Enhanced Methodology and Tools for Exploring Domain-Specific Coarse-grained FPGAs

Husain Parvez, Zied Marrakchi, Habib Mehrez
LIP6, Université Pierre et Marie Curie
4, Place Jussieu, 75005 Paris, France
{parvez.husain, zied.marrakchi, habib.mehrez}@lip6.fr

Abstract

This paper presents a new environment for the exploration of domain-specific coarse-grained FPGAs. An architecture description mechanism is used to define a coarse-grained architecture. A software flow is used to map a netlist on the defined architecture. The software flow not only maps the instances of a target netlist on their respective blocks in the architecture, but also refines the position of the blocks on the architecture. This environment can also be used to define and optimize a domain-specific architecture for a set of netlists to be mapped on it at mutually exclusive times. A set of DSP test-benches are used to show the effectiveness of various techniques used in this work.

1. Introduction

The Versatile Packaging, Placement and Routing tool (VPR) [4] has been widely used for the exploration of fine-grained FPGA architectures. Inherently it does not support coarse-grained blocks. However [3] and [8] have extended VPR to explore specific coarse-grained architectures. Similarly [5] has developed the virtual embedded block methodology (VEB) to model arbitrary embedded blocks on existing commercial FPGAs. One of the key advantages of VEB is that new embedded blocks can be tested in commercial FPGAs. But the VEB methodology can be used only with existing commercial architectures. This deficiency has been resolved in [13] by incorporating VEB methodology in VPR; thus enabling support of architectures other than commercial FPGAs. [12] has also developed a CAD tool for FPGAs with Embedded Hard Cores.

All this previous work propose a pre-determined floor-planning organization and does not consider the problem of optimizing the floor-planning (block positions on the architecture). Our major contribution consists of proposing an environment that refines the architecture floor-planning also. Although an FPGA floor-planning can be achieved manually also. But such a task becomes more difficult when different variety of blocks are to be integrated in the architecture. The manual floor-planning becomes further complicated when an FPGA architectures is required to be optimized for a set of netlists. The method proposed in this work places all the netlists simultaneously and changes the architecture floor-planning to get a trade-off architecture for the given set of netlists. The technique of placing multiple netlists is proposed in [6], where it is used to explore configurable ASICs in a single dimension. Whereas this work extends this methodology to explore two dimensional island-style coarse-grained FPGA architectures.

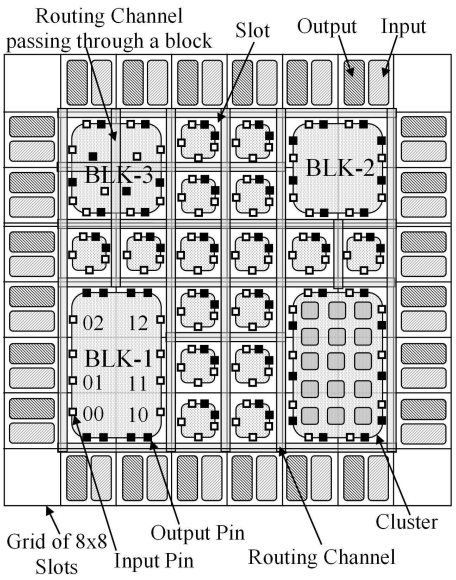


Figure 1. Coarse-grained FPGA

2. Exploration Environment

The coarse-grained architecture is represented on a grid of equally sized SLOTS called the slot-grid. CELLS of different sizes can be mapped on this grid as shown in Figure 1. A CELL can be any block; a soft block like a Configurable Logic Block (CLB); or a hard block like an adder,

multiplier or RAM etc, or even a clusters of CELLS. An instance of a CELL mapped on the slot-grid is called a SITE. Each SITE occupies one or more SLOTS. A routing channel passes between every two neighboring SITES. A SITE occupying more than one SLOT can allow routing channel to pass through it. Once the architecture is defined, the architecture description and an input netlist is passed to the software flow. The software flow maps the instances of the netlist on the SITES of its respective types. The PLACER, a software module, refines both the placement of SITES on the slot-grid (floor-planning) and the mapping of instances on the SITES (binding). The PLACER can also generate a domain-specific FPGA floor-planning for binding a set of netlists on it at mutually exclusive times. This is done by allowing the binding of multiple instances on a single SITE. But instances belonging to the same netlist cannot be mapped on a single SITE. After floor-planning and binding, the ROUTER routes the netlist on the architecture.

Architecture Description: An architecture description file is used to define the FPGA architecture. The parameters Nx and Ny define the size of the slot-grid. A unidirectional mesh [10] or a bidirectional mesh [4] is selected as the routing network. The channel width of the routing network is either fixed to a value W, or a binary search algorithm searches the minimum possible channel width between minimum (Wmin) and maximum (Wmax) channel widths. The position of SITES can be either fixed to an absolute position on the slot-grid, or can be initialized to any position and set as movable, so that the PLACER can refine its position. The CELLS are also defined in the architecture description file. Each CELL is given a name, size (number of slots occupied), and input/output pins. Each pin of the CELL is given a name, a class number, a direction and the slot position on the CELL to which this pin is connected. Figure 1 gives the pictorial view of different CELLS mapped on the slot-grid. The CELL "BLK-1" in the figure is composed of 2x3 slots. Four of its input pins are defined to be on the LEFT side of the slots (0,0), (0,1) and (0,2) of the CELL. Similarly other input and output pins are also assigned a direction and slot position. Pins having the same class are considered equivalent. Thus a driver net targeting a receiver pin of a SITE can be routed to any of the pins of the SITE having the same class. If the routing channel passes through a CELL, the PINS can also be connected to any of the internal channels, as in the case of BLK-3.

Software Flow: A software flow is worked out to map a netlist on the newly defined coarse-grained FPGA. The complete software flow can be seen in the Figure 2. An input to this software flow is a structural netlist in VST (structured VHDL) format. This netlist is composed of traditional standard cell library instances and hard block (HB) instances. VST2BLIF tool is modified to convert the VST file having hard blocks to BLIF format. Later PARSER-1 removes all the instances of hard blocks and passes the remaining netlist to SIS for synthesis into 4-LUT (LookUp Table with

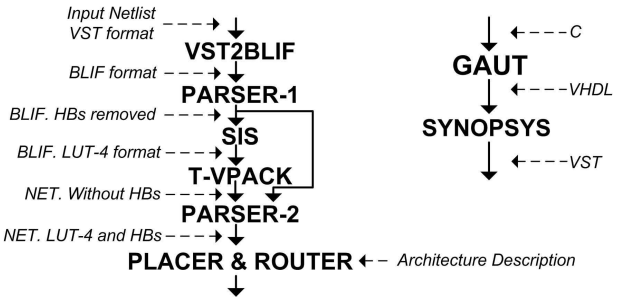


Figure 2. Software Flow

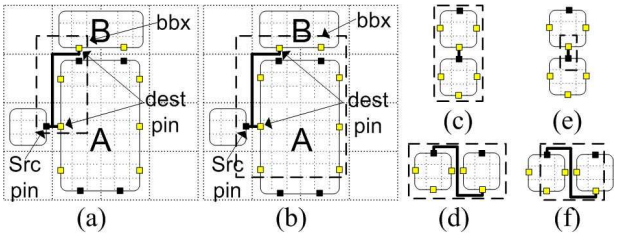


Figure 3. Signal bounding box evaluation

4 inputs) format. All the dependence between the HBs and the remaining netlist is preserved by adding new Input and Output pins to the main netlist. After SIS, and later after the packaging and conversion of the netlist to NET format through T-VPACK, PARSER-2 adds all the removed HBs into the netlist. It also removes all the previously added temporary inputs and outputs. The final netlist in NET format, and the architecture description parameters are passed to the PLACER and ROUTER. In future instead of SIS, we intend to use ABC [1].

This work also uses GAUT [2] to generate VHDL netlist from C Code. GAUT has an extensive library of blocks that it uses in the generated vhdl code. The current limitation of GAUT is that the widths of the input and output pins of a block should be equal. Thus a 16 bit multiplier has two 16 bit inputs and one 16 bit output (instead of a 32 bit output).

3. The Placer

3.1 Bounding box formation

The PLACER uses simulated annealing algorithm [4] [9] to achieve a placement having the minimum sum of the half-perimeters of the bounding boxes of all the nets. The bounding box (bbx) of a signal or a net is the minimum rectangular area that contains the driver instance and all the receiver instances of the net. In this work the position and direction of pins are also considered in the formation of bounding box. Similarly all the input pins of a SITE having same class are also included in the bbx. Thus the definition of the bounding box used in this work is the minimum rectangular area that contains the driver pin and the receiver pins of the net, and

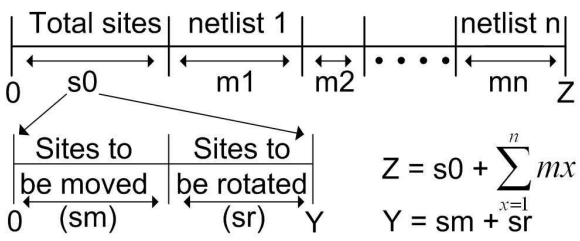


Figure 4. Source Selection

all the input pins of a SITE having the same class as that of the receiver pin of the SITE connected to the net. Figure 3(a) shows a case in which all the input pins of SITE 'A' have different class, whereas in Figure 3(b) all its input pins have same class. The size of the bounding box actually increases in 3(b) as compared to 3(a). This increase does not matter, as we are concerned with improving the relative cost and not the absolute cost. The driver instance targeting such a receiver pin of a SITE (having other peer pins of same class) will be having multiple placement options for achieving the same placement cost.

Figure 3(c) and Figure 3(d) show two cases in which the bounding box is formed without considering the pin positions and directions. In both cases, the bbxs are equally sized, however the placement in Figure 3(c) require lesser number of routing wires than in Figure 3(d). Whereas Figure 3(e) and Figure 3(f) show the same two examples in which bbx is formed using pin positions and their directions. It can be seen from the sizes of bbx that the placement in Figure 3(e) is to be preferred over the placement shown in Figure 3(f).

3.2 Placer Operations

The PLACER either moves an instance from one SITE to another, moves a SITE from one slot position to another, or rotates a SITE at its own axis. After each operation, the placement cost is recomputed for all the disturbed signals. Depending on the cost value and the annealing temperature, the simulated annealing algorithm accepts or rejects the current operation.

Source selection:- The placer performs operations on "source" and "destination". The "source" can be an instance selected from any of the input netlist or a SITE selected from the group of SITES found in the architecture. The probability of selecting a "source" from any of the source group should be proportional to the ratio of the elements in that group to the sum of the elements of all the groups. Thus all the SITES, and the instances of all the netlists are linearly arranged and given a unique ID as shown in figure 4. Z is the sum of the SITES in the architecture that can be moved or rotated, and the instances found in all the netlists. A random number selected from Z decides if the operation is to be performed on a SITE or on an instance of any netlist. If the selected random number is an instance, then the instance of the netlist referred by this random number is chosen as the "source". If it is a SITE, then another random operation will decide as to which SITE should be moved or rotated. Thus all the sites are linearly arranged with Y being the sum of all the sites that can be either moved or rotated. The IO SITES can neither be moved nor rotated. The remaining SITES in the architecture can be moved, and are represented as "sm". The rotation is allowed only for those SITES which have allowed so in the architecture definition of their respective blocks, and such sites are represented as "sr". It is to be noted that Y is not equal to "s0" because there can be sites which can be moved as well as rotated. Thus a random operation on Y selects a SITE to be moved or rotated.

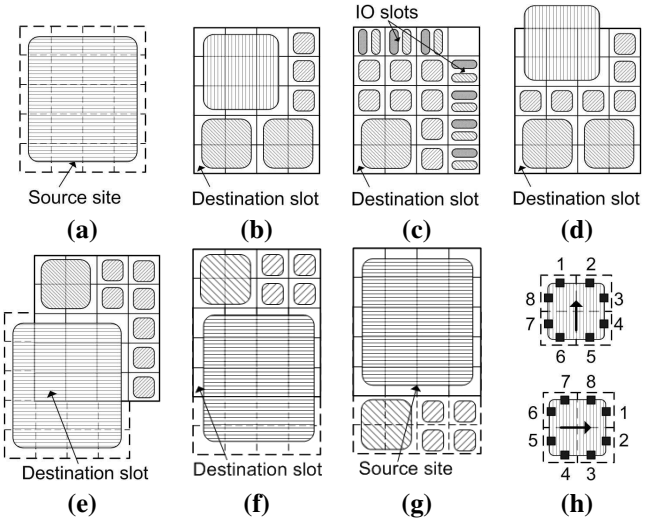


Figure 5. Site Movement Cases

Destination selection:- Once the "source" is selected, a destination is selected where this "source" can be moved. If the source is an instance, then any random matching SITE is selected as its destination. If the source is a SITE to be rotated, the same source position becomes the destination.

If the source SITE is to be moved then any random slot is to be selected as the destination where this SITE can be moved. This destination slot selection is done using the following steps

1. Get the size of the source site. The rectangular window occupied by the source site is called as the source window. The source window depicted in dashed line is shown in figure 5(a).
2. Choose any random slot as destination. The rectangular window starting from this slot, having the same size as the source window is called as the destination window. The source window will always contain a single SITE, whereas the destination window can contain one or more SITES. A valid destination window depicted in solid line is shown in figure 5(b).
3. If the destination window exceeds the boundaries of the

Block Name	Netlists					Block Sizes					
1	2	3	4	5	6	7	8	9	10	11	12
	Fir	Fft	Adac	Dcu	Target-1	Block Size Lamda^2	Inputs	Outputs	No. of slots for ch $>= 9$	Block Size/Shape	No. of shadow CLBs for ch=11
clb	32	94	47	34	94	58500	4	2	1	1x1	-
mul_8_8_16	4	4	-	1	4	1075250	16	16	9	3x3	1.86
slansky_16	3	3	-	1	3	306750	32	16	8	2x4	2.75
sff_8	4	-	2	4	4	36000	8	8	3	2x2	6.5
sub_8	-	6	-	2	6	154500	17	8	4	2x2	0.87
smux_16	-	-	1	2	2	36000	33	16	8	2x4	6.89
	Fir16	Prodmat	Ellipticass		Target-2						CLB4 for ch=22
clb4 (4 clbs)	572	1112	818	-	1112	798000	10	4	1	1x1	-
add_16_16_16	8	11	15	-	15	106750	32	16	3	2x2	4.15
mul_16_16_16	16	27	-	-	27	1908750	32	16	4	2x2	1.89

Table 1. Netlist block utilisation table

slot-grid (as shown in Figure 5(c)) , then reject this destination slot.

4. If the source and the destination windows do not overlap, and if the destination window contains at least one such site which exceeds the limits of the destination window, then reject this slot. Figure 5(d) depicts this case. This is done because in such a case, it would not be possible to move all the destination sites to the source window.
5. If the source and the destination window overlap diagonally (as shown in Figure 5(e)), then reject the destination slot.
6. If the source and the destination windows overlap horizontally or vertically, then accept this destination slot position. Horizontal or vertical translation operation will be applied in these cases. Figures 5(f) shows a valid destination window overlapping vertically with the source window. Figures 5(g) shows the positions of source and destination SITES after vertical translation.
7. If a slot is rejected then the procedure is repeated until a valid destination slot is found.

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

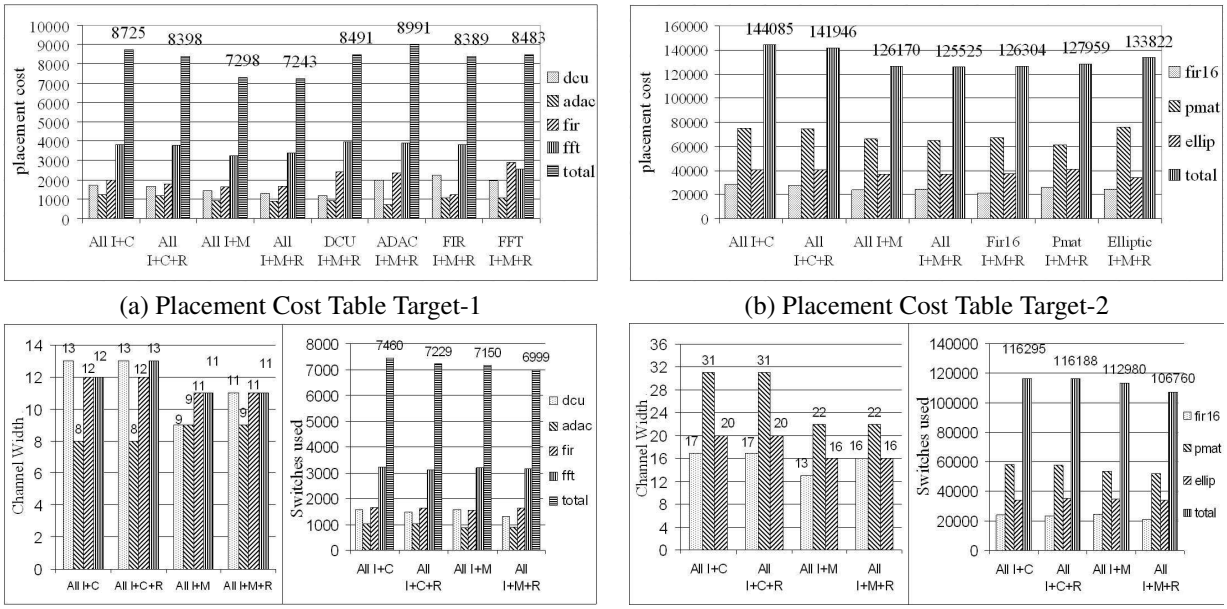
Site Jump: If the source window does not overlap with the destination window, then a JUMP operation is performed. All the SITES in the destination window are moved to the source window, and the source site is moved in the destination window. Each affected SITE breaks its link with the current slots and connects with new slots and vice-versa.

Site Translate: If the source and the destination windows overlap, then a translation is performed. Currently only the horizontal or vertical translation is performed. No diagonal translation is performed. Figure 5(f) and 5(g) show a case of vertical translation. The five sites found in the upper 2 rows of the destination window (as shown in Figure 5f) are moved to the lower 2 rows of the source window (as shown in Figure 5g). The source site is then moved to the destination window.

Site Rotate: The rotation of SITES is important when the classes assigned to each of its pins are different. In such a case the bounding box varies depending upon the pin positions and their directions. A SITE can have an orientation of 0°, 90°, 180° or 270°. The orientation of a SITE is used by the bounding box 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 SITE to another SITE having different orientations, the orientation of both the old site and the new SITE 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 SITEs having a square shape, whereas at the moment only multiples of 180° rotation are allowed for rectangular (non-square) SITEs. A 90° rotation for non-square SITEs involves both rotation and move operations; which is left for future work.

4. Experimentation

Two set of test-benches are used to show the effectiveness of various techniques used in this work. The first set comprise of 4 benches which are a subset of 8-bit fir, fft, adac, and dcu algorithms. These are relatively small benches, and are obtained from their generators written in a procedural language. The second set of 3 benches are generated from GAUT [2]. These are relatively large benches. The CLBs in this set are grouped into a cluster of 4 CLBs connected through a full-crossbar. Each CLB is a 4 input LookUp Table (LUT). All these benches and their block classifications can be seen in Table 1. In the first step, the sizes of the blocks used in the benches are determined using an area model. Later these blocks are defined in the architecture description file and two separate target architectures, Target-1 and Target-2, are defined for these two set of netlists. The block



(a) Placement Cost Table Target-1

(b) Placement Cost Table Target-2

(c) Routing Results Target-1

(d) Routing Results Target-2

Figure 6. Comparison results for 2 target Fpga architectures.

classifications of these architectures can be seen in column 6 of Table 1. Different placement/floor-planning techniques are then applied upon them to refine the architectures for the given netlists.

4.1 Area Model

The area model is devised to determine (i) the area of a single slot in the slot-grid and (ii) the number of slots required by each hard block. This model is derived from a previous work on tile-based FPGA Layout generation methodology [11]. So in this work each slot is considered to be a single tile containing the block, the connection boxes (for connecting inputs and outputs with the routing channel), the routing channels on its top and right side, and the switch box of the top right corner. Thus the area of a slot depends on (i) the area of a block (ii) the total number of inputs and outputs of the block and (iii) channel widths. The basic sizes of the blocks (without connection boxes and routing channel) are measured in ALLIANCE [7] using a symbolic standard cell library 'SXLIB'. These sizes are shown in the 7th column of Table 1. The smallest block in the target architecture is made equivalent to be occupying 1 slot. Considering the basic block sizes and the total number of input and output pins of blocks, the smallest block in Target-1 architecture is a 'clb' for a channel width equal or greater than 9. Fewer channel widths give some false advantage to blocks like smux_16 and sff_8 whose block sizes are very small but have high number of IOs. Once the size of a single slot is determined, the sizes of all the remaining blocks are measured in units of number of slots occupied. Column 10 of Table 1 shows the slots required by each block for a channel width equal to or greater than 9. There is certainly some wastage of area when the blocks are represented in terms of number of slots occupied. Similarly while deciding the shapes of the blocks, the user might require to further increase the area. Like for the sake of experimentation, we have made sff_8 and add_16_16_16 a square (i.e instead of 1x3, a size of 2x2 is used). The final sizes and shapes of blocks are shown in column 11 of Table 1. The free area in each block is also reported by the area model. This free area can be used to integrate shadow clusters [8] so that the precious routing resources can be reutilized if the hard block is not used. Column 12 of the table shows the number of CLBs that can be added in the free space in each block for channel width of 11. The experimental analysis with shadow clusters is left for future work. All the above procedure is repeated for Target-2 architecture also where the single slot is a cluster of 4 CLBs.

4.2 Architecture Evaluation

Firstly an initial architecture (I) is generated in which all the hard blocks are concentrated on the left, whereas the clbs are on the right side of the FPGA. To be a little more realistic the hard blocks are later moved to the center with columns of clbs distributed on its left and right sides. The centered initial architecture (I+C) floor-planning for Target-1 can be seen in figure 7(a). All the netlists are simultaneously mapped on the architecture and the floor-planning is refined using different algorithms. Once the final floor-planning is achieved each netlist is individually placed and routed on it (without changing the floor-planning). A set of 4 different floor-plannings are generated. (1) The centered initial architecture shown in figure 7(a) (I+C). (2) Only the rotation of blocks allowed

in the centered initial architecture (I+C+R). (3) Only block movement allowed (I+M). (4) Both the block movement as well as the rotation of all the blocks (I+M+R). An average reduction of 17% and 13% is noticed in the placement cost of all the netlists between the initial architecture and the I+M+R for Target-1 and Target2 architectures respectively. The optimized Target-1 architecture is shown in figure 7(b). The arrows in the figure show the orientations of the blocks. Figure 6(a) and 6(b) show the placement results for both target architectures. Another important test compares the placement costs of each of the netlists mapped on an architecture optimized for all the netlists together, or for each of the individual netlists. The sum of the placement costs of all the netlists is found to be minimum if all the netlists are used together to get a single floor-planning. Figure 6(c) and 6(d) show how the required channel width and total number of switches used in the routing have reduced significantly with each technique. The decrease in channel widths and the total switches used show that the improvement in the placement costs have well shown their effect.

The implementation of our PLACER module is not yet optimized, but still we feel it important to give some initial run-time results. It is found that the time taken to optimize a common floor-planning for the three netlists for Target-2 I+M+R architecture is on average 8 times more than the total time taken to place the three netlists individually on the Target-2 I+C architecture (having fixed floor-planning). This factor can be significantly reduced by exploiting parallelism in the PLACER. The bounding box formation is independent for different netlists as the nets are not shared between the netlists. Thus, operations on netlist instances can be performed parallely on different machine. However the operations on SITES are to be performed serially as multiple instance of different netlists can be found on the same SITE. We intend to perform a detailed run-time optimization and analysis in future.

5. Conclusion and Future Work

In this paper we have presented new techniques and tools for the conception of coarse-grained FPGAs. This exploration environment can be used to develop application specific FPGAs optimized for a given set of netlists. It has been found that the best possible trade-off architecture floor-planning is achieved when all the netlists are optimized together. In future we intend to gather and test much larger test benches containing coarse-grained components. The SITE movement techniques need to be improved i.e. 90° and 270° rotation for rectangular (non-square) sites, diagonal translation, and finally considering "source window" having multiple sites. The suggested method of reutilizing the free space in each hard block by the addition of shadow clusters need to experimentally analyzed. Finally we intend to target the existing commercial FPGA and judge their floor-planning with this environment.

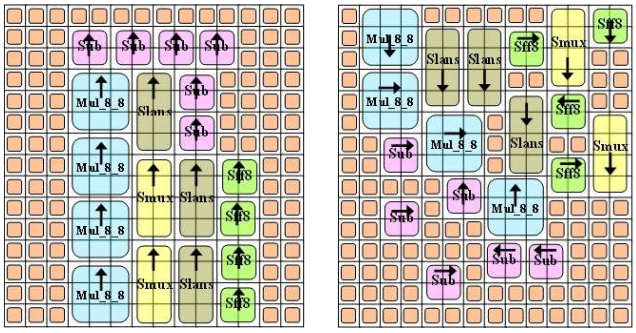


Figure 7. Optimized FPGA Floor-planning

In this work we have optimized only the floor-planning of an FPGA for a set of applications. This work can also be extended towards optimizing the reconfigurable routing channel for the set of input netlists.

References

[1] Berkeley logic synthesis and verification group, ABC: A system for sequential synthesis and verification. *http://www.eecs.berkeley.edu/ alanmi/abc/*.

[2] GAUT - high-level synthesis tool from C to RTL, www-labsticc.univ-ubs.fr/www-gaut/.

[3] M. Beauchamp, S. Hauck, K. Underwood, and K. Hemmert. Embedded floating-point units in fpgas. *International Symposium on Field Programmable Gate Arrays (FPGA)*, pages 12–20, 2006.

[4] V. Betz, A. Marquardt, and J. Rose. *Architecture and CAD for Deep-Submicron FPGAs*. January 1999.

[5] C.H.Ho, P.H.W.Leong, W.Luk, S.Wilton, and S.Lopez-Buedo. Virtual embedded blocks: A methodology for evaluating embedded elements in fpgas. *Proceeding of FCMM*, pages 35–44, 2006.

[6] K. Compton and S. Hauck. Automatic design of area-efficient configurable asic cores. *IEEE Transaction on Computers*, pages 662–672, 2007.

[7] A. Greiner and F. Pecheux. Alliance: A complete set of cad tools for teaching vlsi design. *3rd Eurochip Workshop*, 1992.

[8] P. Jamieson and J.Rose. Enhancing the area-efficiency of fpgas with hard circuits using shadow clusters. *IEEE FPT*, pages 1–8, 2006.

[9] Kirkpatrick, Gelatt, and Hecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[10] G. Lemieux, E. Lee, M. Tom, , and A. Yu. Directional and single-driver wires in fpga interconnect. *IEEE Conference on FPT*, pages 41–48, 2004.

[11] H. Parvez, H. Mrabet, and H. Mehrez. Generic techniques and cad tools for automated generation of fpga layout. *PRIME*, pages 144–144, 2008.

[12] S.Dai and E.Bozorgzadeh. Cad tool for fpgas with embedded hard cores for design space exploration of future architectures. *FCCM*, pages 329–330, 2006.

[13] C. Yu. A tool for exploring hybrid fpgas. *Proceeding of FPL PhD forum 2007*, pages 509–510, 2007.