Non-Rectangular Reconfigurable Cores for System-on-Chip

Pedro Alves[a], João Canas Ferreira[b]
[a]DEEC, Faculty of Engineering, University of Porto - pedro.miguel.alves@fe.up.pt
[b]INESC Porto, Faculty of Engineering, University of Porto - jcf@fe.up.pt

ABSTRACT

Non-rectangular cores of standard-cell-based reconfigurable logic can be used to fill space left on System-on-Chips, thereby providing the system with hardware reconfigurability. The proposed architecture for a non-rectangular reconfigurable core is based on a fixed set of blocks that implement logic functions, interconnections and configurable switching. The basic blocks connect by abutment to form clusters and clusters abut to form a complete reconfigurable core. A software tool was created to generate a gate-level netlist and the floorplan data of the reconfigurable logic core together with a basic testbench. Cores with non-rectangular shapes were created using 90 nm and 45 nm standard-cell technologies and validated by simulation. The results demonstrate the feasibility of a flexible, technology-independent architecture for non-rectangular reconfigurable logic cores that can be physically implemented using a standard digital design flow.

Keywords: Non-rectangular core, reconfigurable circuits, embedded FPGA, core generator

1. INTRODUCTION

Today's embedded systems integrate digital and analog technologies into a single chip, known as System-on-Chip (SoC), in order to achieve increased performance, system reliability, and reduction in packaging and test costs. As fabrication technology advances and processes shrink, design costs escalate due to increasing system complexity, and challenging integration and verification tasks. Part of the digital circuitry of a SoC is comprised of fixed digital logic blocks, but another part is often custom designed, and may benefit from redesigns and updates, or may have a dual use (regular and test mode, for example). This is possible when reconfigurable digital blocks exist, that are capable of supporting different digital functions after silicon fabrication. The use of reconfigurable logic cores on SoCs reduces development risk, because it enables a quick and inexpensive correction or update of the digital core design function, while still maintaining performance levels appropriate for a broad range of applications. Inside a SoC, the space left after placing the analog circuitry and other large IP cores is very often non-rectangular (Figure 1). We propose using a non-rectangular core of standard-cell-based reconfigurable logic to fill that space, thereby adding to the final product the advantage of reconfigurability with a potentially short design cycle. The goal of the current work is to develop a variable-shape reconfigurable core architecture and the tools to support its implementation through a regular ASIC design flow, in order to achieve the cost-efficient inclusion of such cores in SoC-based consumer electronics applications.

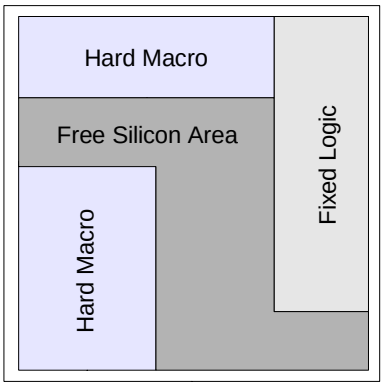

Figure 1. System-on-Chip – Example of space left after placement of other hard macros and fixed logic.

2. PROPOSED ARCHITECTURE

The proposed general core architecture is based on a fixed set of basic cells, commonly available on standard cells libraries, which implement logic, switch and routing blocks. The circuits presented here were devised as a proof of concept of the overall programmable core architecture. The basic blocks connect by abutment to form a higher structural level identified as a cluster. Cluster cells connect by abutment to other cluster cells to form the reconfigurable logic core with the desired shape (Figure 2). Part of the area is occupied with fixed digital logic that supports the programmable core operation and reconfiguration. The configuration of the core is done serially by a scan chain.

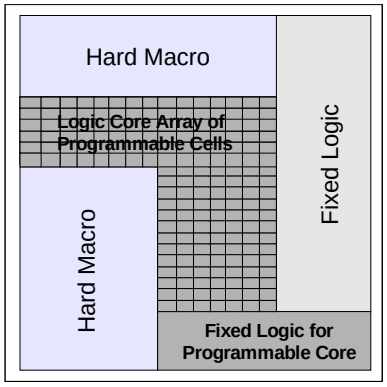

Figure 2. Programmable logic core on SoC implemented as an array of programmable cells and fixed support logic.

2.1 Architectural Overview

The architecture of the embedded programmable logic core proposed is based on a typical island-style FPGA. Three different structures are required: logic, switch and routing blocks.

The logic blocks hold the mapped digital function and connect to adjacent routing blocks that may deliver the data signals to any core location. A switch block is included wherever the routing channels intersect. These three basic blocks connect by abutment to form a more complex cell at a higher structural level identified as a cluster. Cluster blocks connect by abutment to other cluster blocks to form the reconfigurable logic core with the desired shape. They can be snapped together to form a programmable fabric of the desired size and shape. This gives the circuit architecture a fine-grain characteristic, the most suitable to serve non-rectangular shapes.

The implementation of the three types of block is done using just a small set of standard cells. By using standard cells, commonly available CAD tools can be used and the programmable core architecture can be physically implemented through a standard digital logic design flow.

In this architecture the definition of the number of data inputs to the logic block defines the architecture for the routing and switch blocks. In this sense, the logic block architecture defines the routing and switch blocks that surround it and must be used together

2.2 Cluster

The non-rectangular programmable core is composed by a group of macrocells known as clusters. Each cluster block contains one logic block (LB), one horizontal routing block (HRB), one vertical routing block (VRB) and one switch block (SB) (Figure 3). The logic block contains a look-up table to define the logic function assigned to it, and has combinational and sequential outputs interfacing to vertical and horizontal unidirectional tracks on the routing block. These tracks intersect in the switch block and are routed to other tracks through it.

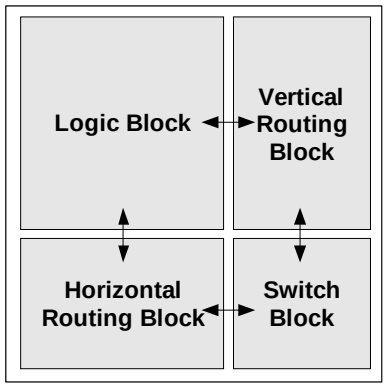

Figure 3. Cluster block hierarchical organization.

The cluster block is physically implemented by instantiating its basic blocks. This is done automatically by procedures written in the scripting language of the EDA tool used. For this work, these procedures were written in *SKILL* programming language and executed within Cadence Virtuoso Layout Editor. Each of the basic blocks should have been previously implemented by the standard digital design flow. For that purpose, their gate-level netlists and floorplan can be used.

The cluster block is the same for all positions on the non-rectangular core. The fixed circuit connecting to the programmable core must take into consideration that all non-used inputs of the core must be tied to a logic value (all unused outputs of the core can be left floating). This situation occurs for all cluster blocks on the boundary of the shape.

2.3 Logic Block (LB)

The logic block (LB) may contain one 2- or 4-input look-up table that is used to implement the reconfigurable logic function assigned to the cluster cell. The lookup table's implementation uses scan flip-flop cells that can be programmed through their scan input. The contents of the flip-flops will define the truth table. The input data signals control the selection signals of the multiplexers that route the look-up table contents to the logic block's combinational and sequential outputs. The sequential output is implemented by a non-scannable flip-flop cell. This cell does not belong to the configuration chain. Additional circuits in the logic block ensure that its sequential and combinational outputs do not toggle during serial programming. The interface signals orientation and the schematic representation of a 2-input LB can be seen in Figure 4.

For an N-input LB, 2^N flip-flop cells and an N-input multiplexer cell or 2^{N-1} 2-input multiplexer cells are needed to implement the look-up table.

In normal operation, the clock signal to the programming flip-flop cells is disabled (gated with combinational logic and the programming mode enable signal) to avoid additional power consumption, since in this mode the programming flip-flop cells need only to maintain their output. The clock signal is enabled when the core is in programming mode. In normal operation, the global reset signal is applied only to the flip-flop cell to the sequential data output. In this way, the circuit mapped to the logic block can be reset without resetting the LUT. The LUTs are reset when the reset global signal is asserted in programming mode.

The schematic representation of the 2-input LB shows the relation between the programmed logic values (*a0..3*) in each of the programming flip-flop cells and the value of the output (*out*) and inverted output (*outz*) for each data input combination (*in0*, *in1*). The logic value programmed in the last flip-flop cell (*a5*) defines whether the output of the logic block is registered or not.

For a cluster surrounded by other clusters, LB input data is provided by the HRB on the adjacent cluster, while its output data is connected to the HRB in the same cluster. The clock, reset and programming mode enable signals are supplied by the VRB in the same cluster. The programming chain input comes from the VRB in the adjacent cluster and its output connects to the VRB in the same cluster.

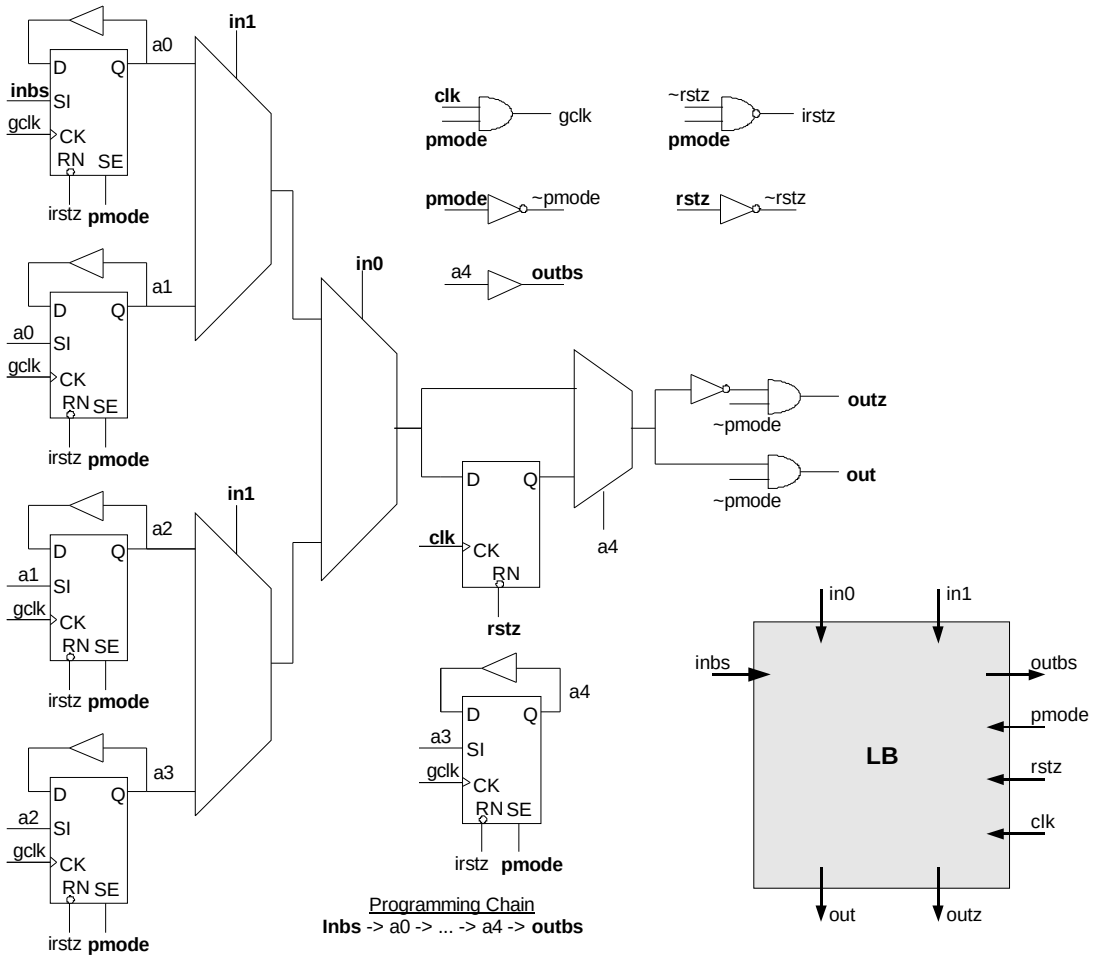

Figure 4. 2-input LB schematic representation and interface signals and orientation in block symbol.

2.4 Routing Block

The routing circuitry in the configurable core implements the connections between all logic blocks and allows the propagation of any data signal to potentially any core location. The routing circuitry of a cluster cell is composed of a vertical and a horizontal routing block. These two blocks interface with the logic and switch block on the same and adjacent clusters.

2.4.1 Horizontal Routing Block

The horizontal routing block (HRB) has 4 horizontal unidirectional tracks (when a LB with a 2-input look-up table is used) or 6 horizontal unidirectional tracks (when a LB with a 4-input look-up table is used). The unidirectional tracks area buffered. A mechanism using three-state buffer cells was implemented to control the drive of the horizontal tracks by the LB output signals. This mechanism was also used to disconnect the tracks from the SBs connecting to the horizontal tracks to avoid conflicts. The access of the LB output signals to the horizontal tracks is implemented using three-state buffers controlled by programmable flip-flop cells. The access of the horizontal tracks to the LB input pins is implemented through a multiplexer tree whose selection signals are controlled by flip-flop cells. The HRB circuit implemented to work together with a 2-input LB is represented, together with the HRB interface signals orientation, in Figures 5 and 6. The same implementation for a 4-input LB differs in the amount of horizontal tracks, and uses an enlarged multiplexer tree, instantiated four times, to connect the horizontal tracks to the 4 LB inputs.

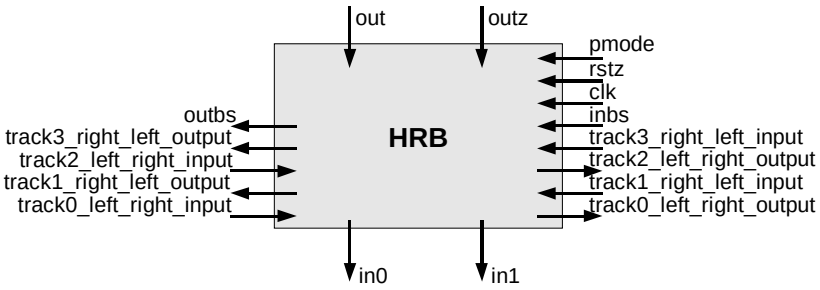

Figure 5. HRB interface signals and orientation in block symbol for a 2-input LB architecture.

Again, the clock signal to the programming flip-flops is disabled during normal operation and enabled during configuration. The programmable flip-flop cells are reset only when the reset signal is asserted in programming mode.

HRB behavior is determined by two groups of signals. One is related to the horizontal track data signals and to the connections to the LB in the adjacent cluster (*a0..3*). The other is related to the LB outputs and their connections to the horizontal tracks in the same cluster (*a4..11*). The two sections are described on Table 2 and Table 3 respectively. It is not possible to connect both LB *out* and *outz* output signals to the same horizontal (Table 3).

Table 2. Horizontal routing block truth table – logic block inputs

a0	***a1***	***a2***	***a3***	**track to *in0***
0	0	X	X	track0_left_right
0	1	X	X	track1_right_left
1	0	X	X	track2_left_right
1	1	X	X	track3_right_left
a0	***a1***	***a2***	***a3***	**track to *in1***
X	X	0	0	track0_left_right
X	X	0	1	track1_right_left
X	X	1	0	track2_left_right
X	X	1	1	track3_right_left

Table 3. Horizontal routing block truth table – logic block outputs

a4	***a5***	***a6***	***a7***	***out* to track**
1	0	0	0	track0_left_right
0	1	0	0	track1_right_left
0	0	1	0	track2_left_right
0	0	0	1	track3_right_left
a8	***a9***	***a10***	***a11***	***outz* to track**
1	0	0	0	track0_left_right
0	1	0	0	track1_right_left
0	0	1	0	track2_left_right
0	0	0	1	track3_right_left

For a cluster surrounded by other clusters in the programmable core, the HRB input data is provided by the LB and SB in the same cluster and by the HRB in the adjacent cluster, while its output data is connected to the SB in the same cluster and to the HRB and LB in the adjacent clusters. The clock, reset and programming mode enable global signals are supplied to the HRB by the SB of its own cluster. The programming chain input comes from the SB in the same cluster and its output is connected to the SB in the adjacent cluster.

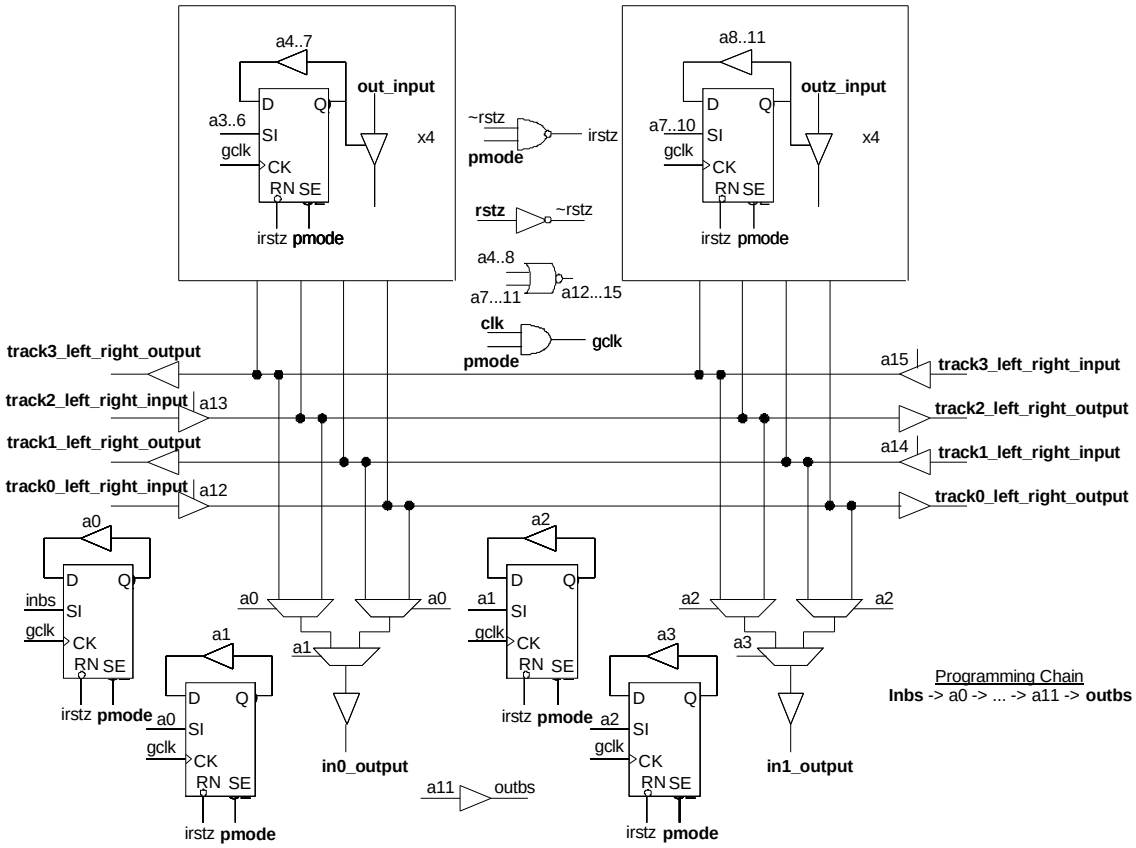

Figure 6. HRB schematic implementation for use with a 2-input LB.

2.4.2 Vertical Routing Block

The vertical routing block (VRB) performs two functions. Like the horizontal routing block, it carries 4 or 6 unidirectional tracks for data propagation (for a 2- or 4-input LB, respectively). In addition, the VRB is used to feed the cluster with the clock, reset and programming mode enable global signals. Figures 7 and 8 show the interface signals orientation and circuit implementation of a VRB for a 2-input LB.

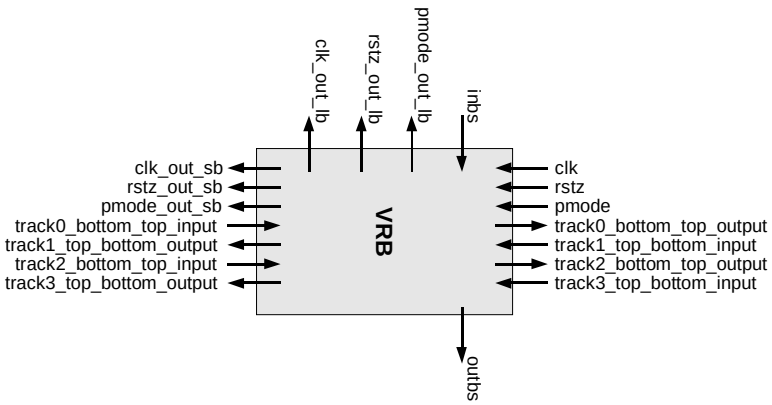

Figure 7. VRB interface signals and orientation on block symbol for a 2-input LB architecture.

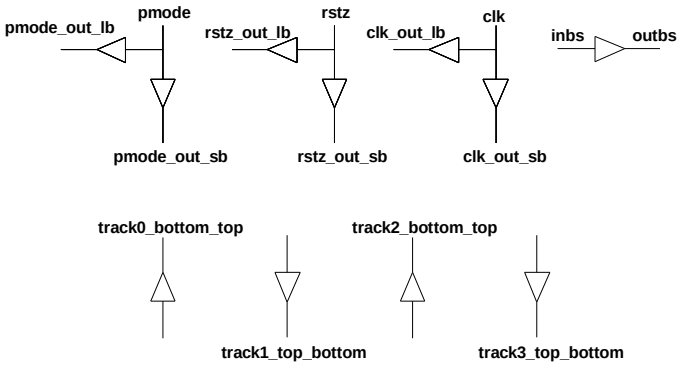

Figure 8. Schematic of a VRB for use with a 2-input LB.

The VRB input and output signals are connected to the SB in the same and adjacent clusters. The clock, reset and programming mode enable global signals are supplied to the VRB by the SB in the adjacent cluster. The programming chain input is supplied by the LB in the same cluster and its outputs are supplied to the LB in the adjacent cluster.

2.5 Switch Block

The switch block, besides propagating the clock, reset and programming mode enable global signals, is responsible for supporting the programmable connection between the tracks of its cluster and the other clusters connected to it. The logic implementation of the switch block is based on the Wilton switch [2] as it is the one that presents the best area-efficiency for architectures where the routing tracks span only one logic block. The track connections are implemented using 3-input multiplexer cells controlled by flip-flop cells. An implementation of this cell for a 4-input LB will be the similar, but will use additional 3-input multiplexer cells to accommodate the larger number of connections. The interface signals and orientation and the circuit representation of a SB for a 2-input LB can be seen in Figures 9 and 10.

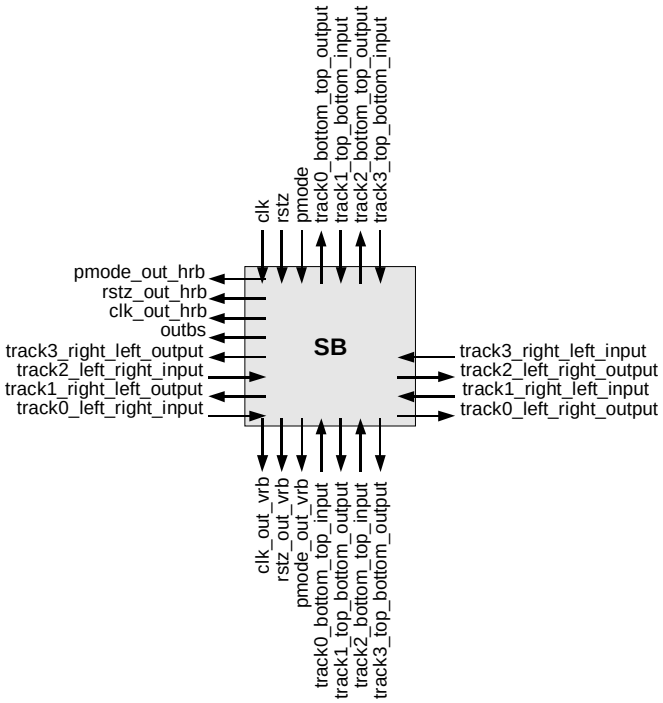

Figure 9. SB interface signals and orientation on block symbol for a 2-input LB architecture.

This block implements the same clock scheme as the LB and the HRB to avoid the additional power consumption in the functional mode. It also implements the same circuit to reset the programmable flip-flop cells only in programming mode.

The SB input and output signals connect to the HRB and VRB blocks that surround it. The clock, reset and programming mode enable signals are supplied to the SB by the VRB in the same cluster. The SB provides these global signals to the VRB in the adjacent cluster. The programming chain input is supplied by the HRB in the adjacent cluster and its output connects to the HRB in the same cluster.

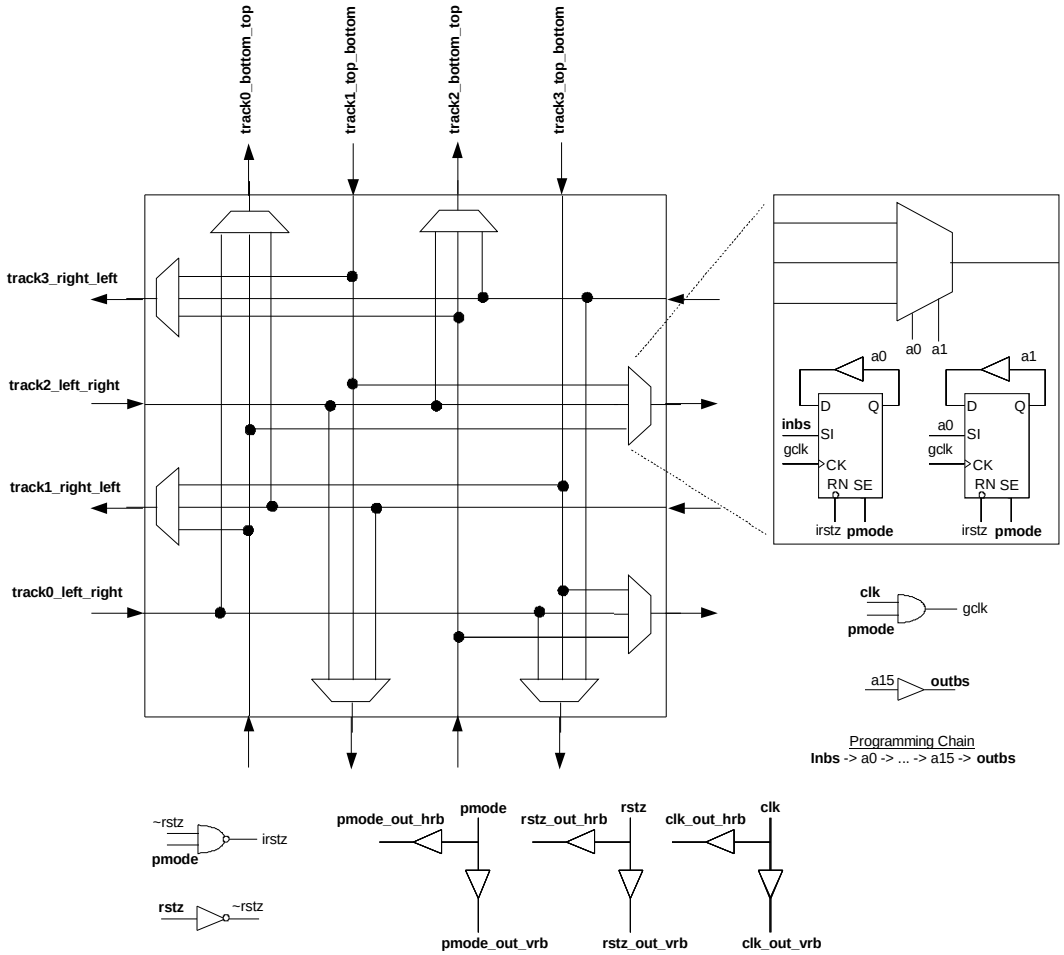

Figure 10. Switch block schematic implementation for a 2-input LB.

The connections between horizontal and vertical tracks that the switch block supports are detailed in table 4 (switch block for a 2-input LB).

Table 4. Switch block truth table (for a 2-input logic block)

track0_left_right_output			***track0_bottom_top_output***			***track3_top_bottom_output***		
a0	***a1***	**input track**	***a2***	***a3***	**input track**	***a4***	***a5***	**input track**
0	0	track0_left_right	0	0	track0_left_right	0	0	track0_left_right
0	1	track2_bottom_top	0	1	track0_bottom_top	0	1	track3_right_left
1	0	track3_top_bottom	1	0	track1_right_left	1	0	track3_top_bottom
1	1	Not allowed	1	1	Not allowed	1	1	Not allowed

track1_top_bottom_output			*track2_left_right_output*			*track2_bottom_top_output*		
a0	***a7***	**input track**	***a8***	***a9***	**input track**	***a10***	***a11***	**input track**
0	0	track2_left_right	0	0	track2_left_right	0	0	track2_left_right
0	1	track1_right_left	0	1	track0_bottom_top	0	1	track2_bottom_top
1	0	track1_top_bottom	1	0	track1_top_bottom	1	0	track3_right_left
1	1	Not allowed	1	1	Not allowed	1	1	Not allowed

track3_right_leftt_output			*track1_right_left_output*		
a12	***a13***	**input track**	***a14***	***a15***	**input track**
0	0	track1_top_bottom	0	0	track3_top_bottom
0	1	track2_bottom_top	0	1	track0_bottom_top
1	0	track3_right_left	1	0	track1_right_left
1	1	Not allowed	1	1	Not allowed

2.6 Configuring the Core

The programmable core is configured through the serial chain that connects the scan flip-flops used to store the programming information. The configuration is performed with a serial bit stream using the available scan chains. With the proposed architecture, it is also possible to program the complete core by programming in parallel each line or group of lines of clusters. The external wrapper cell that surrounds the programmable core is responsible for accessing in parallel the programming chains input and output signals or by connecting them into a single chain. A representation of a non-rectangular programmable core is shown in Figure 11, where a single chain is used to configure the core.

2.7 Design Example

The same non-rectangular programmable core is used on this section with one example digital circuit mapped onto it. This circuit doesn't represent any particular known function. The image presented on Figure 12 shows only the LB input and output signals in use together with the tracks used to propagate those signals through the core. The different dotted lines on the figure are used to represent the different data paths established between the LBs after mapping and they are presented with different patterns for a clearer visualization. The arrows represent the signals input and output direction. The clock, reset and programming mode enable global signals are supplied to all clusters through their VRBs. These signals are not shown in the figure.

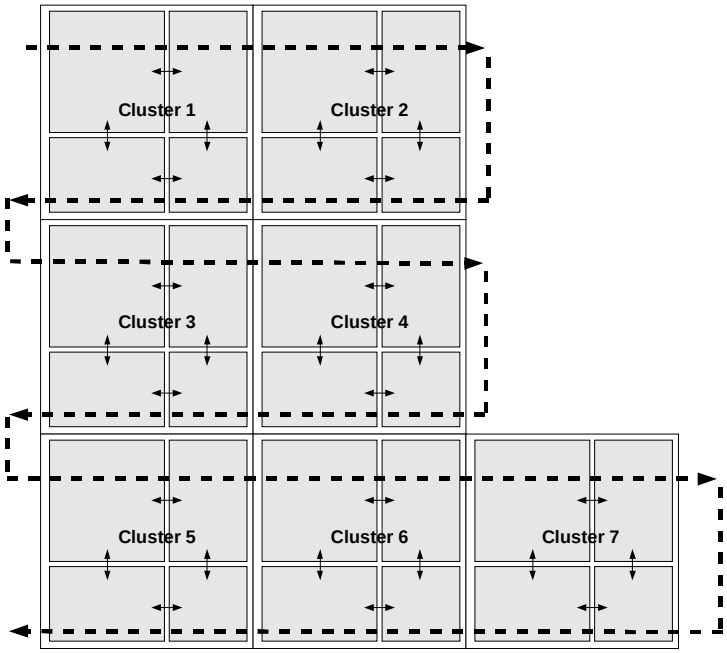

Figure 11. Programmable core serial bit stream configuration.

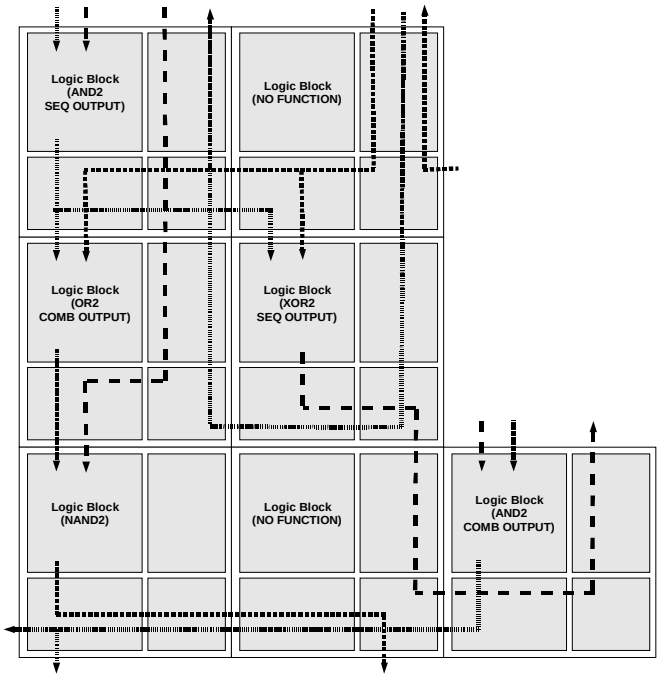

Figure 12. Programmable core with mapped circuit.

3. AUTOMATIC GENERATION OF THE RECONFIGURABLE CORE

A software tool was created that, given the shape of the programmable core, the number of data inputs (2 or 4) and the size of the basic blocks, generates a gate-level netlist of the reconfigurable core, a basic testbench and data for the backend flow required for the physical implementation of the blocks.

The shape of the programmable core defined by a matrix of ”+” and ”-” that represents the desired shape. The tool generates the logic, routing and switch blocks in the form of gate-level netlists that employ technology-dependent standard cells. The generator creates a gate-level netlist of the reconfigurable core by connecting the cluster cells to meet the requested shape. These gate-level netlists can be used in a standard digital design flow and each block can be implemented and validated as a standalone logic block. In this way common CAD tools and optimized standard cells from commercially available libraries can be used, reducing the design risk associated with the reconfigurable core and its impact on the SoC design cycle.

The generator also creates a wrapper top-level cell for the entire core, in order to connect the signals on the boundary clusters to signal buses so as to ease the integration of the core with the rest of the SoC. The wrapper cell that is automatically created is a very simple cell that connects the programming chains of each line of clusters into a single serial chain and maps the programmable core interface signals to data buses. The wrapper cell that needs to be created when integrating the core on a SoC will control the connection of the programmable chains, supply the needed control signals, and tie the unused input signals.

In addition, the generator creates a ready-to-run test-bench implementing a single task that, by serially programming a sequence of 0’s and 1’s, counts the number of programming flip-flop cells on the complete core and compares it to the expected value. With this task, the maximum operating frequency for the programming mode of operation can be easily determined early in the design cycle.

The backend preparation data is created in the form of LEF files [6] (a standard for representing physical layout information) that describe each basic block floorplan and a set of scripts to automatically create the abstract and layout views of the cluster cell and the programmable core.

An additional utility tool was implemented that transforms a user-friendly text file with the mapping information for each cluster into a bit stream that can be used to serially configure the core.

4. EXAMPLES AND RESULTS

Programmable cores with non-rectangular shapes (“S”, ”L”, ”T”, ”U”) were created and different logic functions manually mapped onto them. There are currently no tools to automatically map digital designs onto non-rectangular cores.

Validation was performed in two different ways. The first simulations performed with the gate-level netlists for the programmable core were functional simulations only, with no timing associated. This group of simulations was used to functionally validate the proposed architecture both for the programming and for the normal mode of operation. The second group of simulations used time-annotated standard cell netlists combined with an appropriate wireload model. This group of simulations was done to evaluate with some accuracy the performance and maximum operating frequency of the implemented programmable cores and, in this way, to quantify its performance. The same test-bench for both simulations was applied and a simple digital design was mapped onto a 3×3 programmable core. The programming circuitry operated at more than 850 MHz for a 90 nm commercial CMOS process, under the assumption that the longest path between two flip-flops (in the mapped circuit) spanned two cluster cells. The speed of the mapped logic is design dependent, but this approach establishes an upper limit of 850 MHz. The approximate area for each cluster with 2-input look-up tables is 1600 μm^2 for a 90 nm process and 100μm^2 for a 45 nm process based on a predictive model [7], [8]. At the smaller technology node, a 100×100 programmable core requires around 10mm^2.

An example layout of a cluster cell is presented on Figure 13 for a 90 nm process. Figure 14 shows the non-rectangular core that is automatically created through instantiation of several cluster cells. The size of the cluster is around 2045 μm^2. The core has 318 clusters and the total core area is 0.65mm^2.

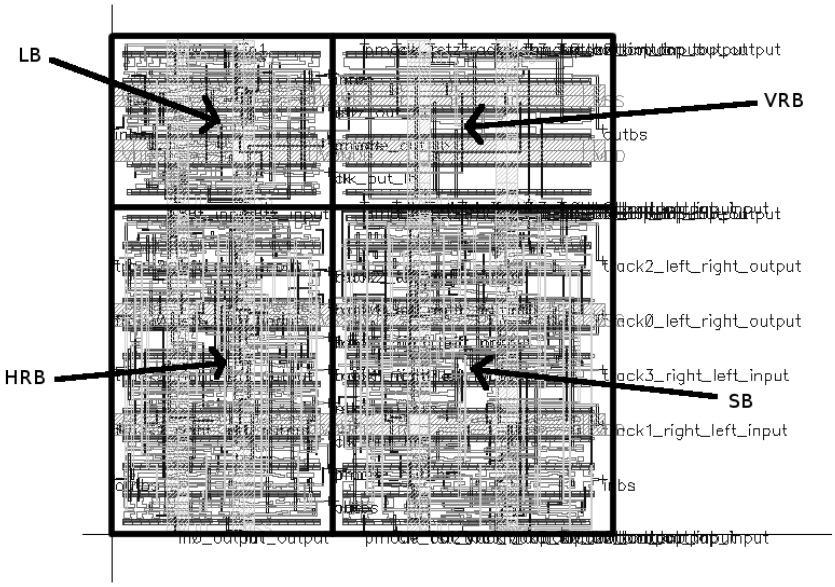

Figure 13. Cluster cell layout

Comparing Figures 3 and 13, it is possible to identify on the layout of a cluster cell the basic blocks that compose it.

Some cluster cells are highlighted on the programmable core layout shown in Figure 15.

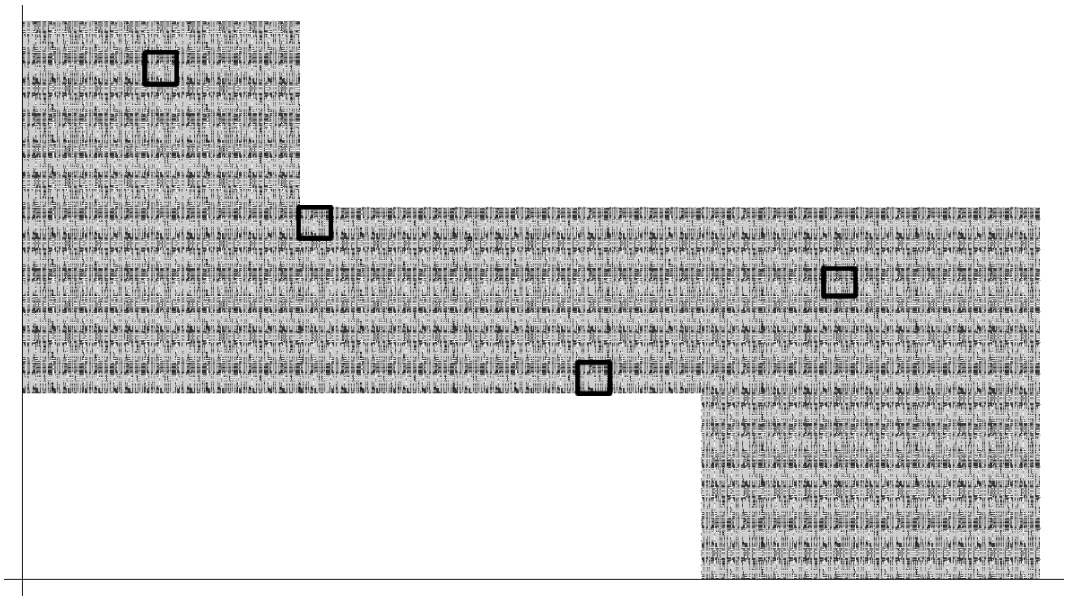

Figure 14. Programmable core layout for a S-shape core

5. CONCLUSIONS AND FUTURE WORK

The results obtained in this work demonstrate the feasibility of a flexible, technology-independent architecture for non-rectangular reconfigurable logic cores, that is supported by an automatic generation tool and that can be physically implemented using a standard digital design flow. The proposed work flow is based on the hierarchical organization of the core circuitry and allows a nonrectangular programmable core to be created and validated on a short design cycle time, as happens with any common fixed digital logic block.

We have identified several points of interest that should be taken into consideration in future work related to the current proposal. One direction is to have a more flexible core organization. A first step would be updating the generator to create cores with variants of the proposed architecture (for example, look-up tables with more inputs and a higher number of tracks on routing blocks). Another important step would be developing tools (or, perhaps, adapting existing tools) for quickly mapping any digital design on a generated core, so that parameters that characterize it, like routability and area-delay and delay-power products, can be easily evaluated.

REFERENCES

[1] Vaughn Betz, Jonatahn Rose and Alexander Marquardt, Architecture and CAD for Deep-Submicron FPGAs, Kluwer Academic Publishers (1999)

[2] Steven J. E. Wilton, Architectures and Algorithms for Field-Programmable Gate Arrays with Embedded Memory, PhD Thesis, University of Toronto (1997)

[3] J. Greenbaum, Reconfigurable logic in SoC Systems, Proceedings of the 2002 Custom Integrated Circuits Conference, pp. 5-8

[4] Steven J.E. Wilton and Resve Saleh, Programmable Logic IP Cores in SoC Design:Opportunities and Challenges, Proceedings of the Custom Integrated Circuits Conference (2001)

[5] Steven J. E. Wilton, Noha Kafafi, James C. H. Wu, Kimberly, A. Bozman, Victor O. Aken'Ova and Resve Saleh, Design Considerations for Soft Embedded Programmable Logic Cores, IEEE Journal of Solid-State Circuits, Vol. 40, No. 2, February 2005

[6] Cadence Design Systems, LEF/DEF 5.5 Language Reference, January 2009

[7] Free 45nm PDK, North Carolina State University, http://www.eda.ncsu.edu/wiki/FreePDK45:Contents

[8] Free 45nm PDK Standard Cells Library, Oklahoma State University, http://vcag.ecen.okstate.edu/projects/scells/