A Learning Tool for Synthesis, Visualization, and Editing of Programming for Simple Programmable Logic Devices

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*—Our experiences in teaching digital circuit design at university level indicate that students find it difficult to understand programmable logic devices (PLDs) such as PALs, PLAs, GALs and FPGAs. This is mainly due to the complexity of the topic and the lack of tools that visualize the inner workings of PLDs and enable students to modify and inspect individual components. The majority of publicly available SPLD-related tools are proprietary, platform-specific and do not expose all elements of the PLD structure to the user. In this work, we propose a learning tool that enables synthesis, visualization and editing the programming of a GAL16v8 SPLD. GAL16v8 has been chosen as it is simple enough that its physical implementation can be observed to the smallest detail, while enabling simultaneous realization of multiple Boolean functions. The tool is platform-independent and specifically tailored towards use in an educational setting, facilitating much better understanding of SPLDs through a hands-on experience.**

I. Introduction

The historical development of programmable logic devices (PLDs) can be traced from the most simple ones, such as one-time programmable read-only memories, through a bit more complicated simple programmable logic devices (SPLDs) such as PALs, PLAs and GALs, to complex programmable logic devices (CPLDs) and field programmable gate arrays (FPGAs) [1], [2]. The basic principles involving internal workings and architectures of such devices are often taught in university-level digital circuit design courses, including the Digital Logic course at our Faculty. It is our experience that students often find the topic hard to understand [3], considering the fact that the presentation of the topic typically covers both low-level details including electronical components such as FGMOS transistors and high level concepts including logical design and PLD programming.

Our previous work [4], [5], [3] indicated that the students at out Faculty readily accept e-learning tools and that such tools have a positive impact on their academic performance. In order to foster student experience with PLD technologies and improve their understanding, we have developed a multipurpose learning tool for GAL16v8. The tool can be used for direct programming, where the student can input each programmable switch programming. It can also be used for visualization of programming and generating the JEDEC file needed to program the physical GAL chip in hardware programmer. Additionally, the tool allows the students to provide a higher level description of the desired design using logical expressions, which the synthesis module then converts into GAL programming. The generated GAL programming can be visualized in a graphical environment and modified if needed.

One of the main advantages of the tool is that it is written in Java so it can be run on many different operating systems; a fact that is not true for many commercial tools. The GAL16v8 has been chosen because it is relatively simple, so the students can understand the function of each of 2194 programmable switches and can set their programming directly. In order to better integrate the rest of the Digital Logic course topics taught at our Faculty which are based on VHDL usage, we have equipped the tool with a module that allows students to provide a logical description of the desired circuits. Using the description, the module determines the GAL programming that the student can then inspect.

II. Related work

Learning basic concepts of digital electronics represents one of the cornerstones of modern computer and electrical engineering education. These concepts are essential for understanding hardware design [6], [7], computer architectures [8], [9], systems-on-a-chip [10], etc. Without deep understanding of these concepts, the students will not be able to understand how the computer actually works, how a microprocessor is designed and implemented, how computer programs are executed and what to consider when thinking about efficient computer code. Given the importance and the ubiquity of digital electronics, there is an interest in developing better learning methods for various digital electronics concepts in the academic education community. These methods involve developing specialized learning tools that include purely software-based solutions [11], [12], [13], as well as remote hybrid software-hardware learning systems [14], [15], [16]. Improvements of the teaching process itself are also considered, e.g. through reframing the teaching to be more centered around using a hardware description language [17], or devising instructive problems that can be solved by small teams of students in a competition-based semi-professional environment [18].

One example of a specialized software learning tool for digital circuit design is Boole-Deusto, proposed by Garcia-Zubia et al. [11]. Boole-Deusto is intended for teaching the design and analysis of bit-level combinatorial and sequential circuits. Combinatorial circuits can be defined using truth tables, minterms or maxterms or Boolean logic expressions. Boole-Deusto supports Veitch-Karnaugh diagrams, finite state machines, and code generation from the designed circuits (OrCAD-PLD, VHDL, and JEDEC are supported). Similarly, Hacker and Sitte [12] propose WinLogiLab, an interactive teaching suite for combinatorial and sequential circuits. The suite is comprised of a set of increasingly complex tutorials, covering a range of topics including Boolean algebra, truth tables, Karnaugh maps, finite state machines, etc. Donzellini and Ponta [13] propose an e-learning simulation environment for digital electronics called Deeds. The environment is tailored towards teaching embedded systems, and it supports combinational and sequential circuits, finite state machines, and microcomputer interfacing and programming. Baneres et al. [19] introduce an online platform for the design and verification of digital circuits through a series of exercises, consisting of a desktop application that communicates with the course server and a web-based management system for instructors.

In hybrid software-hardware learning systems, the goal is to enable the student to interact with a real physical SPLD. For instance, El Medany [14] proposes a remote laboratory that enables the students to interactively control a physical FPGA board over the internet. Garcia-Zubia et al. [15] and Rodriguez-Gil et al. [16] propose a system that combines their previously described Boole-Deusto learning tool with a physical remote FPGA board that controls a virtual simulated water tank. Boole-Deusto is used to generate VHDL code that is synthesized and programmed into the FPGA, and the student can see the remote FPGA controlling the simulated water tank according to the developed digital circuit.

One shortcoming of Boole-Deusto, WinLogiLab and Deeds is that they run exclusively on Microsoft Windows, making them unavailable to students that use other operating systems. Furthermore, Boole-Deusto and WinLogiLab seem to be restricted to basic digital electronics concepts, offering no support for teaching SPLDs such as PALs, PLAs, GALs. Motivated by the importance of teaching basic SPLD concepts and our experience that students find it difficult to understand the inner workings of an FPGA following a classical black box vendor-specific synthesis approach [20], we have previously proposed a platform independent tool for programming, visualization and simulation of simplified FPGAs [3]. In this work, we move a step further by developing a tool that exposes the inner workings of GAL16V8 in a fully interactive manner.

III. The Learning Tool Requirements

In the Digital Logic course at our Faculty, we cover programmable logic circuits in depth: from implementation details up to the logical model and high-level programming. The course covers implementation of permanent read-only

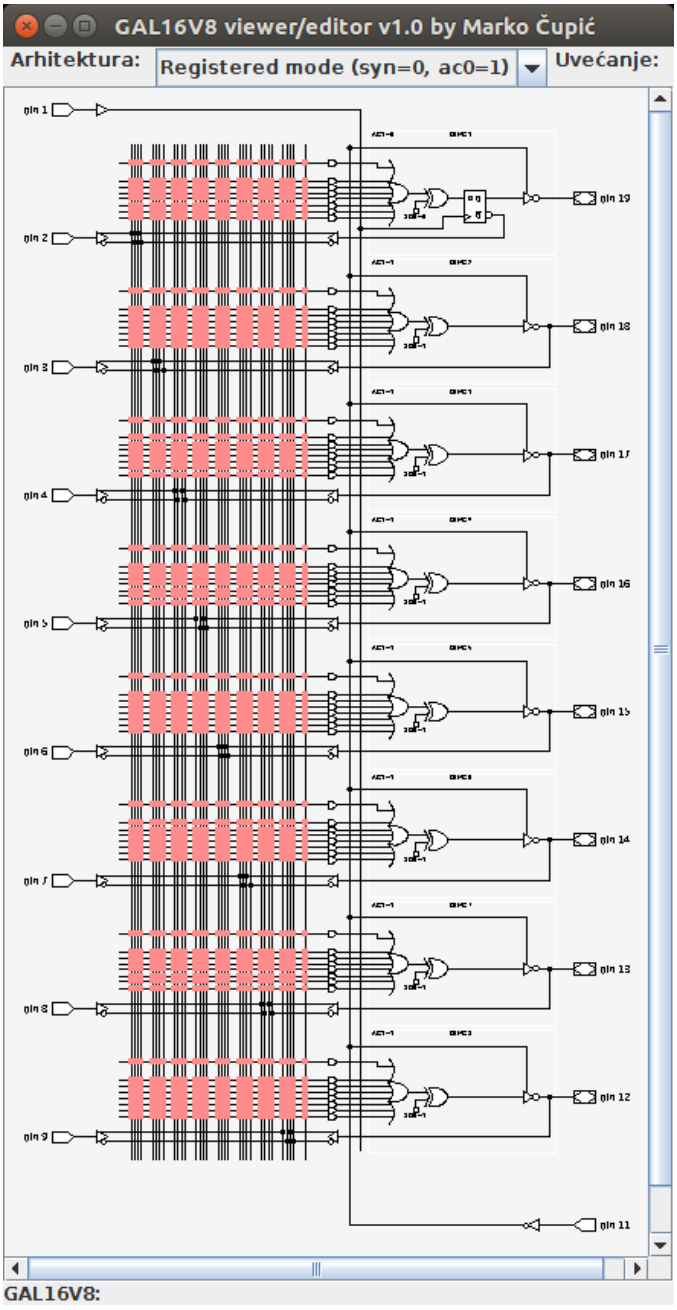

Fig. 1. Logical view of GAL16v8 in registered mode.

memories (using semiconductor diodes and then metal-oxide-semiconductor field-effect transistors, MOSFETs), principles of implementation of erasable ROMs (EPROMs) and electrically erasable ROMs (EEPROMs) which are based on floating-gate MOSFET (FGMOS), various SPLDs (such as programmable logic array - PLA, and programmable array logic - PAL) and finally complex programmable logic devices (CPLDs) and field programmable gate arrays (FPGAs).

In order to clarify the design, programming and application of SPLDs on an instructive example, we decided that our learning tool should model some existing chip which fulfils

the following requirements:

1) it can still be purchased,
2) it is not too expensive,
3) it offers adequate logic programmability so that several Boolean functions can be realized simultaneously,
4) it is simple enough that its physical implementation can be observed to the smallest detail (i.e. which of the FGMOS transistors should have excess charge on its floating-gate and which not in order to realize the desired Boolean function - see Figure 2),
5) support for generating both combinatorial and sequential circuits is desired.

The fourth requirement stems from the fact that within the Digital Logic course (as taught at our Faculty) we teach students the basics of digital circuits implementation (i.e. how can various logic gates, such as NOT, AND, OR, NAND, and NOR, be implemented using bipolar transistors, using MOSFETs and using CMOS) as well as how programmable digital circuits are implemented (i.e. programming by burning fuses, programming by blocking channel creation in MOSFETs, etc). Therefore, we required a chip which is simple enough that such level of detail can be presented.

We also wanted the selected chip to be suitable for teaching how high level SPLD programming can be done. There are two possible ways of doing SPLD programming:

1) the state for each of the programmable elements (FGMOSFET) can be set manually, or
2) the desired functionality can be described by some language for formal specification and then the state of each of programmable element can be set by some automatic procedure.

We have chosen the GAL16v8 chip, as it fulfils all of the aforementioned requirements. It is rather cheap, it can realize eight Boolean functions and it works with a wide range of supply voltages. Additionally, it has a relatively regular structure so that each of 2194 programmable switches can be observed and its function readily understood. This chip can be configured to operate in three different architectural modes defined using two bits of information: simple mode, complex mode and registered mode. This is in itself very instructive, as it offers a great example of multiplexer usage, where different architectures are realized by multiplexers choosing which signal is routed where.

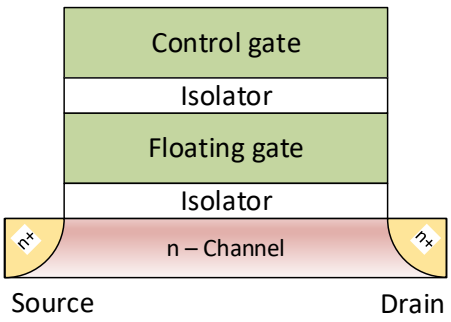

Fig. 2. Structure of FGMOSFET.

For actual chip programming, an additional device is needed (so-called programmer) which is connected to the user's computer and in which the programmable chip is placed. On the computer, the appropriate software must be started and data based on which the programming will be performed must be provided. Today, such data is typically provided as JEDEC-formatted file, so we wanted to be able, while teaching the topic, also to explain how this file is formatted and how it is generated for selected chip.

Having selected the chip, our goal was to develop a learning tool that would satisfy the following requirements:

1) it should be portable (so that the students can use it on various operating systems),
2) it should offer graphical user interface which should show all programmable elements and allow the student to edit each FGMOS transistor state,
3) it should be able to automatically generate a JEDEC file for shown programming,
4) it should be able to read programming from a JEDEC file and adjust current programming,
5) it should support some way of formally describing Boolean functions and automatic programming based on a given formal description,
6) it should offer a command line tool for converting a direct formal circuit description into JEDEC,
7) and lastly, it must be simple enough to be adequate for education purposes (which most commercial professional tools are not).

IV. The developed tool

The learning tool that accommodates the requirements elaborated in Section III has been developed using the Java programming language. This way, we have ensured that the tool is portable across all often used desktop operating systems. When started in interactive mode, the student is presented with a graphical editor showing the logical structure of GAL16v8 (see Figure 1 for a coarse overview).

On the top of the editor window there is a combo box which allows the user to set values for two architectural bits, thus choosing the active GAL configuration. The values of these two architectural bits are connected to several multiplexers in the physical chip implementation. The developed editor, for the sake of clarity, does not show all of these details. Instead, once the user selects values of architectural bits, the editor shows the effective chip architecture.

In simple mode (see more detailed Figure 3), each macrocell can realize a Boolean function in the form of a sum of 8 products. The output of each macrocell can be configured to be permanently enabled or permanently in high-impedance state. Signals from pins 1-9, 11-14 and 17-19 are connected to the input array in the which user can program products. Each programmable element (which determines if the given signal is used in given product) is technologically realized using FGMOSFET.

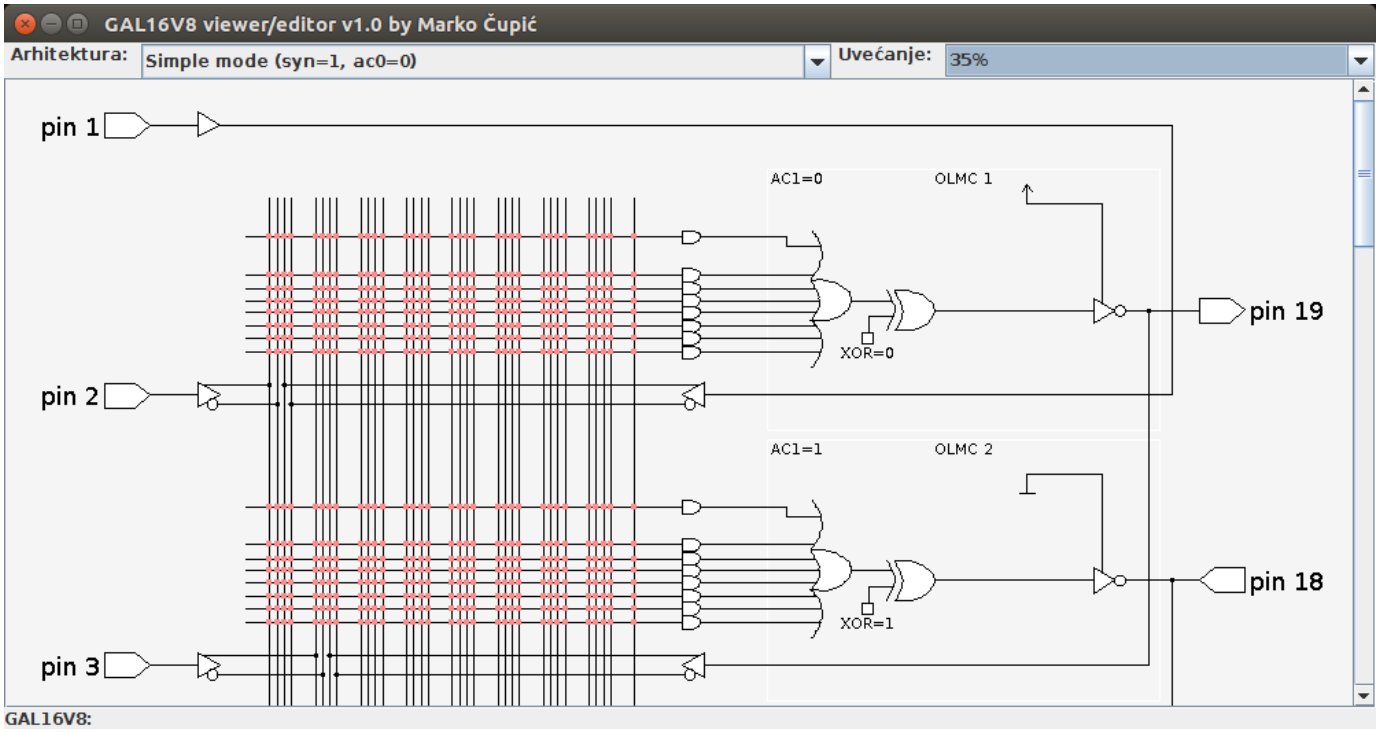

Fig. 3. A detail of the logical view of GAL16v8 in simple mode.

Programming of all the programmable elements can be performed by mouse. The state of FGMOSFET can be toggled by double-clicking on the signal-product connection, which will be visually indicated. Macrocell configuration bits can be defined by double clicking on the macrocell. In simple mode, the user can configure the logical value for lower input of XOR-gate and additionally define whether the output buffer is enabled or disabled. If a low number of Boolean functions is to be realized but of many variables, disabling some macrocells can open output pins to be used as additional inputs. A relevant part of GAL configured to calculate $f(A, B) = A \oplus B$ is shown in Figure 4. Here, A and B were connected to pins 2 and 3 while macrocell OLMC 1 (the top-most one) was used for calculation. Its output buffer was enabled and the bit for output XOR-gate was set to 1 since the output buffer actually also inverts the calculated value.

In complex mode (see smaller Figure 5) each macrocell can realize a Boolean function in the form of a sum of 7 products, while one product controls the output buffer (enabled or in high-impedance).

Finally, in registered mode (see smaller Figure 6), each macrocell can realize a Boolean function in the form of a sum of 8 products. The realized function, depending on the macrocell configuration bit, can be routed to macrocell output or it can be used as input for D-flipflop whose state is then used as macrocell output. In this mode, the output buffer is enabled/disabled for all macrocells by a common signal. In Figure 6, the top-most macrocell is configured to use a D-flipflop and the second macrocell is configured to calculate

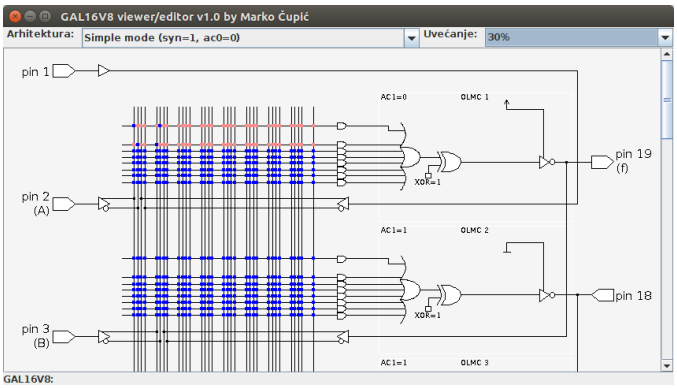

Fig. 4. A GAL configured to generate $f = A \oplus B$.

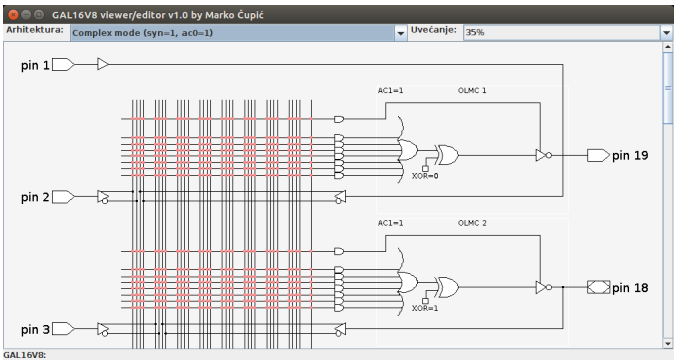

Fig. 5. A detail of the logical view of GAL16v8 in complex mode.

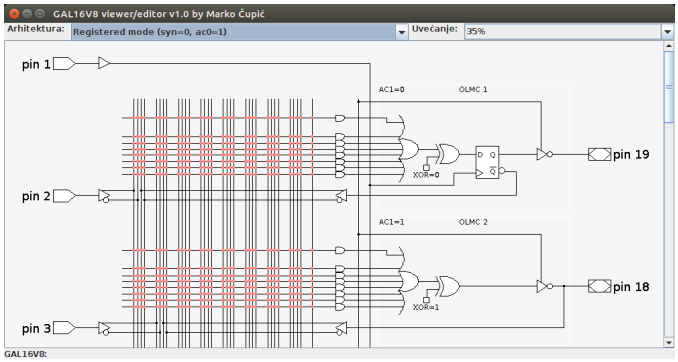

Fig. 6. A detail of the logical view of GAL16v8 in registered mode.

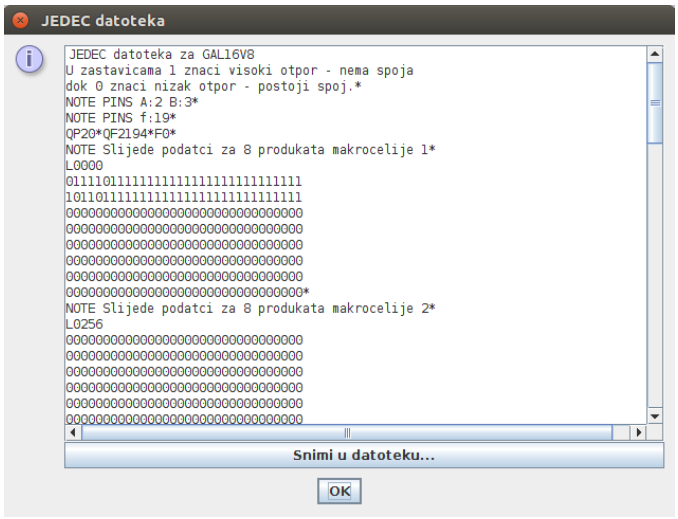

Fig. 7. Generation of JEDEC file from GUI.

combinatorial function.

The developed tool allows students to easily generate the JEDEC file from a popup menu. JEDEC file generation for XOR example from Figure 4 is shown in Figure 7.

The tool also allows user to load configuration from JEDEC file which was generated for this GAL chip.

To support automatic programming based on a formal circuit specification, we decided to reuse a syntax used in older commercial tools, but in a simplified form. An example of such a description is shown below.

```
DEVICE GAL16V8
MODE SIMPLE
PIN 2 A
PIN ? B
PIN ? C
PIN ? f OUTPUT
PIN ? g OUTPUT

EQUATIONS

f = A*/B + C
g = A:+:B + C
```

This formal description defines two Boolean functions to be realized: $f = A \cdot \bar{B} + C$ and $g = A \oplus B + C$. The specification also defines some constraints for the synthesiser: the input A must be attached on pin 2 while other variable-pin assignment can be arbitrary.

Descriptions such as this one can be synthesized through the GUI or the synthesis can be requested directly from the command line. In the latter case, the synthesizer will generate the a appropriate JEDEC file. If the previous description is saved in file `f2.eqn`, the synthesis from the command line can be started by command:

```
java -jar SimpleGAL-1.0.1.jar
      pahdl-to-jedec -in f2.eqn
      -out f2.jedec
```

which will produce the JEDEC file `f2.jedec` as well as additional info (including actual pin assignment). Example of such a file report is given below.

```
DEVICE: GAL16V8
MODE: SIMPLE
F SOP: A*/B+C
G SOP: /A*B+A*/B+C
INFO: Inputs: [A, B, C].
INFO: Outputs: [F, G].
INFO: Pin mapping process succeeded.
INFO: Pin 01: name=NC
INFO: Pin 02: name=A
INFO: Pin 03: name=B
INFO: Pin 04: name=C
INFO: Pin 05: name=NC
INFO: Pin 06: name=NC
INFO: Pin 07: name=NC
INFO: Pin 08: name=NC
INFO: Pin 09: name=NC
INFO: Pin 10: name=GND
INFO: Pin 11: name=NC
INFO: Pin 12: name=NC
INFO: Pin 13: name=NC
INFO: Pin 14: name=NC
INFO: Pin 15: name=NC
INFO: Pin 16: name=NC
INFO: Pin 17: name=NC
INFO: Pin 18: name=G
INFO: Pin 19: name=F
INFO: Pin 20: name=VCC
INFO: Here is the pin assignment:
               GAL16V8
           +----    ----+
           |    \__/    |
        NC |  1      20 | Vcc
         A |  2      19 | F
         B |  3      18 | G
         C |  4      17 | NC
        NC |  5      16 | NC
        NC |  6      15 | NC
        NC |  7      14 | NC
        NC |  8      13 | NC
        NC |  9      12 | NC
       GND | 10      11 | NC
           +------------+

INFO: Synthesis to GAL16V8 completed.
```

V. Example use cases

Example use cases for our tool within the Digital Logic course we teach include illustrations of the topic "Programmable logical devices", where we offer interested students to complete two experiments.

Experiment 1. Elements: 2 DIP switches, 5 resistors, GAL16V8, 3 LEDs. Use 2 DIP switches and two resistors

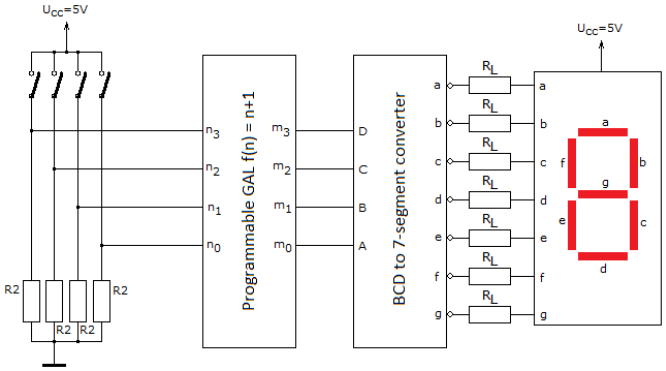

Fig. 8. A digital circuit with GAL16V8.

to create logical values for A and B. Connect values A and B to GAL16V8 inputs, and 3 LEDs to its outputs. Using our tool, program the GAL16V8 by manually clicking the programmable switches, to ensure that the first LED lights only if both A and B are high, that the second LED lights if any of A or B is high, and that the third LED lights only when one of A or B is high. Create the JEDEC file. Program the GAL and test complete circuit on the protoboard.

Experiment 2. Elements: 4 DIP switches, 11 resistors, GAL16V8, BCD to 7-segment converter, 7-segment display. Using 4 DIP switches and 4 resistors create logical values $a_3a_2a_1a_0$. Let us treat $a_3a_2a_1a_0$ as binary-encoded number. Derive by hand logical expressions for function $f(a_3a_2a_1a_0) = a_3a_2a_1a_0 + 1$ (i.e. the next number). Write a formal circuit specification and using our synthesizer create the JEDEC file. Program the GAL and test complete circuit on the protoboard (see Figure 8).

VI. Conclusion

We have presented an interactive learning tool for synthesis, visualization and programming GAL16v8 SPLD. The developed tool can be used in education for courses that cover programmable logic devices. The tool is very useful in bringing insights into the complete design process: starting with a formal description of a circuit, the tool enables learning which steps are needed to complete the programming. This is quite handy because the selected GAL16v8 is simple enough so that the programming can be traced back to each single FGMOSFET transistor.

Using this tool, students can also observe what is the end process of the synthesis - which decisions did the synthesizer make and how the programming was completed. In our experience, if the topic is clearly covered and illustrated in this way, students can more easily proceed to FPGAs, which are much more complex to grasp.

As a future work, we plan to add support for another formal language for hardware specification: a simplified version of VHDL, since the VHDL is used for FPGAs as well. We believe that consistently using a single language from the start can make the transition from simple programmable logic devices to more advanced ones easier and more natural for students.

References

[1] B. J. LaMeres, *Programmable Logic*. Cham: Springer International Publishing, 2017, pp. 371–384.

[2] V. Taraate, *Introduction to PLD*. Singapore: Springer Singapore, 2017, pp. 169–209.

[3] M. Čupić, K. Brkić, and Ž. Mihajlović, "A platform independent tool for programming, visualization and simulation of simplified FPGAs," in *2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, May 2016, pp. 986–991.

[4] M. Čupić and Ž. Mihajlović, "Computer-based knowledge, self-assessment and training," *International Journal of Engineering Education*, vol. 26, no. 1, pp. 111–125, 2010.

[5] Ž. Mihajlović and M. Čupić, "Software environment for learning and knowledge assessment based on graphical gadgets," *International Journal of Engineering Education*, vol. 28, no. 5, pp. 1127–1140, 2012.

[6] J. Staunstrup, *A Formal Approach to Hardware Design*, ser. Kluwer international series in engineering and computer science: VLSI, computer architecture, and digital signal processing. Springer US, 1994.

[7] I. Grout, *Digital Systems Design with {FPGAs} and {CPLDs}*. Newnes, Burlington, 2008.

[8] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Fifth Edition: A Quantitative Approach*, 5th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

[9] D. Patterson and J. Hennessy, *Computer Organization and Design MIPS Edition: The Hardware/Software Interface*, ser. The Morgan Kaufmann Series in Computer Architecture and Design. Elsevier Science, 2013.

[10] 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.

[11] B. S. B. Javier Garcia Zubia, Jesus Sanz Martinez, "A new approach to educational software for logic analysis and design," in *IADAT e2004, International Conference on Education*, 2004.

[12] C. Hacker and R. Sitte, "Interactive teaching of elementary digital logic design with winlogilab," *IEEE Transactions on Education*, vol. 47, no. 2, pp. 196–203, May 2004.

[13] G. Donzellini and D. Ponta, "A simulation environment for e-learning in digital design," *IEEE Transactions on Industrial Electronics*, vol. 54, no. 6, pp. 3078–3085, Dec 2007.

[14] 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.

[15] 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.

[16] 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.

[17] E. G. Breijo, L. G. Sanchez, and J. I. Civera, "Using hardware description languages in a basic subject of digital electronic: Adaptation to high academic performance group," in *2014 XI Tecnologias Aplicadas a la Ensenanza de la Electronica (Technologies Applied to Electronics Teaching) (TAEE)*, June 2014, pp. 1–6.

[18] R. Rengel, M. J. Martin, and B. G. Vasallo, "Supervised coursework as a way of improving motivation in the learning of digital electronics," *IEEE Transactions on Education*, vol. 55, no. 4, pp. 525–528, Nov 2012.

[19] D. Baneres, R. Clariso, J. Jorba, and M. Serra, "Experiences in digital circuit design courses: A self-study platform for learning support," *IEEE Transactions on Learning Technologies*, vol. 7, no. 4, pp. 360–374, Oct 2014.

[20] 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.