Generic Techniques and CAD tools for automated generation of FPGA Layout

Husain Parvez, Hayder Mrabet and Habib Mehrez

Laboratoire d'informatique de Paris 6; Université Pierre et Marie Curie, 4 Place Jussieu, 75005 Paris, France
E-mail : {parvez.husain, hayder.mrabet, habib.mehrez}@ lip6.fr

Abstract— **This paper presents an automated method of generating an FPGA layout. The main purpose of developing a generator is to reduce the overall FPGA design time with limited area penalty. This generator works in two phases. In the first phase, it generates a partial layout using generic parameterized algorithms. The partial layout is generated to obtain a fast bitstream configuration mechanism, an efficient power routing and a balanced clock distribution network. In the second phase, the generator completes the remaining layout using automatic placer and router. This two-phase technique allows better maneuvering of the layout according to initial constraints. The proposed method is validated by generating the layout of an island-style FPGA which includes hardware support for the mitigation of Single Event Upsets (SEU). The FPGA layout is generated using a symbolic standard cell library which allows easy migration to any layout technology. This layout is successfully migrated to 130nm technology.**

1. INTRODUCTION AND RELATED WORK

Developing a new FPGA is a time consuming and a challenging task. It is reported in [3] that a new FPGA creation involves approximately 50 to 200 person years, thus increasing the overall time to market of the final product. It is an interesting option to significantly reduce the time-to-market of the product at the expense of limited area penalty. One way to do this is by automating the complete FPGA design process. The work presented here discusses the automatic generation of FPGA layouts using open-source VLSI tools.

The generator presented here employs an elegant scheme to integrate manual intervention in the automated FPGA generation procedure. This is done with the help of generic parameterized algorithms which generate a partial layout. Later, the automated tools are used to complete the remaining layout. The partial layout is performed on those portions of the design that are either important in one aspect or other, or are too difficult to be handled properly by the automated tools. In this work the partial layout performs the power routing, clock distribution and the configuration memory placement.

A number of previous attempts have been made regarding automated generation of FPGAs. One of the major works in this domain is done in [6] [3]. They have demonstrated the complete automation of FPGA creation with significantly reduced manual labor. The GILES [3] tools are used to generate different tiles which are then abutted together to form a complete FPGA. The clock and power segments are later routed using SKILL [7]. Phillips and Hauck have focused on the automatic layout of domain specific reconfigurable systems [5]. They have reduced the amount of configurability required by an application domain, and thus have generated smaller layouts.

These previous FPGA generators have used the commercial VLSI tools; whereas this work presents an FPGA generator based solely upon open-source VLSI tools. These tools can be adapted easily for specific demands. This work also defines a set of layout parameters to modify the layout according to the initial requirements.

2. FPGA GENERATION

This work focuses on the generation of Island style FPGAs. It comprises an array of configuration logic blocks (CLBs). Each CLB contains a 4-LUT followed by a by-pass flip-flop. Each CLB has 4 inputs (one on each side) and an output that derives adjacent channels on its top and right sides. The CLBs communicate with each other through a disjoint bi-directional routing network. All the inputs and outputs of a CLB connect with all the wires in a channel (i.e. Fc=1). The generated FPGA matrix can have '*Nx'* CLBs in X direction, '*Ny'* CLBs in Y direction, and a channel width '*Ch'*.

A. *Open-source VLSI tools:* An open-source VLSI tool kit ALLIANCE [1] and a python based language STRATUS [2] has been used for the development of this FPGA layout generator. Alliance is a complete set of free CAD tools and portable CMOS libraries for VLSI design. It includes a VHDL compiler and a simulator, logic synthesis tools, and automatic place and route tools. STRATUS generates parameterized VLSI modules. It extends the python language with a set of methods and functions for the procedural generation of netlist and layout views of structural cell based designs.

B. *Tile based approach:* A tile based approach is used to generate the desired FPGA architecture. In this approach a set of tiles are identified in the architecture which are repeatedly abutted to form the whole FPGA matrix. A set of 9 different tiles as shown in figure 1 are used for the generation of the target architecture. The principle tile is the 'basic' tile, whereas the other tiles are its derivations. The tiles on the leftmost column and the bottom row do not contain logic blocks. They only contain a channel which connects the adjacent IO pads and the adjacent logic block input. The rest of the tiles contain a top horizontal channel, a right vertical channel, a switch box, and a logic block. It can be seen in figure 1 that the horizontal repetition of 2nd column and the vertical repetition of 2nd row generate an FPGA of our desired size. An important aspect in the tile based design is that the adjacent sides of two abutted tiles must have same

length. While deciding the sizes of the tiles, priority is given to the tile which is used the most; in this case it is the 'basic' tile. The sizes of the other tiles are adjusted accordingly.

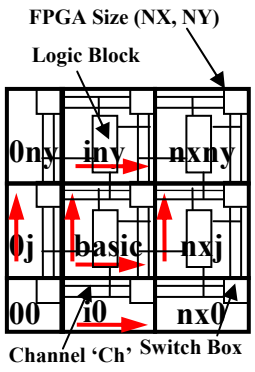

Fig. 1 –Tiles for island-style FPGA

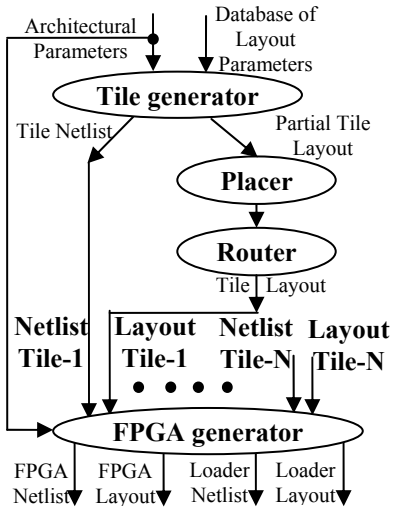

Fig. 2 – Complete FPGA generation CAD Flow

C. Netlist generation: Each tile generator is written in the language STRATUS. The tile generator receives a set of architectural parameters as input. It then generates the netlist of the tile in accordance with the given parameters. Loops and conditional statements are used to generate a tile for different parameters. The netlist of each tile is generated directly using the standard cell library named SXLIB. It is a symbolic cell library which comes with the ALLIANCE tool chain. C++ routines are also merged in the tile generator for generating VHDL model of specific components. These components are synthesized by the Alliance synthesizer named BOOG. After synthesis these components are used by the tile generator. The generated netlists of all the tiles are passed to the FPGA generator which links them together to construct the netlist of a complete FPGA. This generated netlist may be integrated in any larger application.

D. Tile Layout: A tile generator generates both the netlist and the partial layout of a tile. The partial layout is generated with the help of parameterized algorithms which take a set of layout parameters as its input. Currently, the partial layout is performed for the generation of a fast bitstream configuration mechanism, proper buffering of few long wires, power routing and a balanced clock distribution network. Later-on the placer and the router are used to complete the remaining layout.

The partial layout generation algorithm places all the SRAM bits in rows and columns with a fixed distance between each row, as shown in figure 3. Each SRAM bit in a row receives a vertical data signal, and a horizontal strobe signal. The data bits are written in all the SRAMs of a row only when strobe is high for that row and the column is high for the complete tile. The column and strobe signals come from bitstream configurator (loader), which is discussed later in section 4. The column and the data signals from the top are buffered before they exit on the bottom side of the tile. Similarly a strobe signals from the left is buffered before it exits on the right side of the tile.

The algorithm starts placing the bitstream configuration cells from a layout parameter named "Start Position". Similarly the height and width of a tile and the total SRAM bits are also variable parameters which change for each different channel width. These layout parameters change each time there is a change in the number of SRAM bits. For this purpose a small database is created which specify all these variables for different channel widths. The layout algorithm and the database specification are generic enough to handle other architectural parameters that are not yet generic.

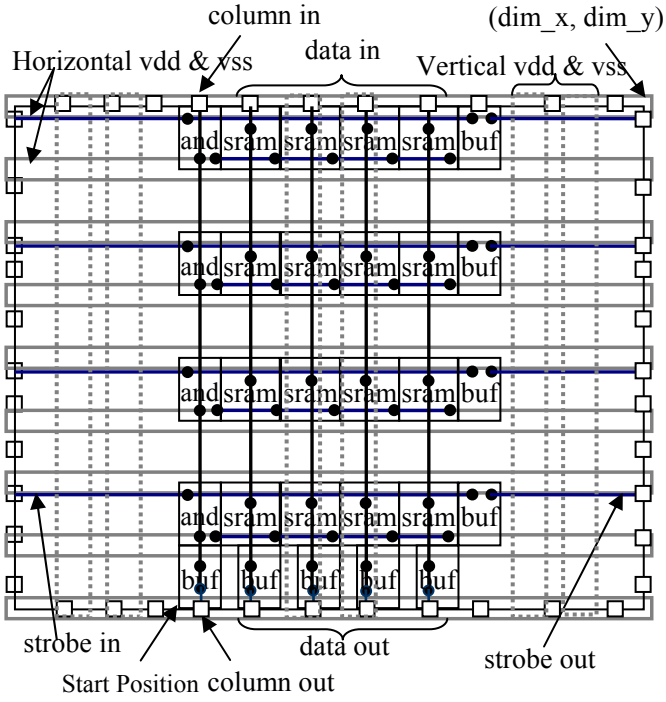

Fig. 3 –Partial layout of a sample FPGA Tile

E. *Power routing:* The layout generation algorithm generates horizontal and vertical power segments as shown in figure 3. The alternating VDD and GND segments in the horizontal direction are fixed whereas the placement of vertical power segments is supported by few layout parameters. The total number of vertical segments for power and ground in a tile, their positions and their widths are defined in the layout database. These values can be changed for tiles of different sizes. The horizontal power segments use the 1st and 2nd routing layer; whereas the vertical power segments use the 5th routing layer.

F. *Clock generation:* In this work, we have used a tile based approach for the routing of a symmetric H-tree clock distribution network. It is found that a group of 13 tiles can be used to generate a clock tree for a matrix of size 2^N x 2^N where N>1. Each corresponding clock tile is automatically merged with the FPGA tiles during the partial layout phase. This results in the generation of multiple copies of the same FPGA tiles having different clock routings. After the merging of FPGA tiles and the clock tiles, 23 different tiles are produced.

These tiles can be abutted together to construct any FPGA of size 2^N x 2^N. All these clock tiles and the sample 8x8 clock distribution matrix is shown in figure 4 and 5. The main advantage of this mechanism is that we have a generic, tile based and a balanced clock distribution network. One of the disadvantage is that it limits the FPGA size in X and Y direction to be equal and power of 2. But since the clock generation algorithms and their merging with the FPGA tiles is totally automatic; we can always implement a generic algorithm for other clock distribution networks found in [4]. The only thing to consider for writing a new clock routing algorithm is that the placement of clock buffers must not overlap the partial layout. Currently the clock is routed in the 5th and 6th routing layer, whereas the partial layout is done on the first 4 routing layers.

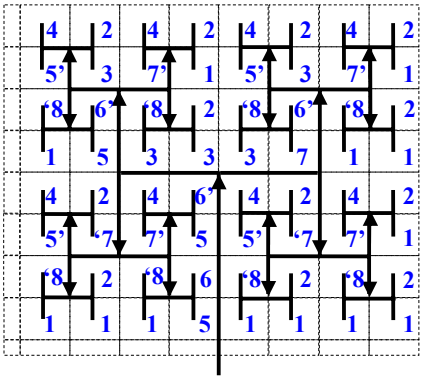

Fig. 4 – H-tree clock distribution network for 8x8 FPGA

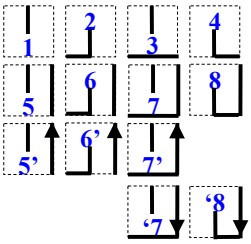

Fig. 5 – Tiles for constructing clock H-tree

G. *Pin generation:* In a tile based FPGA, the tiles connect together by abutment, and the pin locations on the boundaries of adjacent tiles must overlap. The positions of few of these pins are calculated on the basis of the layout parameters found in the database. Since the database is common for all the tiles, thus the pin abutment problem does not arise for these pins. There exist other pins which do not have fixed positions. Since the final automatic placement of all the tiles is done independently; it is difficult for the placer to correctly choose the pin locations of the tiles. So a generic algorithm is written to place all the remaining pins. This algorithm places the pins in all the four directions of the tile and ensures that the pins are not congested to a limited place. It utilizes all the available space and tries to distribute the pins with equal spacing.

H. *Automatic placement & routing:* After the partial layout generation of all the tiles; each tile is separately placed and routed with the help of ALLIANCE automatic placer and router named OCP and NERO respectively. The partial layout information is firstly given to the placer to place the remaining logic. If the placer is unable to place the design, the dimensions of the tile are manually increased in the database. The X and Y dimensions of the tile must be properly adjusted to make sure that a tile does not waste any extra space. The placer automatically adds the empty cells to fill up any extra space. After placement, NERO routes the whole design. All the tiles are successfully routed using 4 routing layers. Only the clock and the vertical power segments are routed on the 5 and 6 routing layer. The overall process of the netlist and layout generation is shown in fig 2.

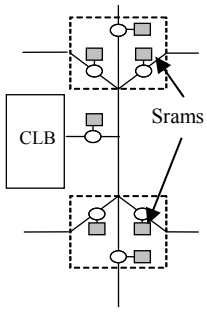

Fig. 6(a) – Standard sytem for deriving a single track

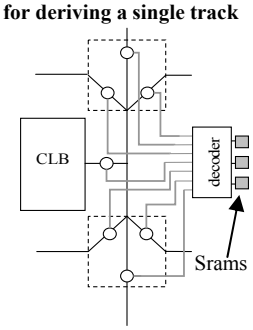

Fig. 6(b) – Decoder system for deriving a single track

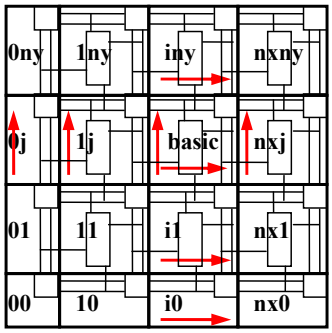

Fig. 6(c) - Set of tiles required to construct island-style FPGA

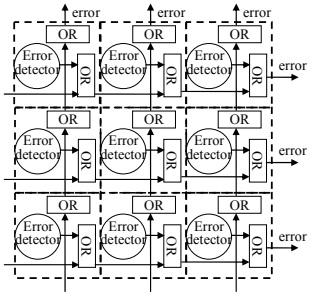

Fig. 6(d) – Scalable error detection method

3. ARCHITECTURE FEATURES

The above process of FPGA generation has been used to generate an island style FPGA with hardware support for the mitigation of Single Event Upsets (SEU) [8].

SEU are induced by energized particles hitting the silicon device. A particle hit with sufficient energy changes the logic state of the memory elements producing a transient error. An SEU on configuration bits may change the functionality of the look-up tables as well as the interconnect controlled by the SRAM cells, thus producing a hard error. These hard errors can be eliminated by using simple decoders, as shown in figure 6(a) and 6(b), to implement a system dependency between switches that derive the same track. An error detection system is integrated in each tile which enables an error signal whenever a change is detected in configuration bits. The error signal propagates through row and column, as shown in figure 6(d).

The addition of this architectural feature increases the total number of tiles to 16 as shown in 6(c). The merging of clock tiles with 16 different FPGA tiles produces a total of 34 different tiles. According to the final application requirements, these tiles are used to generate a 32x32 FPGA matrix with a channel width of 8.

4. VALIDATION

A. *Software flow:* A software flow is followed to test the functionality of the generated architecture. The sample application (in VHDL format) to be mapped onto the FPGA is the input to the software flow. Initially BOOG synthesizes the VHDL input into a netlist of gates VST. VST2BLIF and later

SIS is used to convert it into LUT form. T-VPACK and later VPR is used for the placement and routing of the netlist. A bitstream generator is written which generates a binary stream that contains all the required information for the configuration of the sample application onto the FPGA.

B. Bitstream configuration mechanism: An Nx by Ny FPGA contains (Nx+1) by (Ny+1) tiles; where Nx+1 is the total number of columns and Ny+1 is the total number of rows. Each FPGA tile comprises a set of SRAM bits arranged in multiple rows. The SRAM bits in a row are called a 'word'. For writing data to a word of a tile; a row number, a column number and a word number must be specified. The row and column numbers gives the location of the tile in a matrix, whereas the word number gives the location of word in a tile. All these three parameters are passed to the shift registers. The data to be written in a word is also specified in the same shift registers. With the help of the row, column and word decoders, the exact strobe and column signal is turned on. Thus when write enable turns high, the data is written onto the specific word of the requested tile. This process is repeated for all the words of all the tiles. The shift registers and decoder are implemented in a loader which is also generated by the FPGA generator.

C. Simulation: The generated FPGA netlist is tested on the ALLIANCE simulator called ASIMUT. Different test applications are mapped on the FPGA with the help of the sofware flow. Once the FPGA is programmed, the respective testbench of each test application is applied on the inputs of the FPGA and the outputs are compared. These simulations can also be easily performed on other commercial tools like SYNOPSYS.

D. ***Netlist layout comparison:*** The generated netlist and the generated layout must match with each other. For this purpose the ALLIANCE extraction tool COUGAR is used. It extracts a netlist from a layout. Later the ALLIANCE comparison tool LVX is used to compare the extracted netlist with the generated netlist. This confirms that the generated layout matches with its netlist. This method of layout verification is validated for a set of generated FPGAs. But the flattened 32x32 FPGA matrix is too large to be compared due to the limitations of COUGAR. So, instead of LVX, CALIBRE LVS is used to compare the 32x32 FPGA layout with its netlist.

E. Electric simulation: The ALLIANCE extraction tool COUGAR is used to extract the spice model of each tile. These models are later electrically simulated using ELDO. Our extraction tool is unable to support very large circuits. So it was impossible to electrically simulate the complete 32x32 FPGA. However for the proof of concept we successfully simulated the electric model of a smaller 4x4 FPGA matrix with channel width of 8.

5. TapeOut

The layout generation is done using symbolic standard cell library which works on unit λ (lambda). The ALLIANCE tool S2R (symbolic to real) is used to convert the symbolic design to 130nm technology. The corresponding GDS and LEF files are also obtained. The 32x32 FPGA occupies an area of 3885.6 µm by 3882 µm. It is noticed that 19% of the FPGA area increases due to the hardware support for the mitigation of SEU. The generic symbolic design rules help easy migration to any technology but with some area penalty. Instead of symbolic library, if the netlist of the generated FPGA is laid out in ENCOUNTER using directly a 130nm technology library, 40% area reduction is noticed.

The generated FPGA layout can be used as a black box in any other larger system. For the proof-of-concept, it is used to lay out a complete chip. The pads are placed and routed using ENCOUNTER. The DRC and LVS verification is performed using CALIBRE. The final FPGA chip measures 23.86 mm^2.

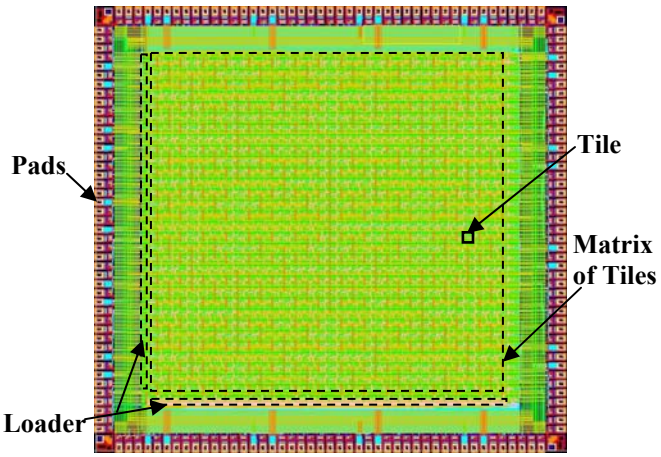

Fig. 7 – A Prototype FPGA chip layout

6. Conclusion and Future Work

In this work we have presented a completely automatic method for the generation of an FPGA using an open-source VLSI tool-kit. We are able to generate FPGAs having different architectural parameters. In future, we intend to increase the number of variable architecture parameters. We also intend to add support for other clock distribution networks.

7. References

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

[2] S. Belloeil, D. Dupuis, C. Masson, J.P. Chaput, H. Mehrez, "Stratus: A procedural description language based upon Python", in Proceedings of the 19th International Conference on Microelectronics, december 2007

[3] K. Padalia, R. Fung, M. Bourgeault, A. Egier, and J. Rose, "Automatic transistor and physical design of FPGA tiles from an architectural specification.", in Proceedings of 2003 ACM/SIGDA 11th international symposium on FPGAs, pp. 164-172. ACM press, 2003.

[4] E. G. Friedman, "Clock Distribution Networks in Synchronous Digital Integrated Circuits", in Proceedings of the IEEE, May 2001

[5] S. Phillips and S. Hauck. "Automatic layout of domain-specific reconfigurable subsystems for system-on-a-chip", In Proceedings of the 2002 ACM/SIGDA tenth International symposium on FPGAs, pages 165-173. ACM Press, 2002

[6] I. Kuon, A. Egier, J. Rose, "Design, layout and verification of an FPGA using automated tools", In Proceedings of the 2005 ACM/SIGDA 13th international symposium on FPGAs, 2005

[7] Cadence. SKILL Programming Language, http://www.cadence.com

[8] J.H. Elder, J. Osborn, W.A. Kolasinski, R. Koga, "A method for characterizing a microprocessor's vulnerability to SEU", IEEE Transaction on Nuclear Science, Dec 1988 v 35 n 6.