On the development of an island-style FPGA

Yang Azevedo Tavares [1], Diomadson Rodrigues Belfort [2], Sebastian Yuri Cavalcanti Catunda [3], Sabiniano Araújo Rodrigues [4], James Tandon [5]

[1] yangtavares@gmail.com. UFRN – Departamento de Engenharia de Computação e Automação. [2] diomadson@imd.ufrn.br. UFRN – Departamento de Engenharia Elétrica. [3] catundaz@gmail.com. UFRN – Departamento de Engenharia de Computação e Automação. [4] sabiniano@ifpb.edu.br. IFPB – Campus Santa Rita. [5] james.tandon@csueastbay.edu. California State University, East Bay – CSUEB Department of Engineering.

ABSTRACT

This paper presents the development of a custom SRAM island-style FPGA, covering the information needed and the steps involved in hardware implementation, bitstream configuration and design alternatives to facilitate the overall implementation effort from an academic point of view. To achieve the state of the art, commercial FPGAs can employ a large team, a high time-to-market, and high non-recurring engineering costs. In contrast, by taking the challenge of building a custom FPGA with a small team of researchers, the development of custom architecture and size focuses on the proof of concept. This baseline methodology result can be a start point for the development of new technologies or circuit enhancements.

Keywords: Field programmable gate arrays. Reconfigurable architectures. Circuit synthesis. Read-write memory. Design methodology.

RESUMO

Este artigo apresenta o desenvolvimento de um FPGA "island-style" com SRAM personalizado, contendo a informação necessária e os passos envolvidos na implementação em hardware, configuração por bitstream e alternativas de projeto para facilitar a implementação geral de um ponto de vista acadêmico. Para obter o estado da arte, FPGAs comerciais podem necessitar de um grande grupo, tempo de implementação e custos. Em contraste, ao aceitar o desafio de desenvolver um FPGA personalizado com um pequeno grupo de pesquisadores, o desenvolvimento do projeto é focado na prova de conceito. Resultados da metodologia apresentada podem ser o ponto de partida para o desenvolvimento de novas tecnologias ou aprimoramentos de circuito.

Palavras-chave: *Portas programáveis em campo. Arquiteturas reconfiguráveis. Síntese de circuito. Memória de leitura e escrita. Metodologia de projeto.*

1 Introduction

With their introduction in 1984, Field-Programmable Gate Arrays (FPGA) innovated the implementation of digital circuits, bringing flexibility and better design speed when compared to old methods such as prototyping using small-scale gates (TRIMBERGER, 2018). Since then, a large variety of applications have emerged with these advantages and the market has grown accordingly with the increased demand over the years. FPGA main benefits come from the fast design due to its reconfigurable behavior, decreasing the time-to-market solution and cost compared to the development of an Application Specific Integrated Circuit (ASIC) (PARVEZ; MEHREZ, 2011).

The ASIC development goes through several processes before being integrated into a final product, and considers the expenses of engineering development time and fabrication facilities. The fabricated chip also has a chance of malfunction due to design mistakes or fabrication process variations, increasing overall expense. Conversely, Commercial Off-The-Shelf (COTS) FPGA devices can be used in a final project product and can be cheaper for small and medium volume applications. They are widely employed for educational purposes and data centers, broadcast, high-performance computing, medical, and aerospace applications (XILINX, 2019).

Despite not reconfigurable, ASICs are optimized for their specific purpose, allowing to achieve higher performance than FPGA for the same job. However, as the IC technology advances and dies shrink, the FPGA area increases, allowing multicore implementation, increasing computational performance, and the gap with ASIC becomes smaller for several applications, such as an accelerating unit for processors applications (TRIMBERGER, 2018).

Unfortunately, the fierce competition in the industry hides the latest knowledge for economic purposes, which creates a gap between academic and industrial resources, as an example seen on Hung (2015). The importance of demystifying the FPGA knowledge then rises for its general behavior, and for improving the cost/benefit. The main building challenges of such architecture lies in the circuit design complexity and configuration with CAD tools, and the efforts involved to create an FPGA may require large teams, such as in industry. Therefore, developing methods to boost the process can be very welcome.

This paper goes through the custom FPGA building steps and challenges. Section 2 gives an overview of previous work on FPGA island-style architecture, considering optimal solutions for various circuit parameters and topology choices. Also, it describes a generic SRAM programmer which can be used in multiple FPGA architectures. Section 3 illustrates the work to implement the described architecture using robust commercial tools, highlighting design approaches and bottlenecks. Section 4 shows the bitstream generation process based on an open-source tool (Verilog-to-Routing). In Section 5, the simulation results of a 4-bit counter programmed into the developed FPGA is presented for testing purposes. Finally, Section 6 concludes the paper.

2 Architecture reference

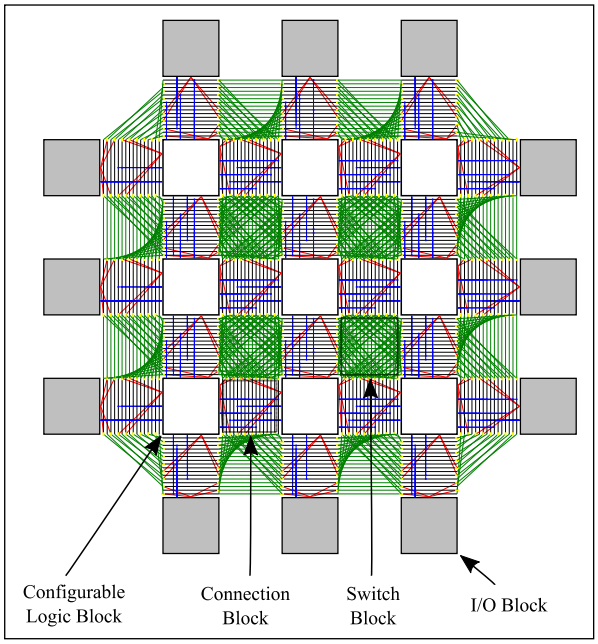
FPGA global routing (macroscopic allocation of wires) architectures can be characterized as either hierarchical or island-style. Depending on the architecture chosen, the designer will face trade-offs among VLSI implementation, CAD flow configuration, and device performance (KUON; TESSIER; ROSE, 2008).

Most commercial SRAM-based FPGAs use the island-style architecture for its notable routing capabilities. This architecture is represented in Figure 1 and consists of Configurable Logic Blocks (CLB) surrounded by routing buses that are responsible for connecting the configured blocks signals together. In this FPGA architecture, other specific function blocks responsible for implementing the matrix are the Connection Block (CB), which consists of programmable switches used to connect the CLB ports to the routing bus; and the Switch Block (SB), that connects the routing busses together.

Among the distinct blocks that an FPGA has the structures responsible for implementing the programmable logic are most important, bringing the device reconfigurability to come true. Despite recent architectures that may have different names for their reconfigurable logic, most of them share a well-known soft logic block (KUON; TESSIER; ROSE, 2008), and the commonly used structure to implement this behavior is known as the Look-Up Table (LUT). A LUT consists of memory to hold programmable data and multiplexer (Mux) for realizing a configurable logic function. The Mux, shown in Figure 2a, uses its addresses as logic function inputs and its input channels as programmable data for realizing the respective truth table. Figure

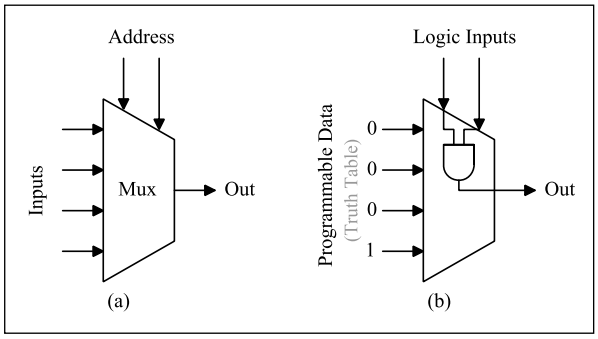
2b exemplifies a 2-input LUT programmed to be an AND gate.

Figure 1 – Illustration of an island-style FPGA. Adapted from: BETZ; ROSE; MARQUARDT (1999)



Source: Self elaboration.

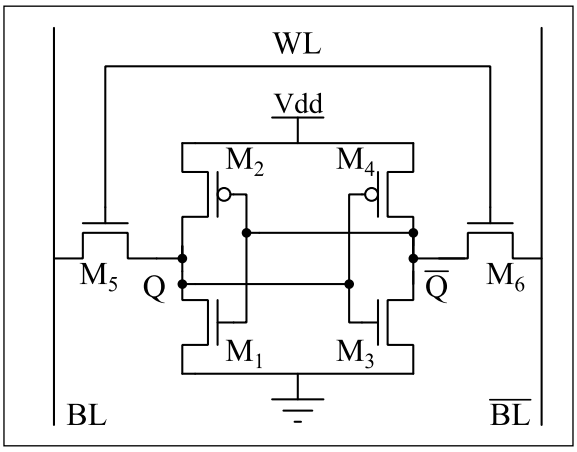
Figure 2 – Mux implementing a Look-Up Table logic



Source: Self elaboration.

With a LUT defined, a memory is required to implement the programmable inputs (as well as controlling the routing switches). Implementing the memory as a Static RAM (SRAM) is an excellent approach since this memory will not be frequently changing after the circuit is placed (compared to a Dynamic RAM). Also, the SRAM cell can be designed to have only 6 transistors to store a bit value (Figure 3), saving circuit area and power if compared with other alternative memory implementations, like using a D-latch.

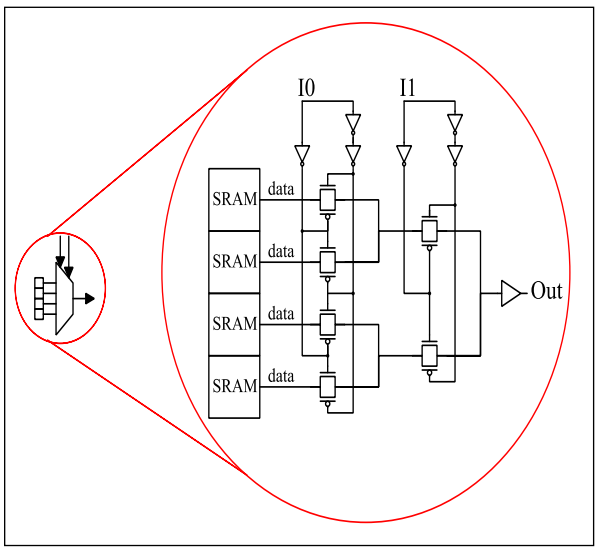
Figure 3 – 6T SRAM cell schematic



Source: Self elaboration.

As the FPGA employs more LUTs the area tends to grow accordingly, leading to use small LUTs to reduce area and improve FPGA logic, allowing more CLBs to fit on the chip die. Figure 4 presents a LUT multiplexer implemented as a binary tree of transmission gates, which is a good approach for size optimization. Another possible implementation consists of using pass transistors that are well known to be employed in these structures, but they require extra circuitry to handle a higher level voltage that drives the gate. The work in Chiasson and Betz (2013) shows the trade-off between both approaches.

Figure 4 – Look-Up Table multiplexer implemented as a binary tree of transmission gates

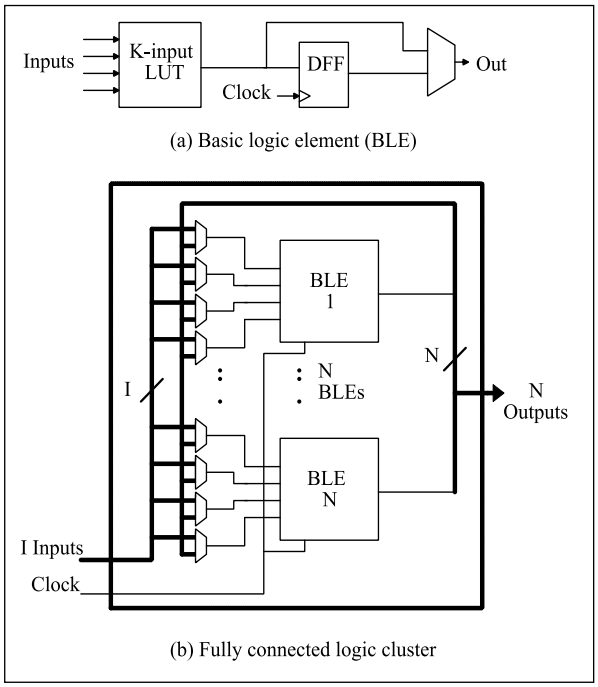


Source: Self elaboration.

The number of ports of a LUT is also a subject of optimization. The work in Ahmed and Rose (2004) explored the effect of a LUT number of inputs (K) on the area and speed performance of a configured circuit, comparing the number of LUTs required, the resulting area and the critical path delay for implementing 28 different circuits using different LUT sizes. The results of the study have shown that a 4-input LUT is optimal considering a homogeneous FPGA.

Heterogeneous FPGA architectures can achieve higher performance than homogeneous ones (PARVEZ; MEHREZ, 2011). Commercial heterogeneous architectures focus on most demanded blocks such as adders and digital signal processing units (INTEL, 2019a) to create hard logic blocks, optimizing speed and power of placed circuits. Such circuitry, if implemented by soft logic would be significantly bigger. Even when creating a homogeneous FPGA, it is common to use a hard logic D-type Flip-Flop (DFF) at the output of each LUT, making it easier to implement combinational or sequential logic. The structure composed of a LUT and a DFF is also known as a Basic Logic Element (BLE) and is represented in Figure 5a.

Figure 5 – FPGA Basic Logic Element and cluster.. Source: KUON; TESSIER; ROSE (2008)



BLEs can be interconnected to form FPGA Configurable Logic Blocks (CLB), as shown in Figure 5b. It is important to pack BLEs together in a CLB to reduce the routing buses since they can take a huge portion of the final area. Internal CLB routing also has its limitations with scalability, principally when using a fully connected architecture, where all inputs to the CLB can connect to any BLE input and so for its outputs, as in Figure 5b (KUON; TESSIER; ROSE, 2008).

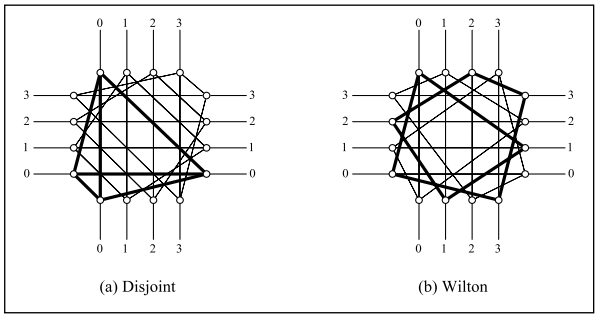
The study from Betz, Rose and Marquardt (1999) has shown that it is not necessary to have all BLEs inputs accessible from the CLB, because some inputs are open (not driven) by anything, and can share signals with each other or take outputs from a BLE within the cluster. A 98% logic utilization is achieved with approximately 60% of BLEs inputs individually accessible to the FPGA routing channels, or equivalently, 2N+2 inputs, where N is the number of BLEs in a cluster.

For the CB topology, the number of switches for routing tracks is another subject that needs to be well defined and optimized. A higher number of routing switches provide more flexibility but results in a bigger FPGA size and higher power consumption. On the other hand, a reduced number of switches turns the FPGA less routable and with a lower performance. The work of Betz, Rose and Marquardt (1999) shows that good flexibility is achieved when the percentage of the connected output () is , where is the number of routing tracks and is the number of BLEs in a CLB. The input connectivity should be somewhat larger.

A SB connects the FPGA routing buses between each other (Figure 6). The number of possible connections that a track can make to others inside a SB is known as SB flexibility. As this number changes, similar trade-offs with area and routability happen, as for the CB. Likewise, the wire length of a routing bus inside an FPGA represents how many CLBs a wire spans, meaning a SB can have some shorted tracks instead of switches to connect two wires apart. Shorted tracks can improve performance by removing the circuitry that comes with routing switches, but at the same time can decrease routability.

There are also different ways to connect the wires together inside the SB. Commonly used topologies such as Disjoint (WU; MAREK-SADOWSKA, 1995) and Wilton (WILTON, 1997) were investigated by its critical path delay (BETZ; ROSE; MARQUARDT, 2012). For a wire length of 1 (spanning only one CB), both topologies present similar critical path delays. However, with bigger wire lengths, the Disjoint SB leads to faster circuits compared to Wilton SB.

Figure 6 –SB topologies. (a) Wilton. (b) Disjoint



Source: Self elaboration.

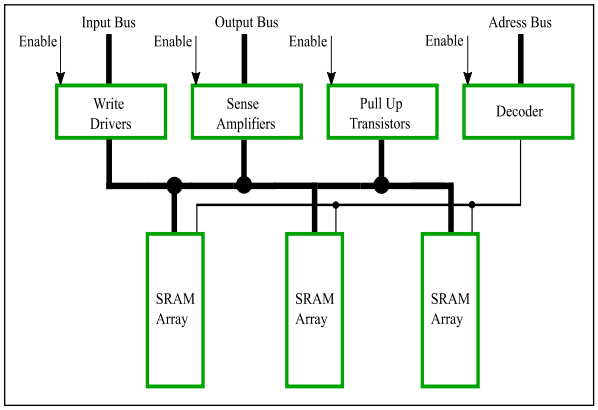
2.1 SRAM manager

FPGA programmable logic and routing switches can be implemented using SRAM cells. Hence, the SRAM configuration and management circuitries compose important pieces on the FPGA itself.

There are some SRAM drawbacks that should be considered. For example, the SRAM volatility brings the need for external devices to keep a circuit configuration on the FPGA. This repeated programming exposes the system to a security issue where the information can be stolen (KUON; TESSIER; ROSE, 2008). Also, the SRAM needs special circuitry to handle the analog nature of the signals for reading and writing data (Figure 3). The SRAM cell overall transistor size should be appropriately designed to avoid the noise on the bitlines to flip the stored bit, and at the same time, to let the access transistors drive the desired data on the cell (WESTE; HARRIS, 2015).

To properly program SRAM cells, the SRAM Manager (Figure 7) holds four different structures: A driver block to write the desired data using buffers, a decoder to select which SRAM row to program, a pull-up block, to prepare the bitlines to be read and a sense Amplifier block, to read the SRAM data.As experienced by our design construction, a common used SRAM flat addressing, where each SRAM connects directly with a specific routing or programmable cell using x and y coordinates, leads to a complex integration between the FPGA hardware description and synthesis tools. In our approach we implemented a resumed SRAM configuration system, which is especially suited to target specific FPGA locations by isolating each FPGA Tile with a decoder system. The different structures, such as LUTs, input multiplexers, and BLE output multiplexers, are selected by their specific address within a tile. The SRAM is programmed by using 16-bits words, which matches the number of SRAM cells required to implement a 4-input LUT.

Figure 7 – SRAM Manager diagram.



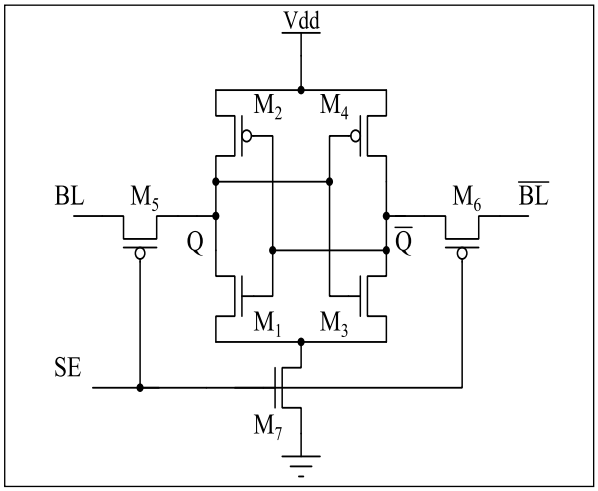
Source: Self elaboration.

Most commercial FPGAs are integrated with special circuitry to handle unexpected data on the SRAM cells. The errors may come from different sources such as the configuration flow itself, a Single-Event Upset (VILLA, 2017) or soft errors (SRINIVASAN *et al*, 2004). A simple approach to handle those unexpected data is to keep checking the SRAM data with reading operations. If an error is observed, the SRAM manager reconfigures the specific part.

The sense amplifier circuit is used to read a SRAM array data (Figure 8). It can distinguish slight differences on the bitlines in a read operation, and is required to accelerate the reading since the SRAM cross-coupled inverters slowly drive the bitlines due to its capacitance. Pulling the "SE" signal down makes the cross-coupled inverters to float with the bitlines, when the "SE" is raised, the node with higher charge will drive the result (BAKER, 2011). Before the sense amplifier operates, pull-up transistors should raise both bitlines to a high level for a fair comparison.

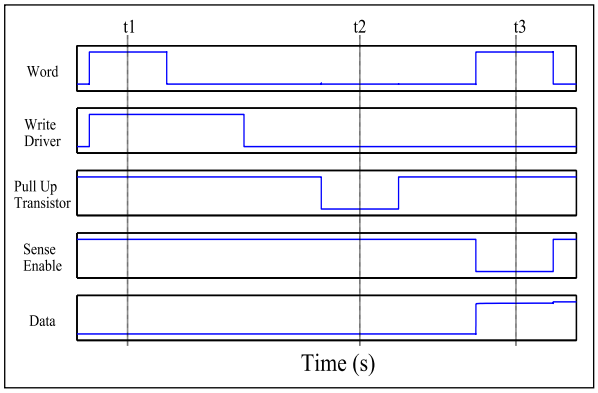
Figure 9 shows an example of the SRAM Manager block operating one bit when there is both a write and read operation happening. At "t1", the "Write Driver" is writing a logical '1' to the selected cell when the respective "Word" is enabled. At "t2", all the bitlines are pulled high with the "Pull Up Transistor" is set to a logical '0', since they are controlled by pMOS. Finally, at "t3" a logical '0' at the "Sense Enable" line permits the cell reading with pMOS access transistors. The sense amplifiers eventually measure which bitline is higher and show the logical value at the "Data" line.

Figure 8 – Sense amplifier schematic.



Source: Self elaboration.

Figure 9 – SRAM programming waves.



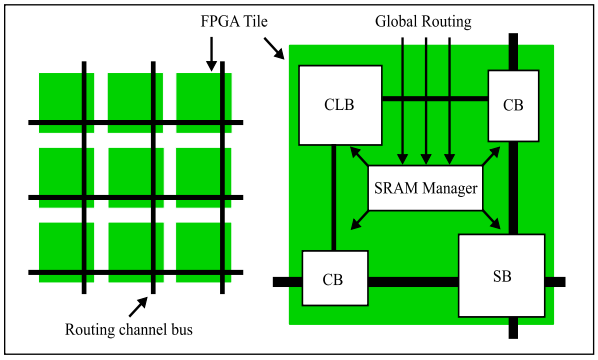
Source: Self elaboration.

This subsection illustrated the configuration flow of a SRAM manager for a single tile. By multiplying this methodology on the FPGA array, direct tile configuration is possible. A large outer decoder with global "Word" lines can select the chosen SRAM manager to execute.

2.2 Target architecture

In this work, we intend to rapidly implement an FPGA to get proof of concept, using common FPGA structures from the last decades (BETZ; ROSE; MARQUARDT, 1999; KUON; TESSIER; ROSE, 2008). We consider the island-style architecture, which can be easily divided into similar Tile structures. As seen in Figure 10, each Tile contains the blocks needed to implement both the reconfigurable logic and routing. A homogeneous FPGA architecture is chosen, meaning the same logic blocks distributed in the FPGA array. This approach enhances the implementation speed of the Tile structure.

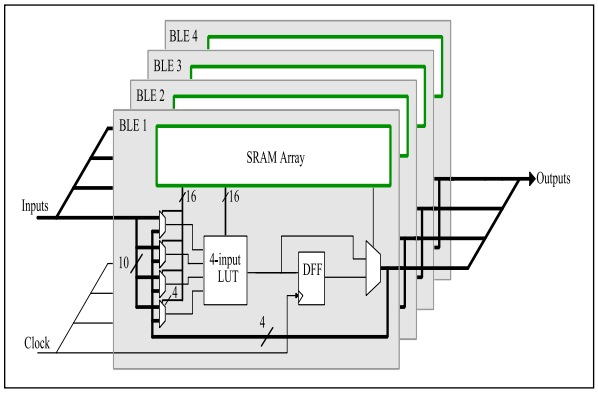
Figure 10 – Tile-based FPGA.



Source: Self elaboration.

The target CLB is composed of 4 BLEs cluster in a fully connected configuration (Figure 11), which means all the inputs and outputs of the CLB can be connected to any input of any LUT. Each LUT is implemented with 4 possible inputs. The CLB ports are evenly distributed around the CLB perimeter, with 10 logical equivalent inputs and 4 outputs. To achieve optimal area, inner muxes are implemented either as a binary tree of transmission gates or as a bench of switches.

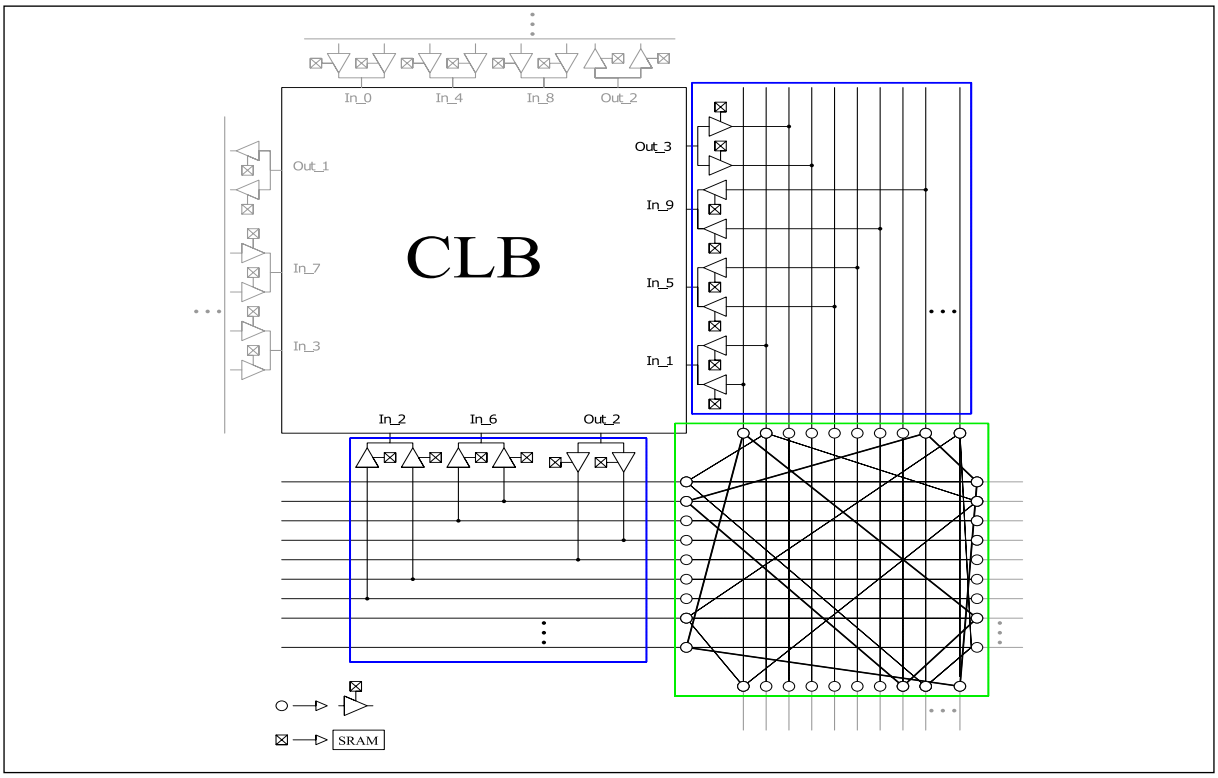
Figure 11 – Configurable Logic Block (CLB) schematic.



Source: Self elaboration.

Figure 12 shows the routing connection logic. For the CB, the connections are driven by the CLB ports. The CB can connect up to 25% of the wires in the routing channel to inputs and outputs of the CLB. The SB is a Wilton type, and each channel wire connects to three other wires from other channels. Tri-state buffers mainly compose the routing switches.

Figure 12 – Routing channel schematic, with Switch Block in green and Connection Blocks in blue.



Source: Self elaboration.

For our purposes, the wires expand only to one Tile, though some advantages on the use of different wire lengths across the FPGA are reported in the references. The channel width (number of tracks in a routing bus) is chosen to be 16, a power of 2 for design easiness, which is beyond the needed for the routing worst case, based on these same references.

3 System implementation method

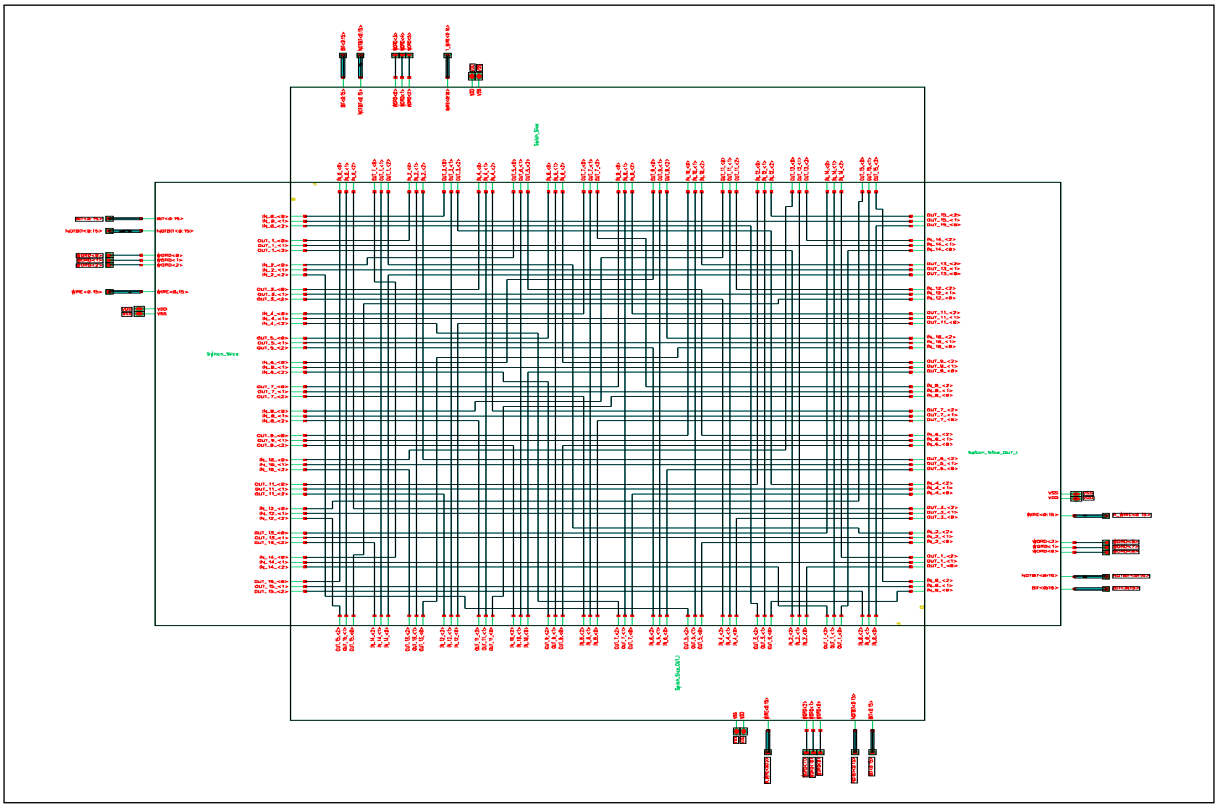
We firstly started building our design on a robust commercial environment such as Cadence Virtuoso, with a functionality test intent for each component with real MOSFET models from a 0.5 um process. Important analyses were made as verifying the SRAM analog behavior, observing the effect of transistor sizing for the read and write operations on the cells, and inspecting the effect of the transmission gate binary tree on the LUT muxes, and on the tri-state buffers in the routing buses. However, as the design gets bigger, more nets are integrated into the schematic, bringing more transistor variables to be taken into account, and increasing the simulation time.

Another issue that rises increasing the design size is that connecting various routing buses manually becomes very prone to human mistakes. These mistakes can only be found through analysis of simulation results from a programmed test bitstream, which can take several hours to find a single misconnected wire.

A small 4-CLB FPGA matrix was designed by hand, with a 3-metal layer 0.5 um technology, using the analog design flow to verify the functionality of several circuits. The main Tile was created with the essential blocks and considering different block connections in the FPGA IO pads perimeter. Figure 13 shows the SB schematic made on Virtuoso with each routing track connected by hand.

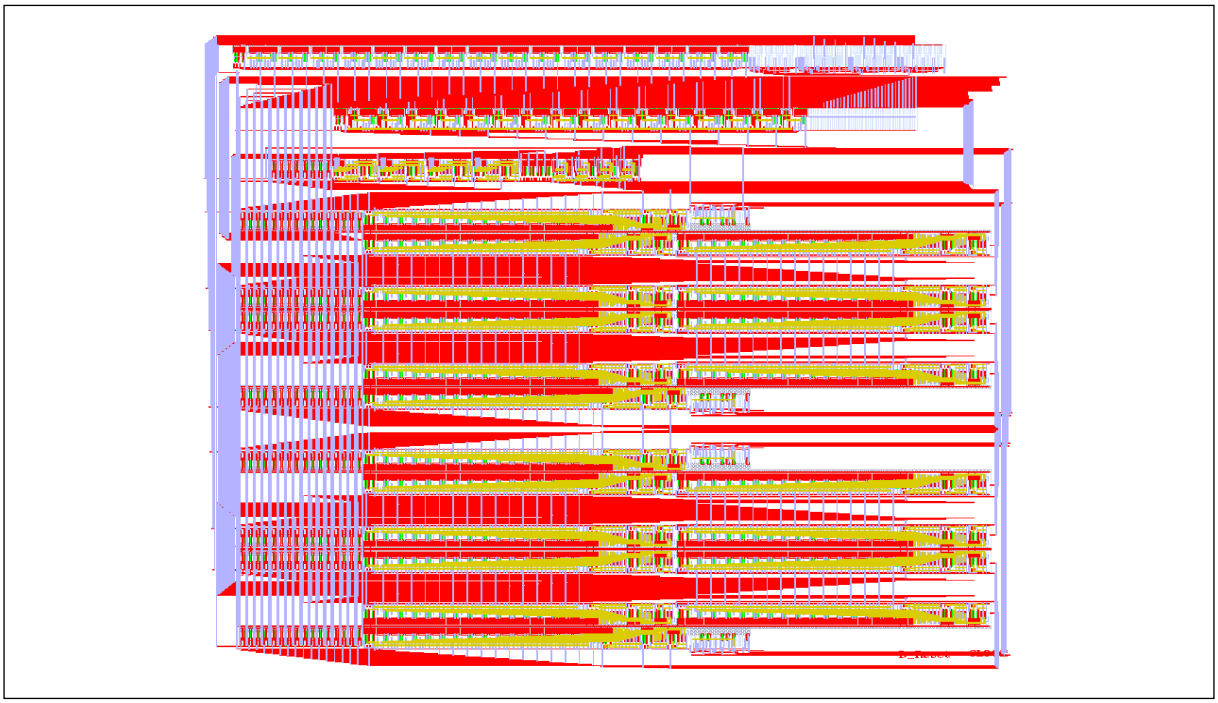
One of the main challenges for the layout is creating the core cells with similar height and proper width for port connections. Those choices influence how well the pieces couple together, possibly sparing design time. Sizing the cells is very important as they reflect directly on the final FPGA area. Conversely, smaller size cells make possible to include more logic cells in the matrix with the same die limitations. A CLB with a SRAM manager layout is shown in Figure 14 for design illustration.

Figure 13 – Switch Block schematic



Source: Self elaboration.

Figure 14 – Configurable Logic Block with SRAM manager layout



Source: Self elaboration.

The 3-metal layer technology brought high implementation effort and time but made by a small team. It is well visible in the figure that, still with a considerable attempt to optimize area, there are huge blank spaces on the layout, making it an inefficient area optimization.

3.1 Design automation

A complex design can bring exhaustive work, high non-recurring engineering costs, and high time-to-market solutions (PARVEZ; MEHREZ, 2011). Nowadays FPGA can go over 5.5 million logic elements (INTEL, 2019b). Padalia *et al* (2003) reported that modern FPGAs could consume anywhere from 50 to 200 man-years merely in the layout step.

A good way to reduce costs on the process is automating the design. The GILES project (PADALIA *et al*, 2003) demonstrated complete FPGA automation reducing significantly the manual labor. Kim and Anderson (2015) showed a CAD flow that takes information from an open-source tool and transforms it into a placed ASIC layout and a bitstream configuration. Both approaches use commercial tools and not necessarily include open-source products.

Parvez and Mehrez (2011) implemented various FPGA architectures based on Alliance and Coriolis open-source tools, by combining standard library cells and dynamic layout placement from subsequent software. This method has a high academic value, enabling the researcher to explore architectures without needing to spend extra money on commercial tools. However, the available standard library cells are not necessarily compatible with industry fabrication due to NDA reasons. The difficulty then rises from the effort of implementing the standard library using compatible software flow.

Our design was also built upon the Cadence digital environment. The design methodology begins by describing the architecture on structural Verilog, instantiating basic FPGA modules (such as multiplexers, d-flip-flops, tri-state buffers, and memory) and combining then to more complex FPGA structures. The HDL code can be generated from a custom software or directly written. Figure 15 shows a BLE Verilog sample.

The described circuit in Verilog then serves as input to the Encounter RTL compiler, which transforms a standard library cell in a synthesized Verilog, based on design constraints. The design uses a 0.5 um CMOS standard library cell and, since these cells are not specified for FPGA purpose, area and performance may be degraded. Likewise, the chosen cell library does not support SRAM architectures but replaces the memory with latches. However, these discrepancies are accepted for academic proof-of-concept purposes.

Figure 15 – Basic Logic Element Verilog.

```
module  ble4(
ble_logic , // LUT programmable logic
sel , // Mux selection
registered , // Registered or not
clk,
reset,
ble_out   // Mux output
);
input [15:0] ble_logic;
input [3:0] sel;
input registered;
input clk, reset;
output ble_out;
wire mux_out, dff_out;
mux_16x1 U0(
.din (ble_logic),
.sel (sel),
.mux_out (mux_out)
);
dff_async_reset U1(
.data (mux_out),
.clk  (clk),
.reset (reset) ,
.q     (dff_out)
);
mux_2x1 U2(
.din_0 (mux_out),
.din_1 (dff_out),
.sel   (registered),
.mux_out (ble_out)
);
endmodule
```

Source: Self elaboration.

The layout can then be generated from the synthesized Verilog. Cadence SoC Encounter takes the HDL description, and places and routes the cells automatically with its optimized algorithms. Figure 16 shows the placed and routed custom FPGA using a top-down synthesis. Kim and Anderson (2015) discuss FPGA synthesis approaches which will be considered for future work.

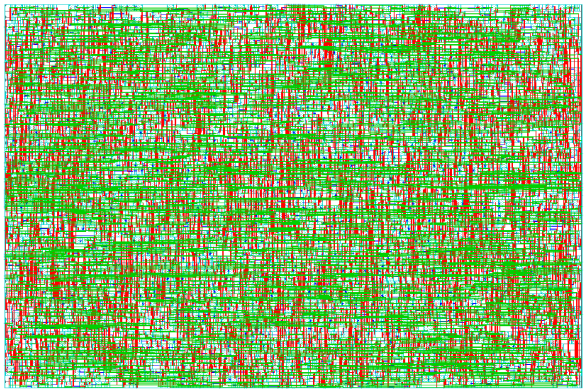
To illustrate the differences between the "hand-made" and automated approaches, Table 1 compares for a single CLB, the time and area. The obtained results are for the design implementation by one person, and the hand-made layout is not precisely optimized or do not use the minimum transistor size.

Table 1 – Design effort comparison

Work topic	Hand-made/ Schematic editor	Automated/ Verilog
Schematic implementation time	1 week	2 seconds
Layout placement time	2 months	30 seconds
Layout are	859000 λ^2	302588 λ^2

Source: Self elaboration.

Figure 16 – 2x2 FPGA placed on Cadence SoC Encounter



Source: Self elaboration.

3.2 FPGA bitstream configuration

The packing, placement, and routing planning for a given FPGA circuit are crucial for the resulting efficiency, and a significant power and speed improvement depends on how each path and logic is configured (BETZ; ROSE; MARQUARDT, 1999). The HDL high-level circuit description must go through a series of algorithms and optimizations before the synthesizer is able to generate a final bitstream. Although there are some open-source tools capable of generating FPGA configuration bitstreams (LAGADEC *et al*, 2001; COOLE; STITT, 2010; SONI; STEINER; FRENCH, 2013), those are target-dependent on commercial FPGAs. While these tools are interesting to explore the state of art point of view configuration, they are not suited to the aim of this work architecture. Therefore, is necessary to develop a bitstream generator compatible with a custom FPGA architecture.

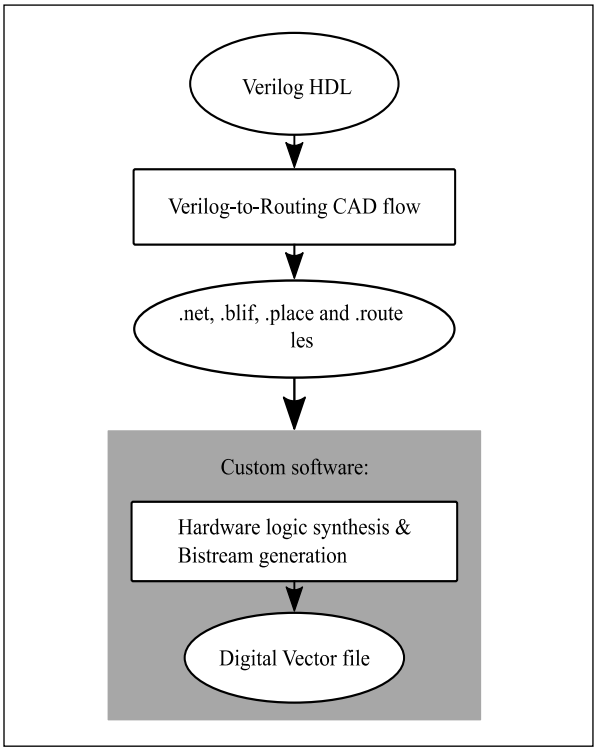
Verilog-to-Routing (VTR) is a good start point to begin studying HDL synthesis for a custom FPGA architecture. The VTR takes as input a circuit described in Verilog, and gives as output path and logic setting file, based on the FPGA architecture. It begins by converting the Verilog HDL design into a netlist of gates considering the different possible blocks the architecture can have. A logic optimization takes place to map the netlist into LUTs and flip-flops. The logic units are packed together into clusters, which will then be placed by using a customized simulated-annealing algorithm, achieving a quick solution close to the global best solution, and routed at the end (ROSE *et al*, 2012; LUU *et al*, 2014).

VTR is an open source CAD flow intended for academic purposes and benchmarking. It outputs a high-level configuration files that are not ready for direct programming a custom FPGA, needing to be interpreted.

3.3 VTR interpreter

Figure 17 illustrates the developed VTR interpreter flow, capable of reading files from the VTR and converting them into a bitstream configuration suitable for programming the designed custom FPGA. The VTR provides four different files, a .blif file in the Berkeley Logic Interchange Format (BLIF) that contains information about the LUTs netlist, truth table, and synchronization tags; a .net file that describes the logic elements cluster netlist in an XML structure; and a .place and a .route files that give information on how to place and route the HDL circuit, respectively.

Figure 17 – Custom Software flow chart.



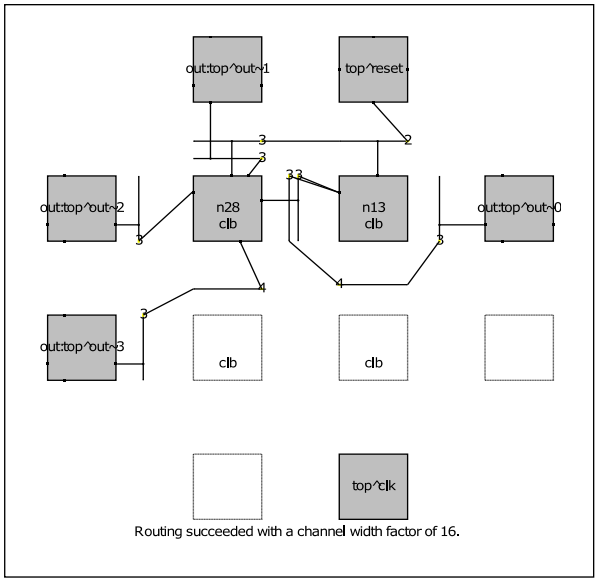
Source: Self elaboration.

The VTR interpreter flow starts with an XML parser (VEILLARD, 2003) that is used to transform the .net file into a table with the tags name, value, and depth in the logic block hierarchy. Next, the BLIF file is read, getting the Boolean Algebra of each LUT and verifying whether the LUT output is registered or not.

Our system then reads the placement file that has all the tile names and respective addresses in the format "Tile name, X coordinate, Y coordinate", and merge these together with the BLIF and Net files information. The next step is the Route file interpretation, which gets information about the interconnections among routing blocks and their switches to an explicit low-level description for the FPGA. Finally, the VTR interpreter uses a custom auxiliary file, containing hardware association with the synthesized VTR output files, to generate the FPGA configuration bitstream.

To help making the link between the VTR and the target FPGA hardware, the developer can use the tool "VPR-Viewer" (BETZ; ROSE, 1997), available in the VTR package. This tool makes possible to verify how the described architecture given by the VTR flow will be implemented in the FPGA architecture. Figure 18 shows an example of a counter circuit generated using the VPR Viewer.

Figure 18 – Counter circuit generated on VPR Viewer.



Source: Self elaboration.

3.4 Configuration interface

As the number of parallel pins needed to configure the FPGA might be expressive, a custom serial-to-parallel and a parallel-to-serial interface were built. The configuration bitstream is written to the FPGA SRAM through the serial-to-parallel interface and can be retrieved from the SRAM as an output bitstream, through the parallel-to-serial interface. Another elegant solution could implement the Joint Test Action Group (JTAG) interface, which is widely used to program and test FPGAs and other digital systems (GRUWELL; ZABRISKIE; WIRTHLIN, 2016).

4 Functionality result

For the proof of concept, the following subsections will demonstrate the functionality of a small-sized FPGA developed. Both simulation and experimental results are presented, which validates the aforementioned methods.

4.1 Simulation result

The architecture of a 4-CLB FPGA, presented in Section 3, and the bitstream generation flow from Section 4 was put together to test all the procedures done so far. A test bench was created consisting of a 4-bit digital counter, described in Verilog and presented in Figure 19, and a stimuli generator for the counter clock and reset, described in Cadence Virtuoso digital vector file. The counter routing visualization is presented in Figure 18, and the circuit was programmed into the FPGA, being simulated for the stimuli signals in the Cadence Virtuoso Analog Environment, for more accurate results in a real-scenario. Simulation results are shown in Figure 20, with the first two signals as stimuli and the last four signals as counter simulated output bits.

Figure 19 – Verilog counter description

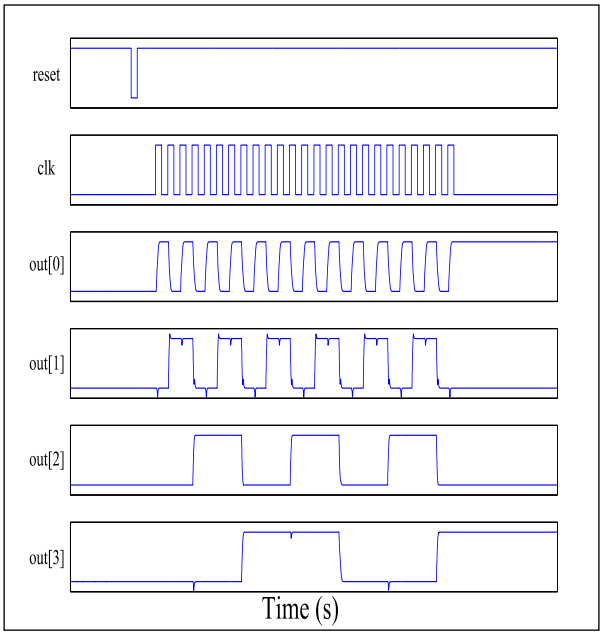
```
module up_counter    (
out     ,  // Output of the counter
clk     ,  // clock Input
reset      // reset Input
);
output [3:0] out;
input clk, reset;
reg [3:0] out;
always @(posedge clk)
if (reset) begin
out <= 3'b0 ;
end else begin
out <= out + 1;
end
endmodule
```

Source: Self elaboration.

From the simulation results, was verified the correct 4-bit counter operation, indicating the correct functionality of the FPGA and programming flow. Some glitches can be observed on the output bits signals, which are revealed by the analog simulation.

Nevertheless, these "nonidealities" do not compromise the FPGA proper operation.

Figure 20 – Simulation results for a 4-bit counter programmed into the FPGA.



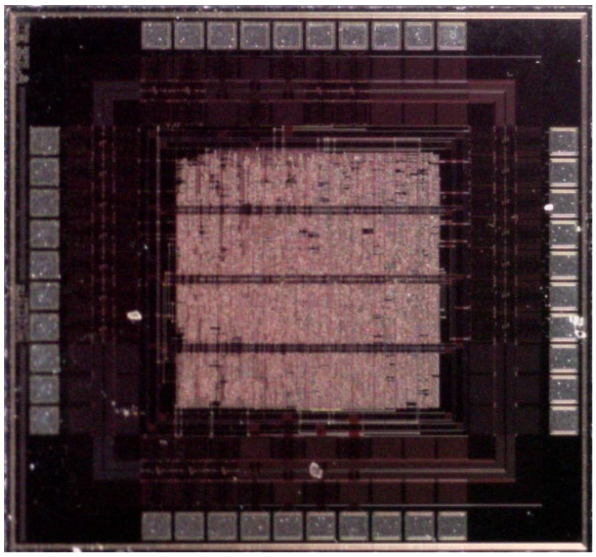
Source: Self elaboration.

4.2 Measurement result

A single CLB architecture was synthesized and fabricated with the same specifications described in subsection 2.2. The die micrograph is shown in Figure 21. Guard rings, power stripes, and other routing techniques were added to improve the overall circuit performance. The National Instruments 6351 (NI 6351) device was used as an interface to configure a 4-bit counter onto the chip, while also providing the appropriate inputs and managing the chip output for further analysis. The MATLAB software was employed to facilitate the NI 6351 device operation and chip data visualization. The data acquisition board was able to work under a frequency of 1 MHz.

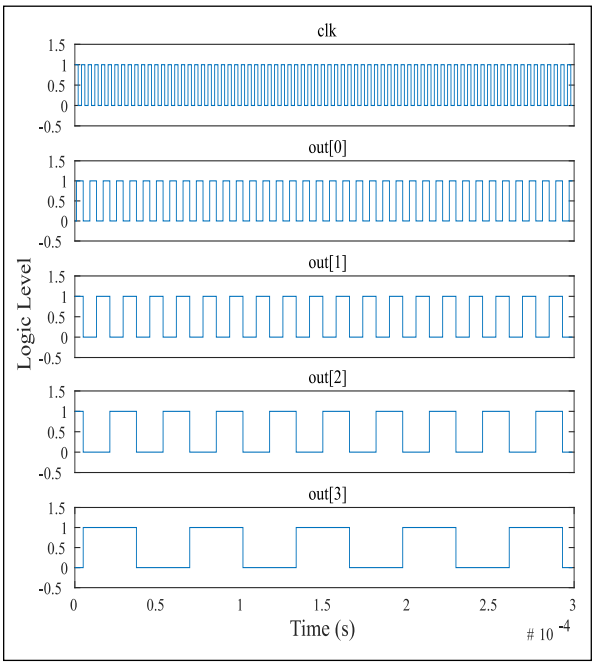
Figure 22 shows the measurement result for the 4-bit counter programmed. It is possible to observe the good functionality of the configured circuit. The analog behavior seen in Figure 20 was suppressed in Figure 22 due to the low-frequency operation and additional interface circuitry.

Figure 21 – Single CLB die micrograph.



Source: Self elaboration.

Figure 22 – Measurement result for a 4-bit counter programmed into the fabricated chip.



Source: Self elaboration.

5 Conclusion

In this paper, the development of a custom SRAM island-style FPGA was proposed and achieved, by a small team. Various topics were considered, such as the steps involved in hardware implementation, bitstream configuration, and design alternatives to facilitate the overall effort.

Firstly, the FPGA CLBs schematic and layout were designed by hand. Although this procedure can be employed to achieve an optimized circuit, this proposed work presents a high design effort, especially considering a small team. Next, the same design was developed CAD automation synthesis from a description of the circuits in Verilog and presented an expressive reduction in design time.

The presented work can help and stimulate researchers to try new FPGA implementations and to make improvements on all steps involved by the main FPGA development flow. Designers can also implement custom FPGAs in new technologies such as those which integrate systems on chip (SoC) to achieve higher performance.

Although the results bring a functional FPGA, the available development resources still need custom integration tools. Future works include open-source hardware tools implementation that couples with synthesis software such as Verilog-To-Routing.

REFERENCES

AHMED, E.; ROSE, J. The effect of LUT and cluster size on deep-submicron FPGA performance and density. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, v. 12, n. 3, p. 288-298, 2004.

BAKER, R. J. **CMOS: circuit design, layout, and simulation**. 3rd Ed. Wiley-IEEE, 2011.

BETZ, V.; ROSE, J. VPR: A new packing, placement and routing tool for FPGA research. In: LUK, W.; CHEUNG, P. Y. K.; GLESNER, M. (eds). **International Conference on Field Programmable Logic and Applications (FPL 1997). Lecture Notes in Computer Science**, vol 1304. Springer, p. 213-222, 1997.

BETZ, V.; ROSE, J.; MARQUARDT, A. **Architecture and CAD for deep-submicron FPGAs**. Springer, 1999.

CHIASSON, C.; BETZ, V. Should FPGAs abandon the pass-gate? In: 2013 23rd International Conference on Field programmable Logic and Applications. 2013, Porto (Portugal), **Proceedings...** 2013. p. 1-8.

COOLE, J.; STITT, G. Intermediate fabrics: virtual architectures for circuit portability and fast placement and routing. In: 2010 IEEE/ACM/IFIP INTERNATIONAL CONFERENCE ON HARDWARE/SOFTWARE DESIGN AND SYSTEM SYNTHESIS (CODES+ISSS). 2010, Scottsdale (United States), **Proceedings....** 2010, p. 13-22.

GRUWELL, A.; ZABRISKIE, P.; WIRTHLIN, M. High-speed FPGA configuration and testing through JTAG. In: 2016 IEEE AUTOTESTCON. 2016, Anaheim (United States), **Proceedings....**, 2016, p. 1-8.

HUNG, E. Mind the (synthesis) gap: Examining where academic FPGA tools lag behind industry. In: 2015 25th International Conference on Field Programmable Logic and Applications (FPL). 2015, London (United Kingdom), **Proceedings....** 2015, p. 1-4.

INTEL. Digital Signal Processing blocks in Stratix series FPGAs. Available in: <https://intel.ly/2LkiHgy**>**. Accessed in: jul., 2019a.

INTEL. Intel Stratix 10 GX/SX device overview. Available in: < https://intel.ly/2PA5ykP**>**. Accessed in: jul., 2019b.

KIM, J. H.; ANDERSON, J. H. Synthesizable FPGA fabrics targetable by the Verilog-to-Routing (VTR) CAD flow. In: 2015 25th INTERNATIONAL CONFERENCE ON FIELD PROGRAMMABLE LOGIC AND APPLICATIONS (FPL). 2015, London (United Kingdom), **Proceedings...** 2015. p. 1-8.

KUON, I.; TESSIER, R.; ROSE, J. **FPGA architecture**: survey and challenges. Now Foundations and Trends, 2008.

LAGADEC, L. *et al.* Placing, routing, and editing virtual FPGAs. In: BREBNER, G.; WOODS, R. (eds). **International Conference on Field Programmable Logic and Applications (FPL 2001). Lecture Notes in Computer Science**, vol 2147. Springer, p. 357-366, 2001.

LUU, J. *et al.* VTR 7.0: Next generation architecture and CAD system for FPGAs. **ACM Transactions on Reconfigurable Technology and Systems (TRETS)**, v. 7, n. 2, p. 6:1-6:30, 2014.

PADALIA, K. *et al.* Automatic transistor and physical design of FPGA tiles from an architectural specification. In: 2003 ACM/SIGDA 11th INTERNATIONAL SYMPOSIUM ON FIELD PROGRAMMABLE GATE ARRAYS. 2003. Monterey (United States), **Proceedings....** 2003, p. 164-172.

PARVEZ, H.; MEHREZ, H. **Application-specific mesh-based heterogeneous FPGA architectures**. Springer, 2011.

ROSE, J. *et al.* The VTR project: architecture and CAD for FPGAs from verilog to routing. In: 2012 ACM/SIGDA 11th INTERNATIONAL SYMPOSIUM ON FIELD

PROGRAMMABLE GATE ARRAYS. 2012. Monterey (United States), **Proceedings....** 2003, p. 77-86.

SONI, R. L.; STEINER, N.; FRENCH, M. Open-source bitstream generation. In: 2013 IEEE 21st ANNUAL INTERNATIONAL SYMPOSIUM ON FIELD-PROGRAMMABLE CUSTOM COMPUTING MACHINES. 2013, Seattle (United States), **Proceedings...** p. 105-112.

SRINIVASAN, S. *et al*. Improving soft-error tolerance of FPGA configuration bits. In: 2004 IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN (ICCAD 2004), 2004, San Jose (United States), **Proceedings...** 2004, p. 107-110.

TRIMBERGER, S. M. S. Three ages of FPGAs: a retrospective on the first thirty years of FPGA technology: this paper reflects on how Moore's law has driven the design of FPGAs through three epochs: the age of invention, the age of expansion, and the age of accumulation. **IEEE Solid-State Circuits Magazine**, v. 10, n. 2, p. 16-29, 2018.

VEILLARD, D. The XML C parser and toolkit of Gnome. 2003. Available in: <**http://xmlsoft.org>**. Accessed in: jul., 2019.

VILLA, P. R. C. *et al*. Analysis of single-event upsets in a Microsemi ProAsic3E FPGA. In: 2017 18th IEEE LATIN AMERICAN TEST SYMPOSIUM (LATS), 2017, Bogota (Colombia), **Proceedings...** 2017. p. 1-4.

WESTE, N. H. E.; HARRIS, D. **CMOS VLSI design: a circuits and systems perspective.** 4th ed. Pearson India, 2015.

WILTON, S. J. E.. **Architectures and algorithms for field-programmable gate arrays with embedded memory**. 1997. 181 f. Thesis (Doctorate in Electrical and Computer Engineering), Department of Electrical and Computer Engineering, **University of Toronto, Toronto (Canada**), 1997.

WU, Y. L.; MAREK-SADOWSKA, M. Orthogonal greedy coupling-A new optimization approach to 2-D FPGA routing. In: 32nd DESIGN AUTOMATION CONFERENCE. 1995, San Francisco (United States), **Proceedings....** 1995. p. 568-573.

XILINX. **FPGA applications**. Available in: <https://www.xilinx.com/applications.html/>. Accessed in: jul., 2019.