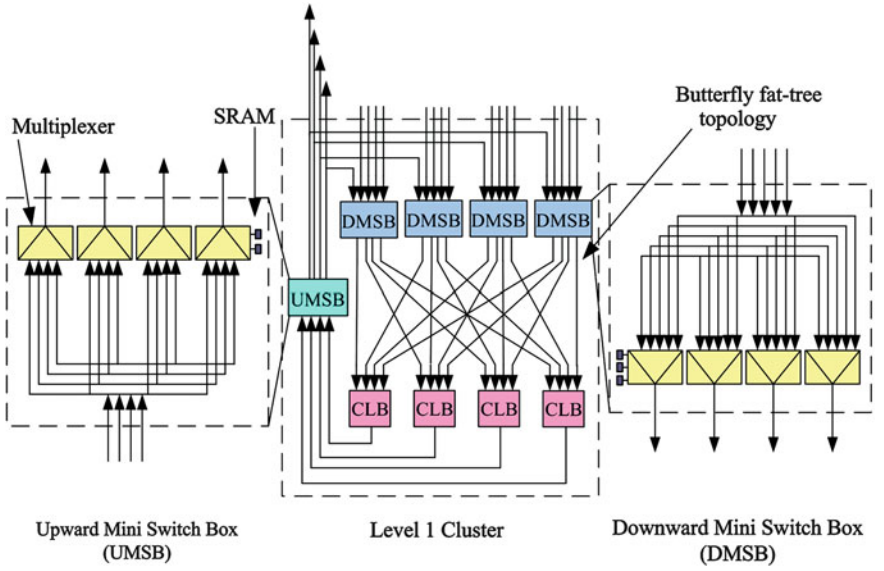

Fig. 4.2 Detailed interconnect of base-cluster of tree-based architecture

placed at any level of hierarchy to meet the best design fit. For example, in Fig. 4.1 HBs are placed at level 2 of hierarchy.

4.2.1.1 The Interconnect Network

Similar to homogeneous architecture, tree-based heterogeneous architecture contains two unidirectional, single length, interconnect networks: a downward network and an upward network. As we move towards the top, signal bandwidth grows in both networks and it is maximum at the top of hierarchy. Downward network is based on butterfly fat tree topology and allows to connect signals coming from other clusters to its sub-clusters through a switch block. The upward network is based on hierarchy and it allows to connect sub-cluster outputs to other sub-clusters in the same cluster and to clusters in other levels of hierarchy. A detailed base-cluster example of two interconnect networks is shown in Fig. 4.2. In this figure, base-cluster contains four CLBs where each CLB contains one LUT with 4 inputs and one output. It can be seen from the figure that switch blocks are further divided into downward and upward mini switch boxes (DMSBs and UMSBs). These DMSBs and UMSBs are unidirectional full cross bar switches that connect signals coming into the cluster to its sub-clusters and signals going out of a cluster to other clusters of hierarchy.

Because of the homogeneity at all the levels, in a tree-based homogeneous architecture, the number of DMSBs in a switch block of a cluster at level ℓ are equal to number of inputs of a cluster at level $\ell - 1$. Similarly, number of UMSBs in a

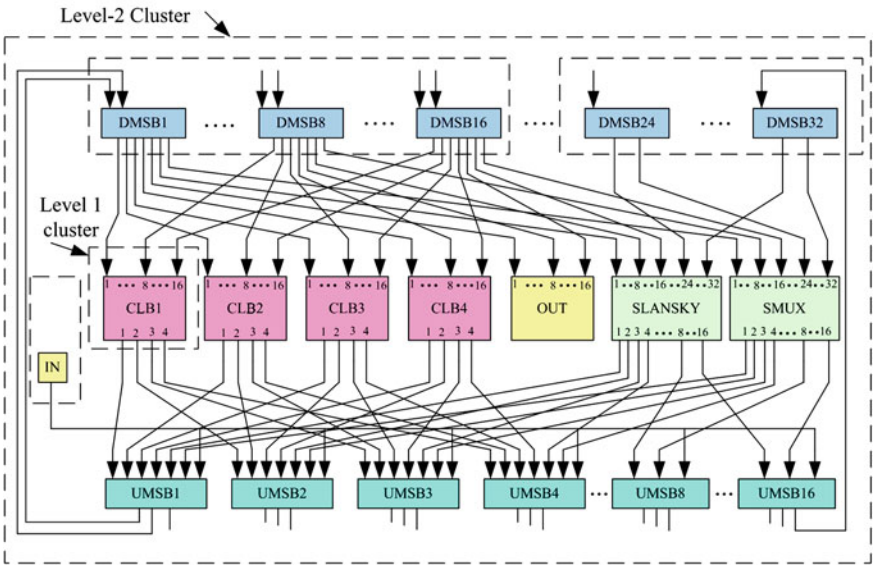

Fig. 4.3 A two level tree-based heterogeneous FPGA architecture

cluster at level ℓ are equal to number of outputs of a cluster at level $\ell - 1$. However, this rule is slightly changed in case of tree-based heterogeneous architectures. In a tree-based heterogeneous architecture, since, there can be clusters at a level with different number of inputs and outputs. So, in this case the number of DMSBs in a cluster at level ℓ are equal to the the highest number of inputs of a cluster at level $\ell - 1$ and number of UMSBs in a cluster at level ℓ are equal to the highest number of outputs of a cluster at level $\ell - 1$. In this way all the inputs of a cluster with maximum inputs can be reached by DMSBs of of upper level cluster and all the outputs of the cluster with maximum outputs can be connected to UMSBs of upper level cluster. By applying this rule we ensure high flexibility and hence improved routability. Also by using this principle we remain inline with the connection patterns of homogeneous architecture explained in previous chapter. This rule can be further explained with the help of example shown in Fig. 4.3. In this figure a two level tree-based heterogeneous architecture is shown. This architecture contains four CLB clusters (detail shown in Fig. 4.2) and two hard-block clusters. We can see that SLANSKY and SMUX (two hard-blocks used by one of the benchmarks) clusters have highest number of inputs among all the clusters of level 1 which is 32. So the number of DMSBs of a cluster at level 2 are equal to 32 and same rule is applied to determine the number of UMSBs at that level. Once the number of DMSBs and UMSBs of a cluster are determined, the inputs per DMSB and outputs per UMSB are determined and connected to the inputs and outputs of lower and upper level clusters in a similar manner as that of tree-based homogeneous architecture.

4.2.1.2 Interconnect Depopulation

Generally, in a tree-based architecture, the interconnect bandwidth grows from bottom to top. However, the number of signals entering into and leaving from the cluster situated at a particular level of a tree-based architecture can be varied depending upon the netlist requirement. The signal bandwidth of clusters is controlled using Rent's rule [74] which is easily adapted to tree-based heterogeneous architecture. This rule states that

$$IO = \left(\underbrace{k.n^{\ell}}_{L.B(p)} + \underbrace{\sum_{x=1}^{z} a_x.b_x.n^{(\ell-\ell_x)}}_{H.B(p)} \right)^{p} \tag{4.1}$$

where

$$H.B(p) = \left\{ \begin{array}{ll} 0 & if(\ell - \ell_x < 0) \\ a_x.b_x.n^{(\ell-\ell_x)} & if(\ell - \ell_x \geq 0) \end{array} \right\} \tag{4.2}$$

In Eq. 4.1 ℓ is a tree level, n is the arity size, k is the number of in/out pins of a LUT, a_x is the number of in/out pins of a HB of type x, ℓ_x is the level where HB is located, b_x is the number of HBs at the level where it is located, z is the number of types of HBs supported by the architecture and IO is the number of in/out pins of a cluster at level ℓ. Since there can be more than one type of HBs, their contribution is accumulated and then added to the $L.B(p)$ part of Eq. 4.1 to calculate p. The value of p is a factor that determines the cluster bandwidth at each level of the tree-based architecture and it is averaged across all the levels to determine the p for the architecture. Normally the value of p is either ≤ 1. However, in particular cases, the initial value of p can be >1 (for details please refer to Sect. 5.6).

4.2.2 Heterogeneous Mesh-Based FPGA Architecture

A mesh-based heterogeneous FPGA is represented as a grid of equally sized slots which is termed as slot-grid. Blocks of different sizes can be mapped on the slot-grid. A block can be either a soft-block like a configurable logic block (CLB) or a hard-block like multiplier, adder, RAM etc. Each block (CLB or a HB) occupies one or more slots depending upon its size. The architecture used in this work is a VPR-style (Versatile Place and Route) [81] architecture that contains CLBs, I/Os and HBs that are arranged on a two dimensional grid. In order to incorporate HBs in a mesh-based FPGA, the size of HBs is quantized with size of the smallest block of the architecture i.e. CLB. The width and height of an HB is therefore a multiple of the width and height of the smallest block in the architecture. An example of such FPGA where CLBs and HBs are mapped on a grid of size 8×8 is shown in Fig. 4.4.

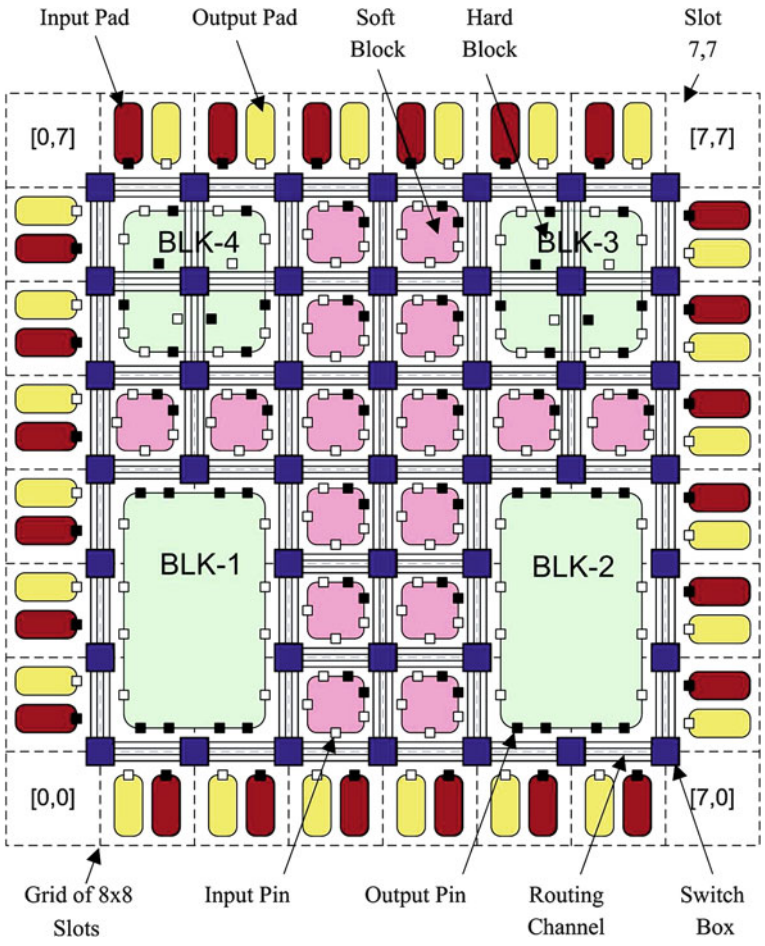

Fig. 4.4 Generalized mesh-based heterogeneous FPGA architecture [94]

In mesh-based FPGA, input and output pads are arranged at the periphery of the slot-grid as shown in Fig. 4.4. The position of different blocks in the architecture depends on the used floor-planning technique. A block (referred as CLB or HB) is surrounded by a uniform length, single driver, unidirectional routing network [77]. The input and output pins of a block connect with the neighboring routing channel. In the case where HBs span multiple tiles, horizontal and vertical routing channels are allowed to cross them [19]. In case of mesh-based heterogeneous FPGA, the detailed interconnect of a CLB with surrounding routing network remains same as homogeneous FPGA (already detailed in Fig. 3.2). For HBs, their input and output pins are connected to the surrounding network in a similar manner as CLBs except that CLBs occupy only one slot and HBs may occupy multiple slots (refer to Fig. 4.4).

In order to explore the two architectures, architecture exploration environments are developed for both of them. Architecture exploration environment of both mesh-based and tree-based architectures starts with respective architecture description. Once the architecture is defined, netlists are placed and routed on both architectures using a software flow. Different steps that are involved in the exploration are detailed in following sections.

4.3 Architecture Description

4.3.1 Architecture Description of Heterogeneous Tree-Based Architecture

Different architecture parameters of the tree-based heterogeneous FPGA architecture are defined using an architecture description file. Some of these parameters are shown in Table 4.1. The parameter Nb_Levels defines the total number of levels of the architecture. Nb_Block_Types parameter defines the total number of types that are supported by the architecture. By default, a tree-based architecture supports two types of blocks which are logic blocks (CLBs or soft blocks) and I/Os. For heterogeneous architectures, however, the types of blocks may vary depending on the netlist requirements that are being implemented on the architecture. The architecture is quite flexible in this sense and it can support any number of block types that can be placed at different levels of hierarchy in order to have a best design fit. In our description mechanism, the architecture description starts with the specification of I/O blocks and once it is done, the rest of the architecture is defined repeatedly using the parameters of lines 5–10 of Table 4.1. These parameters include level number being defined, number of sub-cluster types supported by each cluster, number of sub-clusters contained in each cluster and number of inputs/outputs of the cluster. Definition of clusters starts from bottom level of the architecture and it goes to top until all the levels of the architecture are specified. Once cluster definition is over, binary search parameter is either set to be true or false. If binary search parameter is false, no architecture optimization is performed and the netlist is routed using the given cluster bandwidth. However if this parameter is true, then an architecture optimization is performed using the specified optimization approach which can be bottom_up, top_down or random. Details regarding these optimization approaches are already given in Sect. 3.5.1.

Once cluster definition of all the levels is over, different types of blocks that are supported by the architecture can be defined using Define_Block parameter. Different parameters that are used for the definition of a block are shown in Table 4.2. In a tree-based architecture, definition of a block starts with the name of the block. The parameter "Area" gives the area of the block which is later used for the area calculation of the architecture. Other parameters include the number of input/output pins, the level where the block is located, the arity (i.e. number of blocks per cluster) of the

Table 4.1 Architecture description file parameters of tree-based architecture

	Name	Description
1.	Nb_Levels	Total number of levels in the architecture
2.	Nb_Block_Types	Number of block types that are supported by the architecture
3.	In_Blocks	The level of the cluster and the number of inputs per cluster of the input block
4.	Out_Blocks	The level of the cluster and the number of outputs per cluster of the output block
5.	Level	The level ℓ of the architecture
6.	Nb_Cluster_Type	Number of sub-cluster types supported by a cluster of level ℓ
7.	Arity	Number of sub-clusters of each type supported by a cluster of level ℓ
8.	Nb_Inputs_Per_Cluster	Number of inputs per cluster of each type
9.	Nb_Outputs_Per_Cluster	Number of outputs per cluster type
10.	End_Level	Completes the definition of level ℓ
11.	Optimization	Binary search flag set either true or false
12.	Optimization_approach	Specified as either bottom_up, top_down or random
13.	Define_Block blk	Block definition (See Table 4.2)

Table 4.2 Block definition in tree-based architecture

Definition	Description
Define_Block	
Block_Name	Name of the block
Area	Area of the block
Nb_Inputs	Number of inputs of the block
Nb_Outputs	Number of outputs of the block
Level_Number	Level number where the block is located
Arity	Number of blocks per cluster
Pin_Input	Name and the class number of input pins of the block
Pin_Output	Name and the class number of output pins of the block
End_Define_Block	

block and the definition of its input and output pins. While defining I/O pins of a particular block (logic-block or a hard-block), unique class number are assigned to each block pin to ensure the appropriate routing of the netlist that is mapped on the architecture. An example of the architecture description file that we use to construct the architecture is shown in Fig. 4.5.

Fig. 4.5 An example of architecture description file for tree-based FPGA architecture

```
New_device

Nb_levels:                                  2
Nb_block_types:                             2

In_iobs      level      1      number      4
Out_iobs     level      1      number      4

End_device

Architecture_description:

.level                                      0

        Arity:                              4
        nb_cluster_type:                    1

        new_cluster_type:
                nb_inputs_per_cluster:      16
                nb_outputs_per_cluster:     4
        end_cluster_type:

End_level:                                  0

.level                                      1

        Arity:                              4
        nb_cluster_type:                    1

        new_cluster_type:
                nb_inputs_per_cluster:      0
                nb_outputs_per_cluster:     0
        end_cluster_type:

End_level:                                  1

End_architecture_description

Optimization:                               true
Optimization_algo:                          bottom_up
T-driven:                                   false

Define_block

Block_name:                                 CLB
Area:                                       58500
Nb_inputs:                                  4
Nb_outputs:                                 1
Level_number:                               0
Arity:                                      4
Pin_input:                          i0      0
Pin_input:                          i1      0
Pin_input:                          i2      0
Pin_input:                          i3      0
Pin_output:                         q0      1

End_define_block
```

Table 4.3 Architecture description file parameters of mesh-based architecture

Name	Description
1. Nx num	Slots in the slot-grid in X direction (num >1)
2. Ny num	Slots in the slot-grid in Y direction (num >1)
3. Input_Rate	Number of input pads in each slot on the periphery of slot-grid
4. Output_Rate	Number of output pads in each slot on the periphery of slot-grid
5. Channel_Type T	T is unidirectional or bidirectional
6. Binary_Search F	Binary search flag (F is true or false)
7. Channel_Width num	Channel width if Binary_Search = false (num > 1)
8. Channel_Width_Min num	Minimum channel width if Binary_Search = true (num > 1)
9. Channel_Width_Max num	Maximum channel width if Binary_Search = true (num > 1)
10. Set_Block blk X Y	Place a block named blk at a slot position (X,Y) of slot-grid
11. Set_Block_Auto blk N	Place N instances of blk on first available position of slot-grid
12. Fix_Block_Positions F	The Blocks are movable or fixed (F is true or false)
13. Block_Jump F	If Blocks are moveable, blocks can be moved (F is true or false)
14. Block_Rotate F	If Blocks are moveable, blocks can be rotated (F is true or false)
15. Column_Move W s	If Blocks are moveable, a column can be moved (W is width of the column, s is the starting horizontal slot position of column)
16. Define_Block blk	Block definition (See Table 4.4)

4.3.2 Architecture Description of Heterogeneous Mesh-Based Architecture

Architecture description file of mesh-based FPGA architecture comprises of a number of parameters that are used to construct the architecture. Few major architecture description parameters are shown in Table 4.3. The parameters Nx and Ny define the size of the slot-grid. Channel_Type is used to select a unidirectional mesh [77] or a bidirectional mesh [120] routing network. The channel width of the routing network is either set to a constant value (using the parameter Channel_Width), or a binary search algorithm searches a minimum possible channel width between minimum (Channel_Width_Min) and maximum (Channel_Width_Max) channel width limits. In case of unidirectional mesh, the channel width remains in multiples of 2. The position of blocks can be set to an absolute position on the slot-grid (by using the parameter Set_Block). This parameter takes the name of the block and the position on the slot-grid where it should be placed. Another option to place blocks on the slot-grid is by using the parameter Set_Block_Auto. This parameter automatically places N copies of a block on the first available position on the slot-grid. The blocks on the slot-grid can be either fixed to an initial position or set as moveable (by using the parameter Fix_Block_Positions). In case the blocks are moveable, the placer can refine their position on the slot-grid. The parameter Block_Jump allows the placer to move blocks on the slot-grid. The parameter Block_Rotate allows the placer to rotate blocks at their own axis. The parameter Column_Move allows to move a complete column from one position to another. Column_Move parameter requires the width

Table 4.4 Block definition in mesh-based architecture

Definition	Description
Define_Block	
X_Slots	Number of slots occupied by the block in horizontal direction
Y_Slots	Number of slots occupied by the block in vertical direction
Rotate	A flag set true or false to allow or restrict the rotation of block
Area	Area of the block
Pin_Input	Name, position, class and the direction of input pin of block
Pin_Output	Name, position, class and the direction of output pin of block
End_Define_Block	

of column, W (i.e number of slots as column width), and the starting horizontal slot position of the column. All the blocks in a column must be within the boundaries of the column. This parameter can be repeated if multiple columns are required to be moved.

A new block can be defined in the architecture description file using the Define_Block parameter. The block definition parameters are shown in Table 4.4. Each block is given a name, a size (number of slots occupied), a rotation flag and input/output pins. The rotation flag allows the rotation of individual block by the placer (significance of rotation of a block is explained in Sect. 4.4.3). This rotation flag permits to turn off the rotation of a particular type of block when the global rotation is turned on. Each pin of the block is given a name, a class number, a direction and a slot position on the block to which this pin is connected. An example of the architecture description file that is used to construct a mesh-based FPGA architecture is shown in Fig. 4.6.

4.4 Software Flow

Once the FPGA architectures are defined using their respective architecture description mechanisms, different netlists (benchmarks) are placed and routed on them using a software flow. The software flow used for the exploration of two architectures is shown in Fig. 4.7. The software flow is mainly divided into two parts: first part deals with synthesis and conversion of netlist to .net format while remaining flow deals with the architecture exploration. It can be seen from the figure that netlist synthesis involves a number of steps before it can be placed and routed on the FPGA architecture. These steps are common for both mesh-based and tree-based architectures and they convert the netlist from .vst format to .net format. The netlist in .vst format is obtained using VASY [62] that converts VHDL file to structured VHDL (.vst). Normally a netlist in VST format is composed of traditional standard cell library instances and hard-block instances. The VST2BLIF tool converts the VST file to BLIF format. Later, PARSER-1 removes all the instances of hard-blocks and passes the remaining netlist to SIS [102] for synthesis into 4 input Look-Up Table format.

```
Nx:                                     4
Ny:                                     4

In_rate:                                1
Out_rate:                               1

Interconnect:                           uni_mesh
Binary_search:                          true
Network_width:                          6
Binary_search_min:                      0
Binary_search_max:                      40

Set_automatic                 CLB       16

Block_jump:                             true
Block_rotate:                           true
Column_move:                            false

Define_block

Block_name:                             CLB
X_slots                                 1
Y_slots                                 1
Rotate                                  true
Area:                                   58500
Pin_input:          i0    0     0 0     PIN_LEFT
Pin_input:          i1    0     0 0     PIN_RIGHT
Pin_input:          i2    0     0 0     PIN_TOP
Pin_input:          i3    0     0 0     PIN_BOTTM
Pin_output:         q0    1     0 0     PIN_TOP PIN_RIGHT

End_define_block
```

Fig. 4.6 An example of architecture description file for mesh-based FPGA architecture

All the dependence between hard-blocks and remaining netlist is preserved by adding new input and output pins to the main netlist. SIS generates a network of LUTs and Flip-Flops, which are later packed into CLBs through T-VPACK [120]. T-VPACK generates a netlist in NET format and then PARSER-2 adds all the removed hard-blocks into this netlist. It also removes all the inputs and outputs temporarily added by PARSER-1. This final netlist in NET format, containing CLBs and hard-blocks, is then placed and routed separately on mesh-based and tree-based architectures. In this flow SIS is used for synthesis which we want to replace with ABC [21] in future. Few of the major components of the software flow are detailed below.

4.4.1 Parsers

The output generated by VST2BLIF tool is a BLIF file containing input and output port instances, gates belonging to a standard cell library, and hard-block instances (which are represented as sub circuits in BLIF format). This BLIF file is passed to

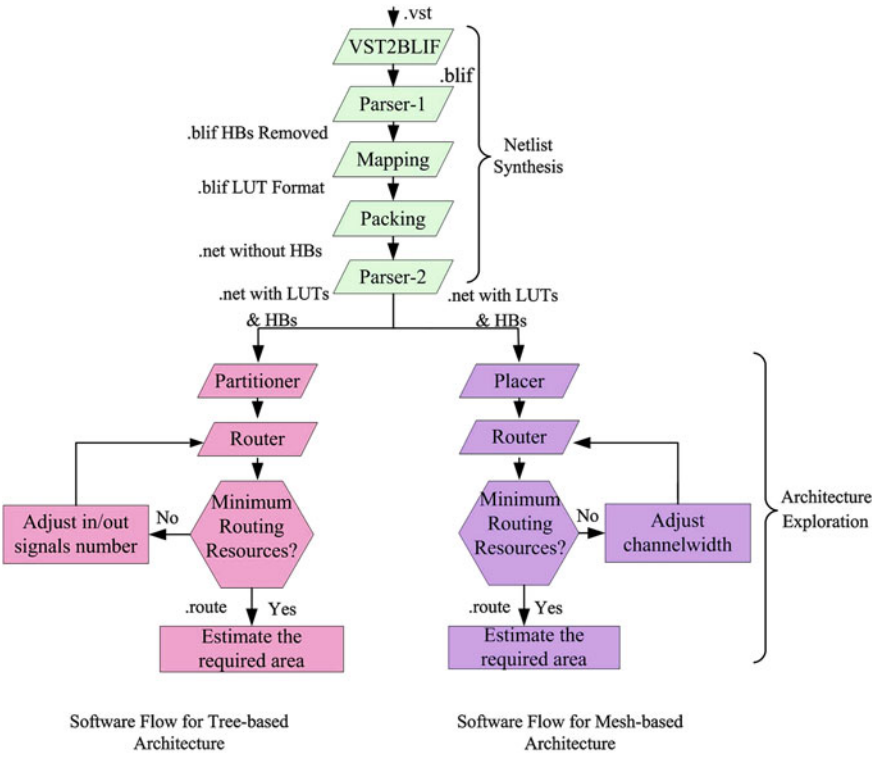

Fig. 4.7 Software flow

SIS [102] for synthesis into LUT format. However, the hard-blocks in the BLIF file are not required to be synthesized. So, the main aim of PARSER-1 is to remove hard-blocks from BLIF file in such a way that all the dependence between the hard-blocks and the remaining netlist is preserved. After synthesis and packing, PARSER-2 will add all the removed hard-blocks in the netlist.

Figure 4.8 shows five different modifications performed by PARSER-1 before removing hard-block instances from the BLIF file. These cases are described as below:

1. Figure 4.8a shows a hard-block whose output pin is connected to the input pin of gate. The output pin of hard-block is detached from the input pin of gate. The detached signal is added as the input pin of main circuit, as shown in Fig. 4.8b.
2. All the output pins of main circuit that are connected by the output pins of hard-block (as shown in Fig. 4.8c) are connected to zero gates (as shown in Fig. 4.8d). This is because, when hard-block is removed, these main circuit outputs do not remain stranded.

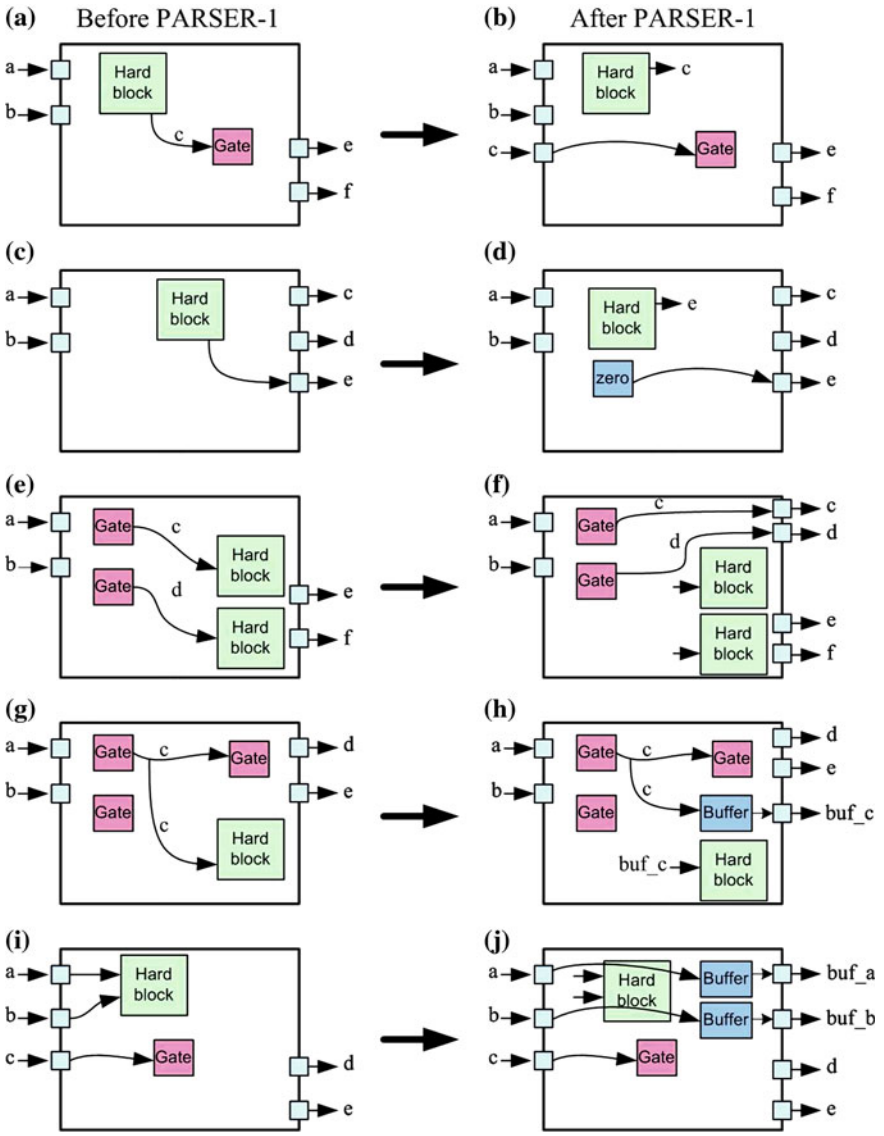

Fig. 4.8 Netlist modifications done by PARSER-1 [94]

3. All the output pins of gates connected only to the input pins of hard-blocks (shown in Fig. 4.8e) are added as the output pins of main circuit (as shown in Fig. 4.8f).
4. For all the output pins of gates connected to the input pins of hard-blocks and also to the input pins of gates (as shown in Fig. 4.8g), add a buffer to this gate

output. The buffered output is added as the output of main circuit. The name of the buffered output should be replaced in all the input pins of hard-blocks (as shown in Fig. 4.8h).

5. Figure 4.8i shows the input pins of main circuit that are connected only to input pins of hard-blocks. After the removal of hard-blocks these inputs will remain stranded and will eventually be removed by SIS. To avoid their removal, these input pins are retained by adding buffers to them, and adding the buffered outputs to the main circuit outputs.

After performing the above changes, PARSER-1 removes all the hard-blocks from the BLIF file. The BLIF file without hard-blocks is passed to SIS, which converts them to LUTs and Flip-Flops. T-VPACK packs LUTs and Flip-Flops together into CLBs. Next, PARSER-2 performs the following changes

1. Adds hard-blocks in the netlist file which is generated by T-VPACK.
2. Removes the "Main circuit input" and "Main circuit outputs" added by PARSER-1.
3. Removes all the zero gates (represented as a CLB after SIS) added by PARSER-1.

The final output file contains I/O instances, CLB instances and hard-block instances. This file is then separately placed and routed on mesh-based and tree-based architectures.

4.4.2 Software Flow for Heterogeneous Tree-Based Architecture

4.4.2.1 Partitioning

Once the netlist is obtained in .net format, it is partitioned using a partitioner. The partitioner is based on the one described in previous chapter. Partitioner partitions CLBs, HBs and I/Os into different clusters in such a way that the inter-cluster communication is minimized. By minimizing inter-cluster communication we obtain a depopulated global interconnect network and hence reduced area. Partitioner is based on hMetis [50] platform. hMetis combines Fidducia-Mattheyses (FM) [47] algorithm with its multi phase refinement approach to optimize the partitioning of the netlist. The main objective of partitioner is to reduce communication between different partitions and FM algorithm achieves this objective using a hill-climbing, non greedy, iterative improvement approach. During each iteration, a block with highest gain is moved from one partition to another and then it is locked and it is not allowed to move during remaining time of iteration. After the block is moved, the gain of all of its associated blocks is recomputed and this process continues until all the blocks are locked. At the end of an iteration, total cost is compared to that of previous iteration

and the algorithm is terminated when it fails to improve during an iteration. After the netlist is partitioned, it is placed and routed on the architecture.

4.4.2.2 Routing

Once partitioning is done, placement file is generated that contains positions of different blocks on the architecture. This placement file along with netlist file is then passed to another software module called router which is responsible for routing of the netlist. In order to route all nets of the netlist, routing resources of the interconnect structure are first assigned to the respective blocks of the netlist that are placed on the architecture. These routing resources are modeled as directed graph abstraction $G(V, E)$. In this graph the set of vertices V represents the in/out pins of different blocks and the routing wires in the interconnect structure and an edge E between two vertices, represents a potential connection between the two vertices. Router is based on PathFinder [80] routing algorithm that uses an iterative, negotiation-based approach to successfully route all nets in a netlist. In order to optimize the FPGA architecture, a binary search algorithm is used. This algorithm determines the minimum number of signals required to route a netlist on FPGA.

4.4.3 Software Flow for Heterogeneous Mesh-Based Architecture

4.4.3.1 Placement

For mesh-based architecture, the netlist obtained in .net format is placed on the architecture using the placement algorithm that determines the position of different block instances of a netlist on their respective block types on FPGA architecture. The main goal is to place connected instances near each other so that minimum routing resources are required to route their connections. The placer uses simulated annealing algorithm [37, 105] to achieve a placement having minimum sum of half-perimeters of the bounding boxes of all the nets. This placer also optimizes floor-planning of different blocks on the FPGA architecture. Different operation that are performed by the placer are detailed as below.

4.4.3.2 Placer Operations

The placer either moves an instance from one block to another, moves a block from one slot position to another, rotates a block at its own axis, or moves an entire column of blocks. After each operation, the bounding box cost (also called as placement cost) is recomputed for all the disturbed signals. Depending on the cost value and the annealing temperature, the simulated annealing algorithm accepts or rejects the current operation.

The placer performs its operation on "source" and "destination" and the slots occupied by source and destination are termed as source window and destination window respectively. Normally, source window contains one block whereas destination window can contain multiple blocks. An example of source and destination windows is shown in Fig. 4.9a and b respectively. Once the source and destination windows are selected, the move operation is performed if:

1. Destination window does not contain any block that exceeds the boundary of destination window. An example violating this condition is shown in Fig. 4.9c.
2. The destination window does not exceed the boundaries of slot-grid (refer to Fig. 4.9d).
3. Destination window does not overlap source window diagonally (refer to Fig. 4.9g). However if the destination window overlaps source window vertically or horizontally, then horizontal or vertical translation operation is performed. Figure 4.9e shows an example where destination window overlaps source window vertically and Fig. 4.9f shows that the move operation is performed using vertical translation.

However, if above three conditions are not met, the procedure continues until a valid destination window is found. After the selection of source and destination, placer either moves an instance, moves a block, rotates a block, or moves an entire column of blocks. The rotation of blocks is important when the class number assigned to the input pins of a block are different; bounding box varies depending upon the pin positions and their directions. A block can have an orientation of 0°, 90°, 180° or 270°. Figure 4.9h depicts a 90° clock-wise rotation. Multiples of 90° rotation are allowed for all the blocks having a square shape, whereas at the moment only multiples of 180° rotation are allowed for rectangular (non-square) blocks. A 90° rotation for non-square blocks involves both rotation and move operations, which is left for future work.

4.4.3.3 Routing

After the placement of netlist on the FPGA architecture, the exploration environment constructs routing graph for the architecture. Few of the architecture description parameters required for the construction of routing graph are taken from the architecture description parameters. These parameters mainly include the type of routing network (unidirectional or bidirectional), channel width, I/O rate, block types and their pin positions on the block. Other parameters depend on the floor-planning details. These parameters include the position of blocks on the slot-grid and their orientation (0°, 90°, 180° or 270°). After the construction of routing graph, the PathFinder routing algorithm [80] is used to route netlists on the routing architecture. In case a binary search operation is used, routing graph is constructed for varying channel widths; routing is tried for each channel width until a minimum channel width is found.

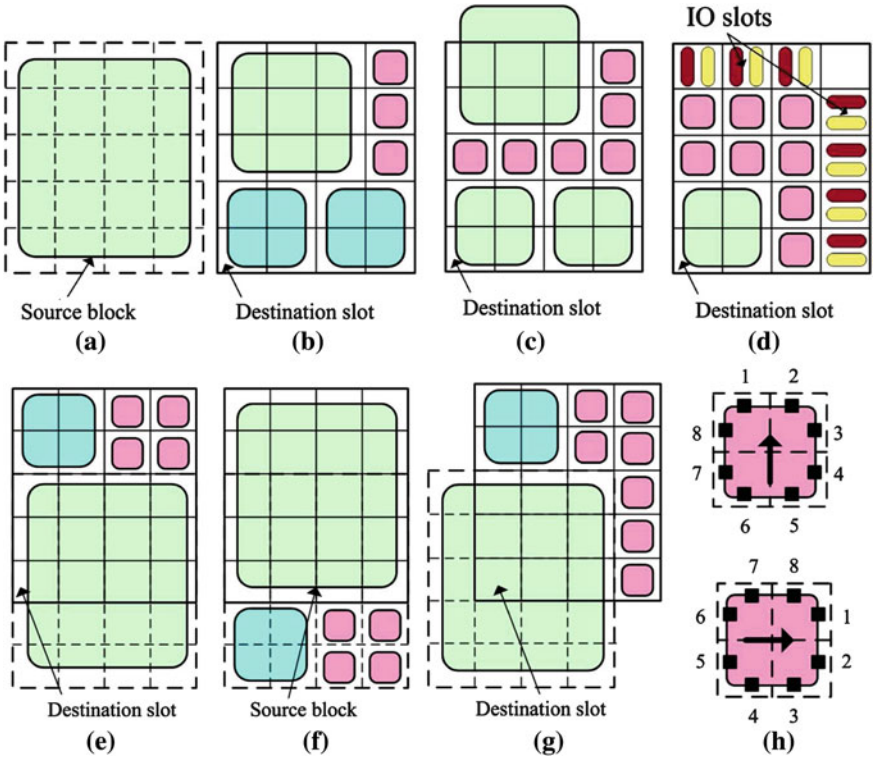

Fig. 4.9 Placer operations

4.4.4 Area Model

Once the optimization of the architecture is over, a generic area model is used to calculate the area of the FPGA (separately for mesh-based and tree-based architecture). The area model is based on the reference FPGA architectures shown in Figs. 4.1 and 4.4 respectively. Area of SRAMs, multiplexors, buffers and Flip-Flops is taken from a symbolic standard cell library (SXLIB [9]) which works on unit Lambda(λ). The area of FPGA is reported as the sum of the areas taken by the switch box, connection box, buffers, soft logic blocks, and hard-blocks.

4.5 Exploration Techniques

Various techniques are explored for both mesh-based and tree-based architectures using the software flow described in Sect. 4.4. A brief overview of different techniques is presented here.

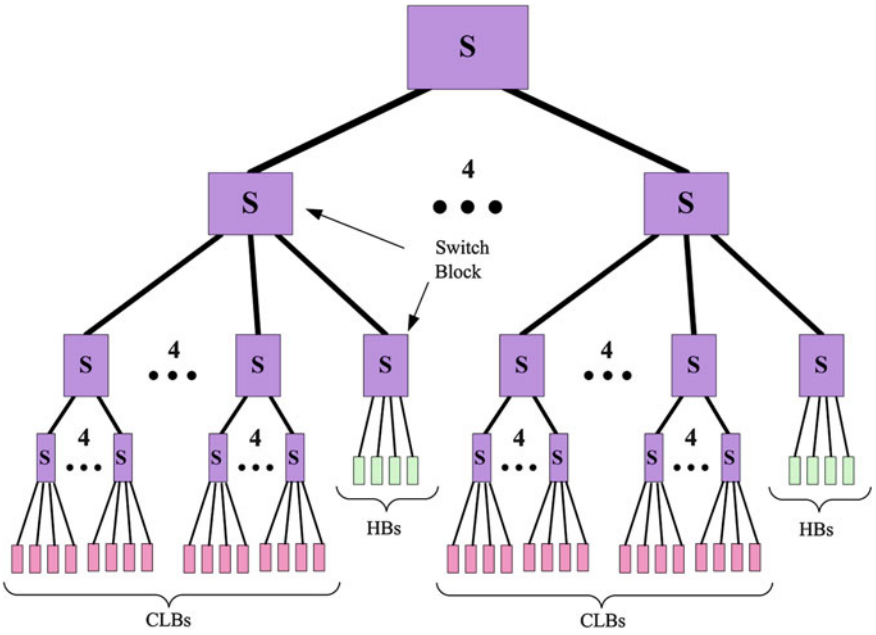

Fig. 4.10 Symmetric tree-based FPGA architecture

4.5.1 Exploration Techniques for Heterogeneous Tree-Based Architecture

Different manual parameters are used in architecture description file to explore two techniques for tree-based architecture. Generalized examples of both techniques are shown in Figs. 4.10 and 4.11 respectively. These techniques are detailed below:

4.5.1.1 Symmetric

A generalized example of first technique is shown in Fig. 4.10. This technique is referred as symmetric (SYM). In this technique clusters of HBs are mixed with those of LBs and HBs can be placed at any level of hierarchy in order to have best design fit. In this technique the symmetry of hierarchy is respected which can eventually result in wastage of HBs and their associated routing resources. For example in Fig. 4.10, it can be seen that this architecture supports four clusters of HBs of a certain type where each cluster contains four HBs. This is because of the fact that this is an arity 4 architecture. However the respect for symmetry of hierarchy may lead to under utilization of HBs and their associated routing resources in the case where a netlist requires less HBs than supported by the architecture.

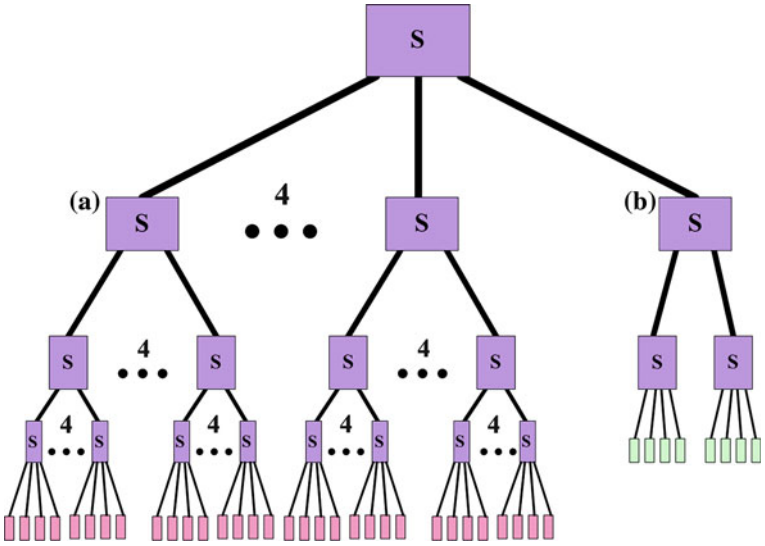

Fig. 4.11 Asymmetric tree-based FPGA architecture

4.5.1.2 Asymmetric

Contrary to the first technique, where the architecture contains only one structure, second technique contains two sub-structures: one sub-structure contains only CLBs while the other contains only HBs. An example of second technique is shown in Fig. 4.11. The main motivation behind this technique is the easy management of logic and routing resources. In this technique both sub-structures are constructed separately and the communication between them is ensured using the switch blocks of their parent cluster. Since the two sub-structures are constructed independently, they do not have to respect the arity of each other; hence leading to more optimized logic and routing resources. This technique is referred as asymmetric (ASYM).

Although this aspect is not fully explored in this work but this type of technique can also be used to exploit arithmetic intensive applications. Arithmetic intensive applications contain a large portion of data-path circuits that contain hard-blocks (e.g. multipliers, adders, memories etc.) that are connected together by regularly structured signals called buses. Conventional FPGAs do not use the regularity of data-path circuits. The regularity of data-path circuits can be exploited by implementing coarse-grain (or bus-based) routing in the sub-structure containing only HBs; routing of the sub-structure containing only CLBs remains unchanged. Exploitation of the regularity of data-path circuits is possible in this technique as the two sub-structures are independent of each other and they communicate with each other only through a parent cluster. By implementing the coarse-grain routing in the sub-structure containing only HBs, the number of SRAMs and switches can be reduced which can eventually lead to smaller area of the architecture.

4.5.2 Exploration Techniques for Heterogeneous Mesh-Based Architecture

By using different placer operations, six floor-planning technique are explored for mesh-based architecture. The detail of these floor-planning techniques is as follows:

4.5.2.1 Apart

In this technique, hard-blocks are placed in fixed columns, apart from the CLBs. This technique is shown in Fig. 4.12a and is termed as Apart (A). Such kind of technique can be beneficial for data-path circuits as described by [29]. It can be seen from the figure that if all HBs of a type are placed and still there is space available in the column then in order to avoid wastage of resources, CLBs are placed in the remaining place of column.

4.5.2.2 Column-Partial

Figure 4.12b shows the Column-Partial (CP) technique where columns of HBs are evenly distributed among columns of CLBs.

4.5.2.3 Column-Full

Figure 4.12c shows Column-Full (CF) technique where columns of HBs are evenly distributed among CLBs. Contrary to first and second techniques, whole column contains only one type of blocks. This technique is normally used in commercial architectures and topologically this technique is equivalent to Symmetric and Asymmetric techniques of tree-based FPGA architecture.

4.5.2.4 Column-Move

In this technique, HBs are placed in columns but unlike first three techniques, columns are not fixed, rather they are allowed to move using the column-move operation of placer. This technique is shown in Fig. 4.12d and it is termed as Column-Move (CM).

4.5.2.5 Block-Move

In this technique HBs are not restricted in columns; and they are allowed to move through block move operation. This technique is termed as Block-Move (BM) and it is shown in Fig. 4.12e.

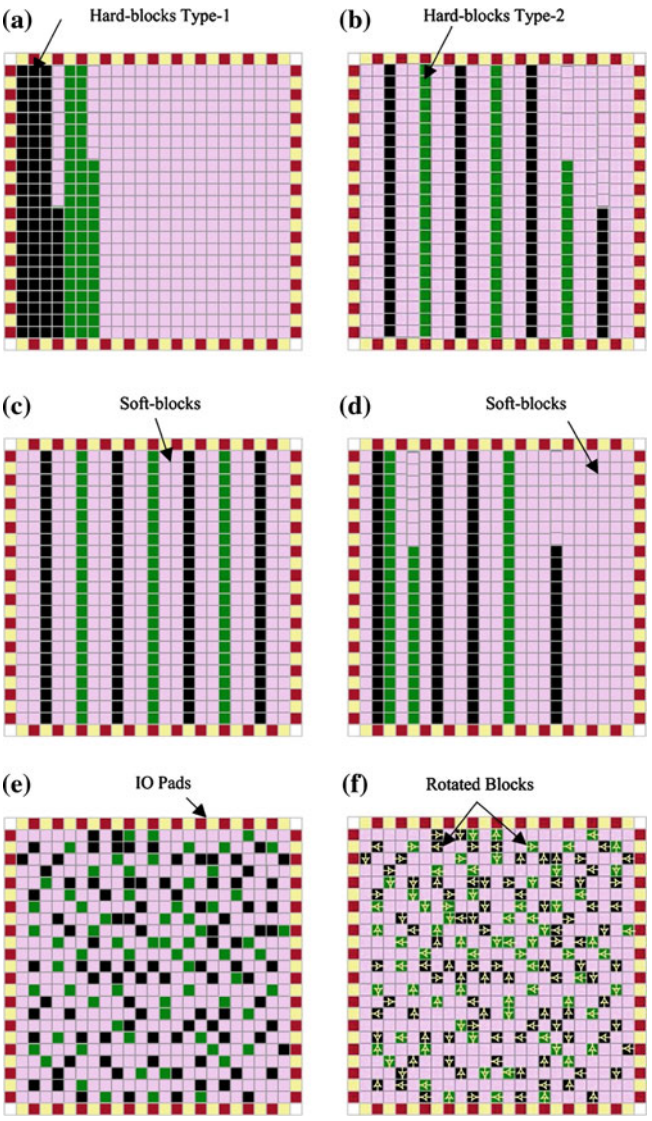

Fig. 4.12 Exploration techniques for mesh-based architecture

4.5.2.6 Block-Move-Rotate

The blocks in this technique are allowed to move and rotate through block move and rotate operations. This floor-planning technique is shown in Fig. 4.12f and it is termed as Block-Move-Rotate (BMR). Among these techniques CF is the technique that is usually used in mesh-based architectures while rest of them are new. Different

floor-planning techniques for mesh-based FPGA are explored here because floor-planning can have a major implication on the area of an FPGA. If a tile-based layout is required for an FPGA, similar blocks can be placed in same column. In this way, width of the entire column can be adjusted according to the layout requirements of the blocks placed in a column. On the other hand, if blocks of different types are placed in a column, the width of the column cannot be fully optimized. This is because the column width can only be reduced to maximum width of any tile in that particular column. Thus, some unused space in smaller tiles will go wasted. Such a problem does not arise if a tile-based layout is not required. In such a case, an FPGA hardware netlist can be laid out using any ASIC design flow.

4.6 Experimentation and Analysis

Different techniques of two architectures are explored using the software flow described in Sect. 4.4. Three sets of benchmarks are used for this exploration. The detail about three sets and their selection criteria is described below.

4.6.1 Benchmark Selection

Generally in academia and industry, the quality of an FPGA architecture is measured by mapping a certain set of benchmarks on it. Thus the selection of benchmarks plays a very important role in the exploration of heterogeneous FPGAs. This work puts special emphasis on the selection of benchmark circuits, as different circuits can give different results for different architecture techniques. This work categorizes the benchmark circuits by the trend of communication between different blocks of the benchmark. So, three sets of benchmarks are assembled having distinct trend of inter-block communication. These benchmarks are shown in Tables 4.5, 4.6 and 4.7. Benchmarks shown in Table 4.5 are developed at [79], the benchmarks shown in Table 4.6 are obtained from [36] and the benchmarks shown in Table 4.7 are obtained from [119]. The communication between different blocks of a benchmark can be mainly divided into the following four categories:

1. CLB-CLB: CLBs communicate with CLBs.
2. CLB-HB: CLBs communicate with HBs and vice versa.
3. HB-HB: HBs communicate with HBs.
4. IO-LB/HB: I/O blocks communicate with CLBs and HBs.

In SET I benchmarks, the major percentage of total communication is between HBs (i.e. HB-HB) and only a small part of total communication is covered by the communication CLB-CLB or CLB-HB. On average, in SET I, the HB-HB communication takes up to 80% of the total communication between different instances of the benchmarks (netlists). Similarly, in SET II the major percentage of total communication

Table 4.5 DSP benchmarks SET I

Circuit name	Inputs	Outputs	LUTs (LUT-4)	Mult (8×8)	Slansky ($16 + 16$)	Sff (8)	Sub ($8 - 8$)	Smux (32:16)	Function
ADAC	18	16	47	–	–	2	–	1	–
DCU	35	16	34	1	1	4	2	2	Discrete cosine transform
FIR	9	16	32	4	3	4	–	–	Finite impulse response
FFT	48	64	94	4	3	–	6	–	Fast fourier transform

Table 4.6 Open core benchmarks SET II

Circuit name	Number of inputs	Number of outputs	Number of LUTs (LUT-4)	Number of multipliers (16×16)	Number of adders ($20 + 20$)	Function
cf_fir_3_8_8_open	42	18	159	4	3	Finite impulse response (8 bit)
cf_fir_7_16_16	146	35	638	8	14	Finite impulse response (16 bit)
cfft16 $\times$ 8	20	40	1,511	–	26	–
cordic_p2r	18	32	803	–	43	Polar to rectangular
cordic_r2p	34	40	1,328	–	52	Rectangular to polar
fm	9	12	1,308	1	19	–
fm_receiver	10	12	910	1	20	–
lms	18	16	940	10	11	–
reed_solomon	138	128	537	16	16	–

is HB-CLB and in SET III, major percentage of total communication is covered by CLB-CLB. Normally the percentage of IO-CLB/HB is a very small part of the total communication for all the three sets of benchmarks.

Table 4.7 Open core benchmarks SET III

Circuit name	Number of inputs	Number of outputs	Number of LUTs (LUT-4)	Number of multipliers (18 × 18)	Function
cf_fir_3_8_8_ut	42	22	214	4	Finite impulse response (8 bit)
diffeq_f_systemC	66	99	1532	4	–
diffeq_paj_convert	12	101	738	5	–
fir_scu	10	27	1366	17	–
iir1	33	30	632	5	Infinite impulse response (16 bit)
iir	28	15	392	5	Infinite impulse response (8 bit)
rs_decoder_1	13	20	1553	13	Decoder
rs_decoder_2	21	20	2960	9	Decoder

4.6.2 Experimental Methodology

Once the benchmarks are selected, they are placed and routed on the two architectures using two different experimental methodologies. An overview of the two methodologies is as follows:

4.6.2.1 Individual Experimentation

In first methodology, experiments are performed individually for each netlist (both for mesh-based and tree-based architectures). The architecture definition, floor-planning, placement/ partitioning, routing and optimization is performed individually for each netlist. Although, such an approach is not applicable to real FPGAs, as their architecture, floor-planning and routing resources are already defined and fixed. However this methodology is useful in order to have detailed analysis of a particular floor-planning technique and usually it is employed to evaluate different parameters of the architecture under consideration. If a generalized architecture is defined for a group of netlists, the netlists with highest logic and routing requirements decide logic and routing resources of the architecture and the behavior of remaining netlists of the group is overshadowed by larger netlists of the group. So, to have more profound analysis, the architecture and floor-planning is optimized for each netlist and the area

Table 4.8 Area of different blocks of three sets

Block name	Inputs	Outputs	Block size (λ^2)
clb	4	1	58,500
mult (8 × 8)	16	16	1,075,250
slansky_16	32	16	306,750
sff_8	8	8	36,000
sub_8	17	8	154,500
smux_16	33	16	36,000
mult (16 × 16)	32	32	1,974,000
adder (20 + 20)	41	21	207,000
mult (18 × 18)	36	36	2,498,300
sram	–	–	1,500
buffer	1	1	1,000
flip-flop	1	1	4,500
mux 2:1	2	1	1,750

of the architecture is calculated individually using the model described in Sect. 4.4.4 and the values shown in Table 4.8. Later average of results of all netlists gives more thorough results.

4.6.2.2 Generalized Experimentation

However, in order to further validate the results, we have also performed experimentation based on the generalized architecture. In this methodology, for mesh-based architecture a generalized minimum architecture is defined for each SET of netlists and the floor-planning is then optimized for this architecture. Generalized floor-planning is achieved by allowing the mapping of multiple netlists on the same architecture where each block of the architecture allows mapping of multiple instances on it, but multiple instances of the same netlist are not allowed. Similarly for tree-based architecture, multiple netlists are partitioned using generalized architecture description where mapping of multiple instances on each block position is allowed but multiple instances of same netlist are not allowed on a single block position. Once generalized floor-planning/partitioning is over, individual netlists are placed and routed separately on both architectures with minimum routing resources that can route any of the netlists of the SET. Since, in this case a generalized architecture is used, optimization of the architecture is not performed for individual netlists.

4.6.3 Results Using Individual Experimentation Approach

Experiments are performed for three sets of benchmarks using the software flow described earlier where 21 benchmarks of three sets are placed and routed individually on the two FPGA architectures using different exploration techniques. Similar

to homogeneous architectures experimentation, for tree-based architecture the LUT size is set to be 4 while arity size is set to be 4 too; for mesh-based architecture LUT size is also set to be 4 and CLB size is set to be 1 for both architectures. Since placement cost and channel width of a mesh-based architecture are directly related to its area, we first present the effect of different floor-planning techniques of mesh-based architecture on these two values.

4.6.3.1 Placement Cost and Channel Width Comparison Results

Placement cost results for six floor-planning techniques of mesh-based architecture are shown in Fig. 4.13. In this figure, for each benchmark circuit, placement cost of five floor-planning techniques (Apart (A), Column-Partial (CP), Column-Full (CF), Column-Move (CM) and Block-Move (BM)) of mesh-based FPGA is normalized against the placement cost of Block-Move-Rotate (BMR) technique. Placement cost is the sum of half perimeters of bounding boxes (BBX cost) of all the NETS in a netlist. The results for benchmarks 1–4, 5–13 and 14–21 correspond to SET I, SET II and SET III respectively. For example in Fig. 4.13 column 1 gives normalized results for "ADAC", column 5 gives results for "cf_fir_3_8_8_open" and column 21 gives results for "rs_decoder_2". The avg1, avg2 and avg3 corresponds to the geometric average of these results for SET I, SET II and SET III respectively while avg corresponds to the average of all netlists. As it can be seen from the figure that in general Apart (A) gives the worst and BMR gives the best placement cost results whereas the results of remaining techniques are in between these two techniques. In Apart the average placement cost is higher than the other floor-planning techniques because in this technique columns of hard-blocks are fixed and they are separated from CLBs. Although this kind of floor-planning technique can give good results for data-path circuits, it gives poor placement solution for control path circuits as the columns of HBs are fixed and they are not mixed with CLBs. This situation further aggravates if there are more than one types of HBs that are required to be supported by the architecture. Although columns of HBs are fixed in CF and CP, they give better placement cost results when compared to Apart as in those techniques the columns of hard-blocks are not placed apart rather they are interspersed evenly among CLBs; hence leading to smaller placement costs. BMR gives the best placement cost results because it is the most flexible technique among the six floor-planning techniques. Although the only difference between BM and BMR is that of hard-block rotation, it gives a slight edge to BMR which might lead to smaller bounding box and eventually lower placement costs of the architecture. Figure 4.14 gives the channel width results of the six floor-planning techniques of mesh-based architecture. In this figure, for 21 benchmarks, channel widths of 5 floor-planning techniques are normalized against the channel width of BMR. Similar to the results in Fig. 4.13, BMR gives the best results and Apart gives the worst results. The two figures (i.e. Figs. 4.13 and 4.14) look similar to each other as

1. Both figures give normalized results.

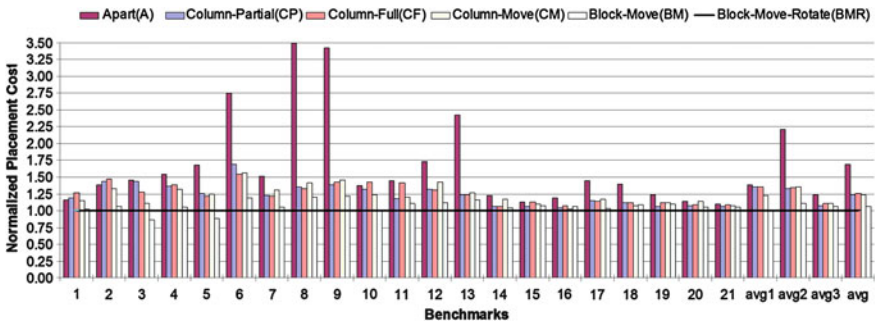

Fig. 4.13 Placement cost comparison between different techniques of mesh-based architecture

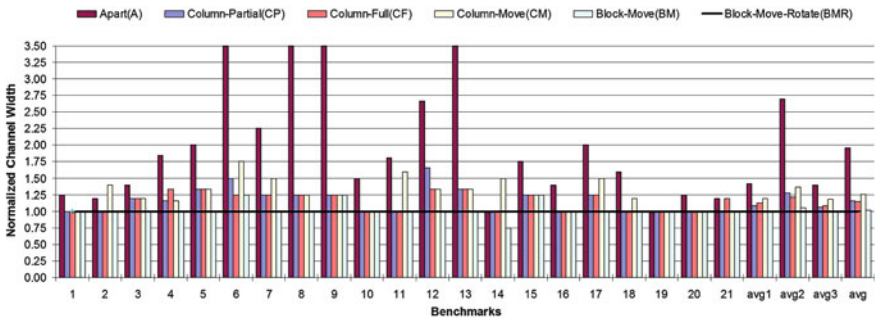

Fig. 4.14 Channel width comparison between different techniques of mesh-based architecture

2. Placement cost and channel width are closely related to each other.

Generally an architecture with higher placement cost indicates a poor placement solution as in this solution instances connected to each other are placed far from each other. A poor placement solution normally leads to higher channel width of the architecture as the instances placed far from each other require more routing resources than the ones placed close to each other. Analysis of the results in Figs. 4.13 and 4.14 show that, on average, CF (the technique used in most of the work cited at the start of the chapter) gives 35, 35 and 11% more placement cost than BMR, for SET I, SET II and SET III benchmark circuits respectively. Figure 4.14 shows that, on average, CF requires 13, 22 and 9% more channel width than BMR for SET I, SET II and SET III respectively.

The reason that BMR gives better placement cost and channel width results when compared to other techniques lies in its flexibility. In four out of five remaining techniques (i.e. Apart, Column-Partial, Column-Full and Column-Move), hard-blocks are fixed in columns and they are not free to move and the placer can move only logic-blocks which eventually leads to higher placement costs and larger channel widths. In case of BMR, however, both HBs and CLBs can be moved which increases the flexibility of the architecture; hence smaller placement cost and smaller channel widths.

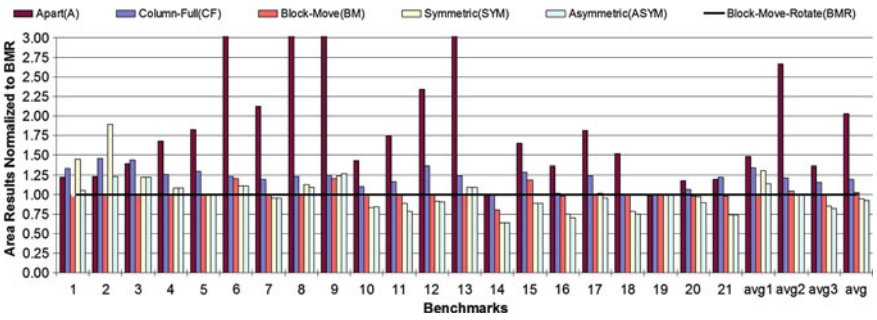

Fig. 4.15 Area comparison between different techniques of mesh-based and tree-based architecture

It is interesting to note that the only difference between BM and BMR techniques is that of block rotation. Although block rotation has no effect on CLBs, it affects the bounding box cost of HBs which can decrease the overall placement cost; hence a small gain in channel width is observed.

4.6.3.2 Area Comparison Results

Area results for different exploration techniques of mesh-based and tree-based architectures are shown in Fig. 4.15. Similar to the results of Figs. 4.13 and 4.14, area results of different techniques are normalized against the BMR technique of mesh-based architecture. In order to avoid congestion, results for only 4 out of 6 techniques of mesh-based architecture are presented here. It can be seen from the figure that gains of BMR technique observed in Figs. 4.13 and 4.14 remain valid here too and BMR technique gives either equal or better results when compared to other floor-planning techniques of mesh-based architecture. Among four floor-planning techniques of mesh-based architecture, on average, Apart gives the worst results and BMR gives the best results whereas the results of CF are in between Apart and BMR. However, when compared to exploration techniques of tree-based architecture, for SET I benchmark circuits, SYM requires 35% more area than BMR, and ASYM requires 12% more area than BMR. However for SET II benchmark circuits, on average BMR is almost equal to SYM and ASYM. For SET III benchmark circuits BMR is worse than SYM and ASYM by 14 and 18% respectively. Further, the comparison of ASYM with CF shows that for three sets of benchmarks, on average, tree-based architecture consumes 15, 21 and 29% less area than mesh-based architecture. As compared to the mesh-based architecture, tree-based architecture slightly under utilizes its logic resources because of its hierarchy. This under utilization leads to natural congestion spreading in routing resources; hence leading to smaller switch sizes and ultimately reduced area.

In this work, the BMR floor-planning serves as a near ideal floor-planning with which other floor-planning techniques are compared. It can be noted that results of CF compared to BMR vary depending upon the set of benchmarks that are used. For SET I benchmark circuits, where the types of blocks for each benchmark are two or

more than two and communication is dominated by HB-HB type of communication, CF produces worse results than the other two sets of benchmarks. This is because of the fact that columns of different HBs are separated by columns of CLBs and HBs need extra routing resources to communicate with other HBs. However in BMR there is no such limitation; HBs communicating with each other can always be placed close to each other. For other two sets the gap between CF and BMR is relatively less. The reduced HB-HB communication in SET II and SET III benchmark circuits is the major cause of reduction in the gap between CF and BMR. However 21 and 16% area difference for SET II and SET III is due to the placement algorithm. In CF, the simulated annealing placement algorithm is restricted to place hard-block instances of a netlist at predefined positions. This restriction for the placer reduces the quality of placement solution. Decreased placement quality requires more routing resources to route the netlist; thus more area is required. The results show that BMR technique produces least placement cost, the smallest channel width and hence the smallest area for mesh-based heterogeneous FPGA. However, BMR floor-planning technique is dependant upon target netlists to be mapped upon FPGA. Although such an approach is not suitable for generalized FPGAs, it can be beneficial for domain specific FPGAs. Moreover, the hardware layout of BMR requires more efforts than CF.

For tree-based FPGA, ASYM produces better results than SYM because of its better logic resource utilization and further it is better than the best technique of mesh-based FPGA (i.e. BMR) by an average of 8% for a total of 21 benchmark circuits. The gain of tree-based FPGA is not large when compared to the best technique of mesh-based FPGA. This is because of the fact that tree-based FPGA requires more resources because of its hierarchy. Although it helps in spreading the congestion, it leads to extra logic and routing resources which decrease the area gain of tree-based FPGA. However, when the best technique of tree-based FPGA is compared to equivalent mesh-based FPGA (i.e. CF), it gives on average a gain of 15, 21 and 29% for SET I, SET II and SET III benchmarks respectively.

4.6.3.3 Critical Path Comparison Results

In order to evaluate the performance of different techniques of two architectures, we have calculated the number of switches crossed by critical path. Since we are exploring a number of techniques for both mesh-based and tree-based architectures, it would be very difficult to perform layout for each technique and determine the exact critical path delay. So, we use a simple model that gives an overview of the impact of active routing resources (switches) on the overall performance of the architecture. Similar to area results, critical path results are normalized against BMR floor-planning of mesh-based FPGA. These results are shown in Fig. 4.16. To avoid congestion, results for only 6 out of 8 techniques are shown. It can be seen from Fig. 4.16 that due to its higher flexibility, BMR gives higher performance results than other floor-planning techniques of mesh-based FPGA. On average, CF critical path crosses 5, 7 and 10% more switches than BMR technique for SET I, SET II and SET III bench-

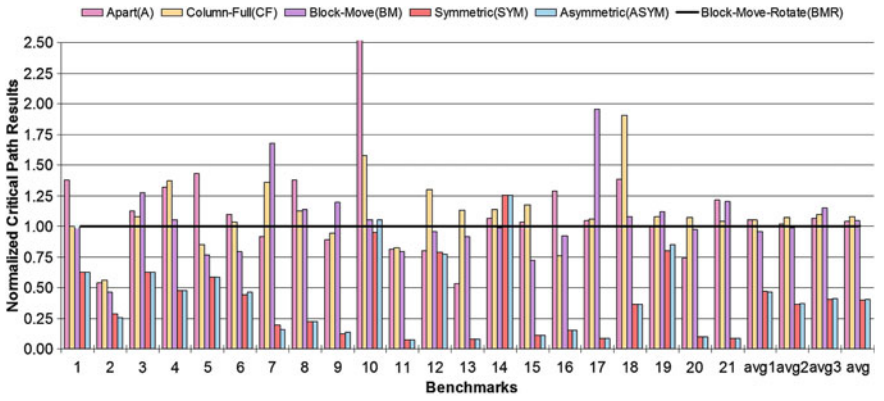

Fig. 4.16 Critical path comparison between different techniques of mesh-based and tree-based architecture

marks respectively. Although Apart (A) gives worst results in terms of placement cost, channel width and area, it is quite interesting to note that critical path results of Apart (A) are comparatively better than CF. This is because of the fact that in Apart columns of HBs are placed close to each other and apart from the CLBs. Since in SET I benchmarks majority of the communication involves HB-HB communication, there is a strong probability that critical path involves HBs which ultimately leads to 50% of benchmarks of SET I crossing less number of switches than CF. For SET II benchmarks this percentage drops to 44% as there is more communication between CLBs and HBs. However, in case of SET III benchmarks 75% of benchmarks cross less number of switches for Apart than CF as the communication pattern is dominated by CLB-CLB and in case of Apart there are no columns of HBs interspersed in between CLBs; hence leading to smaller number of switches that are crossed by critical path. Although Apart gives better results than CF, BMR manages to produce the best overall results among floor-planning techniques of mesh-based architecture due to its higher flexibility.

However, compared to the tree-based FPGA, both SYM and ASYM techniques of tree-based FPGA produce far better performance than BMR technique due to the inherent characteristic of tree-based architecture. Compared to BMR technique of mesh-based architecture, on average, SYM and ASYM techniques of tree-based architecture cross 53%, (54% less switches for SET I), 64%, (63% less switches for SET II) and 60%, (59% less switches for SET III) benchmarks respectively. It can also be observed from these results that on average ASYM technique crosses 1% more switches than SYM technique. In ASYM technique, HBs have a separate substructure and if critical path involves HBs and CLBs then it can lead to an increase in number of switches crossed by critical path (refer to Figs. 4.10 and 4.11). However if critical path involves no HBs or only HBs and I/Os, it can lead to smaller number of switches than SYM technique (refer to Fig. 4.16 results for benchmark 2 and 7).

4.6.3.4 SRAM and Buffer Comparison Results

Power optimization of FPGAs has become very important with the advancement in process technology. Although in this work a detailed power analysis of mesh-based and tree-based FPGA architectures is not performed, it gives a brief overview of the static power consumption of the two architectures; which has become increasingly important for smaller process technologies [12]. Static power of the FPGAs is directly related to the configuration memory and the number of buffers in an FPGA architecture [132]. Therefore, a comparison of configuration memory and number of buffers for different techniques of the two architectures is shown in Figs. 4.17 and 4.18 respectively.

Figure 4.17 shows number of SRAMs for different techniques normalized against the BMR technique of mesh-based FPGA. Comparison of BMR with CF shows that, on average, CF consumes 23, 16 and 9% more SRAMs than BMR for SET I, SET II and SET III respectively. Comparison of BMR with tree-based architecture techniques shows that, on average, SYM consumes 9% more and ASYM consumes 10% less SRAMs for SET I. However, for SET II and SET III SYM and ASYM consume 11%, 7% and 13%, 15% less SRAMs than BMR respectively. Similarly Fig. 4.18 shows that, compared to BMR, CF consumes 9, 22 and 18% more buffers for SET I, SET II and SET III respectively. Comparison of SYM and ASYM with BMR shows that both consume 6% more buffers for SET I, 3% less buffers for SET II and 15%, (18% less buffers for SET III). Although the comparison presented in Figs. 4.17 and 4.18 does not give detailed power estimation of the two architectures, it gives an empirical estimate of the static power of the two architectures and as stated by [63], it closely correlates to the average area results of the two architectures.

4.6.4 Results Using Generalized Experimentation Approach

In this approach experiments are performed for three sets of benchmarks shown in Tables 4.5, 4.6 and 4.7 respectively. Different exploration techniques are explored for each architecture. For each technique, a minimum common architecture is defined for each set of benchmarks that can implement any of the netlists of the benchmark set. Area comparison results for different exploration techniques of both mesh-based and tree-based architectures are shown Fig. 4.19. For each of the three sets of benchmark, the results of five floor-planning techniques of mesh-based architecture and two exploration techniques of tree-based architecture are normalized against the BMR technique of mesh-based architecture. Contrary to the individual experimentation approach where separate architectures are defined and optimized for each netlist of benchmark set, a common architecture is defined for each set in this approach.

The area comparison results of different floor-planning techniques show that compared to other floor-planning techniques of mesh-based FPGA, BMR produces equal or better results. However, compared to individual experimentation approach, the gain of BMR compared to CF is reduced from 23%, 10 to 5%, 3% for SET II and SET III

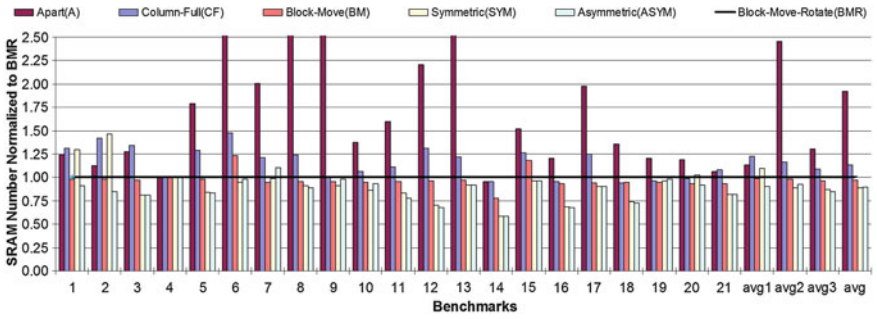

Fig. 4.17 SRAM comparison between different techniques of mesh-based and tree-based architecture

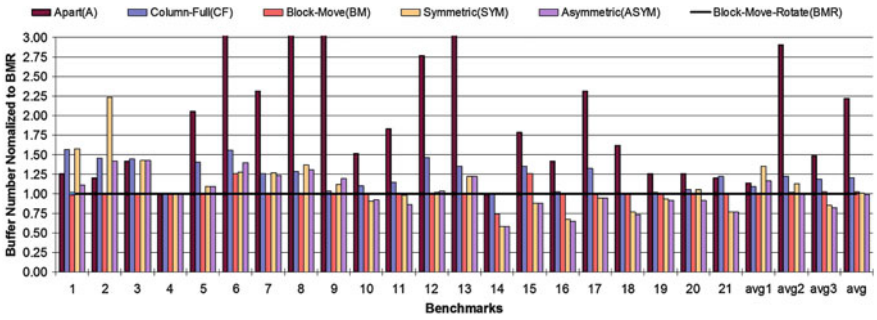

Fig. 4.18 Buffer comparison between different techniques of mesh-based and tree-based architecture

benchmarks respectively, while the gain for SET I benchmarks remains unchanged. This drop in gain is mainly due to the combined floor-planning optimization of all the netlists of a SET where the routing requirements of smaller netlists are overshadowed by those of larger netlists. As far as the comparison of BMR with SYM and ASYM techniques of tree-based FPGA is concerned, the results of tree-based topologies are further improved. For SET I benchmarks, SYM and ASYM techniques are only 4 and 3% worse than BMR and for SET II, their gain is increased from 0 to 22 and 24% and for SET III their gain is increased from 14–18% to 22–24% respectively.

In case of generalized experimentation, increase in the area gain of both techniques of tree-based architecture is because of the fact that in this experimentation the netlist having the highest routing requirements decides the logic and routing resources of the architecture. Routing requirement of a benchmark not only depends on the number of CLBs and HBs but also on the connection density. In our case the benchmarks with the highest routing requirements are benchmarks 4, 10 and 21 for SET I, SET II and SET III respectively and results of these benchmarks in Fig. 4.15 correspond well to the results of Fig. 4.19.

Generalized critical path comparison results for different techniques of mesh-based and tree-based architectures are presented in Fig. 4.20. It can be seen from the

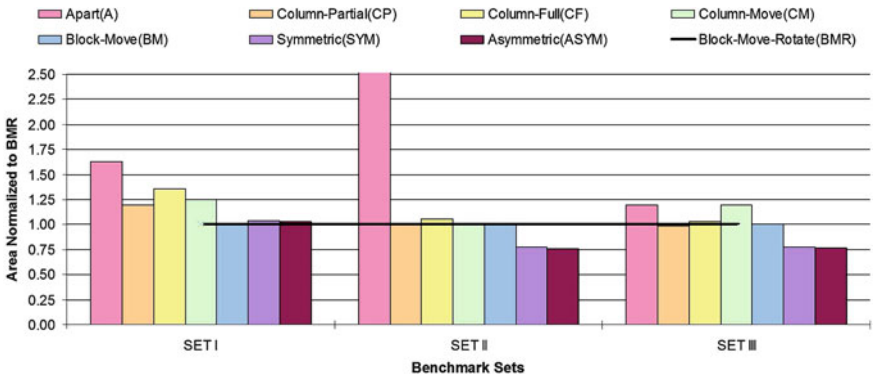

Fig. 4.19 Generalized area comparison between different techniques of mesh-based and tree-based architecture

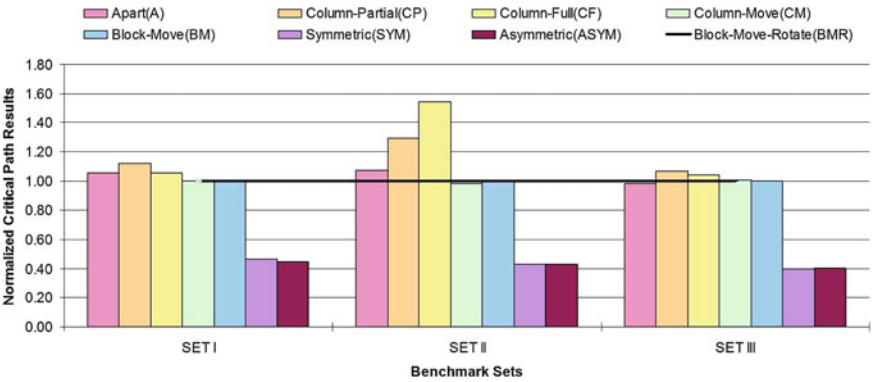

Fig. 4.20 Generalized critical path comparison between different techniques of mesh-based and tree-based architecture

figure that similar to results of individual experimentation methodology, techniques of tree-based architecture give far better results compared to the techniques of mesh-base architecture. On average, SYM crosses 54, 57 and 60% less switches than BMR and ASYM crosses 56, 57 and 60% less switches than BMR for SET I, SET II and SET III benchmark sets respectively. Further, SRAM and buffer comparison results are shown in Figs. 4.21 and 4.22 respectively. These results give an empirical estimate of the static power consumption of the two architectures which has become increasingly important with the advancement in the process technology. Figure 4.21 shows that CF consumes 37, 8 and 3% more SRAMs than BMR technique for SET I, SET II and SET III respectively whereas ASYM consumes almost same number of SRAMs for SET I and 24%, 14% less SRAMs than BMR for SET II and SET III respectively. Similarly the buffer comparison shows that ASYM produces the best overall results and as stated by [63] these results are in compliance with the area results of the two architectures. In this chapter we have mainly emphasized on the comparison between heterogeneous mesh-based and tree-based FPGA architectures

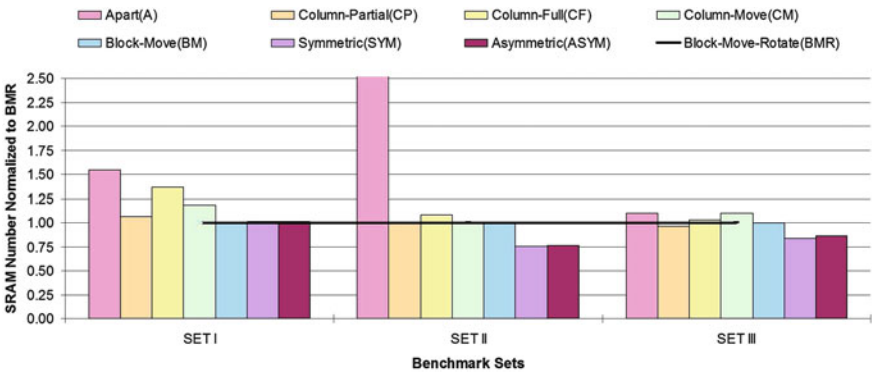

Fig. 4.21 Generalized SRAM comparison between different techniques of mesh-based and tree-based architecture

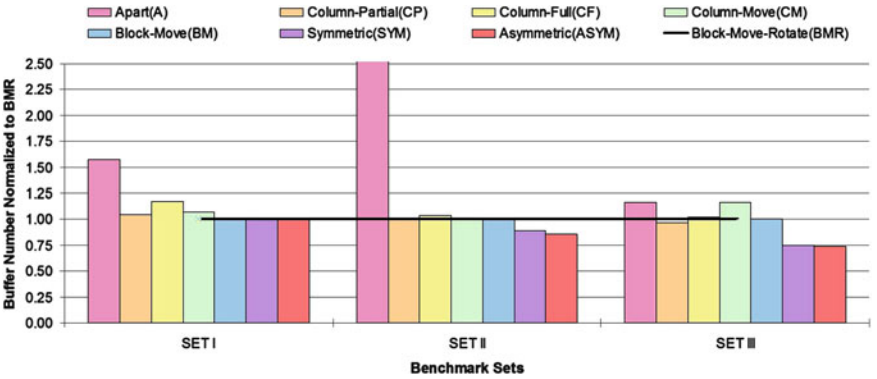

Fig. 4.22 Generalized buffer comparison between different techniques of mesh-based and tree-based architecture

and no comparison is presented between tree-based heterogeneous architectures and their homogeneous counterparts. However, in [45] we have performed a comparison between heterogeneous and homogeneous tree-based FPGA architectures and results show that on average heterogeneous architecture gives 41% area gain as compared to tree-based homogeneous FPGA architecture. So, it can be stated here that the introduction of hard-blocks in tree-based architectures improves their density as compared to their homogeneous counterparts and further they give better overall results when they are compared with mesh-based heterogeneous architectures.

The experimental results presented from Figs. 4.13 to 4.22 are concluded below. These conclusions are valid both for individual and generalized experimentation unless otherwise specified.

- Among the six floor-planning techniques of mesh-based FPGA, "Apart (A)" technique gives the worst area and static power results while "Block-Move-Rotate (BMR)" gives the best overall results. The "Apart" floor-planning is not a suitable

floor-planning; atleast not for the netlists used in this work. This floor-planning might be advantageous if

1. Control-path portion of a circuit implemented on CLBs is relatively small as compared to data-path portion of circuit implemented on hard-blocks.
2. Routing network of control and data-path sections of the FPGA architecture are optimized independently.

- Column based floor-planning (CF) of hard-blocks is advantageous for an optimized tile-based layout generation; the widths of hard-blocks placed in columns can be appropriately adjusted to optimize the layout area. However, column-based floor-planning is unable to decrease the placement cost as some other floor-plannings do. This difference in placement costs can sometimes result in as high as 35% difference in total area of FPGA.
- The floor-planning achieved through Block Move and Rotate (BMR) operation gives the least possible placement cost, and eventually least FPGA area as compared to other floor-planning techniques. However, such a floor-planning can be achieved only if the set of netlists are known in advance. Such can be a case if an application specific FPGA is desired for a product.
- Among the two exploration techniques of tree-based architecture, "Asymmetric(ASYM)" technique gives the best overall results (area, critical path delay and static power) because of its better resource utilization (see Sect. 4.5.1). When compared to the equivalent mesh-based floor-planning technique (i.e. CF floor-planning technique), ASYM gives better overall results. However, when compared to the best floor-planning technique of mesh-based architecture (i.e. BMR), ASYM gives either equal or better results except for the SET I benchmarks which makes tree-based architecture unsuitable for benchmarks involving excessive communication between HBs. However, this deficiency can be remedied by incorporating the architecture modifications that are suggested in Sect. 4.5.1.

4.7 Heterogeneous FPGA Hardware Generation

The hardware of heterogeneous FPGA is generated in a similar manner as that of homogeneous FPGA. However, modifications are performed to incorporate the effect of different types of blocks that are used by different netlists. Similar to the homogeneous FPGA, the VHDL model generator of heterogeneous FPGAs is integrated with their exploration environments and the parameters that are supported by the exploration environment are also supported by the VHDL generator.

The FPGA generation flow remains exactly the same except that now the block database contains details about a variety of blocks that are used by different netlist being mapped on the architecture. As far as the remaining steps involved in the VHDL model generation are concerned, they remain the same. The only difference between the VHDL generation of homogeneous and heterogeneous FPGAs is that of the support for hard-blocks.

4.8 Summary and Conclusion

This chapter presented a new exploration environment for tree-based heterogeneous FPGA architecture which remains relatively unexplored despite its attractive characteristics. Different architectural techniques of tree-based architecture are then explored using its exploration environment. Exploration of heterogenous tree-based FPGA architecture is mainly related to the architecture description and architecture optimization. For the architecture description, a detailed architecture description mechanism is designed to define a heterogeneous FPGA architecture. Once the architecture is defined, it is optimized using different parameters of architecture exploration environments. Also, this chapter presented an exploration environment for mesh-based heterogeneous FPGA architecture. Contrary to the existing environments of mesh-based architecture that use pre-determined floor-planning, the environment presented in this chapter automatically optimizes the floor-planning of the architecture. The major feature of the exploration environments of two FPGA architectures is that they are flexible and they can be used to explore different exploration techniques for two FPGA architectures. In order to evaluate the exploration environments of two architectures, a generalized software flow is designed which maps different applications on the two architectures separately. The software flow used to explore the two architectures is flexible in the sense that it can be used to implement different types of applications having different types of blocks. Since communication trends of applications play a very important role in the architecture evaluation, special care has been taken while selecting these applications. We have selected 21 benchmarks (applications) where we have covered different aspects of inter-block communication.

A number of techniques are explored for both architectures using their respective software flows and experimental results show that for 21 benchmarks, on average, a column-based (CF) island-style FPGA takes 19% more area, crosses 8% more switches on critical path, consumes 13% more memories and 20% more buffers than the best non-column (BMR) based island-style FPGA. These differences increase as the number of different types of hard-blocks increase in the FPGA architecture. However, these gains might decrease due to difficulties associated with layout of non-column based heterogeneous FPGA. Further, the comparison between different techniques of mesh-based (island-style) and tree-based (hierarchical) architecture shows that, on average, the best technique of tree-based architecture is 8.7% more area efficient, crosses 60% less switches on critical path, consumes 11% less memories and almost same number of buffers than the best non-column based technique of mesh-based architecture. These gains further increase when the best technique of tree-based architecture is compared to the equivalent column-based technique (i.e. CF) of mesh-based architecture. These results are averaged for 21 benchmarks which cover different aspects of heterogeneous benchmarks. The work and results presented in this chapter are also published in [116].