A New Coarse-Grained FPGA Architecture Exploration Environment

Husain Parvez, Zied Marrakchi, Umer Farooq, Habib Mehrez
LIP6, Université Pierre et Marie Curie
4, Place Jussieu, 75005 Paris, France
parvez.husain@lip6.fr

Abstract

This paper presents an exploration environment for the design of 2D island-style coarse grained FPGA architectures. An architecture description file defines various architectural parameters including the definition of new coarse grained blocks, the positioning of blocks in the architecture and the selection of routing network. Once the initial architecture is defined, a software flow places and routes a target netlist on the generated architecture. The placement cost of a netlist is optimized either by changing the position of netlist instances on its respective blocks or by changing the position of blocks on the architecture. A single FPGA architecture can also be obtained for mapping a set of netlists at mutually exclusive times. It has been found that the sum of the placement costs of all the netlists is found to be minimum if all the netlists are used to get a single architecture. A set of DSP test-benches is used to show the effectiveness of the various techniques used in this work.

1. Introduction

This work proposes a new environment for the exploration of coarse-grained FPGAs. Previously VPR [3] (Versatile Place and Route) has been extensively used for the exploration of fine grained FPGAs. Inherently it does not support coarse grained blocks. However [2] and [8] have extended VPR to explore specific coarse grained architectures. In the same context, [5] has developed the virtual embedded block methodology (VEB) to model arbitrary embedded blocks on existing commercial FPGAs. Later [12] has incorporated VEB methodology in VPR, thus enabling the support of architectures other than commercial FPGAs. [10] 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 finding the block positions in the architecture. Our major contribution consists of proposing an environment that refines the FPGA floor-planning.

2. Exploration Environment

The basic working ground of the exploration environment is a grid of equally sized SLOTS. CELLS of different sizes can be mapped on this grid. 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. A CELL when mapped on the slot-grid is referred to as 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. A software flow maps the instances of a 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 an application specific FPGA floor-planning for binding a set of netlists on it at mutually exclusive times. This technique of placing a set of netlists is proposed in [6], where it is used to explore configurable ASICs in a single dimension. This work extends the same methodology to explore two dimensional island-style FPGA architectures. After floor-planning and binding, the ROUTER routes the netlist on the architecture.

Architecture Description: An architecture description file is used to define the complete architectural details. Nx and Ny define the size of the slot-grid. The channel width of the routing architecture is either fixed to a value W, or a binary search algorithm searches the minimum possible channel width between the minimum (Wmin) and the maximum (Wmax) channel widths. The position of SITES can be either fixed to an absolute position on the slot-grid, or it can be set as variable so that the PLACER can modify it lateron. Each type of CELL is defined in the architecture description file. The pins of the CELL are given a name, a class number, a direction and the slot position on the CELL to which this pin is connected. Pins having the same class are considered equivalent. Thus a driver net targeting a receiver pin of a SITE (CELL instance) 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 be assigned to any of the internal channels also. The routing network

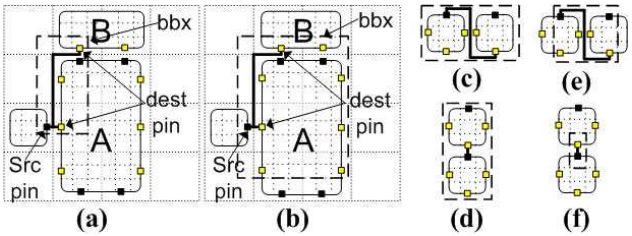


Figure 1. Signal bounding box evaluation

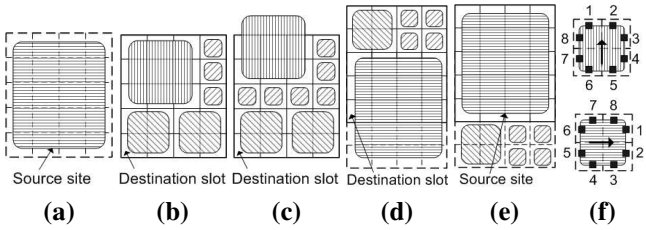


Figure 2. Site Movement Cases

supported by the environment can be either a bidirectional mesh [3] or a unidirectional mesh [9].

Software Flow: Once the architecture is properly defined, a software flow maps the netlists on it. The input to this software flow is a structural netlist in VST (structured VHDL) format. This netlist is composed of the traditional standard cell library instances and the hard block instances. VST2BLIF tool is modified to convert the VST file having hard blocks to BLIF format. Later a PARSER removes all the instances of hard blocks and passes the remaining netlist to SIS [11] for synthesis into 4-LUT format. All the dependence between the hard blocks and the remaining netlist is preserved by adding new input and output pins to the main netlist. After SIS, and later by the conversion of the netlist to NET format through T-VPACK [4], another parser adds all the removed hard blocks into the netlist. It also removes all the previously added inputs and outputs. This final netlist in NET format, containing LUTs and hard blocks, is passed to the PLACER and the ROUTER. In future instead of SIS, we intend to use ABC [1].

3. The Placer

3.1 Bounding box formation

This work considers the position and direction of pins in the formation of bounding box (bbx). All the input pins of a SITE having same class are also included in the bbx. Thus the bounding box of a net used in this work is the minimum box which encompasses the driver and the receiver pins of the net, and all the input pins of a SITE having the same class as that of the receiver pin of the SITE connected to the net. The PLACER tries to achieve a placement having the minimum sum of the half-perimeters of the bounding boxes of all the nets. Figure 1(a) shows a case in which all the input pins of SITE 'A' have different class, whereas in Figure 1(b) all its input pins have same class. The size of the bounding box actually increases in 1(b) as compared to 1(a); but this increase does not matter. The driver instance targeting such a receiver pin (having other peer pins of same class) will be having multiple placement options for achieving the same placement cost.

Figure 1(c) and Figure 1(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. Whereas Figure 1(e) and Figure 1(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 1(f) is to be preferred over the placement shown in Figure 1(e).

3.2 Placement Move Generation

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 nets. Depending on the cost value and the annealing temperature the operation is accepted or rejected. Multiple netlist files can be placed together to get a single architecture floor-planning for all the netlists. With multiple netlist placements, each SITE can allow multiple instances to be mapped on it; but multiple instances of the same netlist cannot be mapped on a single SITE.

The PLACER performs an operation on a "source" and a "destination". The "source" is randomly selected to be either an instance from any of the input netlists or a SITE from the architecture. 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 as well.

If the source SITE is to be moved, a random slot is selected as its destination. The rectangular window starting from this destination slot, having the same size as the source SITE is called as the destination window, whereas the rectangular window occupied by source SITE is called as source window. The dashed line in Figure 2(a) is the source window, and the solid line in Figure 2(b) depicts a valid destination window. The source window will always contains a single SITE, whereas the destination window can contain multiple SITES. The destination slot is rejected if (i) the destination window exceeds the boundaries of the slot-grid, (ii) the destination window contains atleast one such SITE which exceeds the limits of the destination window (as shown in Figure 2(c)) and (iii) the source window overlaps the destination window diagonally (i.e partially horizontal, partially vertical overlap). 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, a JUMP operation is performed. All the sites in the destination window are moved to the source window, and the source site is moved into the destination window. Each affected SITE breaks its link with the current slots and connects with new slots.

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

Site Rotate: The rotation of SITES is important when the classes of each of its pins are different. In such a case the bounding box varies depending on the pin positions and their directions. Multiples of 90° rotation are allowed for all the SITES having a square shape, whereas only 180° rotation is allowed for rectangular (non-square) SITES. A 90° rotation for rectangular SITES involves both rotation and move operation; which is left for future work. The orientation of SITE is used by the bounding box cost function to correctly calculate the exact position and direction of each of its PINS. Figure 2(f) depicts a 90° clock-wise rotation.

4. Experimental Results

In this work a set of 4 testbenches is used to design an FPGA. The goal is to generate a single coarse-grained FPGA floor-planning for the given testbenches. In the first step, these benches are experimentally analyzed to determine which components should be made as hard blocks. The selected blocks are defined, and different placement and floor-planning techniques are applied on them to minimize the overall placement cost. The total switches in the architecture and the switches used for routing are measured with bi-directional and uni-directional routing channels.

Netlist generation: The netlists are extracted from generators written in a procedural language. These netlists are generated in a modular fashion which makes it much easier to extract different blocks. Only those blocks are selected as hard blocks which are used at least by 2 netlists. The remaining blocks are converted to soft blocks (CLB) using the software flow. The set of blocks used by each of the netlists are shown in Table 1. The final target architecture, as shown in the last row of Table 1, contains the maximum number of blocks required by any of the netlist. Once the blocks are identified, the sizes of each of these blocks are measured and then compared with a CLB. This is done by placing and routing them in standard cell library using ALLIANCE [7]. Table 2 shows the related area information. This information is used to decide the size of each block on the slot-grid. Another major factor in deciding the sizes of these blocks is their pin-demand. An imbalanced pin-demand gives rise to under utilization of routing resources in a uniform routing channel. Thus, in this work all hard blocks are defined with routing channel passing through them. The pins are not only limited to the boundaries of the blocks, but are also attached to the routing channel passing through them. The sizes of the hard blocks are increased to make them rectangular shaped, as well as to balance their pin demand; causing some area loss. One method to overcome this area loss can be to employ the shadow cluster technique [8]. The empty space can be filled by shadow CLBs that can be used if the hard block is not used.

	clb	mul_8_8_16	slans_16	sff_8	sub_8	smux_16_1_cmd
FIR	32	4	3	4	-	-
FFT	94	4	3	-	6	-
ADAC	47	-	-	2	-	1
DCU	34	1	1	4	2	2
Target	**94**	**4**	**3**	**4**	**6**	**2**

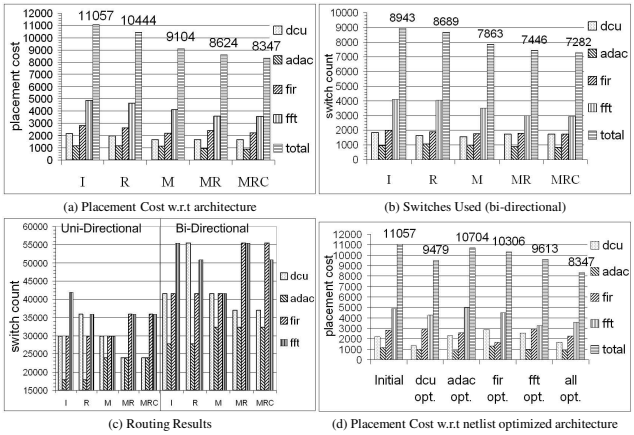
Table 1. Netlist block utilisation table

	Size Lamda^2	**Ratio w.r.t clb**	**MB Size**
clb	280x250	1	1x1
slans_16	615x600	5.27	3x3
sff_8	215x200	0.61	2x2
sub_8	410x450	2.63	2x2
smux_16_1_cmd	215x200	0.61	3x3
mul_8_8_16	1120x1150	18.4	4x5

Table 2. Block size table

Architecture evaluation: From the required number of blocks of each type an initial architecture is generated by firstly placing all the hard blocks and then placing the CLBs. The initial architecture floor-planning achieved is shown in the Figure 4(a).

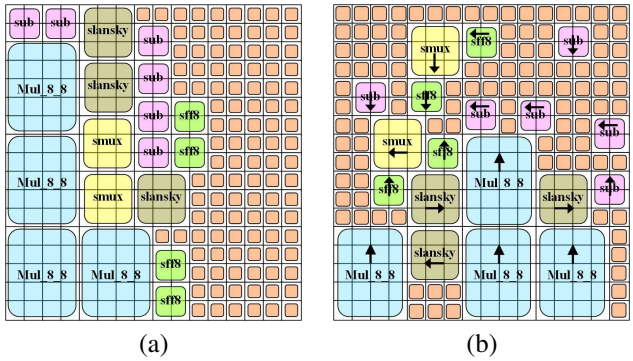
All the netlists are mapped together on the architecture and different floor-planning techniques are tested. Once the final floor-planning is achieved each netlist is individually placed and routed on it (without changing the floor-planning). A set of 5 different combinations of floor-planning are generated. (1) The initial architecture shown in Figure 4(a) (I). (2) Only the rotation of blocks allowed on the initial architecture (R). (3) Only block movement allowed (M). (4) Both the block movement as well as the rotation of all the blocks (MR). (5) The block movement and the rotation of all the blocks allowed (except the CLBs) (MRC). Each netlist is individually routed on these architectures with both the bi-directional and uni-directional routing channel. Figure 3(a) shows how the placement cost



(a) Placement Cost w.r.t architecture (b) Switches Used (bi-directional)

(c) Routing Results (d) Placement Cost w.r.t netlist optimized architecture

Figure 3. Comparison results



(a) (b)

Figure 4. Optimized FPGA Floor-planning

of each netlist reduces with different floor-planning architectures. An average reduction of 25% in the placement cost of all the netlist is noticed between the initial architecture and the MRC. The MRC architecture is shown in Figure 4(b). The arrows in the figure show the orientations of the blocks. Since all the four input pins of CLBs have the same class, and the single output pin is found both on the right and top side, the rotation of CLB is not of greater advantage in reducing the placement cost. Rather turning off its rotation gives higher probability of movement/rotation to other hard blocks. That is why a slight benefit in the placement cost is noticed with no rotation of CLBs. Figure 3(b) shows how the total switches used for routing reduce with each floor-planning. Figure 3(c) shows the total number of the switches in the architecture. These switches are proportionate to the number of routing channels required to route a netlist. It has been noted that the minimum number of switches among all the netlists are found with the architecture (M). Moreover, the uni-directional routing network takes on average 28% less number of switches than the bi-directional network.

Figure 3(d) compares the placement costs of the netlists on the architectures whose floor-planning is optimized either for individual netlists (netlist_name opt.) or for all the netlists together (all opt.). The floor-planning optimized for an individual netlist gives relatively better results only for that particular netlist, but worse results for all the remaining netlists; thus increasing the total placement cost of all the netlist. Whereas the sum of placement costs of all the netlists is found to be minimum with the "all opt." floor-planning. However the placement cost of a single netlist on "all opt." floor-planning is on average 17% higher than on the floor-planning optimized for that particular netlist.

5. Conclusion and Future Work

In this paper we have presented an exploration environment 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. In future we intend to gather and test more 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 hard blocks whose sizes have been increased for balancing the pin demand produce some free space. This space can be used to incorporate shadow clusters in those blocks. Finally we intend to judge the floor-planning of existing commercial FPGAs using this environment.

References

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

[2] M. Beauchamp, S. Hauck, K. Underwood, and K. Hemmert. Embedded floating-point units in fpgas. *FPGA*, 2006.

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

[4] V. Betz and J. Rose. VPR: A New Packing Placement and Routing Tool for FPGA research. *International Workshop on FPGA*, pages 213–22, 1997.

[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. *FCMM*, 2006.

[6] K. Compton and S. Hauck. Automatic design of area-efficient configurable asic cores. *IEEE Transaction on Computers*, 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. *FPT*, 2006.

[9] G. Lemieux, E. Lee, M. Tom, , and A. Yu. Directional and single-driver wires in fpga interconnect. *FPT*, 2004.

[10] S.Dai and E.Bozorgzadeh. Cad tool for fpgas with embedded hard cores for design space exploration of future architectures. *FCCM'06*, 2006.

[11] E. M. Sentovich and al. Sis: A system for sequential circuit analysis. *Tech. Report No. UCB/ERL M92/41, University of California, Berkeley*, 1992.

[12] C. Yu. A tool for exploring hybrid fpgas. *FPL*, 2007.