Automatic Layout of Scalable Embedded Field Programmable Gate Array

Hayder MRABET, Zied MARRAKCHI*, Habib MEHREZ* and André TISSOT***

**LIP6-ASIM Laboratory*
Université Pierre et Marie Curie
4, Place Jussieu, 75252 Paris, France

***CEA-DAM IDF*
Commissariat á l'Energie Atomique
email:{hayder.mrabet,zied.marrakchi,habib.mehrez}@lip6.fr, andre.tissot@cea.fr

Abstract*—This paper presents a layout technique for scalable embedded Field Programmable Gate Array architecture (eFPGA). It describes the total flow to generate a variety of eFPGA architectures using parameterized generators and Alliance CAD developed in the university of Paris6. We will show one example of realization using a symbolic library of cells. Our test eFPGA have a symmetric mesh architecture (Island-style) composed of five main tiles. The scalability of this tiles can be varied to obtain the best design fit on the System on Chip device.*

I. Introduction

The ability of an FPGA to implement any circuit simply by being programmed appropriately gives a high flexibility for a system on chip (SoC) that include FPGA as an embedded performance accelerator. This makes it possible to reuse a portion of the chip for a variety of tasks and application domains like image analysis and signal processing. Unfortunately the advent of new fabrication technologies increases devices complexity and transistors count. Designers of modern SoC have a larger design space to consider and more difficult job to combine heterogeneous components. However, automating the generation of the specific embedded FPGA will reduce the overall cost of the design.

In this paper, we present a flexible technique for eFPGA creation, focusing particularly on the process to create physical eFPGA model. This process that automates the major part of the transistor level design and layout is based on generators using the procedural GENLIB language [2]. GENLIB uses the C programming language as a procedural layout language oriented toward development or portable parameterized generators for CMOS VLSI. Mostly dedicated to libraries design, GENLIB addresses both the process and software portability issues. Advantages of using superset of C functions for symbolic layout and netlist description permits us to create generic generators (described in section 3). Those generators use architecture parameters to generate the required structural netlist and layout. GENLIB ensures the access to a large number of symbolic components : custom, semi-custom, standard cell libraries and soon macro cells.

We adopt hierarchical methodology to make up the eFPGA, starting from the bottom level where we generate the different components. Then those components have been combined to form core tiles. Finally the tiles can be used to generate arrays of different sizes. Figure 1 gives an overview of the flow of this methodology. Previous attempts at this type of work offered eFPGA cores in the form of a hard layout [3, 4]. Other efforts to synthesize eFPGA core have been reported in [8]. Padalia et al.[5] describes a system for automating the tile layout . The latter system uses an appropriate library of asymmetric coarse grained cells. In this work, we present a general methodology with the adequate CAD flow that permit designer to create their proper eFPGA with their proper specifications. The final layout is not dedicated to a single fabrication process thanks to symbolic method.

This paper is organized as follows: the following section describes the architecture of our test-prototype. Section 3 describes different steps to generate the eFPGA layout. Section 4 describes the programming technique, including description of the exploration environment flow necessary to extract the configuration Bitstream. Section 5 concludes and presents the perspectives for future work.

II. Architecture Overview

To integrate eFPGA onto SoC, we require that the programmable structure be flexible enough to be implemented easily with any physical constraints. Our test eFPGA is regular structure in Island style. It consists of an array of logic blocks, which can be linked together thanks to uniform horizontal and vertical programmable routing channels. The logic and routing must be programmed to realize the required function, the configuration is stored in SRAM cells.

This style has been the subject of considerable work by Rose et al. [6] looking at the effect of different architecture parameters on area and delay efficiency. We also mention the work of Rabey et.al [7] looking for energy consumption. The goal of our work is to support any architecture in this style regardless the parameters. The main building block of our SRAM-based eFPGA prototype is the programmable Tile presented in figure 2. It combine the logic

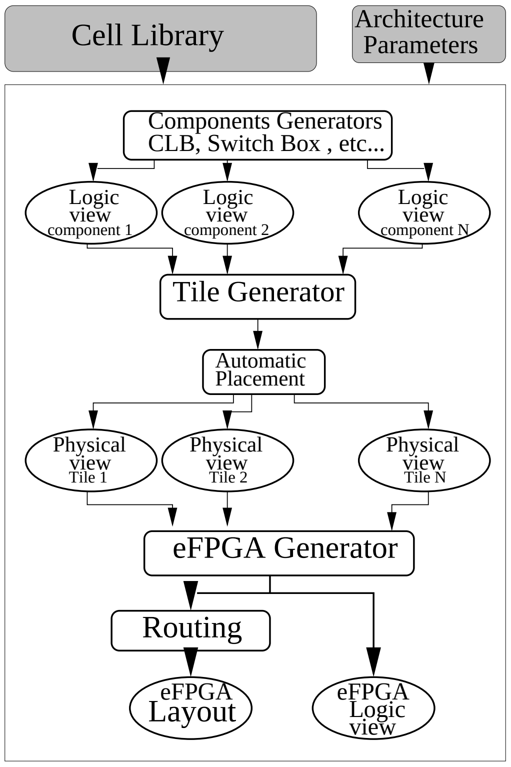

Figure 1: Overview of Flow

block, the connection box and the switch box:

- The Configurable Logic Block (CLB) structure is capable of implementing two 4-inputs random logic function or 2-bit arithmetic function using 2 Look-Up-Table (LUT) of 4 inputs. A LUT is a digital memory that stores the truth table of the boolean function. The 2 outputs of the CLB can be registered if required using a 2 to 1 Mux controlled by one SRAM cell. The flip-flop use the global clock. Figure 2 describes the CLB in details.

- The connection box controls connectivity between the pins of the CLB and the adjacent routing channels. Its purpose is to select the right input track to send to the CLB using programmable muxes. It is also selecting the right outputs track driven by the CLB using programmable tristates.

- The switch box is located in the cross of vertical and horizontal channels. It is a disjoint with the flexibility of three. Each track have three programmable connections with the corresponding tracks on the other three sides.

III. Design Technique and Layout

To create the transistor-level circuit it is sufficient to generate a high cell-level netlist describing the tiles that can be replicated to create an eFPGA array. In our model design we have five different tiles. Each one represents a netlist based on multiplexers, tristates, flip-flops and SRAM. Difference between tiles is due to the border effects. These latters represent external interface.

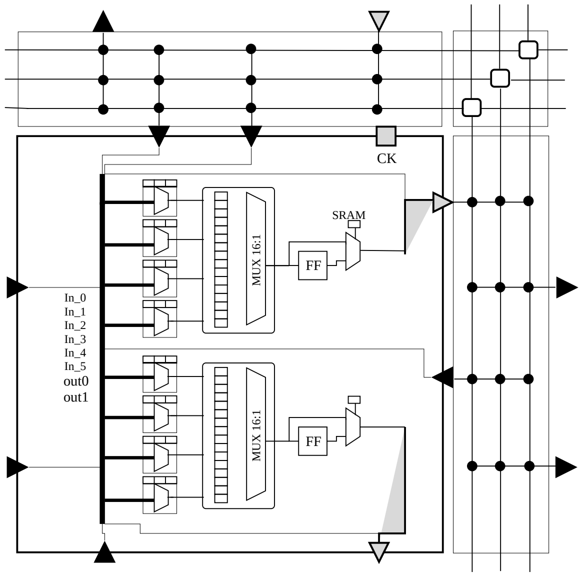

Figure 2: Main Tile in the center of the architecture

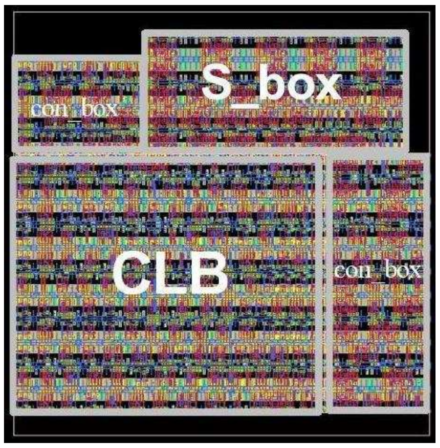

Figure 3: Example of Tile Layout Placement

A. Tile Layout

As discussed above, a netlist of high-level modules are combined to form the Tile. Currently the connection box module is manually described on GENLIB format. However CLB and switch box modules are generated automatically using two generators : genCLB and genSBOX.

- GenCLB uses three parameters to generate the structural netlist of the CLB: Number of inputs per CLB, Number of LUT per CLB, LUT size.

- GenSBOX needs two parameters: the topology of the switch box, the channel width.

Each netlist is passed to the Placement tool to generate rectangular layout area, then GenTile combines the three modules and generates the logic and the physical views. Figure 3 correspond to the placement of physical Tile. The dimensions of the tile abutment-box are 96μ x 108.62μ in CMOS 0.13. Cells that compose the tile are symbolic standard cells (mux2:1, mux4:1, static flip-flop).

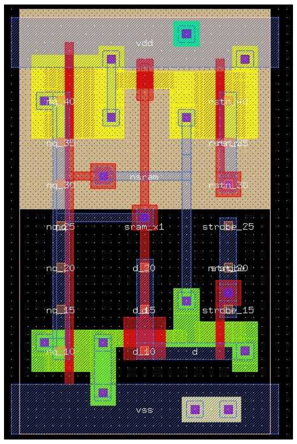
Figure 4: Sram Cell with asynchrone reset. Symbolic layout with the dimension 30λ x 50λ (3.9μx6.42μ in CMOS 0.13μ)

We add some specific cells like the SRAM provided with a reset mechanism shown in figure 4.

B. eFPGA generation

The regularity of the island-style simplifies the automatic structuring of the eFPGA Layout. The parameters for the generator are the number of CLB per Row and the number of CLB per column. This generator assumes the last step before the final global routing . It generates the global physical view (see figure 5)and all the other needs of the router: structural vhdl netlist and connector placement file.

The final routed layout of the test-eFPGA is given in figure 6. It is an array of size 4x4, with an equivalent logic capacity of 32 4-input lookup tables. The layout measures $0.453mm$x$0.476mm$ in CMOS 0.13 and routed using 4 layers of metal (ignoring the clock distribution). It is reported from the symbolic to the real using s2r[1]. Table 1 illustrates details of different arrays generated using genFPGA.

array size logic capacity	number of transistor	measure in symbolic	measure in CMOS 0.13
4 x 2 16 4-LUTs	25588	3770λx2000λ	$0.453mm$x$0.240mm$
2 x 6 24 4-LUTs	37728	1970λx5600λ	$0.237mm$x$0.672mm$
4 x 4 32 4-LUTs	48288	3770λx3800λ	$0.453mm$x$0.476mm$
4 x 8 64 4-LUTs	93688	3770λx7400λ	$0.453mm$x$0.888mm$
8 x 8 128 4-LUTs	181328	7370λx7400λ	$0.885mm$x$0.888mm$

IV. Configuration

A. programming access

The functionality of the logic block is controlled by programming the multiplexers and the content of the look-up-table. In our test eFPGA the CLB needs 58 bits. Concerning the programmable

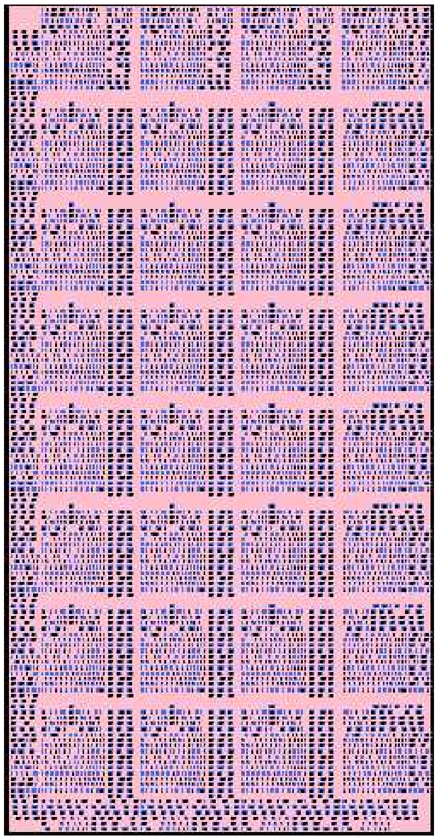
Figure 5: Placement of 4x8 eFPGA

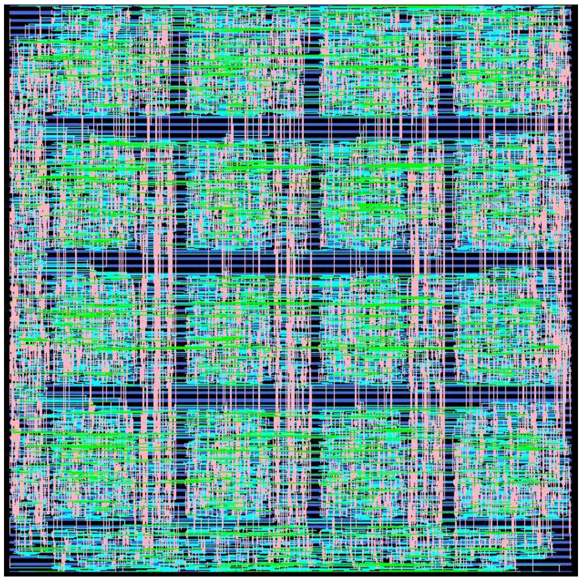
Figure 6: routed 4x4 eFPGA architecture

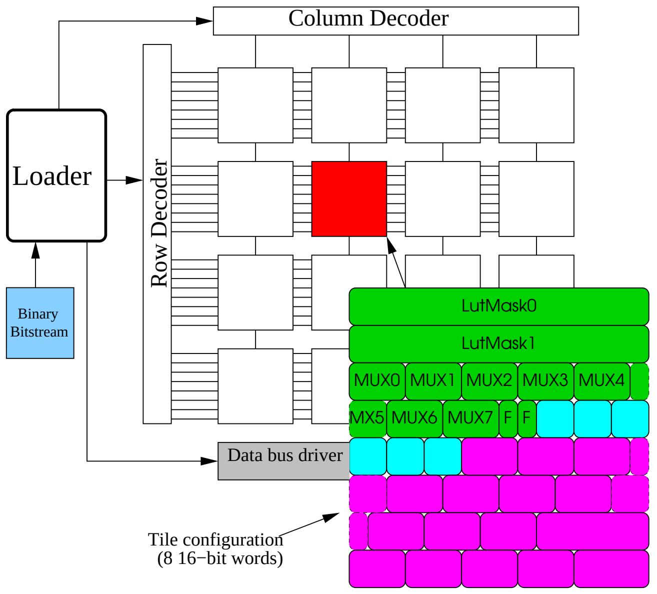

Figure 7: eFPGA Configuration Technique

routing inside the tile including connection boxes and the switch box, it needs 70 bits. The total number of configuration bits in the Tile is 128 bits grouped as eight word of 16 bits that can be selectively programmed.
If the implementation of the array is made, we obtain an array of reconfigurable cells grouped as 16-bit words and can be programmed similar to RAM.
The random access method is used to write the configuration. Each configuration cycle the row and column decoders identify the word being programmed. The data is distributed using a config bus driver. The configuration circuitry is shown in figure 7.

Using this technique of programming, the number of pins required for configuration (address pins) is proportional to the array size. This number is not critical in the case of embedded FPGA (no extern pins). However we profit of the random access for dynamic configuration.

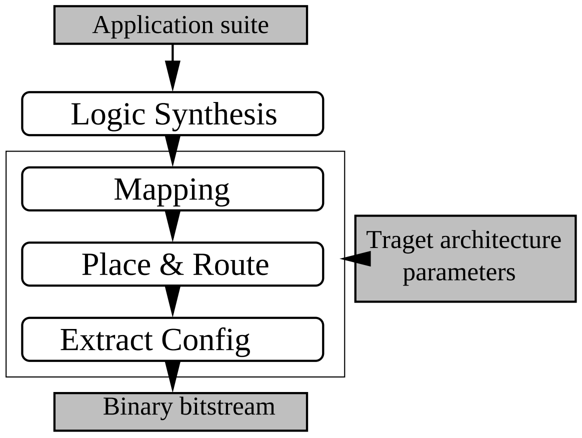

Figure 8: Environment for eFPGA exploration

B. Bitstream configuration

The exploration of the implemented eFPGA requires the flow (figure 8) involving logic synthesis, placement, routing and extraction of the implementation on the target architecture named Bitstream. This flow uses freeware tools like ’boog’ [1] for logic synthesis, SIS [10] for mapping, T-vpack+VPR [9]for place and route. We developed a generic extractor of bitstream that analyzes all the results of the previous tools to generate the configuration memory.

V. Conclusions and Future Work

We presented a technique for implementation of embedded FPGA. And we have shown that automation of layout generation is possible. Using this technique we are capable of producing a large spectrum of different architectures and different array sizes. As we noted, our work is not completely automated. To achieve better quality of layout some manual specific-tasks were suggested. Those tasks are specific to the target FPGA. We are investigating ways that include those method of placement optimization and SRAMs organization. Making a local routing for tiles will be beneficial with respect to global routing.

VI. References

[1] www-asim.lip6.fr/recherche/alliance, ALLIANCE CAD.

[2] F.Pétrot, A.Greiner ”Using C to Write Portable CMOS VLSI Module Generators” Proceedings of the European Design Automation Conference (EURO-DAC), Grenoble, France, Septembre 1994, pp.676-681.

[3] Actel Corp, VariCore Embedded Programmable Gate Array Core (EPGA) 0.18m Family. Datasheet, December 2001.

[4] M2000, Inc, M2000 FLEXEOStm Configurable IP Core,http://www.m2000.fr.

[5] K.Padalia, R.Fung, M.Bourgeault, A.Eiger, and J.Rose, ”Automatic Transistor and Physical Design of FPGA Tiles from an Architectural Specification”, FPGA’03, February 23-25, Montery, California, USA.

[6] Vaughn Betz, Alexander Marquardt, Jonathan Scott Rose ”Architecture and CAD for Deep-Submicron FPGAs”. January 1999. Kluer Academic Publishers

[7] Varghese George, Jan M.Rabaey ”LOW-ENERGY FPGA Architecture and Design”. Kluwer Academic Publishers

[8] James C.H. Wu, Victor Aken Ova, Steven J.E. Wilton, Resve Saleh”SoC Implementation Issues for Synthesizable Embedded Programmable Logic Cores”. IEEE Custom Integrated Circuits Conference, Sept. 2003

[9] www.eecg.toronto.edu/ vaughn/vpr/vpr.html

[10] www-cad.eecs.berkeley.edu/Software/software.html