Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2011, Article ID 121404, 18 pages
doi:10.1155/2011/121404

Research Article

Exploration of Heterogeneous FPGA Architectures

Umer Farooq,[1] Husain Parvez,[1] Habib Mehrez,[1] and Zied Marrakchi[2]

[1] *LIP6, UPMC, 75005 Paris, France*
[2] *FLEXRAS Technologies, 93521 Saint-Denis Cedex, France*

Correspondence should be addressed to Umer Farooq, umer.farooq@lip6.fr

Received 27 August 2010; Revised 11 January 2011; Accepted 10 February 2011

Academic Editor: Michael Hübner

Mesh-based heterogeneous FPGAs are commonly used in industry and academia due to their area, speed, and power benefits over their homogeneous counterparts. These FPGAs contain a mixture of logic blocks and hard blocks where hard blocks are arranged in fixed columns as they offer an easy and compact layout. However, the placement of hard-blocks in fixed columns can potentially lead to underutilization of logic and routing resources and this problem is further aggravated with increase in the types of hard-blocks. This work explores and compares different floor-planning techniques of mesh-based FPGA to determine their effect on the area, performance, and power of the architecture. A tree-based architecture is also presented; unlike mesh-based architecture, the floor-planning of heterogeneous tree-based architecture does not affect its routing requirements due to its hierarchical structure. Both mesh and tree-based architectures are evaluated for three sets of benchmark circuits. Experimental results show that a more flexible floor-planning in mesh-based FPGA gives better results as compared to the column-based floor-planning. Also it is shown that compared to different floor-plannings of mesh-based FPGA, tree-based architecture gives better area, performance, and power results.

1. Introduction

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 in [1] shows that the use of embedded memory in FPGA improves its density and performance. Beauchamp et al. [2] have incorporated floating point multiply-add units in the FPGA and have reported significant area and speed improvements over homogeneous FPGAs. Ho et al. [3] have proposed a virtual embedded block (VEB) methodology that predicts the effects of embedded blocks in commercial FPGA devices, and they have shown that the use of embedded blocks causes an improvement in area and speed efficiencies. Also Govindu et al. [4] and Underwood and Hemmert [5] suggest the use of embedded blocks in FPGAs for better performance regarding complex scientific applications. The work in [6] 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 [7] and Altera [8] are also using HBs (e.g., multipliers, memories, and DSP blocks).

Almost all the work cited above considers mesh-based (island-style) 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 island-style, column-based heterogeneous FPGA lies in its simple and compact layout generation. When tile-based layout for an FPGA is required, the floor-planning of similar type blocks in a column simplifies the layout generation. The complete width of the entire column, having same type of blocks, can be adjusted appropriately to generate a very compact layout. 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 the number of types of blocks that are required to be supported by the architecture.

In order to reduce the amount of useless resources and increase the area density of FPGA architecture, in this work, we explore different floor-planning techniques of mesh-based FPGA. Although some domain-specific architectures [9–11], targeting multimedia and DSP domains and having a wide range of routing architectures, are proposed to address the problem of useless logic and routing resources, these architectures cannot be related to FPGA architectures as domain-specific architectures use a different routing structure compared to FPGAs. Also, unlike previous research [1–6] that mainly compares heterogeneous mesh-based FPGA architectures with their homogeneous counterparts, this work presents a detailed comparison between heterogeneous mesh- and tree-based architectures. Contrary to mesh-based architecture, a tree-based architecture is a hierarchical architecture where logic and routing resources are arranged in a multilevel clustered structure. A comparison between homogeneous mesh- and tree-based FPGA architectures was presented in [12] and in this work we extend that comparison to their heterogeneous counterparts.

Mainly six floor-planning techniques are explored for mesh-based architecture, four of which are column-based and two are non column-based. In order to evaluate these floor-planning techniques, this work also compares a tree-based heterogenous FPGA architecture [13] with different floor-planning techniques of mesh-based heterogeneous FPGA architecture. Contrary to mesh-based heterogenous FPGA, routability of a tree-based FPGA is independent of its floor-planing and the number of types of HBs required to be supported by the architecture. So, tree-based heterogenous FPGA can be advantageous as compared to mesh-based FPGA. Two different techniques are explored for tree-based architecture. First technique respects the symmetry of hierarchy, which is one of the characteristics of tree-based architectures. However, in order to ensure the optimal use of available resources, the second technique does not respect the symmetry of hierarchy. The details of the architectures under consideration and their respective techniques are presented in the sections that follow.

The remainder of the paper is organized as follows: Section 2 gives a brief overview of mesh- and tree-based architectures. Section 3 presents exploration environments and the floor-planning techniques that are explored using these environments. Section 4 describes the experimental flow. Section 5 presents experimental results, and Section 6 finally concludes the paper.

2. Reference FPGA Architectures

2.1. Mesh-Based Heterogenous 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, and so forth, 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

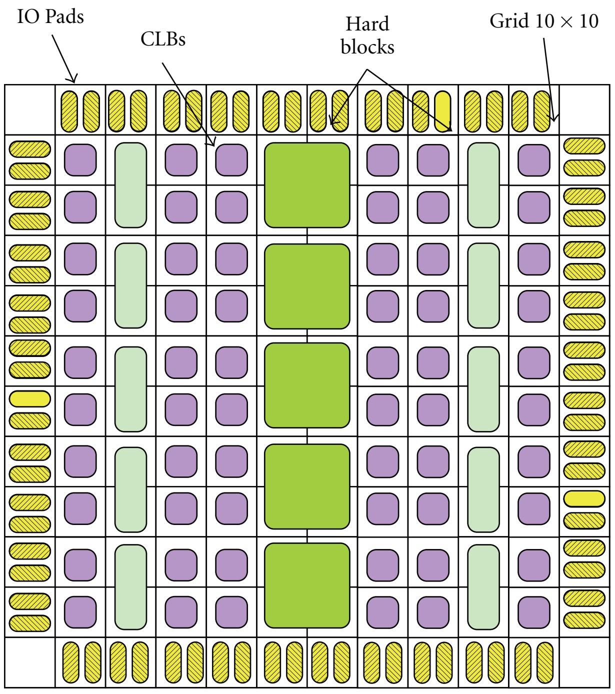


Figure 1: Mesh-based heterogeneous FPGA architecture.

(Versatile Place & Route) [14] 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 the size of the smallest block of the architecture, that is, CLB. The width and height of an HB are 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 10 × 10 is shown in Figure 1. In mesh-based FPGA, input and output pads are arranged at the periphery of the slot-grid as shown in Figure 1. 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 [15]. 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 [2].

An FPGA tile showing the detailed connection of a CLB with its neighboring routing network is shown in Figure 2. In this figure, 4 inputs of the CLB are connected to 4 adjacent routing channels. The output pin of the CLB is connected to the routing channel on its top and right through the diagonal connections of the switch box. The switch box uses unidirectional, disjoint topology to connect different routing tracks together. The connectivity of a routing track incident on a switch block with routing tracks of other routing channels that are incident on the same switch block, termed as switch block flexibility (Fs), is set to be 3. The connectivity of the routing channel with the input and output pins of a block, abbreviated as Fcin and Fcout, is set to be 1.

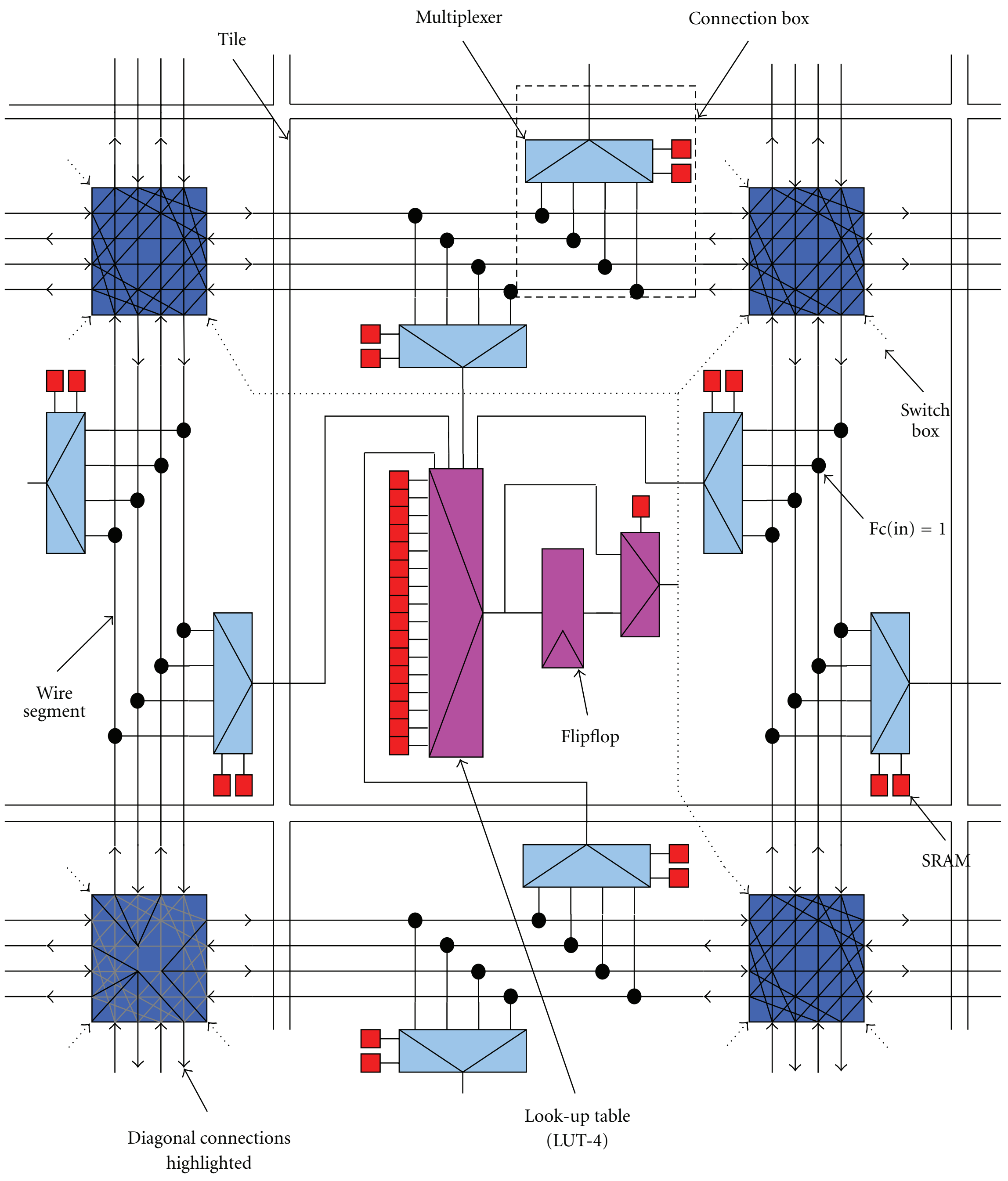


Figure 2: Detailed interconnect of a CLB with its neighboring channels.

The channel width is varied according to the netlist requirement but remains a multiple of 2 [15].

2.2. Tree-Based Heterogeneous FPGA Architecture. A tree-based architecture is a hierarchical architecture having unidirectional interconnect. Unlike mesh-based architecture where logic and routing resources are arranged in island-style, in a tree-based architecture, logic and routing resources are arranged in hierarchical manner. Tree-based architecture exploits the locality of connections that is inherent in most of the application designs. In this architecture, CLBs, I/Os, and HBs are partitioned into a multilevel clustered structure where each cluster contains subclusters and switch blocks allow to connect external signals to subclusters. Figure 3 shows generalized example of a heterogeneous tree-based architecture used in this work. In a heterogenous tree-based architecture, CLBs and I/Os are placed at the bottom of hierarchy whereas HBs can be placed at any level of hierarchy to meet the best design fit. For example, in Figure 3 HBs are placed at level 2 of hierarchy.

Tree-based architecture contains two unidirectional, single length, interconnect networks: a downward network and

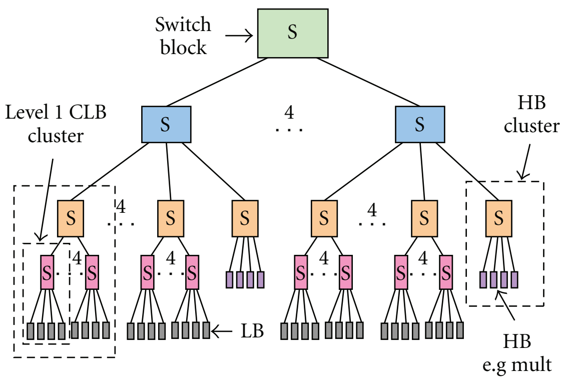


Figure 3: Tree-based heterogeneous FPGA architecture.

an upward network. downward network is based on butterfly fat tree topology and allows to connect signals coming from other clusters to its subclusters through a switch block. The upward network is based on hierarchy and it allows to connect sub-cluster outputs to other subclusters in the same cluster and to clusters in other levels of hierarchy. A level 1 cluster example of two interconnect networks, demonstrating the detailed connection of a CLB with its neighboring CLBs, is shown in Figure 4. It can be seen from the figure that switch blocks are further divided into downward and upward miniswitch boxes (DMSBs & UMSBs) where DMSBs are responsible for downward interconnect and UMSBs are responsible for upward interconnect. DMSBs and UMSBs are combined together to route different signals of the netlists that are mapped on the architecture. These DMSBs and UMSBs are unidirectional full cross bar switches that connect signals coming into the cluster to its subclusters and signals going out of a cluster to the other clusters of hierarchy. The number of DMSBs in a switch block of a cluster at level l are equal to the number of inputs of a cluster at level $l-1$ and the number of UMSBs in a cluster at level l are equal to the number of outputs of a cluster at level $l-1$. The number of signals entering into and leaving from the cluster can be varied depending upon the netlist requirement. However, they are kept uniform over all the clusters of a particular level. The signal bandwidth of clusters is controlled using Rent's rule [16] which is easily adapted to tree-based architecture. This rule states that

$$\mathrm{IO} = \left(\underbrace{k \cdot n^{l}}_{L\cdot B(p)} + \underbrace{\sum_{x=1}^{z} a_x \cdot b_x \cdot n^{(l-l_x)}}_{H\cdot B(p)} \right)^{p}, \qquad (1)$$

where

$$H \cdot B(p) = \left\{ \begin{matrix} 0 & \text{if } (l - l_x < 0) \\ a_x \cdot b_x \cdot n^{(l-l_x)} & \text{if } (l - l_x \geq 0) \end{matrix} \right\}. \qquad (2)$$

In (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, l_x is the level where HB is located, b_x is the number of HBs at the level where it is located, 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 $L \cdot B(p)$ part of (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.

2.3. Characteristics of Mesh-Based and Tree-Based Architectures. Both tree-based and mesh-based architectures have particular characteristics that are mainly dependant on the basic interconnect structure and the arrangement of different blocks in the architecture. For example, the major advantage of a tree-based heterogeneous FPGA is its predictable routing, timing behavior, and its independence of the types and position of blocks supported by the architecture. In a tree-based architecture, the number of paths required to reach a destination are limited and hence the number of switches crossed by a signal to reach a destination from a source do not vary greatly. It can be seen from Figure 3 that any CLB can reach any HB by traversing between four to six switches. Unlike tree-based FPGAs, routability of mesh-based FPGA is greatly dependent upon the position of different blocks on the architecture. In mesh-based FPGAs, routability is not predictable and the number of paths available to reach a destination is almost unlimited. Hence the number of switches crossed to reach a destination varies with respect to the position of blocks in the architecture. For example, any CLB in the leftmost column of Figure 1 crosses at least eight switches to reach an HB in the second last column of the architecture. However, this number of switches is reduced to only one if that CLB is placed beside the HB of the second last column of architecture. So, floor-planning plays a very important role in island-style heterogeneous FPGAs and this role becomes more important with the increase in types of HBs that are required to be supported by the architecture.

3. Exploration Environments

In this section, the exploration environments of two FPGA architectures are presented along with different floor-planning techniques that are explored using these exploration environments.

3.1. Exploration Environment of Mesh-Based Architecture. This work uses the mesh-based architecture exploration environment presented earlier in [17] which is further improved by implementing Range Limiter [18] and column-move operation for heterogeneous architectures. In this environment, an FPGA architecture is initially defined using an architecture description file. Architecture description file includes a certain number of parameters that are used for the exploration of the architecture. Some of these parameters include the size of slot-grid, the types of blocks supported by the architecture, the type of routing network (either unidirectional or bidirectional), initial channel width, parameters regarding the optimization of architecture, and the parameters regarding the position of different blocks on the

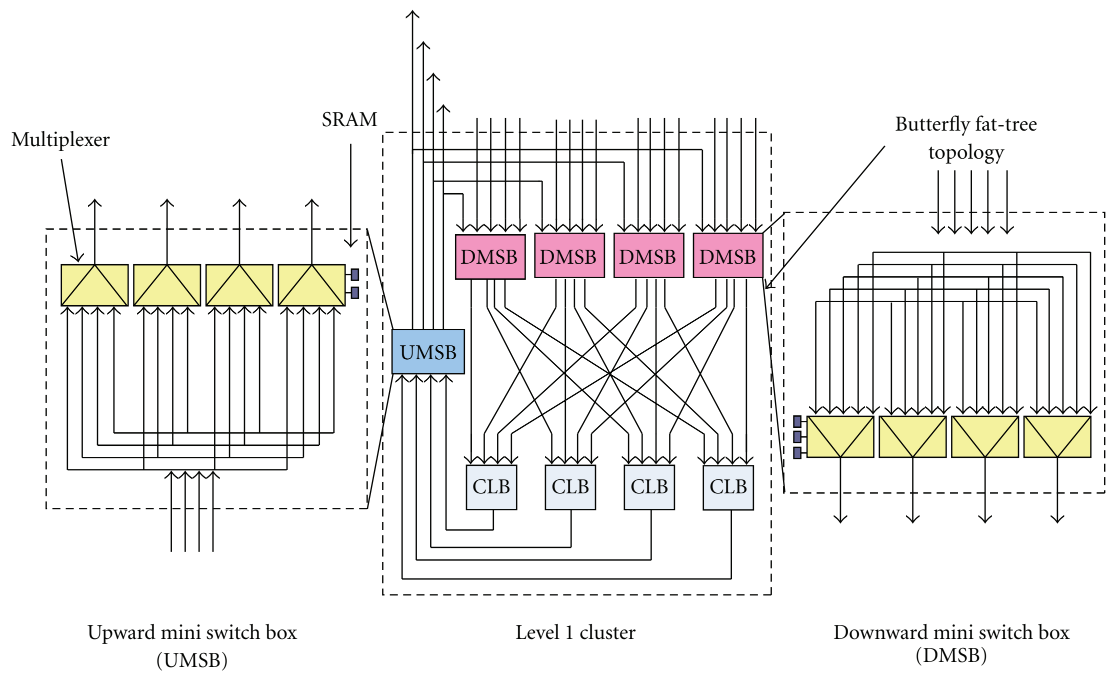

Multiplexer
SRAM
Butterfly fat-tree topology
DMSB
DMSB
DMSB
DMSB
UMSB
CLB
CLB
CLB
CLB
Upward mini switch box (UMSB)
Level 1 cluster
Downward mini switch box (DMSB)

FIGURE 4: Detailed interconnect of level 1 cluster of tree-based FPGA.

architecture. Blocks of different sizes are defined, and later mapped on a grid of equally sized slots, called a slot-grid. Each block occupies one or more slots. The type of the block and its input and output pins are used to find the size of a block. 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. Pins with the same class number are considered equivalent; thus a NET targeting a receiver pin of a block can be routed to any of the pins of the block belonging to the same class. Once the architecture of FPGA is defined, the benchmark circuit is placed on the architecture using a placer that performs a number of operations to explore different floor-planning techniques of the architecture. An overview of different operations performed by placer is given below.

3.1.1. PLACER Operations. For mesh-based architecture, a simulated annealing-based [19, 20] PLACER is used to place connected instances near to each other so that placement cost of the architecture is minimized and minimum routing resources are required to connect the instances that communicate with each other. In order to minimize the routing resource and optimize the placement solution, PLACER performs a number of operations that are summarized below.

(i) Moving an instance from one block to another.

(ii) Moving a block from one slot to another.

(iii) Rotating a block around its own axis.

(iv) Moving a complete column of blocks from one slot position to another.

After each operation, the placement cost is recomputed for all the disturbed nets. Depending on the cost value and the annealing temperature, the operation is accepted or rejected. Multiple netlists can be placed together to get a single architecture floor-planning for all the netlists. For multiple netlist placement, each block allows mapping of multiple instances on itself, but multiple instances of the same netlist cannot be mapped on a single block.

PLACER performs its move and rotate operations on a “source” and a “destination”. The “source” is randomly selected to be either an instance from input netlist or a block from the architecture. If the “source” is an instance to be moved, any random matching block is selected as its “destination”. If the “source” is a block, then a slot position is selected as its “destination”. If a “source” block is to be rotated, the same source slot position becomes the “destination”. If the “source” block is to be moved, then any random slot is selected as its “destination”. The rectangular window starting from this destination slot and having same size and shape as that of source is called destination window whereas the window occupied by the source is called source window. Normally, source window contains one block whereas destination window can contain multiple blocks. An example of source and destination windows is shown in Figures 5(a) and 5(b), respectively. Once the source and destination windows are selected, the move operation is performed if

(i) destination window does not contain any block that exceeds the boundary of destination window. An example violating this condition is shown in Figure 5(c),

(ii) the destination window does not exceed the boundaries of slot-grid (Figure 5(d)),

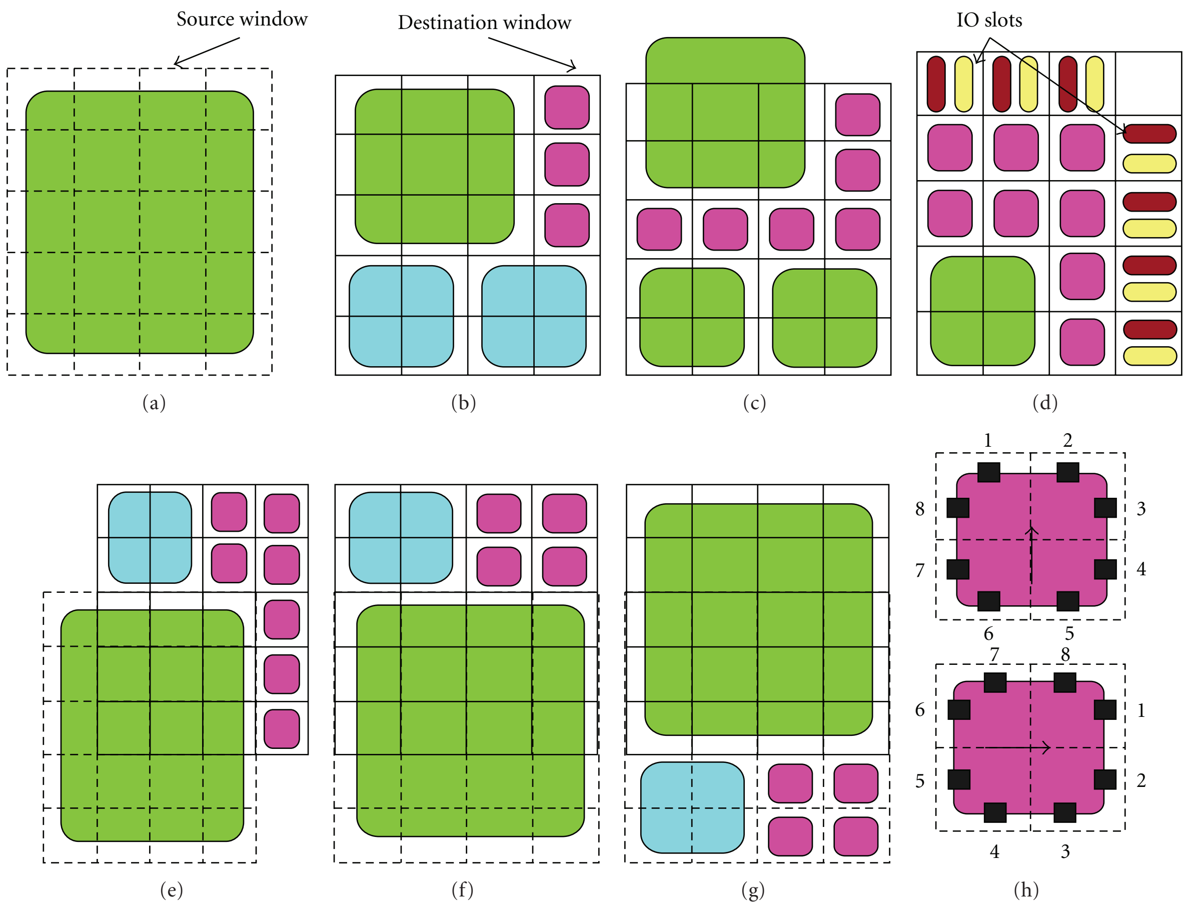


Figure 5: Placer operations.

(iii) destination window does not overlap source window diagonally (Figure 5(e)). However, if, the destination window overlaps source window vertically or horizontally, then horizontal or vertical translation operation is performed. Figure 5(f) shows an example where destination window overlaps source window vertically, and Figure 5(g) shows that the move operation is performed using vertical translation.

However, if the above three conditions are not met, the procedure continues until a valid destination window is found. After the selection of source and destination, any one of the following operations is performed by the placer.

(i) Instance Move: In this case, a move operation is applied on the source instance and the destination block. If the destination block is empty, the source instance is simply moved to the destination block. If the destination block is occupied by an instance, then an instance swap operation is performed.

(ii) Block Jump: If the source window does not overlap with the destination window, then a JUMP operation is performed. All the blocks in the destination window are moved to the source window, and the source block is moved in the destination window. The instances mapped on a block also move along with the block.

(iii) Block Translate: If the source and the destination windows overlap, then a translation operation is performed. Only horizontal and vertical translations are currently performed; diagonal translation is not performed in this work.

(iv) Block Rotate: 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°. The orientation of a block is used by the bounding box (the minimum rectangle containing the source instance and all its destination instances) evaluation function to correctly calculate the exact position and direction of each of its pins. When an instance of a netlist is moved from one block to another block having different orientations, the orientation of both the old block and the new block are used to compute the difference in the bounding box. Figure 5(h) 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 (nonsquare) blocks. A 90° rotation for nonsquare blocks involves both rotation and move operations, which is left for future work.

(v) Column Move: The column move operation moves a complete column of blocks from one slot position to another. If the source block is restricted to remain in a column, a column move operation is performed.

3.1.2. Floor-Planning Techniques. By using different PLACER operations, six floor-planning techniques are explored. The details of these floor planning techniques are as follows.

(i) Apart: In this technique, hard blocks are placed in fixed columns, apart from the CLBs. This technique is shown in Figure 6(a) and is termed as Apart (A). Such kind of technique can be beneficial for data-path circuits as described by [21]. 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 the column.

(ii) Column-Partial: Figure 6(b) shows the Column-Partial (CP) technique where columns of HBs are evenly distributed among columns of CLBs.

(iii) Column-Full: Figure 6(c) shows Column-Full (CF) technique where columns of HBs are evenly distributed among CLBs. Contrary to the first and second techniques, the whole column contains only one type of blocks. This technique is normally used in commercial architectures.

(iv) Column-Move: In this technique, HBs are placed in columns but unlike the 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 Figure 6(d) and it is termed as Column-Move (CM).

(v) 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 Figure 6(e).

(vi) 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 Figure 6(f) and it is termed as Block-Move-Rotate (BMR).

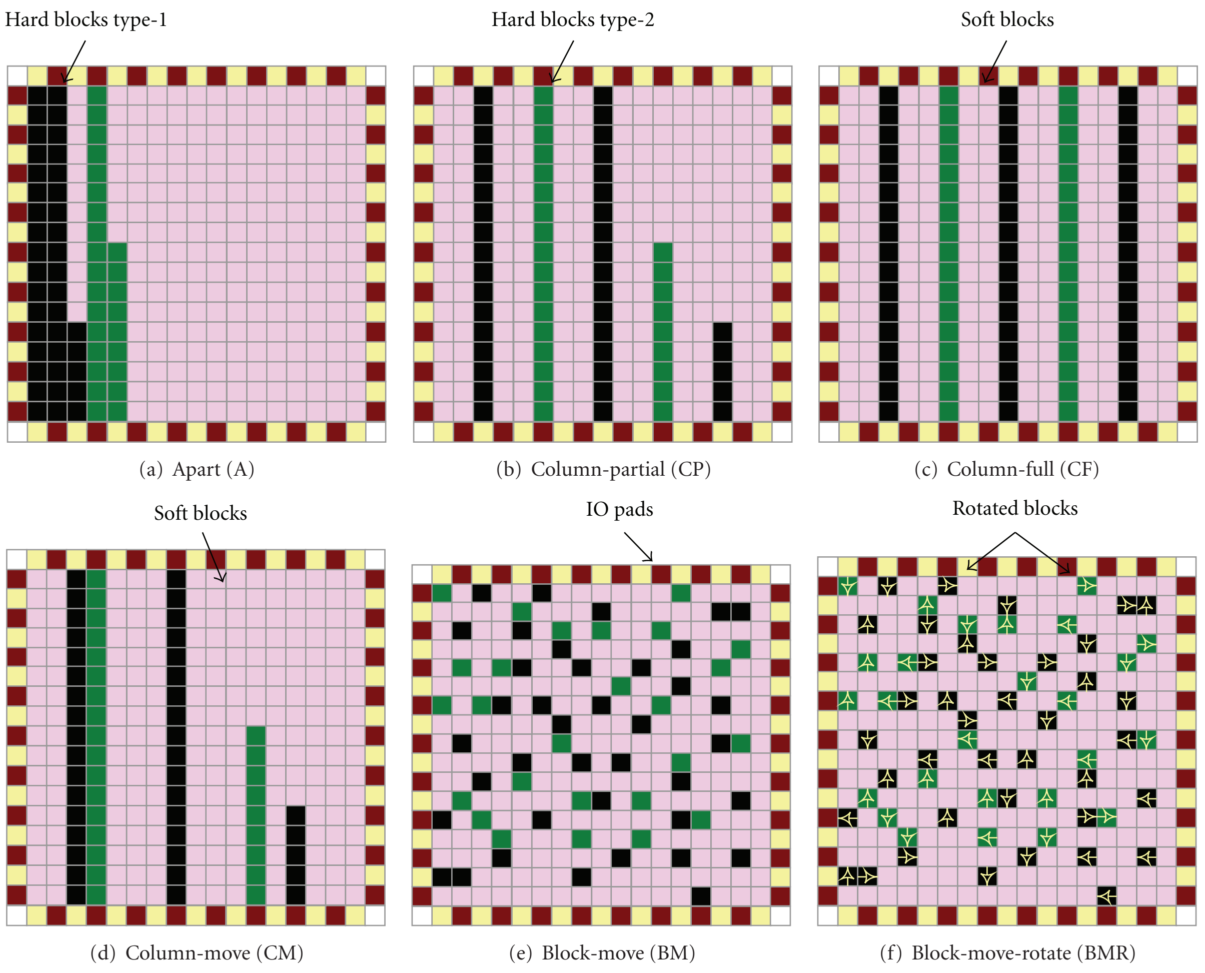
3.2. Exploration Environment of Tree-Based Architecture. A tree-based architecture is defined using an architecture description file. The architecture description file contains different architectural parameters along with the definition of different blocks used by the architecture. Some of these architecture parameters include the number of levels in the architecture, the types of blocks supported by the architecture, initial signal bandwidths of different clusters situated at different levels of hierarchy, the arity of different clusters of the architecture and so forth. In a tree-based architecture, the definition of a block (CLB or HB) includes the type, area, the level where it is located, and the class numbers for each of its input and output pins. Similar to mesh-based architecture, in this architecture, pins with the same class number are considered equivalent.

Once the architecture is defined, a Fidducia-Mattheyses-(FM) [22] based PARTITIONER partitions the netlist using a top-down recursive partitioning approach. The main objective of PARTITIONER is to reduce communication between different partitions (clusters), and FM algorithm achieves this objective using a hill-climbing, nongreedy, iterative improvement approach. During each iteration, a block with the highest gain is moved from one partition to another and then it is locked and it is not allowed to move during the 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.

3.2.1. Exploration Techniques. In order to explore the architecture, we have used two exploration techniques.

(i) Symmetric: A generalized example of first technique is shown in Figure 7. This technique is referred to as symmetric (SYM). In this technique, HBs can be placed at any level of hierarchy in order to have best design fit. However, 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 Figure 7, it can be seen that this architecture supports 4 clusters of HBs of a certain type where each cluster contains 4 HBs. This is because of the fact that this is an arity 4 architecture. However, the respect for the symmetry of hierarchy may lead to underutilization of HBs and their associated routing resources in the case where a netlist requires less HBs than supported by the architecture.

(ii) Asymmetric: Contrary to the first technique, where the architecture contains only one structure, the second technique contains two substructures: one substructure contains only CLBs while the other contains only HBs. An example of the second technique is shown in Figure 8. The main motivation behind this technique is the easy management of logic and routing resources. Also the substructure containing only HBs does not have to respect the arity of the substructure containing only CLBs, hence leading to more optimized logic and routing resources. This technique is referred to as asymmetric (ASYM).



(a) Apart (A) (b) Column-partial (CP) (c) Column-full (CF) (d) Column-move (CM) (e) Block-move (BM) (f) Block-move-rotate (BMR)

Figure 6: Floor-planning techniques.

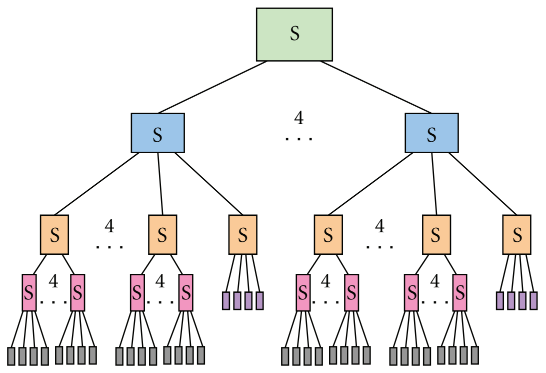


Figure 7: Symmetric tree-based heterogeneous FPGA architecture.

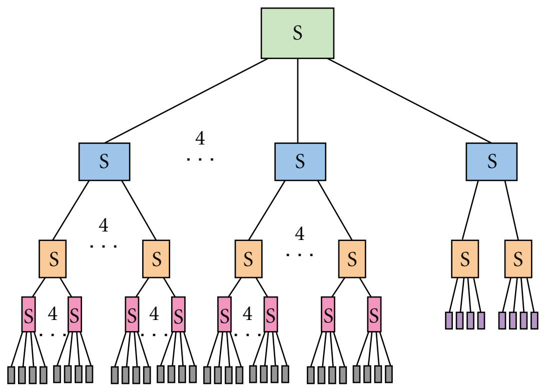


Figure 8: Asymmetric tree-based heterogeneous FPGA architecture. (a) substructure containing only CLBs. (b) substructure containing only HBs.

4. Experimental Flow

Evaluation of different floor-planning/exploration techniques of the two architectures is performed using a specifically designed experimental flow. Details of this experimental flow are explained in this section.

4.1. 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

Table 1: DSP benchmarks set I.

Circuit name	Inputs	Outputs	CLBs (LUT4)	Mult (8×8)	Slansky $(16 + 16)$	Sff (8)	Sub $(8 - 8)$	Smux (32 : 16)
ADAC	18	16	47	—	—	2	—	1
DCU	35	16	34	1	1	4	2	2
FIR	9	16	32	4	3	4	—	—
FFT	48	64	94	4	3	—	6	—

Table 2: Open core benchmarks set II.

Circuit name	No of inputs	No of outputs	No of LUTs	No of multipliers (16×16)	No of adders $(20 + 20)$
cf_fir_3_8_8	42	18	159	4	3
cf_fir_7_16_16	146	35	638	8	14
cfft 16 × 8	20	40	1511	—	26
cordic_p2r	18	32	803	—	43
cordi_r2p	34	40	1328	—	52
fm	9	12	1308	1	19
fm_receiver	10	12	910	1	20
lms	18	16	940	10	11
reed_solomon	138	128	537	16	16

Table 3: Open core benchmarks set III.

Circuit name	No of inputs	No of outputs	No of LUTs	No of multipliers (18×18)
cf_fir_3_8_8	42	22	214	4
diffeq_f_system C	66	99	1532	4
diffeq_paj_convert	12	101	738	5
fir_scu	10	27	1366	17
iir1	33	30	632	5
iir	28	15	392	5
rs_decoder_1	13	20	1553	13
rs_decoder_2	21	20	2960	9

the selection of benchmark circuits, as different circuits can give different results for different architecture floor-planning 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 interblock communication. These benchmarks are shown in Tables 1, 2, and 3 and they are obtained from [23–25], respectively. The communication between different blocks of a benchmark can be mainly divided into the following four categories.

(i) CLB-CLB: CLBs communicate with CLBs.

(ii) CLB-HB: CLBs communicate with HBs and vice versa.

(iii) HB-HB: HBs communicate with other HBs.

(iv) IO-CLB/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. Similarly, in SET II the major percentage of total communication 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.

4.2. Software Flow. The software flow used to place and route different benchmarks (netlists) on the two heterogeneous FPGAs is shown in Figure 9. The input to the software flow is a VST file (structured vhdl). This file is converted into BLIF format [26] using a modified version of VST2BLIF tool. The BLIF file is then passed through PARSER-1 which removes HBs from the file and adds temporary inputs and outputs to the file to preserve the dependance between HBs and the rest of the netlist. The output of PARSER-1 is then passed through SIS [27] that synthesizes the BLIF file into LUT format which is later passed through T-VPACK [28] which packs and converts it into. NET format. Finally, the netlist is passed through PARSER-2 that adds previously removed HBs and also removes temporary inputs and outputs. The final netlist in. NET format contains CLBs, HBs, and I/O instances that are connected to each other via NETS. After obtaining the netlist in. NET format, floor-plannings of Section 3 are explored separately for both tree-based and mesh-based architectures using their respective flow.

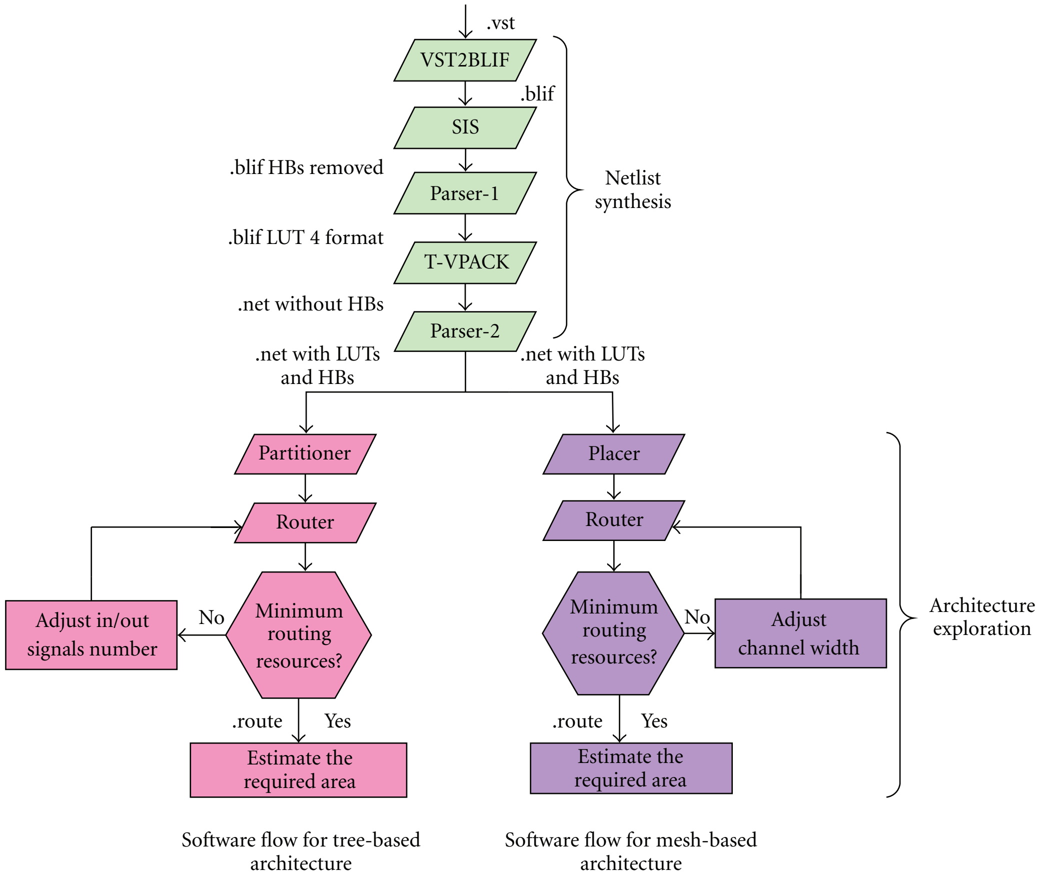


Figure 9: Software flow.

4.2.1. Software Flow for Tree-Based Architecture. For tree-based architecture, the netlist obtained in. NET format is first partitioned using a software module called PARTITIONER. This module partitions CLBs, HBs, and I/Os into different clusters in such a way that the intercluster communication is minimized. By minimizing intercluster communication we obtain a depopulated global interconnect network which leads to smaller intercluster signal bandwidths and eventually gives good tradeoff both in terms of delay and area. PARTITIONER is based on hMetis [29] platform. hMetis combines FM algorithm with its multiphase refinement approach to optimize the partitioning of the netlist. These phases include coarsening, initial partitioning, uncoarsening, and refinement phase [30]. Once partitioning is done, a 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 the routing of netlist. In order to route all the NETS of 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 $G(V, E)$ where 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 [31] 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. Once the optimization is over, area of the architecture is estimated using an area model which is based on symbolic standard cell library SXLIB [32].

4.2.2. Software Flow for Mesh-Based Architecture. For mesh-based architecture, the netlist file is passed to a software module called PLACER that uses simulated annealing algorithm [18, 19] to place CLBs, HBs, and I/Os on their respective blocks in FPGA. The main objective of the PLACER is to place connected instances close to each other so that minimum routing resources are required to route their connection. For this purpose, PLACER optimizes the placement cost of the architecture which is equal to the sum of half-perimeters of the bounding boxes of all NETS. The bounding box (BBX) of a NET is a minimum rectangle that contains the driver instance and all receiving instances of a NET. PLACER moves an instance randomly from one block position to another; the BBX cost is updated. Depending on cost value and annealing temperature, the operation is accepted or rejected. After placement, a software module named ROUTER routes the netlist on the architecture.

Table 4: Area of different blocks of three sets.

Block name	Inputs	Outputs	Block size (λ^2)
clb	4	1	58500
mult (8×8)	16	16	1075250
slansky_16	32	16	306750
sff_8	8	8	36000
sub_8	17	8	154500
smux_16	33	16	36000
mult (16×16)	32	32	1974000
adder ($20 + 20$)	41	21	207000
mult (18×18)	36	36	2498300
sram	—	—	1500
buffer	1	1	1000
flip-flop	1	1	4500
mux 2 : 1	2	1	1750

Similar to the ROUTER of tree-based FPGA, mesh-based FPGA uses a pathfinder algorithm [31] to route the netlist using FPGA routing resources. In order to optimize the FPGA resources, a binary search algorithm similar to the one used for tree-based FPGA is used to determine the smallest channel width required to route a netlist.

4.2.3. Architecture Evaluation. Once the netlist routing is completed, the area, performance, and power estimation of architecture is performed (separately for mesh- and tree-based architectures). The area of FPGA architecture is estimated by combining areas of CLBs, HBs, multiplexors of interconnect, and all associated programming bits. Our area model is based on standard cell library SXLIB [32] and the area of different cells that are used for the calculation of area is shown in Table 4. Since we do not have accurate wire length estimation, performance evaluation of the architecture is performed by counting the number of switches that are crossed by critical path and static power estimation is performed by combining the number of configuration memories and buffers.

4.3. Experimental Methodology. In order to have a detailed analysis of different techniques, we have employed two different methodologies for the experimentation: individual experimentation and generalized experimentation.

4.3.1. Individual Experimentation. In the first methodology, experiments are performed individually for each netlist (both for mesh- and tree-based architectures). The architecture definition, floor-planning, placement, 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, 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 the 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 get more profound results, the architecture and floor-planning is optimized individually for each netlist; later average of all netlists gives more thorough results.

4.3.2. 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 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 nelists 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. Once generalized floor-planning optimization/partitioning is over, individual netlists are placed and routed separately on both architectures using the above described flow except that optimization of the architecture is not performed for individual netlists.

5. Experimental Results

5.1. Experimental Results Using Individual Methodology. Since placement cost and channel width of mesh-based architecture is directly related to its area, we first present the effect of different floor-planning techniques of mesh-based architecture on these two values. Placement cost and channel width results obtained for three sets of benchmarks are shown in Figures 10 and 11, respectively. In these figures, the results for benchmarks 1 to 4, 5 to 13, and 14 to 21 correspond to SET I, SET II, and SET III, respectively. The avg1, avg2, and avg3 in the Figures 10 and 11 correspond to the geometric average of these results for SET I, SET II, and SET III, respectively. The avg corresponds to the average of all netlists.

As explained earlier, placement cost is the sum of the bounding box cost of all the nets of the netlist being implemented on the architecture and this cost gives us a measure of the quality of the placement solution that is provided by a certain floor-planning technique. In Figure 10, for each of the 21 benchmark circuits, the placement cost given by five floor-planning techniques (i.e., Apart (A), Column-Partial (CP), Column-Full (CF), Column-Move (CM), and Block-Move (BM)) is normalized against the placement cost of Block-Move-Rotate (BMR). 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

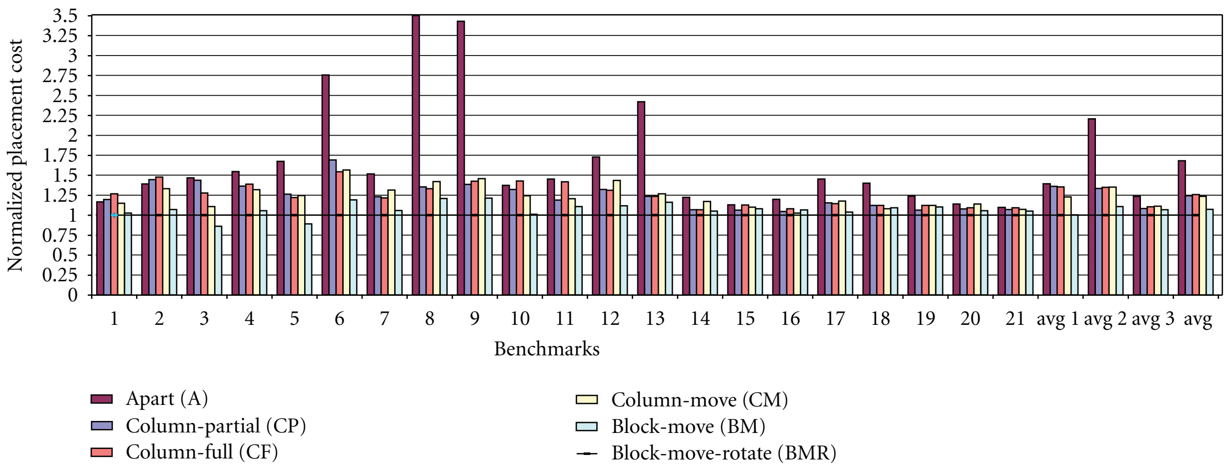


Figure 10: Placement cost comparison between different techniques of mesh-based architecture.

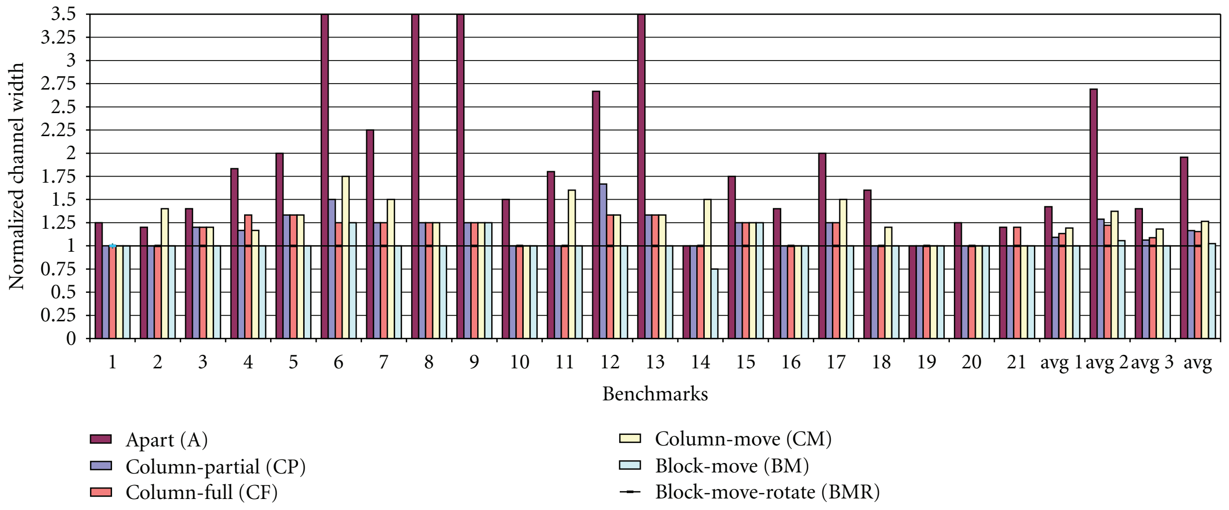


Figure 11: Channel width comparison between different techniques of mesh-based architecture.

can give good results for datapath 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 type of HBs that are required to be supported by the architecture. Although the 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 slightly more flexibility to BMR which might eventually lead to smaller BBX and eventually lower placement costs of the architecture.

Figure 11 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 Figure 10, BMR gives the best results and Apart gives the worst results. The two figures (i.e., Figures 10 and 11) look similar to each other as (i) both figures give normalized results and (ii) 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 Figures 10 and 11 shows that, on average, CF gives 35%, 35%, and 11% more placement cost than BMR, for SET I, SET II, and SET III benchmark circuits, respectively. Figure 11 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 increase in channel width increases the overall area of the architecture, as shown in Figure 12. In this figure, the area results of A, CF, BM floor-planning techniques of

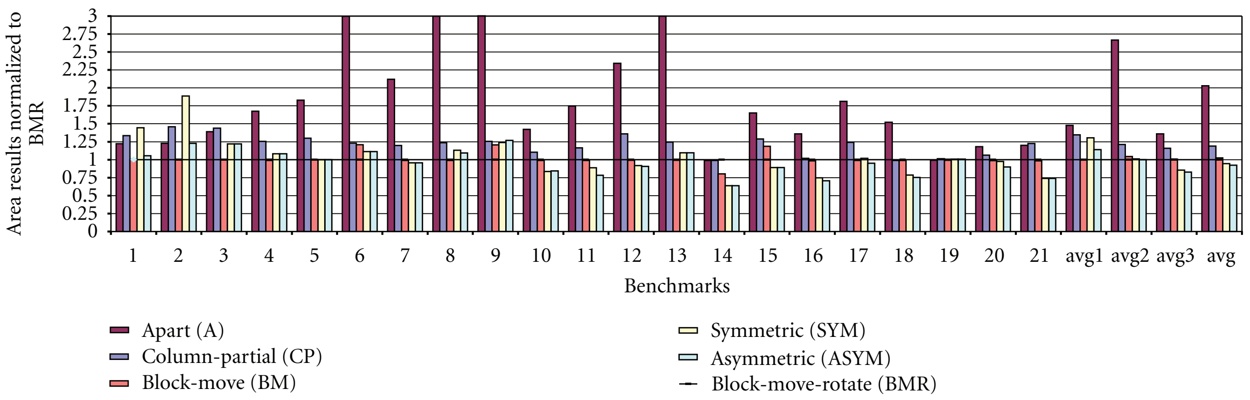


Figure 12: Area comparison between different techniques of mesh- and tree-based architectures.

mesh-based FPGA and SYM and ASYM techniques of tree-based FPGA are normalized against the area results of BMR floor-planning technique of mesh-based FPGA. For the sake of clarity, the results for CP and CM floor-planning techniques are not presented. On average, CF requires 36%, 23%, and 10% more area than BMR for SET I, SET II, and SET III, respectively. For SET I benchmark circuits, SYM requires 35% more area than BMR, and ASYM requires 10% 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.

Analysis of the results shown in Figure 12 reveals that area of CF compared to BMR varies 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 as HBs can always be placed close to each other. For the other two set 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, 23% and 10% 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 reduces the quality of placement solution which leads to the demand for more routing resources to route the netlist; thus more area is required. The results show that BMR technique produces the least placement cost, the smallest channel width, and hence the smallest area for mesh-based heterogeneous FPGA. However, BMR floor-planning technique is dependent 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 the best results in terms of area and it is better than the best technique of mesh-based FPGA (i.e., BMR) by an average of 5% for a total of 21 benchmarks. The major advantage of a heterogeneous tree-based FPGA is that the maximum number of switches required to route a connection between CLB-HB or HB-HB remain relatively constant. However, in case of SET I benchmarks, extensive HB-HB communication gives rise to total switch requirement of tree-based architecture; hence giving poor area results as compared to mesh-based architecture. The architecture floor-planning of tree-based FPGA does not affect the switch requirement of the architecture. However, the floor-planning of mesh-based FPGA causes a drastic impact on the switching requirement of the architecture.

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- and tree-based architectures, it will 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 Figure 13. To avoid congestion, results for only 6 out of 8 techniques are shown. It can be seen from Figure 13 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 benchmarks, 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 the 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 a smaller number of switches than CF. For SET II benchmarks this percentage drops to 44% as there is more

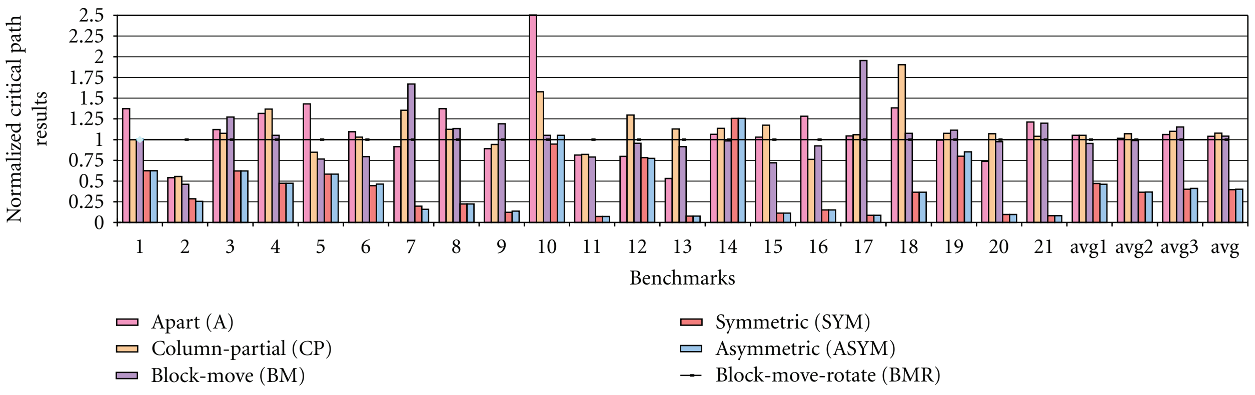


Figure 13: Critical path comparison between different techniques of mesh- and tree-based architectures.

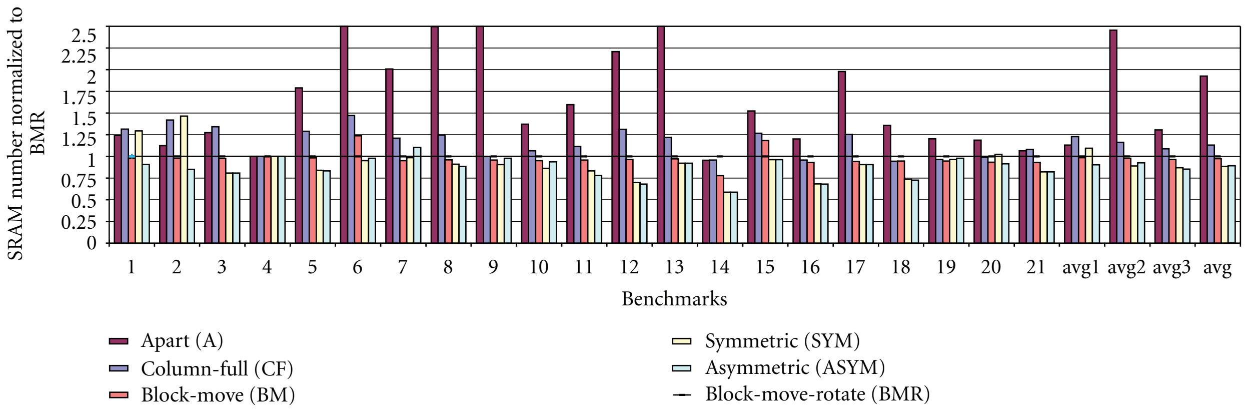


Figure 14: Configuration memory comparison between different techniques of mesh- and tree-based architectures.

communication between CLBs and HBs. However, in case of SET III benchmarks, 75% of benchmarks cross a smaller 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 (see Section 2.3). 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 the number of switches crossed by critical path (Figures 7 and 8). However, if critical path involves no HBs or only HBs and I/Os, it can lead to a smaller number of switches than SYM technique (Figure 13 results for benchmark 2 and 7).

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 [33]. Static power of the FPGAs is directly related to the configuration memory and the number of buffers in an FPGA architecture [34]. Therefore, a comparison of configuration memory and number of buffers for different techniques of the two architectures is shown in Figures 14 and 15, respectively.

Figure 14 shows the 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 Figure 15 shows that, compared to BMR, CF consumes 9%, 22%, and 18%

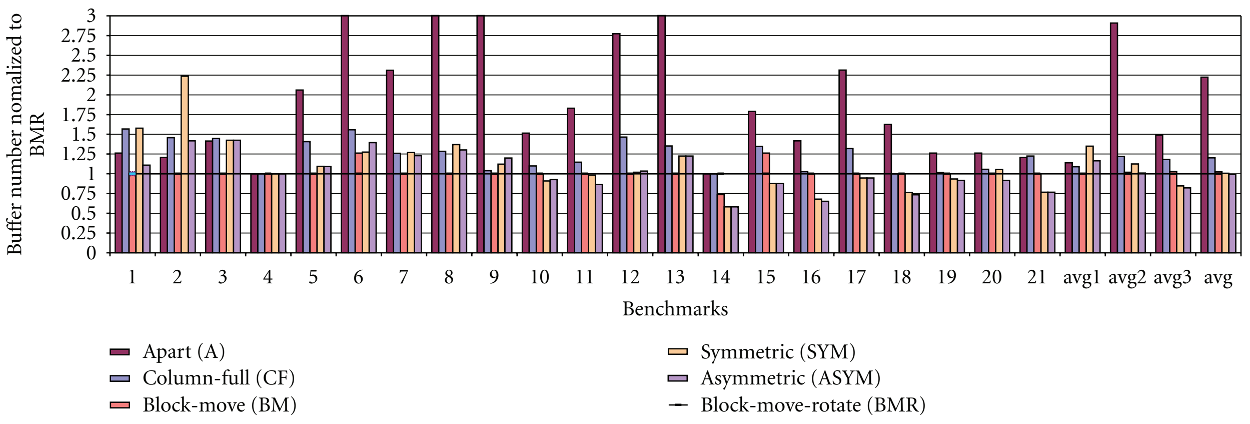


Figure 15: Buffer comparison between different techniques of mesh- and tree-based architectures.

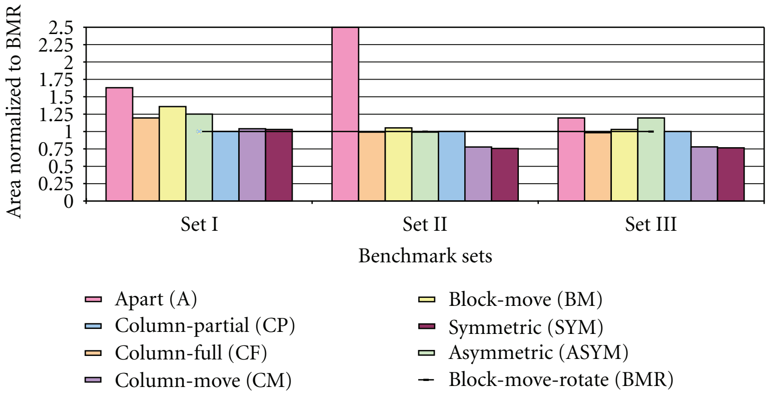


Figure 16: Generalized area results of mesh- and tree-based architectures.

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 Figures 14 and 15 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 [35], it closely correlates to the average area results of the two architectures presented in Figure 12 (avg1, avg2, and avg3 of Figure 12).

5.2. Experimental Results Using Generalized Methodology. Figure 16 shows the area results obtained using generalized experimental methodology (Section 4.3.2). In this methodology, a generalized architecture is defined for each SET of netlists that can place and route all netlists of that particular SET. It can be seen from the figure that this methodology further enhances the results obtained by the first experimental methodology. In this methodology too, compared to other floor-planning techniques of mesh-based FPGA, BMR produces equal or better results. However, the gain of BMR compared to CF is reduced from 23%, 10% to 5%, 3% for SET II and SET III 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 larger netlists. As far as the comparison of BMR with SYM and ASYM techniques 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% and 18% to 22% and 24%, respectively. Figures 17, 18, and 19 show the generalized critical path, configuration memory and buffers results that are obtained using generalized experimental methodology. The results shown in these figures further reinforce the observations made using individual experimental methodology. Although in our case the results of individual and generalized experimental methodologies comply with each other, there can be cases where the two methodologies contradict each other. So, in order to have a profound insight about different exploration techniques, it is good to perform both kinds of experimentation.

6. Conclusion

This paper has explored two heterogeneous FPGA architectures. Different mesh-based floor-planning techniques are compared. The floor-plannings of mesh-based FPGA

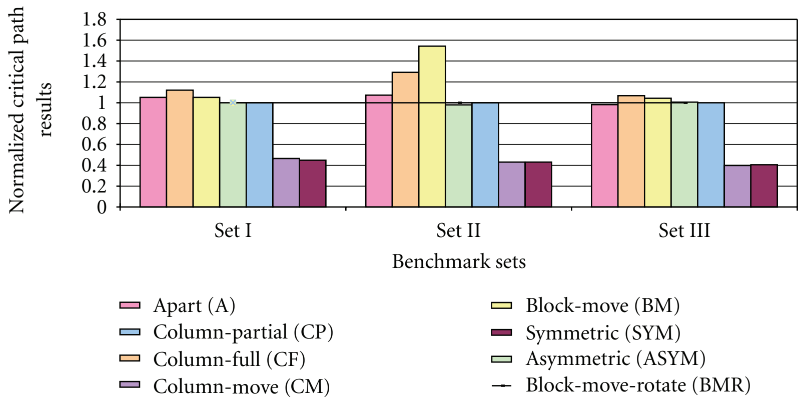


Figure 17: Generalized critical path results of mesh- and tree-based architectures.

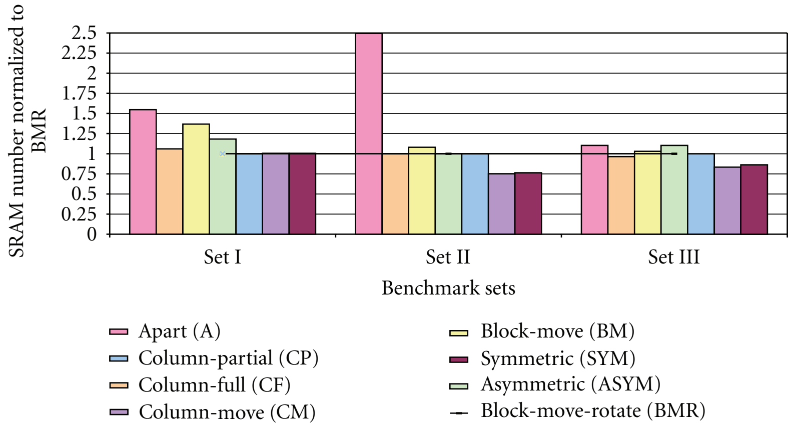


Figure 18: Generalized configuration memory results of mesh- and tree-based architectures.

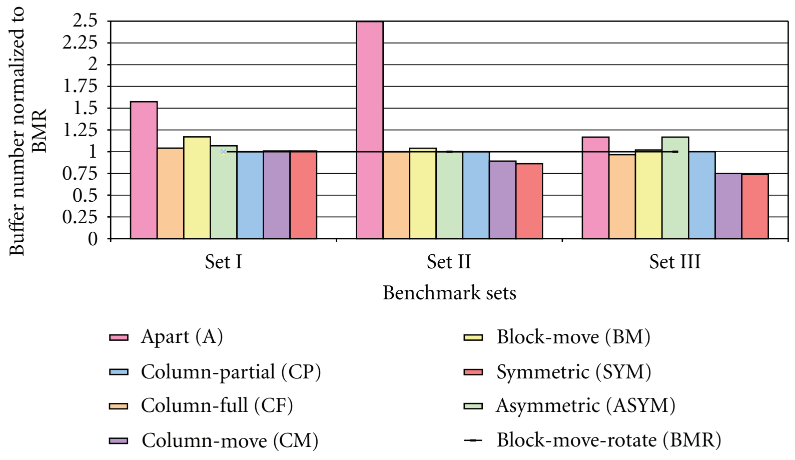


Figure 19: Generalized buffer comparison results of mesh- and tree-based architectures.

influence the routing network requirement of the architecture. Individual experimentation results show that CF floor-planning is on average 36%, 23%, and 10% more area consuming than the BMR floor-planning for three sets of netlists. The tree-based architecture is independent of its floor-planning; however, its layout is relatively less scalable than the mesh architecture. The CF floor-planning is on average 18%, 21%, and 40% larger than the ASYM technique of tree-based FPGA architecture for the same sets of netlists. Performance evaluation results show that BMR technique of mesh-based architecture crosses 4.7%, 6.5%, and 9% less switches and ASYM technique of tree-based architecture crosses 56%, 66%, and 63% less switches than CF technique of mesh-based FPGA. Static power estimation results show that, for three sets, BMR technique consumes 18%, 13% and 8% less SRAMs and 8%, 18%, and 15% less buffers than CF floor-planning of mesh-based FPGA. Similarly, ASYM consumes 26%, 20%, and 22% less SRAMs and 3%, 20%, and 30% less buffers than CF floor-planning of mesh-based FPGA. Among all the techniques compared, ASYM technique of tree-based FPGA gives the best area, performance and power estimation results for all three benchmark sets. However, hardware layout efforts are required to maintain these area benefits on tree-based FPGAs. A mesh containing smaller trees architecture can be designed to resolve scalability issues of tree architecture.

References

[1] S. Wilton, *Architectures and algorithms for field-programmable gate arrays with embedded memory*, Ph.D. dissertation, Citeseer, 1997.

[2] M. J. Beauchamp, S. Hauck, K. D. Underwood, and K. S. Hemmert, "Embedded floating-point units in FPGAs," in *Proceedings of the 14th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '06)*, pp. 12–20, February 2006.

[3] C. H. Ho, P. H. W. Leong, W. Luk, S. J. E. Wilton, and S. Lopez-Buedo, "Virtual embedded blocks: a methodology for evaluating embedded elements in FPGAs," in *Proceedings of the 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '06)*, pp. 35–44, April 2006.

[4] G. Govindu, S. Choi, V. Prasanna, V. Daga, S. Gangadharpalli, and V. Sridhar, "A high-performance and energy-efficient architecture for floating-point based LU decomposition on FPGAs," in *Proceedings of the18th International Parallel and Distributed Processing Symposium (IPDPS '04)*, pp. 2035–2042, April 2004.

[5] K. D. Underwood and K. S. Hemmert, "Closing the gap: CPU and FPGA trends in sustainable floating-point BLAS performance," in *Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '04)*, pp. 219–228, April 2004.

[6] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," in *Proceedings of the 14th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '06)*, pp. 21–30, ACM, Monterey, Calif, USA, February 2006.

[7] Xilinx, "Xilinx," http://www.xilinx.com, 2010.

[8] Altera, "Altera," http://www.altera.com, 2010.

[9] C. Ebeling, D. Cronquist, and P. Franklin, "RaPiDReconfigurable pipelined datapath," *Field-Programmable Logic Smart Applications, New Paradigms and Compilers*, pp. 126–135, 1996.

[10] S. C. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Matt, and R. R. Taylor, "PipeRench: a reconfigurable architecture and compiler," *Computer*, vol. 33, no. 4, pp. 70–77, 2000.

[11] A. Abnous and J. Rabaey, "Ultra-low-power domain-specific multimedia processors," in *Proceedings of the 9th IEEE Workshop on VLSI Signal Processing*, pp. 461–470, November 1996.

[12] Z. Marrakchi, H. Mrabet, E. Amouri, and H. Mehrez, "Efficient tree topology for FPGA interconnect network," in *Proceedings of the 18th ACM Great Lakes Symposium on VLSI (GLSVLSI '08)*, pp. 321–326, March 2008.

[13] U. Farooq, H. Parvez, Z. Marrakchi, and H. Mehrez, "A new tree-based coarse-grained FPGA architecture," in *Proceedings of the 5th International Conference on Ph.D. Research in Microelectronics and Electronics (PRIME '09)*, pp. 48–51, July 2009.

[14] J. Luu, I. Kuon, P. Jamieson et al., "VPR 5.0: FPGA CAD and architecture exploration tools with single-driver routing, heterogeneity and process scaling," in *Proceedings of the 7th ACM SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '09)*, pp. 133–142, ACM, February 2009.

[15] G. Lemieux, E. Lee, M. Tom, and A. Yu, "Directional and single-driver wires in FPGA interconnect," in *Proceedings of the IEEE International Conference on Field-Programmable Technology (FPT '04)*, pp. 41–48, December 2004.

[16] B. Landman and R. Russo, "On a pin versus block relationship for partitions of logic graphs," *IEEE Transactions on Computers*, vol. 20, no. 12, pp. 1469–1479, 1971.

[17] H. Parvez, Z. Marrakchi, U. Farooq, and H. Mehrez, "A new coarse-grained FPGA architecture exploration environment," in *Proceedings of the International Conference on Field-Programmable Technology (ICFPT '08)*, pp. 285–288, December 2008.

[18] V. Betz and J. Rose, "VPR: a new packing placement and routing tool for FPGA research," in *Proceedings of the 2nd International Workshop on Field-Programmable Gate Arrays (FPGA '97)*, pp. 213–222, IEEE Press, 1997.

[19] C. C. Skiścim and B. L. Golden, "Optimization by simulated annealing: a preliminary computational study for the tsp," in *Proceedings of the 15th Winter Simulation Conference (WSC '83)*, pp. 523–535, IEEE Press, Piscataway, NJ, USA, 1983.

[20] C. Sechen and A. Sangiovanni-Vincentelli, "The timberwolf placement and routing package," *IEEE Journal of Solid-State Circuits*, vol. 20, no. 2, pp. 510–522, 1985.

[21] D. Cherepacha and D. Lewis, "DP-FPGA: an FPGA architecture optimized for datapaths," *VLSI Design*, vol. 4, no. 4, pp. 329–343, 1996.

[22] C. M. Fiduccia and R. M. Mattheyeses, "A linear-time heuristic for improving network partitions," in *Proceedings of the 19th ACM/IEEE Design Automation Conference*, pp. 175–181, 1982.

[23] http://www-asim.lip6.fr.

[24] http://www.opencores.org/.

[25] http://www.eecg.utoronto.ca/vpr/.

[26] "Berkeley logic Synthesis and Verification Group,University of California, Berkeley. Berkeley Logic Interchange Format (blif)," http://vlsi.colorado.edu/vis/blif.ps.

[27] E. M. Sentovich, K. J. Singh, and L. Lavagno, "Sis: a system for sequential circuit analysis," Tech. Rep. UCB/ERL M92/41, University of California, Berkeley, Calif, USA, 1992.

[28] A. Marquardt, V. Betz, and J. Rose, "Using cluster-based logic blocks and timing-driven packing to improve FPGA speed and density," in *Proceedings of the ACM/SIGDA 7th International Symposium on Field Programmable Gate Arrays (FPGA '99)*, pp. 39–46, Monterey, Calif, USA, February 1999.

[29] G. Karypis and V. Kumar, "Multilevel k-way hypergraph partitioning," in *Proceedings of the 36th Annual Design Automation Conference (DAC '99)*, pp. 343–348, June 1999.

[30] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: application in VLSI domain," in *Proceedings of the 34th Design Automation Conference*, pp. 526–529, ACM, June 1997.

[31] L. McMurchie and C. Ebeling, "PathFinder: a negotiation-based performance-driven router for FPGAs," in *Proceedings of the ACM 3rd International Symposium on Field-Programmable Gate Arrays*, pp. 111–117, February 1995.

[32] A. Greiner and F. Pecheux, "Alliance: a complete set of cad tools for teaching vlsi design," in *Proceedings of the 3rd Eurochip Workshop on VLSI Design Training*, 1992.

[33] Altera, "40-nm FPGA Power Management and Advantages," http://www.altera.com, 2010.

[34] M. Zied, M. Hayder, F. Umer, and M. Habib, "FPGA interconnect topologies exploration," *International Journal of Reconfigurable Computing*, vol. 2009, 2009.

[35] P. Jamieson, W. Luk, S. J. E. Wilton, and G. A. Constantinides, "An energy and power consumption analysis of FPGA routing architectures," in *Proceedings of the International Conference on Field-Programmable Technology (FPT '09)*, pp. 324–327, December 2009.

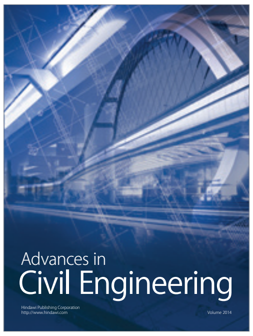
Journal of Engineering

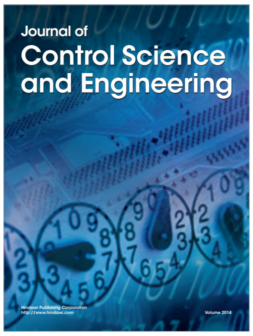
The Scientific World Journal

International Journal of Rotating Machinery

Journal of Sensors

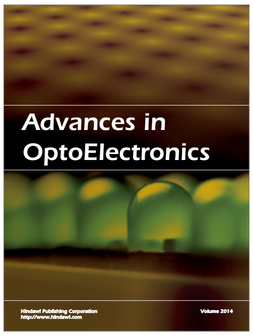
International Journal of Distributed Sensor Networks


Advances in Civil Engineering


Journal of Control Science and Engineering

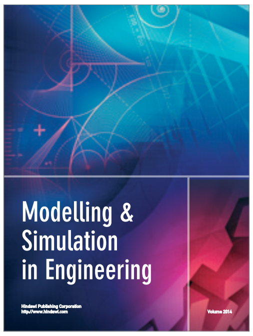
Journal of Robotics

Journal of Electrical and Computer Engineering

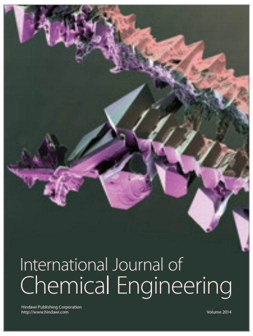

Advances in OptoElectronics

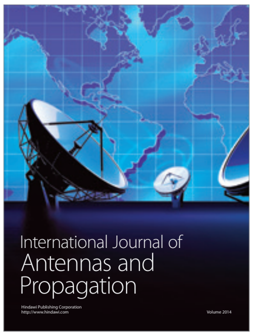
VLSI Design

International Journal of Navigation and Observation

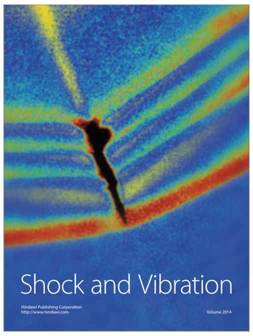

Modelling & Simulation in Engineering

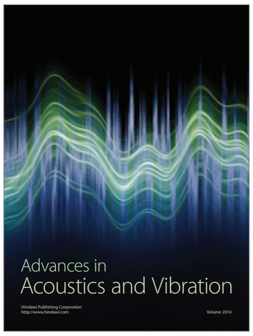
International Journal of Aerospace Engineering


International Journal of Chemical Engineering


International Journal of Antennas and Propagation

Active and Passive Electronic Components


Shock and Vibration


Advances in Acoustics and Vibration