A platform independent tool for programming, visualization and simulation of simplified FPGAs

Marko Čupić, Karla Brkić, Željka Mihajlović
University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10 000 Zagreb, Croatia
{marko.cupic, karla.brkic, zeljka.mihajlovic}@fer.hr

***Abstract*—During the last few decades complex programmable circuits have seen a widespread usage in various digital circuit applications. One prominent example are Field Programmable Gate Arrays (FPGAs). Teaching FPGA technology has become an integral part of introductory digital logic courses. However, implementing Boolean functions in this technology requires understanding of several steps that are not trivial, including Boolean function decomposition, mapping the design into physical programmable units and routing. We have developed a portable Java-based tool which allows students to experiment with the required steps for arbitrary Boolean functions and to simulate the obtained implementation. In this paper, we give an overview of the developed tool and discuss its usage in class.**

I. Introduction

At the University of Zagreb, Faculty of Electrical Engineering and Computing, Digital Logic is a first-year course. On this course students are taught various concepts including basics of Boolean algebra, combinatorial circuits, sequential circuits, D/A and A/D conversions, implementations of memory modules as well as transistor implementations of basic logical gates and the implementations of standard programmable circuits. We cover Programmable Array Logic - PAL, Programmable Logic Array - PLA and Field Programmable Gate Array - FPGA. PLAs and PALs are rather simple, intuitive and easy to understand, so students can see the meaning of each bit used in their programming. This is not true for FPGAs, whose conceptual structure is known, but the direct correspondence between programming stream bits (generated by commercial development tools) and each programmable element (either pass transistors, LUT SRAM cells etc.) is unknown.

The working of an FPGA can be demonstrated to students by describing some digital circuit using a hardware description language (e.g. VHDL), then applying some magic (use multi-gigabyte commercial programming tools) which will properly configure the black box (i.e. the FPGA) and then observing that the black box behaves as the described digital circuit should. However, this approach is not adequate if deeper understanding of the technology is desired. In order to demystify this process, students are often taught the conceptual structure of FPGAs (for example, LUT-based implementations) and then are left to imagine how everything else works.

Our research goal is to propose a tool that allows students to explore the complete process which starts with a Boolean function to be implemented, goes through each implementation step and ends with a fully programmed FPGA. We believe that introducing such a tool in class lectures could help students to attain a much better understanding of the topic. Additionally, it would be beneficial if for the resulting FPGA students would be able to see the complete programming (every switch, every memory cell, etc.) and if it would be possible to interactively change the values on input pins and observe how signal changes in each programmed element eventually result in a new output value. Therefore, adhering to constructive research methodology, we have developed a graphical portable Java-based tool which emulates a variety of simplified LUT-based FPGA architectures and which allows students to define an FPGA architecture and a Boolean function to be implemented, performs all implementation steps and finally offers a visual inspection of the programmed FPGA as well as an interactive simulation of its work.

II. Related work

The behavior of an FPGA can be defined either through a schematic design or using a hardware description language such as VHDL [1] or Verilog [2]. The process of mapping the desired behavior to FPGA architecture is then carried out, typically using proprietary software specific to each FPGA manufacturer [3]. This process consists of Boolean function decomposition, which determines how the Boolean function will be fitted to CLBs, placement, which determines which physical CLBs will be used, and routing, which determines how the physical CLBs will be connected [4]. We now briefly outline some important considerations in each stage and the solutions proposed in literature.

1) Boolean function decomposition: The decomposition of the user-specified Boolean functions needs to be carried out in a manner that ensures that the functions can be realized using the available type of CLBs. For instance, it could be the case that a Boolean function is defined with n inputs, and the available CLBs have k inputs where $k < n$. Furthermore, redundancy should be minimized, so that CLBs should be shared between different Boolean functions whenever possible. As there are no computationally efficient algorithms that guarantee optimal decomposition, a number of solutions have been proposed in related work, optimizing according to various criteria. For instance, Łuba and Selvaraj [5] propose representing the function by a set of r-partitions over the set of minterms and interleaving serial and parallel decomposition strategies.

Nowicka et al. [6] propose an approach that produces a number of solutions that meet different optimization criteria.

2) Placement and routing: When mapping to physical CLBs, one major consideration is how to assign physical CLBs so the latency between CLBs is minimized. Moreover, determining which routing channels to use to connect two CLBs is a complex optimization problem with many trade-offs. For instance, one might connect two CLBs with the shortest possible route to ensure minimum latency, which might have the consequence that some other CLBs cannot be connected using that channel resulting in unwanted latency in some other components. Enzler et al. [7] propose a measure of quantification of an FPGA design that enables early trade-off considerations. Placement and routing can be optimized using heuristic methods, such as e.g. genetic algorithms [8], combined genetic algorithms and simulated annealing [9], or other optimization techniques such as min-cut or quadratic optimization [10].

3) Hardware constraints: When the logical design is transferred onto an actual physical chip, physical constraints of the hardware environment should be respected. Physical FPGAs are typically connected to other hardware in a fixed way (for instance, pin 1 might be connected to a physical LED). As an additional constraint, placement and routing should take into account this hardware environment, so that logical signals are mapped to appropriate pins. Being able to constrain placement and routing is also important when optimizing power consumption [11].

In our experience teaching FPGA design in early stages of university education, students find it difficult to use proprietary vendor-specific software for FPGA programming. We attribute this to the fact that this type of software is mainly intended for hardware professionals, exposing many complex options that students do not learn in their curriculum. Additionally, many of the inner workings of the synthesized FPGAs are hidden from view, being protected as a trade secret. The tool proposed in this paper help us to alleviate the described problems.

Existing e-learning solutions for FPGA design [12], [13] mainly emphasize having the students interact with a physical board. El Medany [12] proposes an FPGA hardware remote laboratory that can be accessed by students over the internet. Garcia-Zubia et al. [13] propose a system intended for remotely teaching digital electronics that combines a step-by-step digital circuit designer, a real FPGA board and a virtual simulation of a complex watertank that is controlled by the physical board. The virtual simulation aims to expand the range of possible exercises that can be done using a remote FPGA board, given that the outputs of such board are limited. Further details on the proposed system can be found in [14].

We believe in the importance of teaching FPGAs to students, as FPGA technology is widely used and constantly developed. For instance, recently there has been an interest in combining FPGAs with embedded microprocessors, as in e.g. Xilinx Zynq-7000 programmable SoC [15], offering the possibility to easily integrate and intermix the software programmability of a processor and the hardware programmability of an FPGA.

III. Simplified FPGA architecture

In order to explain the functioning and the programming of an FPGA we have developed a simplified FPGA architecture combined with a Java-based visualization and simulation program. The main parts of this FPGA architecture are I/O blocks, configurable logic blocks (CLBs), wiring segments, switch boxes and programmable switches for signal input and output from CLBs.

I/O blocks can be used to bring signals from the FPGA's environment into the FPGA (when configured as inputs) or to bring Boolean functions calculated by the FPGA to the FPGA's environment (when configured as outputs). In our simplified architecture, once programmed, a pin can not change directionality.

Configurable logic blocks in our architecture mimic typical CLBs in todays mainstream FPGAs. Each CLB (see Figure 1) is composed from a look-up table (LUT), a D flip-flop and a multiplexer which selects what is routed to CLBs output: the output of the LUT or the output of the flip-flop output.

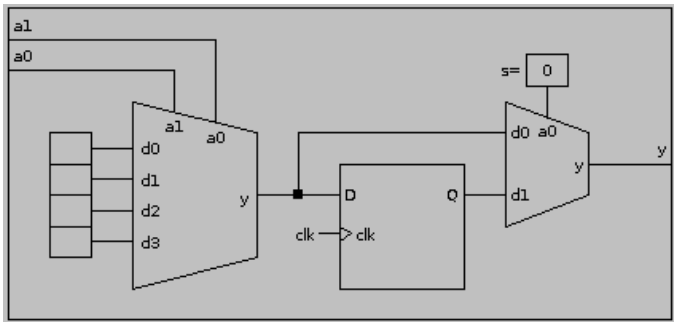

Fig. 1. The structure of a CLB

A LUT is a structure which uses 2^k user-programmable SRAM cells and one multiplexer $2^k/1$ with k address inputs which are also CLB inputs. Each SRAM cell is connected to one data input of the multiplexer. This structure allows us to directly realize any Boolean function of up to k variables simply by writing its truth table into SRAM cells. To illustrate how this works, Figure 2 shows an implementation of the Boolean function $f(A, B) = A \cdot B$ using a two-variable LUT.

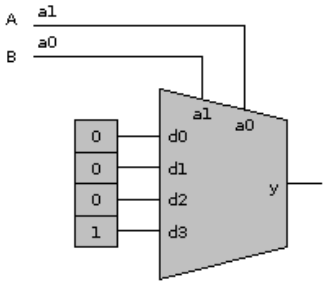

Fig. 2. A LUT programmed to calculate $f(A, B) = A \cdot B$

In our simplified architecture, each CLB receives inputs from the wiring segment located left from the CLB and

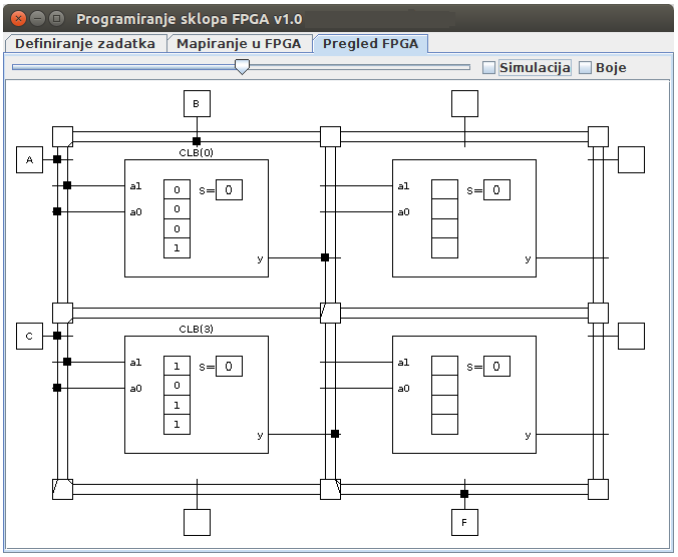

Fig. 3. FPGA programmed to generate function $f = A \cdot B + \bar{C}$

generates output which can be placed onto the wire from the wiring segment located right from the CLB. Each CLB input has its own programmable multiplexer which can be configured to route a signal from a single segment wire. Outputs are routed into a segment wire using pass transistors.

The function of switch boxes is to route signals from input pins or CLB outputs to output pins and CLB inputs. There are various implementations of switch boxes, for example the Disjoint Switch Box [16], [17], the Universal Switch Box [18] and the Wilton Switch Box [19]. The switch boxes we use in our simplified architecture allow the creation of connections between wires from each segment and wires from remaining segments.

An illustrative example is shown in Figure 3, where FPGA is used to implement the Boolean function $f = A \cdot B + \bar{C}$. The FPGA shown has four two-input CLBs in a 2×2 matrix, eight I/O pins, six horizontal and six vertical wire segments, each comprised of two wires. Wire segments are connected using nine switch-boxes. Only two CLBs are used for the implementation of the given Boolean function (the left side of FPGA). There are three input pins (denoted A, B and C) and one output pin (denoted F) on which the function is produced. The LUT programming in each CLB is also shown, as well as the routing configured in switch boxes. Connections established in each switch box are visualized in Figure 3 by joining the connected signals with a line within the switch box. For example, the upper-left switch box only establishes a connection between the second horizontal wire and the second vertical wire.

IV. From Boolean functions to programmed FPGAs: the workflow

For students to acquire a deeper understanding of FPGA technology (and to grasp why the commercial tools are so complex), it is important to demonstrate the complete work-

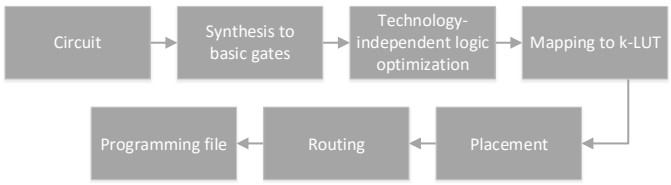

Fig. 4. The FPGA programming workflow

flow: how does one come from an abstract Boolean function to a programmed FPGA chip. A simple diagram shown in Figure 4 illustrates this workflow.

One starts with a circuit description which is usually given in some high level hardware description language such VHDL [1] or Verilog [2]. This description is in a process called *synthesis* converted into networks of basic logic gates. Using rules of Boolean algebra, this representation can then be converted into more adequate one (e.g. using minimization). Then, the resulting Boolean functions which can be functions of arbitrarily many variables should be decomposed into a network of k-LUTs, where each LUT implements a Boolean function of at most k-variables (k is typically a fixed parameter of the available architecture). Then, for each LUT generated in the decomposition procedure a physical LUT (i.e. CLB) must be assigned in the FPGA chip and configured appropriately; this process is called *placement*. After placement is done, I/O pins must be allocated as well and then the switch boxes must be configured so that each LUT gets the required input values. This process is called *routing*.

Each of the aforementioned steps is non-trivial. For example, how can we optimally create a function decomposition? If we have more than one function to implement, how can we decide if there are common subfunctions which can be implemented once and then shared among function implementations? As mentioned previously, today these problems are solved using heuristic algorithms.

When considering routing, one could start with the idea to find shortest path between connected CLBs. However, the routing problem we face here is the problem of simultaneously finding routes for many pairs of CLBs which, considering limited available wires in wire segments, can be hard or even impossible for some bad LUT placements.

Additionally, the routing procedure can have an additional constraint if the FPGA chip is in advance placed into a bigger circuit where the external wiring is already decided so the procedure does not have the freedom to select I/O pins for each variable and function, but has to obey predefined placement (e.g. function f must be routed to pin 5).

We will illustrate the difficulties one meets with the decomposition procedure on a simple example of finding the optimal decomposition for the Boolean function

$$f(A, B, C) = A \cdot \bar{B} \cdot C + B \cdot \bar{C}$$

using 2-LUTs.

One approach could be to utilize the Shannon decomposition theorem [20] (sometimes also referred to as an expansion)

which states that for each Boolean function F the following holds: $F = \bar{x} \cdot F_{\bar{x}} + x \cdot F_x$ where x is any variable, $F_{\bar{x}}$ is the function obtained from F by setting $x = 0$ and F_x is the function obtained from F by setting $x = 1$. We proceed recursively to decompose $F_{\bar{x}}$ and F_x until we end up with subfunctions of only two variables. This decomposition is shown in Figure 5.

$$f(A, B, C) = \underbrace{\overline{A}\ \underbrace{\underbrace{(\overline{B}\,C)}_{f_1(B,C)}}_{f_3(A,f_1)} + A\ \underbrace{\underbrace{(B\,\overline{C} + \overline{B}\,C)}_{f_2(B,C)}}_{f_4(A,f_2)}}_{f_5(f_3,f_4)}$$

Fig. 5. Decomposition based on the Shannon decomposition theorem

Using this approach we have used five 2-LUTs. The 2-LUTs are used to implement following subfunctions.

$$f_1(B,C) = \bar{B} \cdot C$$

$$f_2(B,C) = B \cdot \bar{C} + \bar{B} \cdot C$$

$$f_3(A,f_1) = \bar{A} \cdot f_1$$

$$f_4(A,f_2) = A \cdot f_2$$

$$f = f_5(f_3,f_4) = f_3 + f_4$$

We see that this decomposition could not be implemented into the FPGA from Figure 3 which only has four 2-LUTs. We can try a different approach hoping to end up with a smaller number of 2-LUTs. Since the 2-LUTs cannot implement a Boolean function of three variables, a naive approach would be to group variables and subfunctions into subfunctions of only two variables. This approach is shown in Figure 6.

$$f(A, B, C) = \underbrace{A\ \underbrace{\underbrace{B\,\overline{C}}_{f_1(B,C)}}_{f_2(A,f_1)} + \underbrace{\overline{B}\,C}_{f_3(B,C)}}_{f_4(f_2,f_3)}$$

Fig. 6. Naive decomposition

Using this approach we have used only four 2-LUTs. However, the described approach lacks a clear strategy on the grouping order.

Finally, it is possible to rewrite the given function in such a way that only three 2-LUTs are required for its implementation. This decomposition is shown in Figure 7.

For commercial tools non-introductory course would now continue to describe many heuristic approaches for solving this problem. However, on an introductory course, examples such as the described one are quite enough to illustrate the problem, to create interest in students into the possible ways for solving it and to demonstrate that such problems can be solved more-or-less successfully, and various solvers are an integral part of commercial packages for programming FPGAs.

$$f(A, B, C) = A\,B\,\overline{C} + \overline{B}\,C = \underbrace{\underbrace{(A + C)}_{f_1(A,C)} \cdot \underbrace{(B \oplus C)}_{f_2(B,C)}}_{f_3(f_1,f_2)}$$

Fig. 7. Better decomposition

The final step of the described workflow is preparing a configuration file for FPGA. This configuration file (called *bitstream* and often saved in JEDEC file format) is a stream of bits where each bit represents some configuration detail in the FPGA chip (for example, content of SRAM cells in LUTs, configuration of the output multiplexer in a CLB, configuration for each pass transistor, configuration for each I/O block, configuration for each input multiplexer which passes a value from single segment wire to address input of the LUT's multiplexer, etc). For commercially available FPGA chips the meaning of each bit in this bitstream is not publicly known; that way FPGA chip producers tie the users to buy and work with their own set of tools.

V. The Developed Tool

In order to demonstrate the most of the described workflow steps, we have developed a Java based tool for programming, visualization and simulation of a family of simple FPGA chips.

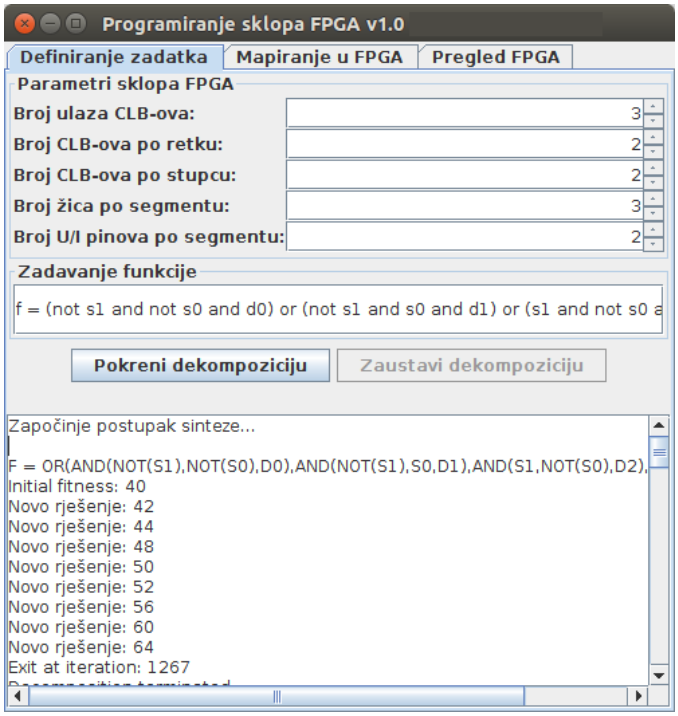

Fig. 8. The developed tool: first step

When the program is started, the user begins by specifing the FPGA architecture (see the top part of Figure 8). The user can define the following parameters.

- Number of CLB inputs (the k in the previous section). The user can decide to use 2-LUTs, 3-LUTs, etc.
- Number of CLBs in each row.
- Number of CLBs in each column.
- Number of wires per each wire segment.
- Number of I/O pins per wire segment.

Using these parameters a wide variety of FPGAs can be simulated. The architecture defined by parameters shown in Figure 8 used four 3-LUTs in 2×2 matrix having 2 I/O pins per wire segment and 3 wires per wire segment.

In the middle of Figure 8 a text box is shown, in which the user can specify a Boolean function to be implemented. At this moment, we only allow one function. The function can be entered using a simple infix notation with standard Boolean operators AND, OR, XOR, NAND, NOR, XNOR and NOT. The function shown in Figure 8 is a 4/1 multiplexer having data inputs d0, d1, d2, d3 and address inputs s1 and s0:

$$f = \bar{s}_1 \cdot \bar{s}_0 \cdot d_0 + \bar{s}_1 \cdot s_0 \cdot d_1 + s_1 \cdot \bar{s}_0 \cdot d_2 + s_1 \cdot s_0 \cdot d_3.$$

The decomposition process is started when the left central button is pressed (Croatian: *Pokreni dekompoziciju*). The progress and output is written in the text area on the bottom. In this example, the decomposition process finished with following:

```
Function is at output: 2
CLB(0): D2 S0 D3
00011101
CLB(1): D0 D1 S0
11100100
CLB(2): CLB(1) S1 CLB(0)
11010001
CLB(3): D0 D2 CLB(2)
00011000
```

For each CLB the signals for LUT inputs are written (starting with the most significant) and then the LUT configuration is written. Let us examine a bit closely the function of CLB(0). The function it realizes is

$$\begin{aligned} f(D_2, S_0, D_3) &= \sum \text{minterm}(3, 4, 5, 7) \\ &= \bar{D}_2 \cdot S_0 \cdot D_3 + D_2 \cdot \bar{S}_0 \cdot \bar{D}_3 + D_2 \cdot \bar{S}_0 \cdot D_3 + D_2 \cdot S_0 \cdot D_3 \\ &= \bar{S}_0(D_2 \cdot \bar{D}_3 + D_2 \cdot D_3) + S_0(\bar{D}_2 \cdot D_3 + D_2 \cdot D_3) \\ &= \bar{S}_0 \cdot D_2 + S_0 \cdot D_3 \end{aligned}$$

which is multiplexer 2/1 which multiplexes D_2 or D_3 based on S_0. The CLB(1) is also a multiplexer 2/1 which multiplexes D_0 or D_1 based on S_0. Finally, the CLB(2) on which the function is realized is also multiplexer 2/1 which multiplexes output of CLB(0) and CLB(1) based on S_1. It is trivial to see that the decomposition procedure has created a multiplexer tree realizing multiplexer 4/1 using a tree of multiplexers 2/1.

Once the decomposition is done, the user can proceed to the second tab which is shown in Figure 9. On this tab one can select the mapping algorithm (currently, we offer only one) and then can start the mapping procedure by pressing the central left button (Croatian: *Pokreni mapiranje*). This will start both the placement procedure and the routing procedure. The results and the final report will be written in the text area at the bottom of the window.

Once this is done, the user can proceed to the last tab which

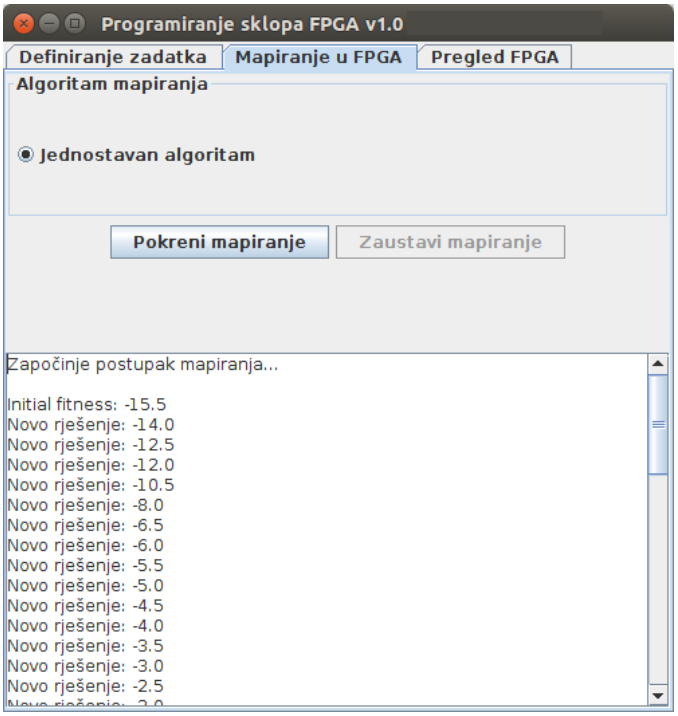

Fig. 9. The developed tool: second step

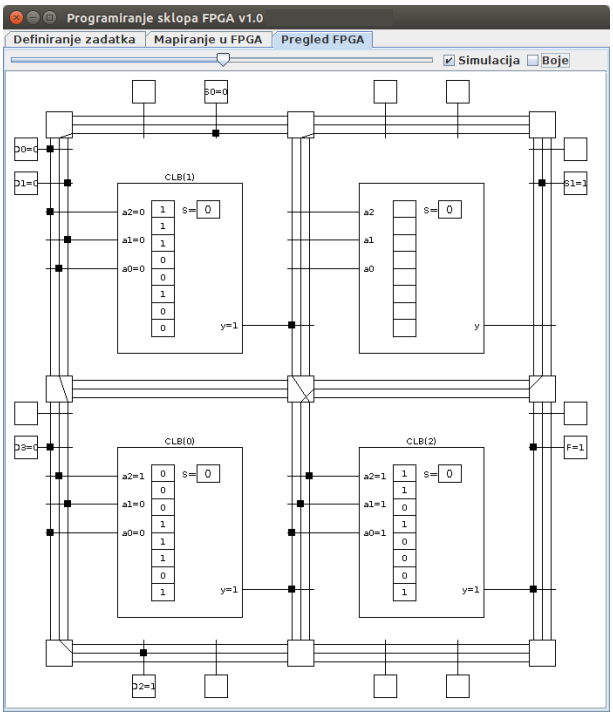

Fig. 10. The developed tool: third step

is shown in Figure 10.

Here we can see the final configured FPGA chip. We can inspect all wiring and switch-boxes and CLB configuration, and by clicking on any CLB while holding the CRTL key, we can get a detailed schematic of the CLB. In the shown case, the placement and routing algorithm placed inputs D0, D1 and D3 along the left side of chip, D1 along the bottom side, S0 along the top side and S1 and output F along the right side. If user enables Simulation mode (checkbox *Simulacija*), values on input pins can be selected by clicking while the outputs will be automatically calculated for each CLB. In the shown example we have $d_3d_2d_1d_0 = 0100$, $s_1s_0 = 10$ so the multiplexer output should be equal to d_2 and so $F = 1$.

By using the synthesised FPGA in interactive mode, students can observe each aspect of its functioning in real time. Using this tool allows us to demystify the fundamentals of FPGA technology and explain what commercial tools do.

It is important to note that the algorithms implemented in this tool for various stages are not commercial-grade algorithms: our intention was to build an educational tool. The algorithms we used are based on evolutionary computations. Specifically, we have implemented a variant of a genetic algorithm for Boolean function decomposition which is similar to Cartesian Genetic Programming [21]. Another variant of a genetic algorithm was used to solve the placement and the routing problem as well as configuration simplification.

VI. Conclusion and future work

In this paper we have described a platform independent Java based tool for programming, visualization and simulation of simplified FPGAs. We have developed this tool in an effort to enable students to better understand what an FPGA is, how it works and how it is programmed.

In academic year 2015/2016 we have introduced the tool in class and were able to obtain a much better interaction and discussion with students while presenting the FPGA topic. Students tended to ask much more focused and related questions which was not the case in previous years. This is a strong indication that the students acquired a better understanding of the topic, which was our research goal.

Currently, the developed tool allows the user to customize the FPGA architecture but to enter only a single Boolean function to be synthesized. For educational purposes this is quite enough since it allows us to illustrate most of the steps which commercial tools do. However, we are planning an extension of this work on several fronts. We plan to allow the synthesis of multiple Boolean functions as well as the synthesis of sequential circuits. Also, to make the user experience closer to the one offered by the commercial tools, we plan to implement support for circuit synthesis described by the user using a simple subset of the VHDL language (which we also use on laboratory exercises). Finally, another extension of current work will be to allow the user to manually program the FPGA circuit directly by mouse (clicking on a connection, dragging to configure switchboxes, clicking to set SRAM content in LUTs etc.).

References

[1] P. J. Ashenden, *The Designer's Guide to VHDL*, 3rd ed. Burlington, MA, USA: Morgan Kaufmann, 2008.

[2] D. E. Thomas and P. R. Moorby, *The Verilog Hardware Description Language*, 3rd ed. Burlington, MA, USA: Springer, 1996.

[3] A. Sangiovanni-Vincentelli, A. El Gamal, and J. Rose, "Synthesis method for field programmable gate arrays," *Proceedings of the IEEE*, vol. 81, no. 7, pp. 1057–1083, Jul 1993.

[4] K. Compton and S. Hauck, "Reconfigurable computing: A survey of systems and software," *ACM Comput. Surv.*, vol. 34, no. 2, pp. 171–210, Jun. 2002.

[5] T. Łuba and H. Selvaraj, "A general approach to boolean function decomposition and its application in FPGA based synthesis," *VLSI Design*, vol. 3, no. 3-4, pp. 289–300, 1995.

[6] M. Nowicka, T. Łuba, and M. Rawski, "FPGA-based decomposition of boolean functions. algorithms and implementation," *Proc. of Sixth International Conference on Advanced Computer Systems*, pp. 502–509, 1999.

[7] R. Enzler, T. Jeger, D. Cottet, and G. Tröster, *Field-Programmable Logic and Applications: The Roadmap to Reconfigurable Computing: 10th International Conference*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, ch. High-Level Area and Performance Estimation of Hardware Building Blocks on FPGAs, pp. 525–534.

[8] M. del Solar, J. Perez, J. Pulido, and M. Rodriguez, "Placement and routing of boolean functions in constrained FPGAs using a distributed genetic algorithm and local search," in *Parallel and Distributed Processing Symposium*, April 2006, p. 7.

[9] M. Yang, A. Almaini, L. Wang, and P. Wang, "FPGA placement using genetic algorithm with simulated annealing," in *ASICON 2005. 6th International Conference On*, vol. 2, Oct 2005, pp. 806–810.

[10] S.-J. Lee and K. Raahemifar, "FPGA placement optimization methodology survey," in *Electrical and Computer Engineering, 2008. CCECE 2008. Canadian Conference on*, May 2008, pp. 001 981–001 986.

[11] A. Gayasen, Y. Tsai, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, and T. Tuan, "Reducing leakage energy in FPGAs using region-constrained placement," in *Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays*, ser. FPGA '04. New York, NY, USA: ACM, 2004, pp. 51–58.

[12] W. El Medany, "FPGA remote laboratory for hardware e-learning courses," in *Computational Technologies in Electrical and Electronics Engineering, 2008. SIBIRCON 2008. IEEE Region 8 International Conference on*, July 2008, pp. 106–109.

[13] J. Garcia-Zubia, I. Angulo, L. Rodriguez-Gil, P. Orduna, O. Dziabenko, and M. Guenaga, "Boole-WebLab-FPGA: Creating an integrated digital electronics learning workflow through a hybrid laboratory and an educational electronics design tool," *International Journal of Online Engineering (iJOE)*, vol. 9, 2013.

[14] L. Rodriguez-Gil, P. Orduna, J. Garcia-Zubia, I. Angulo, and D. Lopez-de Ipina, "Graphic technologies for virtual, remote and hybrid laboratories: WebLab-FPGA hybrid lab," in *Remote Engineering and Virtual Instrumentation (REV), 2014 11th International Conference on*, Feb 2014, pp. 163–166.

[15] L. H. Crockett, R. A. Elliot, M. A. Enderwitz, and R. W. Stewart, *The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable SoC*. UK: Strathclyde Academic Media, 2014.

[16] N. H. E. Weste and D. Money Harris, *Principles of CMOS VLSI Design: A Systems Perspective*, 4th ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2011.

[17] G. G. Lemieux and S. D. Brown, "A detailed routing algorithm for allocating wire segments in field-programmable gate arrays," in *In Proc. ACM/SIGDA Physical Design Workshop*, 1993, pp. 215–226.

[18] Y.-W. Chang, D. F. Wong, and C. K. Wong, "Universal switch modules for FPGA design," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 1, no. 1, pp. 80–101, Jan. 1996. [Online]. Available: http://doi.acm.org/10.1145/225871.225886

[19] S. J. E. Wilton, "Architectures and algorithms for field-programmable gate arrays with embedded memory," Ph.D. dissertation, University of Toronto, 1997.

[20] C. E. Shannon, "The synthesis of two terminal switching circuits," *Bell System Technical Journal*, vol. 28, pp. 59–98, 1949.

[21] J. F. Miller and P. Thomson, "Cartesian genetic programming," in *Genetic Programming*. Springer, 2000, pp. 121–132.