Chapter 6
Tree-Based ASIF Using Heterogeneous Blocks

An Application Specific Inflexible FPGA (ASIF) is an FPGA with reduced flexibility and improved density that can implement a predetermined set of application circuits which will operate at mutually exclusive times. A homogeneous ASIF is presented in the previous chapter; this chapter extends the work to heterogeneous domain. An ASIF that is reduced from a heterogeneous FPGA is called as heterogeneous ASIF. A heterogeneous ASIF can contain hard-blocks such as multipliers, adders, RAMS etc. In order to generate a heterogeneous ASIF, first a minimal FPGA architecture is defined that can implement any of the applications under consideration. Later these application circuits are efficiently placed and routed to minimize total routing switches required by the heterogeneous FPGA architecture. After that, all unused routing switches are removed from the FPGA to generate a heterogeneous ASIF.

This chapter presents a new tree-based heterogeneous ASIF. Four ASIF generation techniques are explored for tree-based heterogeneous ASIF using 17 benchmarks that use heterogeneous blocks. These benchmarks are further divided into two sets and results show that an ASIF generation technique with efficient logic and routing resource sharing produces optimal results for both benchmark sets. Further experiments are performed to determine the effect of LUT and arity size on tree-based heterogeneous ASIF. Later, tree-based ASIF with the best LUT-arity combination is compared to mesh-based ASIF. Also a quality analysis of tree-based heterogeneous ASIF and a quality comparison between mesh-based and tree-based ASIFs is performed.

6.1 Reference Heterogeneous FPGA Architectures

This section gives a brief overview of heterogeneous mesh-based and tree-based FPGA architectures and the related software flow that is used to place and route different benchmarks on these architectures. Application circuits are efficiently placed and routed on these architectures and later they are reduced to their respective ASIFs.

DOI: 10.1007/978-1-4614-3594-5_6,

6.1.1 Heterogeneous Tree-Based FPGA Architecture

A tree-based heterogeneous architecture [45] is a hierarchical architecture having unidirectional interconnect. In this 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. The number of signals entering into and leaving from the cluster 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 architecture. This rule states that

$$IO = \left(\underbrace{k.n^{\ell}}_{L.B(p)} + \underbrace{\sum_{x=1}^{z} a_x.b_x.n^{(\ell-\ell_x)}}_{H.B(p)} \right)^{p} \tag{6.1}$$

where

$$H.B(p) = \begin{Bmatrix} 0 & if(\ell - \ell_x < 0) \\ a_x.b_x.n^{(\ell-\ell_x)} & if(\ell - \ell_x \geq 0) \end{Bmatrix} \tag{6.2}$$

In Eq. 6.1 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, 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. 6.1 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. Further details regarding the tree-based heterogeneous FPGA architecture can be found in Chap. 4.

6.1.2 Heterogeneous Mesh-Based FPGA Architecture

The architecture used in this work is a VPR-style (Versatile Place & Route) [81] architecture that contains configurable logic blocks (CLBs), I/Os and hard-blocks (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 width and height of the smallest block in the architecture. Further details regarding the architecture can be found in Sect. 4.2.2.

6.1.3 Software Flow

The software flow used to place and route different benchmarks (netlists) on the two architectures is same as discussed in Chap. 4. The flow starts with the conversion of vst (structured vhdl) file to BLIF format [20]. The BLIF file is then passed through a parser which removes HBs from the file and passes it to SIS [14] that synthesizes it into LUT format of a given size (K). The file is then passed to T-VPACK [14] which packs and converts it into net format. A netlist in net format contains CLBs and I/O instances that are connected together using nets. The size of a CLB is defined as the number of LUTs contained in it and in this work this size is set to be 1 for both mesh-based and tree-based architectures. After T-VPACK the netlist is passed through another parser that adds previously removed HBs and finally it is placed and routed separately on tree-based and mesh-based FPGAs. A detailed description regarding the placement and routing of netlists on the two architectures is already given in Sect. 4.4.

6.2 Heterogeneous ASIF Generation Techniques

A heterogeneous ASIF is a modified form of a heterogeneous FPGA architecture. A heterogeneous ASIF has reduced flexibility but improved density compared to heterogeneous FPGA and it can contain hard-blocks like multiplier, adders, RAMs etc. A group of netlists are placed and routed on the FPGA architecture. Different automatic floor-planning techniques are used to optimize the position of different blocks on the architecture. Among these floor-planning techniques, the technique Block-Move-Rotate (BMR) gives the best results (refer to Chap. 4) and this is the technique that is used for the generation of mesh-based heterogeneous ASIF. Similarly, for tree-based ASIF ASYM techniques gives best results (refer to Chap. 4) and this is the technique that is used for tree-based ASIF.

In order to generate a mesh-based heterogeneous ASIF, initially a heterogeneous FPGA architecture is defined using an architecture description file. Blocks of different sizes are defined, and later mapped on a grid of equally sized slots. The placer maps multiple netlists together to get a single architecture floor-planning for all netlists. Efficient placement and routing techniques are exactly the same as discussed in previous chapter. However, necessary changes are performed to handle heterogeneous blocks. Efficient placement tries to place the instances of different netlists in such a way that minimum routing switches are required in an FPGA. Later, efficient routing encourages different netlists to route their nets on an FPGA with maximum common routing paths. After all netlists are efficiently placed and routed on FPGA, unused switches are removed from the architecture to generate a heterogeneous ASIF.

For tree-based ASIF, similar to mesh-based architecture, initially an FPGA architecture is defined using an architecture description file. Architecture description file contains complete detail about different parameters of the architecture including the

Table 6.1 SET I benchmark details

Index	Circuit name	In	Out	Mult 18×18	LUT-3	LUT-7
1.	cf_fir_3_8_8_ut	42	22	4	217	186
2.	diffeq_f_systemC	66	99	4	2,114	1,366
3.	diffeq_paj_convert	12	101	5	1,013	471
4.	fir_scu	10	27	17	2,267	629
5.	iir	33	30	5	524	322
6.	iir1	28	15	5	898	410
7.	rs_decoder_1	13	20	13	2,367	900
8.	rs_decoder_2	21	20	9	4,204	1,835
9.	Maximum	66	101	17	4,204	1,835

signal bandwidth of different clusters located at different levels of hierarchy, types of different blocks, the level where they are located and their pin details etc. Once the architecture is defined, netlists are efficiently partitioned and then placed and routed on the architecture. Different ASIF generation techniques are employed for tree-based architecture in a manner similar to the previous chapter except that necessary modification are made to handle the heterogeneity of the architecture. Once the netlists are efficiently placed and routed on the architecture, unused resources of the architecture are removed to generate an ASIF.

6.3 Experimentation and Analysis

6.3.1 Experimental Benchmarks

This work uses 17 open core benchmarks to explore the effect of different ASIF generation techniques on the tree-based ASIF. These benchmarks are further divided into two sets. The division of benchmarks into two sets is basically based on the type of hard-blocks that are supported by these benchmarks. The details of two sets of benchmarks is shown in Tables 6.1 and 6.2 respectively. As it can be seen from these tables that one set supports only 18×18 multipliers while the second set supports 16×16 multipliers along with $20 + 20$ adders. Apart from hard-blocks, these benchmarks also contain logic blocks; size of whom in terms of different LUT sizes is shown in the last two columns of these tables. These benchmarks are converted into .net format using the same flow as described in Chap. 4. Netlists with different LUT sizes are generated where number of I/Os and hard-blocks remain unchanged, but the number of LUTs change with change in LUT size. Details of netlists for two sets with LUT size 3 and 7 are shown in last two columns of Tables 6.1 and 6.2 respectively.

Table 6.2 SET II benchmark details

Index	Circuit name	In	Out	Mult 16×16	Add 20+20	LUT-3	LUT-7
1.	cf_fir_3_8_8_open	42	18	4	3	159	159
2.	cf_fir_7_16_16	146	35	8	14	639	638
3.	cfft16x8	20	40	–	26	1,937	1,122
4.	cordic_p2r	18	32	–	43	803	801
5.	cordic_r2p	34	40	–	52	1,497	1,178
6.	fm	9	12	1	19	1,508	1,046
7.	fm_receiver	10	12	1	20	1,004	677
8.	lms	18	16	10	11	965	935
9.	reed_solomon	138	128	16	16	537	537
10.	Maximum	146	128	16	52	1,937	1,178

This section is basically divided into three parts. First the effect of different ASIF generation techniques on tree-based heterogeneous ASIF is explored for both sets of benchmarks. For each ASIF generation technique, separate ASIFs are generated for both sets of benchmarks. Results of different ASIF generation techniques are then compared and later results of ASIF generation techniques are also compared with those of equivalent tree-based heterogeneous FPGA. Secondly, experiments are performed to determine the effect of LUT and arity size on tree-based heterogeneous ASIF. Experiments are performed for a range of architectures that are generated using different combinations of LUT and arity sizes and netlists are placed and routed on them. Finally, after determining the best LUT and arity size combination for tree-based ASIF, a comparison between mesh-based and tree-based ASIFs is performed using their best ASIF generation techniques.

6.3.2 *Effect of Different ASIF Generation Techniques on Heterogeneous Tree-Based ASIF*

Figure 6.1 shows the variation in routing area of tree-based architecture using different ASIF generation techniques. In this experimentation, for all ASIF generation techniques, LUT size is set to be 4, arity size is 4, and CLB size is 1. In this figure columns 1–5 correspond to results for SET I benchmarks whereas columns 6 to 10 correspond to results for SET II benchmarks. For each set of benchmarks, ASIFs of four different kinds are generated over a range of signal bandwidths. The range of signal bandwidth is varied from the minimum required to the maximum value beyond which a further increase in signal bandwidth does not improve the routing area of ASIF. It can be seen from the figure that for each ASIF generation technique (for both sets), an increase in signal bandwidth decreases the routing area of architecture. Although the rate of decrease in routing area of the architecture with increasing signal bandwidth varies, a decrease in the routing area of the architecture for each ASIF generation technique remains valid. The difference in the routing areas

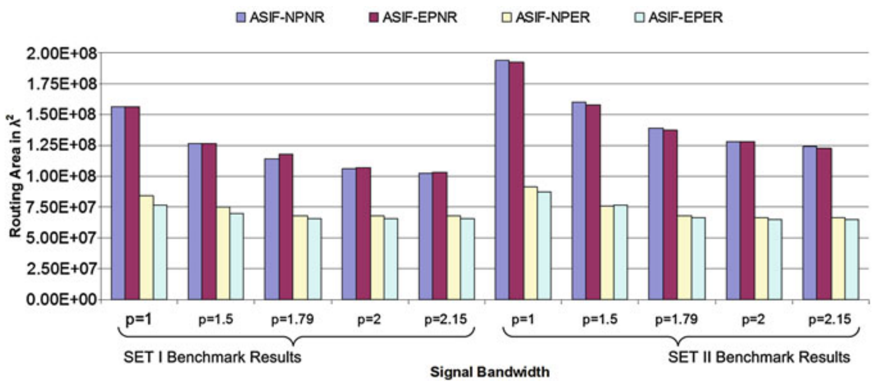

Fig. 6.1 Routing area comparison between different tree-based heterogeneous ASIFs

of different ASIF generation techniques is mainly because of the different optimization techniques that are employed in each ASIF generation technique. As can be seen from the figure that the ASIF generation technique with no optimization (i.e. ASIF-NPNR) produces the worst results among the four ASIF generation techniques for both sets. Also the technique with only efficient partitioning solution (i.e. ASIF-EPNR) is not as efficient as the technique with only efficient routing solution (i.e. ASIF-NPER). Efficient partitioning alone is unable to produce any significant results because it is unable to exploit the routing resources of the architecture that contribute a major percentage of the total area. However, when it is combined with efficient routing, it gives the best overall results compared to the rest of ASIF generation techniques.

Figure 6.2 shows the variation in the number of wires that are used by different ASIF generation techniques. Similar to area results, first 5 columns give results for SET I while the next 5 columns give results for SET II benchmarks. For both sets, signal bandwidth is varied from the minimum required to the maximum, and results show that, at maximum signal bandwidth, ASIF-EPNR uses the least number of wires while ASIF-NPER uses the most number of wires for both sets. Similar to the results for homogeneous ASIFs, the techniques using efficient routing resource sharing consume more wires as compared to the techniques that are not using efficient routing resource sharing. Similarly, techniques with efficient logic resource sharing consume less number of wires than the techniques that are not using efficient logic sharing. These two facts combined together result in the trends that are shown in Fig. 6.2.

Figure 6.3 shows the comparison between different ASIF generation techniques for different number of netlists. In this figure, first 8 columns give results for SET I and next 9 columns give results for SET II respectively. For fist 8 columns, column 1 means that ASIF is generated for first netlist of Table 6.1, column 2 means that ASIF is generated for first two netlists of the table and same rule applies to the netlists of Table 6.2. For all four techniques, the signal bandwidth is set to be maximum

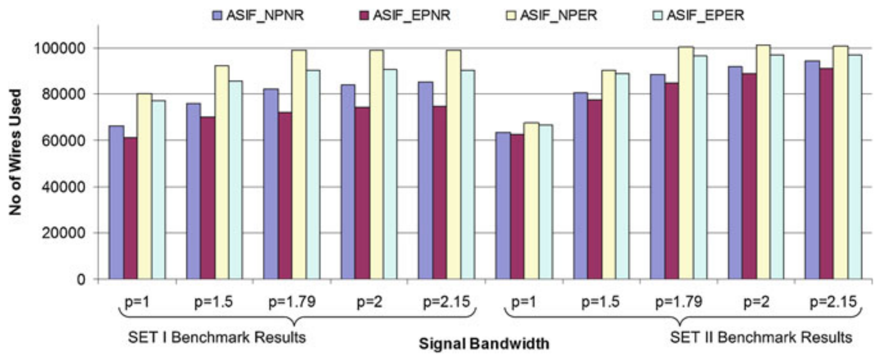

Fig. 6.2 Wire comparison between different heterogeneous ASIF techniques

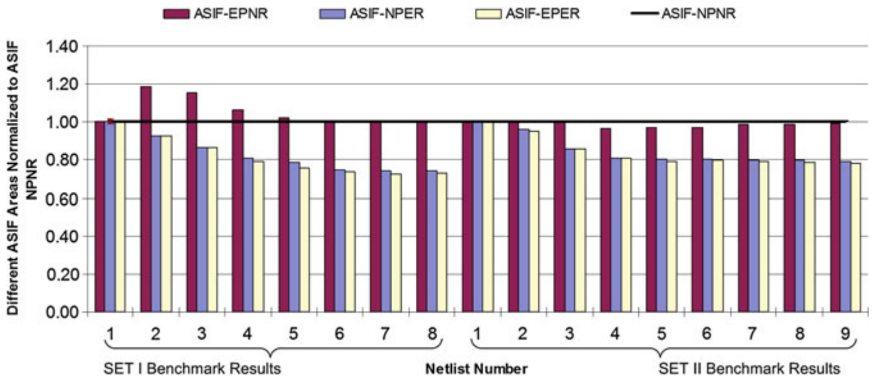

Fig. 6.3 Different ASIFs normalized to ASIF-NPNR

that produces the best results. As it can be seen from the figure that ASIF-NPNR gives worst results and ASIF-EPER gives best results in terms of area. Further the gap between the best and the worst ASIF techniques increases with increase in the number of netlists for which the ASIF is being generated.

In order to perform the performance comparison of different ASIF generation techniques, we have employed a model that calculates the number of switches that are crossed by different applications mapped on ASIF. Figure 6.4 shows the performance comparison results between different techniques for a varying range of signal bandwidths. In this figure results of first 5 columns correspond to SET I benchmarks while remaining 5 correspond to SET II benchmarks. Since each ASIF is generated for a group of netlists and each netlist crosses different number of switches on its critical path, an average of critical path switch number is presented here for each technique. It can be seen from the figure that for both sets of benchmarks and for all generation techniques, an increase in signal bandwidth results in a decrease in average number of switches that are crossed by critical path. These results are in direct correspondence with those of Fig. 6.1 and further this observation is enhanced

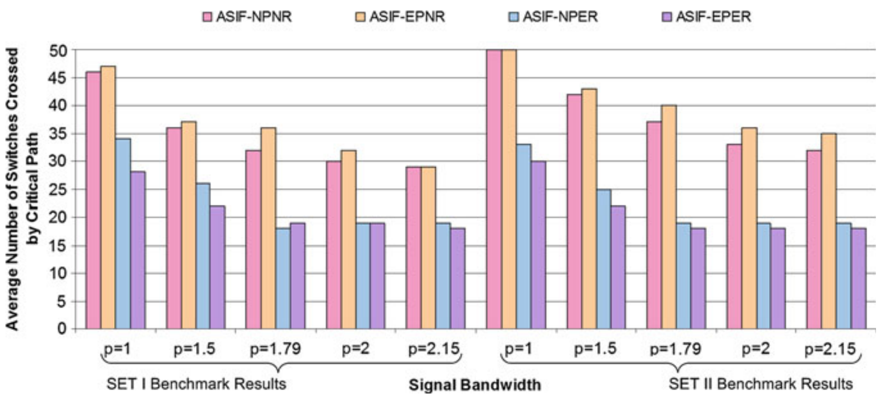

Fig. 6.4 Performance comparison between different tree-based heterogeneous ASIFs

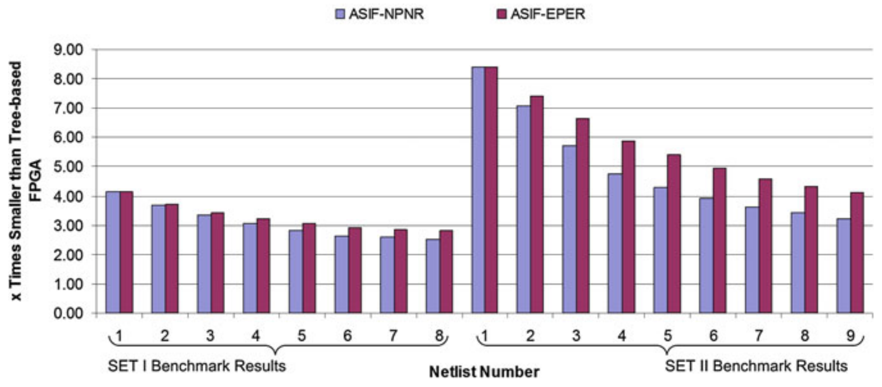

Fig. 6.5 Tree-based heterogeneous FPGA vs tree-based heterogeneous ASIFs

by the fact that ASIF-EPER produces the best results in both cases. Although the performance comparison presented here is based on a simple model and it does not give the exact temporal delay, yet it gives us an idea about the performance of each ASIF generation technique.

Figure 6.5 shows the area comparison between tree-based FPGA and the worst and the best ASIF generation techniques. In this figure first 8 columns give results for SET I benchmarks and remaining 9 columns give results for SET II benchmarks respectively. Signal bandwidth is set to be maximum for ASIF while equivalent FPGA has minimum signal bandwidth which is capable of mapping any of the benchmarks of respective sets. In this figure, starting from 1, ASIFs are generated for a varying range of netlists and results are then compared to those of tree-based FPGA. For SET I benchmarks, when varying the number of netlists, the order shown in Table 6.1 is followed. For example, netlist 1 means an ASIF is generated for "cf_fir_3_8_8_ut" only and netlist 2 means that an ASIF is generated for "cf_fir_3_8_8_ut" and "diffeq_f_SystemC" and so on. Similarly, for SET II benchmarks, the order shown in Table 6.2 is followed. Results of ASIFs of both sets for varying number of netlists are compared against their respective LUT-4, arity-4 tree-based FPGAs that can implement any netlist of the Table 6.1 for SET I and any netlist of Table 6.2 for SET II respectively. It can be seen from the figure that for best ASIF generation technique, for SET I, for 1 netlist ASIF is 4.14 times or 76% smaller than tree-based FPGA. However as the number of netlists increase, the gap between the two decreases and for 8 netlists ASIF is 2.8 times or 64% smaller than the tree-based FPGA. Similarly for SET II benchmarks, for 1 netlist the best ASIF is 8.4 times smaller than the corresponding tree-based FPGA and when the count of netlists is increased to 9 the tree-based ASIF is 4.12 times or 75% smaller than the corresponding tree-based FPGA. The results in Fig. 6.5 suggest that ASIF gives excellent area gain for a small group of netlist. However, the advantage of ASIF begins to fade as the size of the set of netlists increases and for a very large group of netlists it might not be able to give any significant area advantages.

So far in this section we have seen that ASIF-EPER produces best results for both sets of benchmarks that are under consideration. Although we have fixed LUT and arity sizes to 4, ASIF-EPER gives the best results even if we change this combination; hence this is the technique that we will use to determine the effect of LUT and arity size on a tree-based heterogeneous ASIF.

6.3.3 Effect of LUT and Arity Size on Heterogeneous Tree-Based ASIF

In order to determine the effect of LUT and arity size on a tree-based heterogeneous ASIF, experiments are performed for a group of 17 open core benchmarks and these benchmarks are further divided into two sets (refer to Tables 6.1 and 6.2). For benchmarks of SET I and SET II, LUT size is varied from 3 to 7 while arity size is varied from 4 to 8 and 16. So with 5 LUT sizes and 6 arity sizes, we have explored 30 architectures for SET I and SET II where ASIF is generated for each architecture using the ASIF-EPER technique as it gives the best overall results compared to other ASIF generation techniques.

Effect of varying LUT and arity size on total area of tree-based heterogeneous ASIF is shown in Fig. 6.6 where first 5 columns give results for SET I and next 5 columns give results for SET II. It can be seen from the figure that smaller LUTs give better area results as compared to the larger LUTs. Also it can be noticed from the figure that, in general, for a fixed LUT size the total area of the ASIF decreases with an increase in the arity size. On the other hand, for a fixed arity size, an increase in LUT size increases the total area of the architecture. The observations made in this figure are further elaborated using Figs. 6.7 and 6.8.

Figure 6.7 shows the effect of varying K and N on the logic area of tree-based heterogeneous ASIF. It can be seen from the figure that for all arity sizes, an increase

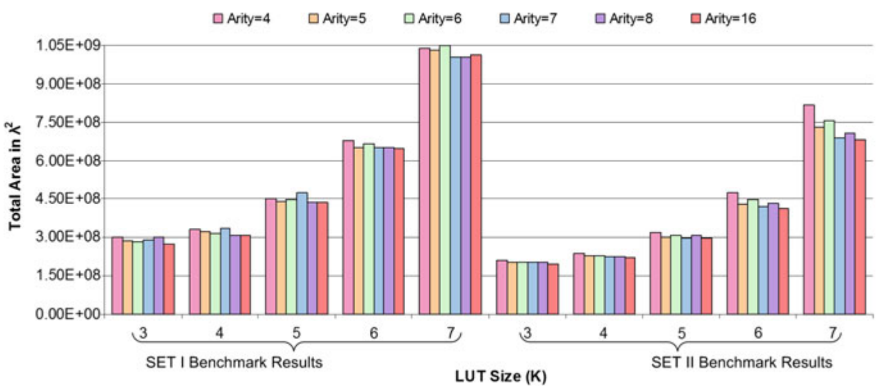

Fig. 6.6 Effect of LUT and arity size on total area of tree-based heterogeneous ASIF

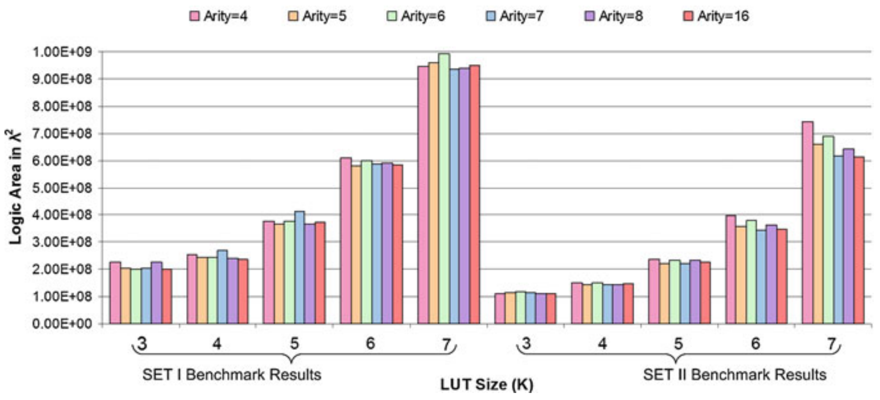

Fig. 6.7 Effect of LUT and arity size on logic area of tree-based heterogeneous ASIF

in K increases the logic area of the architecture. In fact, for a fixed arity size, increase in LUT size decreases the total number of LUTs required. But at the same time LUT area increases exponentially with increase in its size. Normally the increase in LUT area overshadows the decrease in its number; hence causing a sharp rise in the logic area of the architecture.

Figure 6.8 shows the effect of varying K and N on the routing area of tree-based heterogeneous ASIF. It can be seen from the figure that for all arity sizes, in general, a raise in K causes a decline in the routing area of the architecture. For a fixed N, elevation in K reduces average number of LUTs required by the architecture but it raises average routing area per LUT. However, the fall in average number of LUTs required by the architecture outweighs the rise in routing area per LUT; hence leading to overall decline in routing area of the architecture. Although there is a decrease in routing area of the architecture (refer to Fig. 6.8) with an increase in LUT and arity size but this decline is overshadowed by the increase in the logic area of the

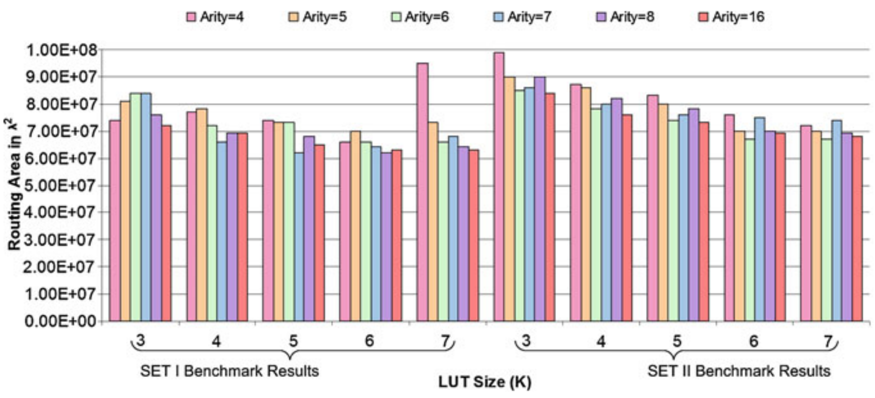

Fig. 6.8 Effect of LUT and arity size on routing area of tree-based heterogeneous ASIF

architecture (refer to Fig. 6.7); therefore leading to an increased total area of the architecture (refer to Fig. 6.6).

Figure 6.9 shows the effect of LUT and arity size on the critical path switch number of a tree-based heterogeneous ASIF. This work considers only an elementary model to have an idea about the critical path delay and wire delays are not considered here. In Fig. 6.9 columns 1–5 give the results for SET I benchmarks while columns 6–10 give results for SET II benchmarks. It can be seen from the figure that an increase in LUT and arity size decreases the number of switches that are crossed by critical path. For both sets of benchmarks, for a fixed K, an increase in N decreases the average number of switches crossed by critical path. Similarly, for a fixed N, an increase in K decreases the average number of switches crossed by critical path. However, for a fixed N, the two benchmark sets have different rates of decrease in average critical path switch number with increasing K. For example in case of SET I, for N=4, as the K is varied from 3 to 7, a 28% decline in average switch number is observed. On the other hand, in case of SET II, a decline of only 7% is observed for N=4 and variation in K from 3 to 7. This is because of the fact that for a fixed N, when K is varied from 3 to 7, for 5 out of 9 benchmarks of SET II, LUT requirement remains almost same (refer to Table 6.2). Number of switches of these 5 benchmarks constitute around 88% of the average switch numbers of SET II; hence a significant decrease in the average switch numbers is not observed. Figures 6.6 to 6.9 give an overview about the effect of K and N on tree-based ASIF. Results shown in these figures imply that ASIFs with small K and large N (K = 3 or 4 and N = 16) give best area results and large K with large N give best performance results. So, we have to find a trade-off between area and performance results. For this purpose, area-delay products of ASIFs with varying LUT and arity size are shown in Fig. 6.10. In this figure, first five columns give results for SET I benchmarks while next five columns give results for SET II benchmarks. It can be seen from the figure that K = 4 with N = 16 gives best results for SET I benchmarks and K = 3 with N = 16 gives best results for SET II benchmarks. Since, we have determined the best LUT-arity

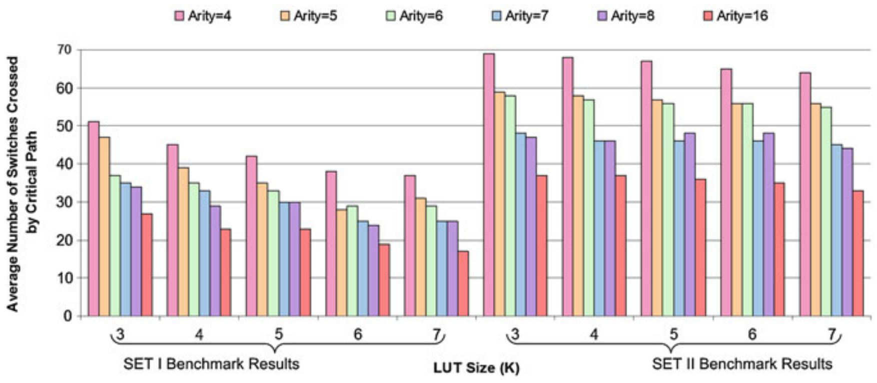

Fig. 6.9 Effect of LUT and arity size on critical path of tree-based heterogeneous ASIF

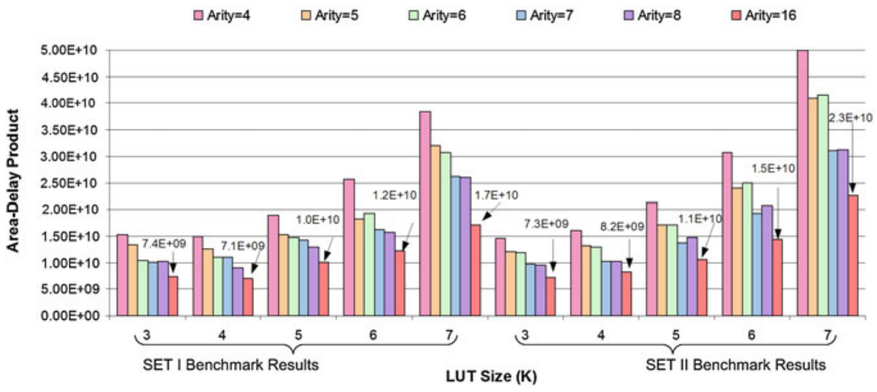

Fig. 6.10 Effect of LUT and arity size on area-delay product of tree-based heterogeneous ASIF

combinations for the two sets, these combinations will be used for comparison of heterogeneous mesh-based and tree-based ASIFs.

6.3.4 Comparison Between Heterogeneous Mesh-Based and Tree-Based ASIFs

Now that we have determined the best LUT-arity combination for tree-based ASIF, here we present a comparison between the best techniques of tree-based ASIF and equivalent mesh-based ASIF. The comparison results of heterogeneous mesh-based and tree-based ASIFs are shown in Fig. 6.11. In this figure columns 1–4 present routing area comparison between mesh and tree architectures for SET I benchmarks whereas columns 5–8 present comparison results for SET II benchmarks. For both

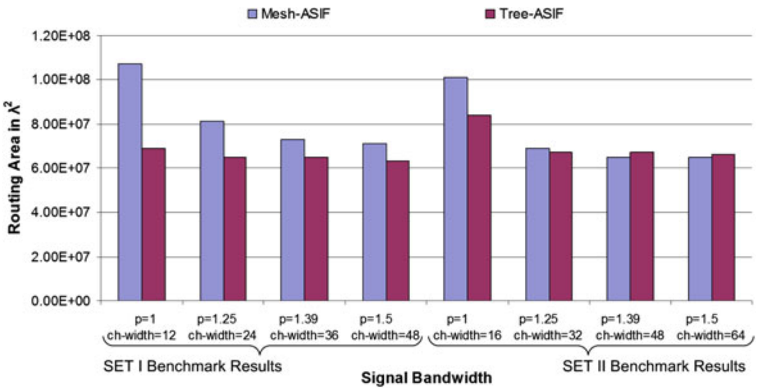

Fig. 6.11 Routing area comparison between heterogeneous mesh-based and tree-based ASIFs

sets, mesh-based and tree-based ASIFs are generated using ASIF-EPER technique as it gives the best results. Signal bandwidth is varied for both sets starting from a minimum required to a maximum value. An increase of signal bandwidth beyond maximum value does not improve further the routing area of the architecture. For tree-based ASIF, for SET I, LUT size is set to be 4 and arity size is set to be 16 and for SET II LUT size is set to be 3 and arity size is set to be 16. In case of mesh-based ASIF, for SET I, LUT size is 4 and for SET II LUT size is 3 while CLB size is 1 for both architectures.

Since the variation in signal bandwidth does not affect the logic area, only routing area results are presented in Fig. 6.11. In this figure "p" values correspond to the signal bandwidth of tree-based ASIF and it is calculated using Eq. 6.1 whereas "ch-width" values correspond to the signal bandwidth values of mesh-based ASIF. It can be seen from the figure that for SET I, tree-based ASIF produces better results when compared to mesh-based ASIF for all signal bandwidths. For maximum signal bandwidth (i.e. $p = 1.5$, ch-width $= 48$) tree-based ASIF gives 11.27% better routing area than mesh-based ASIF. For SET II benchmarks, however, tree-based ASIF gives better routing area results for smaller signal bandwidths. But with larger bandwidth tree-based ASIF looses the gain and for a maximum signal bandwidth value mesh-based ASIF is 1.5% better than tree-based ASIF. Intuitively if we look at the ASIF generation mechanism of the two architectures, they should give same results as unused resources are removed in both architectures after the mapping of benchmarks under consideration. However, it is the arrangement of resources and quality of tools that makes the difference. For example for mesh-based architecture, the ASIF generation mechanism uses BMR floor-planning at its core (for details regarding BMR floor-planning refer to Chap. 4). This is the most flexible floor-planning technique where hard-blocks are allowed to move and rotate and this is the reason that mesh-based ASIF performs better in case of SET II benchmarks. Since, for SET II benchmarks, the major percentage of total communication is between CLBs and HBs and BMR is flexible enough to take full advantage of this trend. However, for

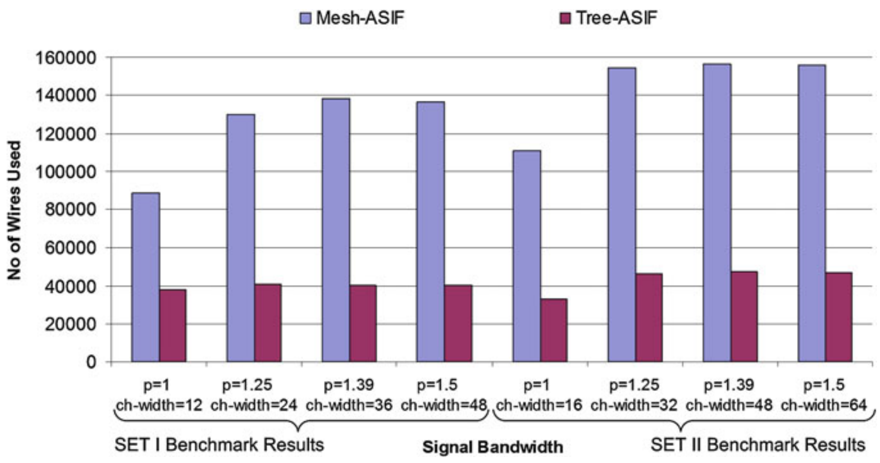

Fig. 6.12 Wire comparison between heterogeneous mesh-based and tree-based ASIFs

tree-based ASIF, ASYM exploration technique is used (for details regarding ASYM technique refer to Chap. 4) which is comparatively less flexible than BMR; hence causing an increased area than mesh-based ASIF. For SET I benchmarks however, the major percentage of total communication is between the CLBs and tree-based ASIF is not affected in that case.

Area results are not the only comparison performed between mesh-based and tree-based ASIFs, the two architectures are also compared on the basis of number of wires consumed by the two architectures and the results are shown in Fig. 6.12. Similar to area results, first four columns present comparison results for SET I benchmarks while last four present results for SET II benchmarks. As it can be seen from the figure that for both sets of benchmarks, tree-based ASIF consumes far less wires than mesh-based ASIF. For maximum signal bandwidth values, tree-based ASIF uses 70, 69% less wires than mesh-based ASIF. The number of wires can play a pivotal role while making a choice for the architecture. For mesh-based ASIF, the number of wires are high and they increase significantly with the increase in the signal bandwidth and at higher bandwidth values the architecture might become wire dominant and it might be obliged to use a lower signal bandwidth. So in such a case, tree-based ASIF is clearly a better choice when compared to mesh-based ASIF as it gives relatively better area and consumes significantly less wires than mesh-based ASIF.

6.4 Quality Analysis of Heterogeneous Tree-Based ASIF

Similar to the quality analysis of tree-based homogeneous ASIF, the quality analysis of tree-based heterogeneous ASIF is also performed. Quality analysis of tree-based heterogeneous ASIF is performed for two sets separately and results for SET I and

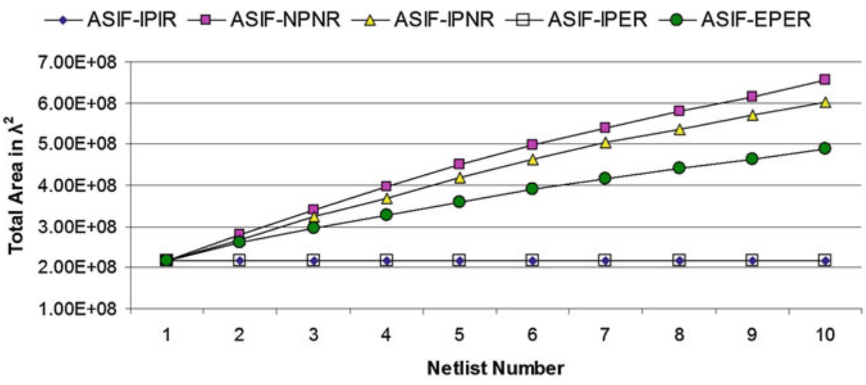

Fig. 6.13 Quality analysis of tree-based heterogeneous ASIF (SET I)

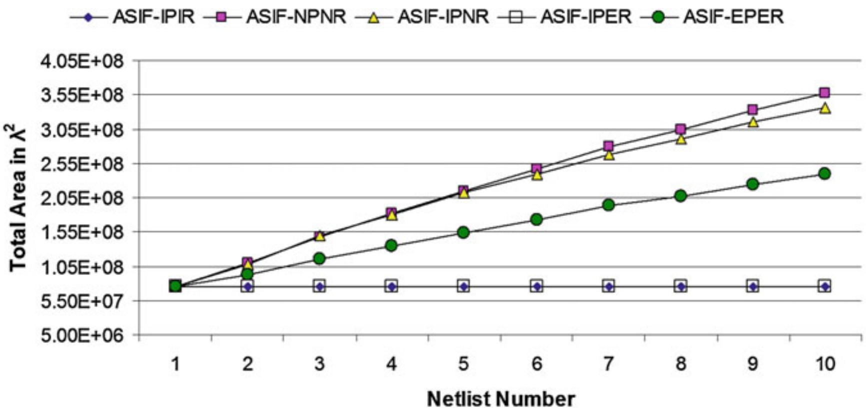

Fig. 6.14 Quality analysis of tree-based heterogeneous ASIF (SET II)

SET II are shown in Figs. 6.13 and 6.14 respectively. In these figures, the X-axis shows the number of same netlists used in the generation of ASIF, whereas Y-axis shows the total area of ASIFs. For SET I, quality analysis is performed using "rs_decoder_2" and for SET II quality analysis is performed using "cfft16x8". The two netlists are chosen as they are the biggest among the two sets. Similar to the quality analysis of homogeneous ASIF, five different techniques are used to perform the quality analysis of tree-based heterogeneous ASIF. It can be seen from these figures that ASIF-NPNR produces the worst results whereas ASIF-IPIR produces the best results and an ASIF generated using efficient placement and routing technique lies almost in the middle of the best and the worst solutions. The major area loss in ASIF is mainly due to the placement/partitioning inefficiency. New placement techniques need to be explored to further improve area of an ASIF.

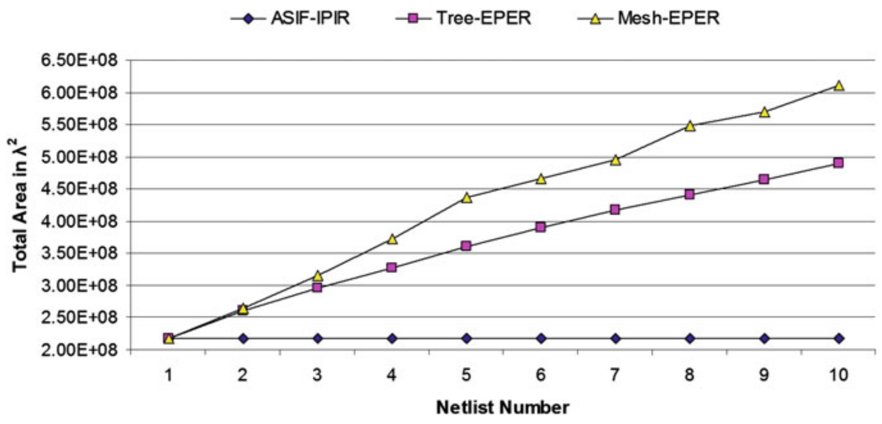

Fig. 6.15 Quality comparison between heterogeneous mesh-based and tree-based ASIF (SET I)

6.4.1 Quality Comparison Between Heterogeneous Mesh-Based and Tree-Based ASIF

In order to evaluate the quality of mesh-based and tree-based architectures, a quality comparison between heterogeneous mesh-based and tree-based ASIFs is performed. The quality comparison is performed separately for SET I and SET II benchmarks by using ASIF-EPER techniques for mesh-based and tree-based architectures and it is shown in Figs. 6.15 and 6.16 respectively. Similar to experiments in previous sub-section, ASIFs are generated for both architectures using "rs_decoder_2" and "cfft16x8" netlists and results are also compared against an ideal solution. As it can be seen from these figures that for both architectures, the quality of ASIFs of both architectures deteriorates with increase in the count of netlists. However, tree-based ASIF produces better results as compared to mesh-based ASIF. For small number of netlists, the gap between mesh-based and tree-based ASIFs is not significant but as we increase the count of netlists this gap becomes more and more significant and finally for a group of 10 netlists, tree-based ASIF is around 19%, 20% more area efficient than mesh-based ASIF for SET I and SET II benchmarks respectively.

6.5 Heterogeneous ASIF Hardware Generation

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

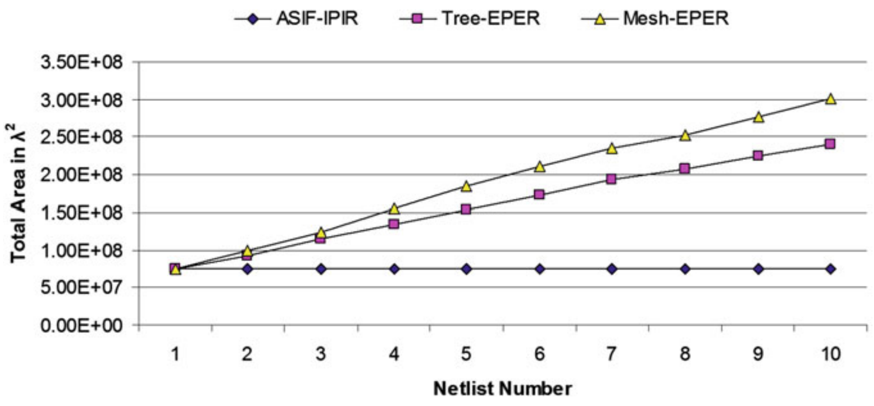

Fig. 6.16 Quality comparison between heterogeneous mesh-based and tree-based ASIF (SET II)

The ASIF generation flow remains exactly the same except that now the block database contains details about a variety of blocks that are used by different netlists 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 model generation of homogeneous and heterogeneous ASIFs is that of the support for hard-blocks. In homogeneous ASIFs all blocks are of uniform size as there is only a single type of blocks in the architecture. However, in heterogeneous architectures there are blocks of different types. Although the basic tile size remains the same, there might be blocks that occupy multiple tiles. However, it does not leave much of an effect on the the VHDL generation mechanisms. So different processes involved in the hardware generation of homogeneous ASIFs remain the same for the hardware generation of heterogeneous ASIFs.

The generated VHDL model of heterogeneous ASIF is translated to 130 nm standard cell library of STMicroelectronics, and then passed to Cadence encounter for layout generation. Some of the layout results are shown in Fig. 6.17 where tree-based ASIFs and ASICs are generated for a varying number of netlists. Figure 6.17a, c, e show the layout generated for tree-based ASIF and Fig. 6.17b, d, f show the layout of ASIC generated for 1, 5 and 9 netlists of SET II benchmarks. Here from netlist 1 we mean that layout is generated for first netlist of Table 6.2. Similarly from netlist 5 and 9, we mean that layout is generated for first 5 and all the 9 netlists of Table 6.2. The hardware of tree-based ASIF is generated using LUT-3, Arity-16 combination with maximum channel width that gives best area results. It can be seen from the figure that for 1 netlist ASIC is 20% smaller than tree-based ASIF and for 9 netlists this gap increases to almost 40%. Although for 1 netlist both ASIF and ASIC contain no switches or memories, but ASIF consumes more area than ASIC. This is because of the fact that ASIF layout is generated using a VHDL description which is the result of software flow of ASIF (refer to Fig. 4.7). During that flow different tools are used starting from the synthesis of the netlist to its routing on the architecture and these tools have added different imperfections in the design which at the end result

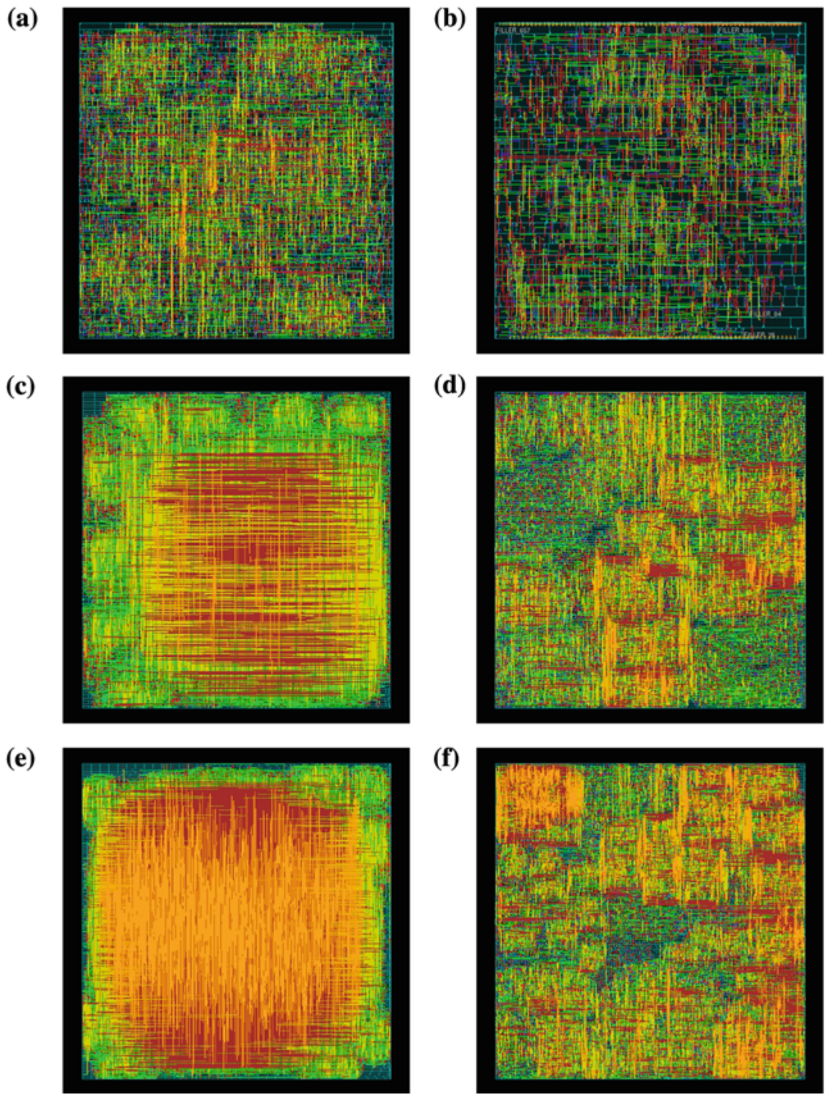

Fig. 6.17 ASIF/ASIC layout for varying number of netlists using encounter (SET II). **a** Tree-ASIF, netlists=1, 0.05 mm^2. **b** ASIC, netlists=1, 0.04 mm^2. **c** Tree-ASIF, netlists=5, 0.56 mm^2. **d** Sum of ASICs, netlists=5, 0.37 mm^2. e Tree-ASIF, netlists=9, 1 mm^2. **f** Sum of ASICs, netlists=9, 0.6 mm^2

in a larger area. Also tree-based ASIF uses LUTs which remain as configurable as in FPGAs. On the other hand, ASIC is generated directly from hardware description of the netlist and it uses gates that require no configuration bits.

Further, the increase in number of netlists increases the gap between ASIF and ASIC and for 9 netlists ASIC is 40% smaller than ASIF. This is because of the fact that increase in number of netlists increases the switches and configuration memory for ASIF while there are no configuration memories in ASICs. Also the standard cell library of STMicroelectronics does not contain SRAM cell. So LATCH cell is used instead of SRAM for configuration memory. Generally LATCH cells are larger than SRAMs and for an ASIF generated for 9 netlists, SRAMs take up to 23% of total routing area. The percentage of configuration memory area is measured from VHDL model generated using symbolic standard cell library (i.e. measure before mapping it to ST standard cell library). The high percentage of area taken by SRAMs indicates that the gap between ASIF and ASIC can further be reduced by replacing LATCH cells with SRAMS cell in ST standard cell library.

6.6 Summary and Conclusion

In this chapter, a new tree-based heterogeneous ASIF is presented. Four ASIF generation techniques are explored for 17 open core benchmarks. Based on the types of hard-blocks used, benchmarks are divided into two sets where first set contains a mixture of logic blocks and multipliers (SET I) while second set contains a mixture of logic blocks, multipliers and adders (SET II). Comparison of different techniques with tree-based FPGA shows that the most efficient ASIF generation technique is 64%, 75% more area efficient than an equivalent tree-based FPGA for SET I and SET II respectively. Further the exploration of LUT and arity size on tree-based ASIF shows that an ASIF with LUT size 4 and arity size 16 produces the most efficient results for SET I and an ASIF with LUT size 3 and arity size 16 produces the most efficient results for SET II. Comparison of tree-based ASIF with mesh-based ASIF reveals that tree-based ASIF is 11.27% better and 1.5% worse in terms of area for SET I and SET II while it consumes 69%, 70% less wires than mesh-based ASIF for two sets respectively. Finally the quality analysis of two architectures reveals that for a group of 10 similar netlists, tree-based ASIF is 19%, 20% more area efficient than mesh-based ASIF for SET I and SET II benchmarks respectively. The results presented in this chapter are also published in [115].

Layout of the tree-based heterogeneous ASIF is also performed in this chapter using Cadence encounter and its comparison with equivalent sum of ASICs shows that ASIC is 40% smaller than ASIF for a group of 9 open core netlists (SET II). However, the gap between sum of ASICs and ASIF can be reduced by replacing LATCH cells with the SRAM cells. Further, full-custom design of repeatedly used cells can be performed to further decrease the area of ASIF and reduce the gap between ASIF and ASIC. These repeatedly used cells can be groups of similar size SRAMs, switches and logic blocks etc. However, full-custom design can not be much of a benefit for ASICs as there are no major repeatedly used components in them.