A Double-Phase Evolvable Hardware Architecture Learning Platform: Design, Simulation, and Prototyping Testbed

Bernardo Guerra Pereira Cunha
Flávia Magalhães Freitas Ferreira
Pontifical Catholic University of Minas Gerais
Post-Grad. Prog. of Electrical Engineering
Minas Gerais, Brazil
bernardogpcunha@gmail.com
flaviamagfreitas@gmail.com

Carlos Augusto Paiva da Silva Martins
Pontifical Catholic University of Minas Gerais
Post-Grad. Prog. of Informatics
capsm@pucminas.br

Abstract

The world is changing. Scientists and engineers create solutions for daily problems every day. Time is a crucial factor in many contexts, for example, academia. Undergraduate, master, or doctorate tasks in engineering and computer science tend to use enhanced methods that are computationally expensive for regular computers. It is the case of Hardware development. When a research issue deals with complex and multidisciplinary topics, like Evolvable Hardware, things become even worse. In this case, students and researchers spend a huge amount of time prototyping and making experiments in traditional EDA tools. This work shows the initial results of a double-phase evolvable hardware design learning platform, a combination of a computational interface for logic simulation and hardware prototyping.

1. Introduction

EHW (Evolvable Hardware) is a field of knowledge that emerged in the early 1990's due to the development of Reconfigurable Computing Devices, such as the Field Programmable Gate Arrays, or FPGAs. According to Haddow and Tyrrell [1], it is the embodiment of evolution in a physical media. EHW has been widely applied in contexts such as digital image processing [2] [3] and digital circuit design [4] [5] [6], but there are also applications in signal processing [7], computer networks [8], robotics [9], among other areas. Evolvable Hardware can also be comprehended as the intersection of two areas of computing: Reconfigurable Computing (RC) and Evolutionary Computing (EC) [10]. That is, it is the application of nature-inspired abstractions in the codification of algorithms (EA) implemented in reconfigurable hardware (RC) [11] [12]. The most recognized digital devices in the RC literature are the FPGAs [4].

There are critical issues related to hardware development in general: Firstly, digital designers must have expertise in the use of the aimed IDE (Integrated Development Environment), and also in some specific concepts related to electronic design and reconfigurable computing; Furthermore, hardware design is technology-dependent, that is, you may not change the FPGA distributor (Altera/Intel, Xilinx, etc.) because the IDEs are different and the projects work only in the aimed FPGA's distributor; Finally, digital design and simulation is something time-consuming. Simulating microseconds of hardware operation may take seconds or minutes in traditional hardware development platforms, as shown in Fig.1. For complex projects, such as Evolvable Hardware Architectures, the simulation time can make it unfeasible to explore the design possibilities. For EHW students and researchers, it would be necessary to investigate the architecture parameters and explore the evolutionary algorithm, even to absorb the concepts and have quick preliminary results, something not viable to make in traditional platforms, like Quartus II, the IDE presented in Fig.1.

This paper's objective is to expose a learning platform for evolvable hardware architecture learning and development. This artifact was implemented in a Master's Degree in Electrical Engineering, whose goal was to develop an EHW Architecture to solve problems related to combinational logic, like in [13]. The learning platform consists of two different approaches, addressing two different phases of development: one in the research and design, and another in the testing and execution of the application in the final device, analyzing the project in the real world. The first phase removes the necessity of acquiring equipment and coding in Hardware Description Languages, like VHDL (Very High-Speed Integrated Circuits Hardware De-

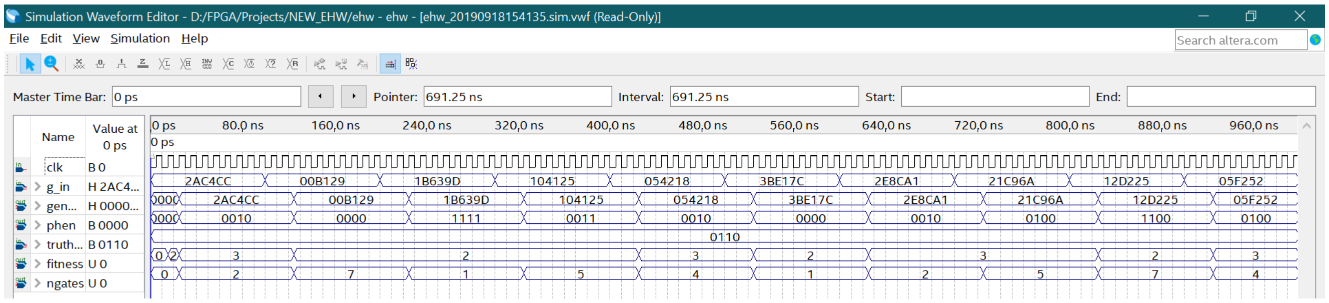

Simulation Waveform Editor - D:/FPGA/Projects/NEW_EHW/ehw - ehw - [ehw_20190918154135.sim.vwf (Read-Only)]
File Edit View Simulation Help
Search altera.com
Master Time Bar: 0 ps
Pointer: 691.25 ns
Interval: 691.25 ns
Start:
End:
Name
Value at 0 ps
0 ps 80,0 ns 160,0 ns 240,0 ns 320,0 ns 400,0 ns 480,0 ns 560,0 ns 640,0 ns 720,0 ns 800,0 ns 880,0 ns 960,0 ns
0 ps
clk B 0
g_in H 2AC4... 2AC4CC 00B129 1B639D 104125 054218 3BE17C 2E8CA1 21C96A 12D225 05F252
gen... H 0000... 0000 2AC4CC 00B129 1B639D 104125 054218 3BE17C 2E8CA1 21C96A 12D225 05F252
phen B 0000 0000 0010 0000 1111 0011 0010 0000 0010 0100 1100 0100
truth... B 0110 0110
fitness U 0 0 2 3 2 3 2 3 2 3
ngates U 0 0 2 7 1 5 4 1 2 5 7 4

Figure 1. Post-synthesis simulation considering the main variables in EHW. Every 100 microseconds represent a new genotype, considering the time limit of 1 miliseconds.

scription Language). It is also independent of the FPGA technology and vendor. The student/developer only focuses on the concepts and the abstraction of the architecture.

These two phases are financially feasible and technically approachable. They shouldn't add hardships to the researchers. On the contrary, they are supposed to offer more technical and scientific mechanisms. The first one includes modeling the system on a PSE (Problem Solving Environment) coded and developed in MATLAB, a software simulation program. One of the advantages of this approach is that MATLAB has graphical interfaces that enable to automate of the process of coding and testing, and it is more advantageous to export the results, either in graphics or in spreadsheets format. The second phase is based on the use of low-cost hardware to interact with the aimed hardware device (the FPGA), working as a prototyping verification testbed. In this paper, it is used an Arduino Uno, plugged into the terminal of a personal computer, which can tell precisely how the hardware system is working. Real stimula can be given to the hardware device to check the FPGA circuit's functional behaviors, in real-time.

2. Evolvable Hardware Basic Concepts

The concepts around Evolutionary Computing lays in the foundations of biology and genetics. First of all, Evolutionary Computing has abstractions from nature-based evolution, such as mutation, crossover and elitism [11]. But the overall objective of this optimization algorithm is that the fittest individual or candidate solution survives until the next generation. The candidate solution is also called genotype, and in EHW it represents a set of bits, '0' and '1'. In the present work's context, it can be interpreted as a code that implements a digital circuit. The phenotype is the manifestation of the genotype in the environment, or the description of the circuit's behavior, applying signals stimula as the its input. But how does the circuit evolve? Where is the evolution mechanism?

In order to have an artificial selection, the individuals must be evaluated. This final result of this process is called fitness. in digital circuit design, a usual metric for this evaluation is the accuracy according to the desired problem's truth table and the number of logic gates utilized. This value is used to determine the genotype(s) that are good enough to reproduce with other genotypes (crossover). Mutation is a feature to increase the diversity in the population (the whole group of individuals in a generation), and elitism is a feature that increases the chance that the fittest individual survive to the next generation. As the time goes by, the quality of the individuals will be improved, and they would be fitter to execute the finality of the circuit design [4].

3. A double-phase supportive methodology for hardware development

The present article describes the development of an Evolvable Hardware Architecture Learning Platform. As EHW is a complex issue, involving different areas of knowledge (Computer Architecture, Evolutionary Computing, Reconfigurable Computing), two approaches supported the process of implementing an EHW Architecture.

The presented methodology could be implemented by other students, researchers or developers, by undergraduation/ post-graduation students or professionals, in fields like Electrical Engineering, Computer Science/Engineering and correlated areas. Fig.2 shows the flow chart for the development of an EHW, including the double-phase methodology.

The first phase consists on using the project requirements and definitions and configurating the virtual environment to start designing the architecture. In the process, the PSE user adjusts and simulates the architecture until it presents a reasonable solution to be implemented. It is notable that this kind of simulation requires less development workforce and less time to achieve a solution than traditional methodologies, such as IDEs and electronic CADs (Computer-Aided

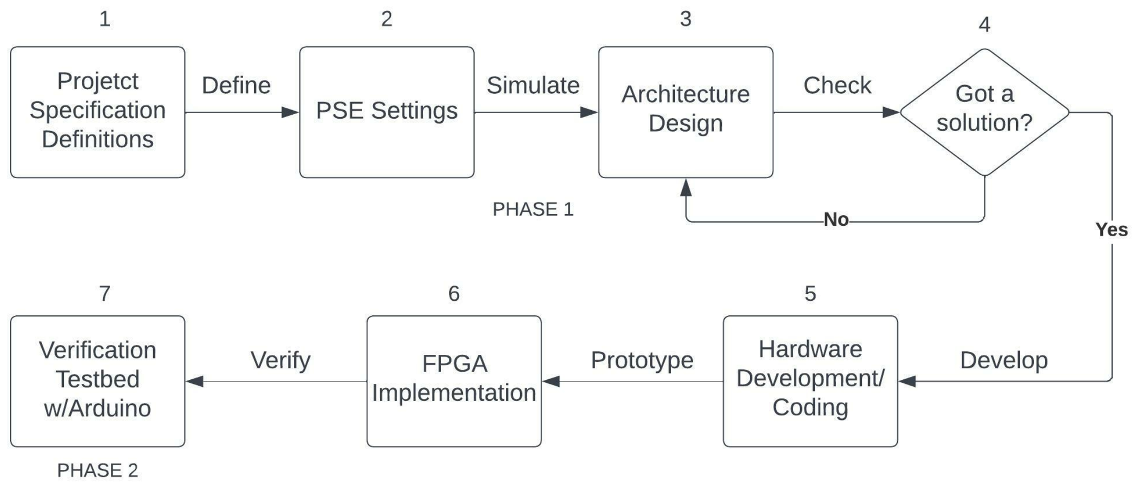


Figure 2. The double-phase methodology flow chart, containing the design simulation in the PSE (blocks 2 and 3), and the verification testbed (block 7).

Design) tools. The second phase is the use of low-cost hardware to assist testing and validating the architecture design. This second phase automates the manual process of testing the implemented architecture or algorithm, because the Arduino was programmed to send verification signals and to give a response according to what it receives.

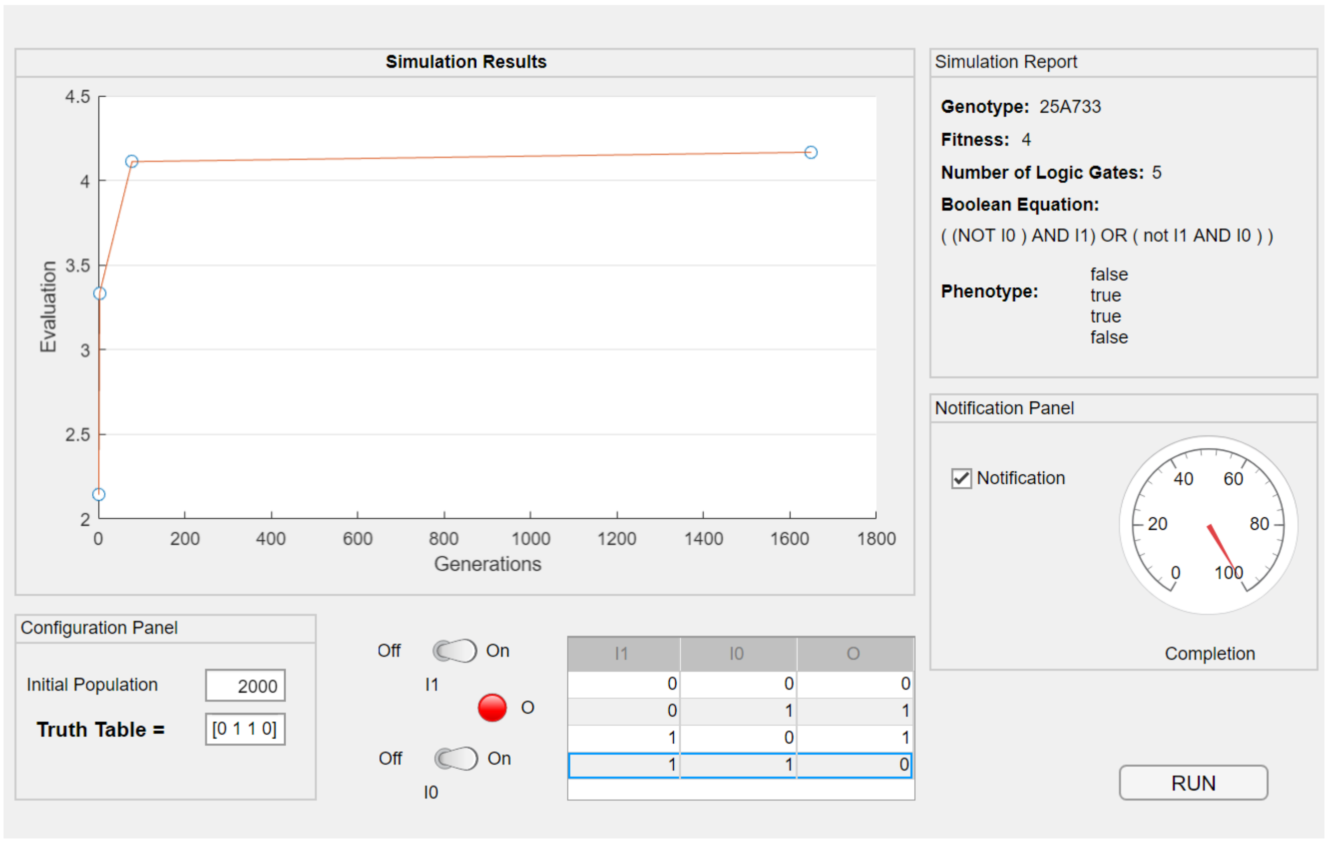
Both phases supported the development of the EHW architecture. They can be used by any Computer Science/Engineering student or professional in order to improve their knowledge about the issue or to develop better EHW solutions in a shorter period of time.

3.1. Phase 1: Evolvable Hardware Logic Simulation PSE

During the initial research period of the author's Master's degree, plenty of reading and preliminary/investigatory implementations were made. Technical implementations in Evolvable Hardware aren't usually available on Internet, even in detailed papers. However, by participating in specific disciplines, the implementation of each basic module of the EHW was possible. The process of creating an EHW Architecture, delivering accurate results, took around 10 months, including hours of coding time, designing, research, brainstormings with colleagues, professors and advisors. Some methodological aspects were still pending at that time. For example, when the Architecture passed through tests and analysis, there wasn't an automatic mechanism to averiguate the accuracy/correctness of the result. The resulting output variables are directly related to the inputs, so it is possible to calculate the answers, but the manual process usually took around ten minutes, for one individual, due to the VRC circuit's complexity.

Therefore, a question emerged in the summer break of the University: what if there was an interface capable of answering back the final output metrics easily and fast? The programming language chosen to implement this feature was MATLAB, because it was the language utilized in previous disciplines and experiences of the researchers. Deciding which language or platform was going to be used was not the focus at that time, because the interface would be working as an auxiliar tool. And the best part is that is was coded in a couple of days, and it is able to deliver the circuitry output (boolean equation, number of logic gates used, phenotype, fitness value) in a matter of seconds, given the inputs (problem's truth table and the genotype). A remarkable fact is that the genotype verification is a basic and important process that was executed until the last experiments of the research.

Since the MATLAB and VHDL VRC implementations were completely matched, and the genotype verification time was reduced from minutes to seconds, another question was brought to mind: What if there was a basic evolutionary algorithm interface, flexible and parametrized, in a way that it would be easier to investigate the parameters and explore the EHW design space? Considering the problem mentioned about the time hardware simulation might take, a basic MATLAB instance of the VHDL EA was im-



Simulation Report

Genotype: 25A733

Fitness: 4

Number of Logic Gates: 5

Boolean Equation:

((NOT I0) AND I1) OR (not I1 AND I0))

Phenotype: false true true false

Configuration Panel

Initial Population 2000

Truth Table = [0 1 1 0]

I1	I0	O
0	0	0
0	1	1
1	0	1
1	1	0

Figure 3. PSE tab for circuit evolution. Given the truth table and the number of generations limit, it evolves and presents the genotype, its fitness, phenotype and the final boolean equation, beyond the evolution curve.

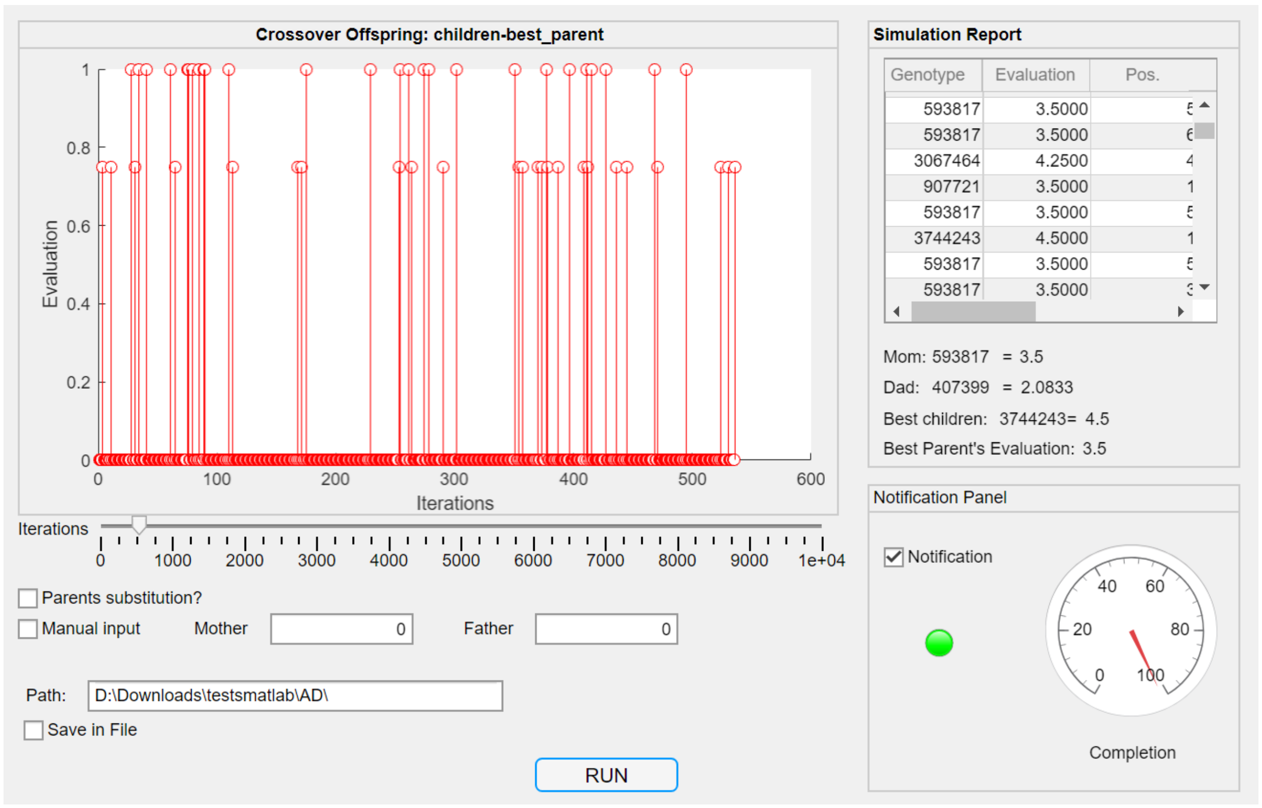
plemented and tested. As the PSE (Problem Solving Environment) strategy was presented at a previous discipline, a basic alphanumeric interface was developed to assist the calculations and validating future features of the EHW Architecture.

A PSE is a user-friendly interface that makes the process of solving a particular problem easier, and it is usually used in complex problems development, study, or research [14],[15],[16].

The concepts of evolutionary computation were still being assimilated by the author, several implementation issues were answered by using the PSE. For example, answering the question: how could the crossover be better implemented? Fig. 4 represents an attempt to generate better genotypes from two different ”parents”. Different possibilities and approaches were tested and the interface delivered the comparison metrics in a tabular and graphic point of view. Would there be an optimal cutoff bit to start the crossover? Would there be only one cutoff bit? Would the order of arrangement be important to the final result? These questions were also answered in a set of experiments in the PSE.

Then the PSE interface was developed to represents a sample of the tested genotypes regarding the validation of the MATLAB PSE, so future experiments are considered reliable in terms of the ENA’s characteristics and response. The PSE also simulates the evolution of a desired gate, as seen in Fig. 3.

The PSE has many tabs, each one with a distinct objective. The one presented in Fig. 3 show all the desired outputs of the EA: The best genotype, its fitness, the number of logic gates, the phenotype, and the boolean equation, which is a differential feature (in hardware simulation the equation cannot be displayed). The user can insert the maximum number of generations and the reference vector (truth table), and, after the evolution is complete, he/she can verify the circuit simulated in virtual IOs (two switches and a lamp). In this tab the user can make a global simulation of the Evolutionary Algorithm, including all the features of the implemented EA. On the other hand, the tab presented in Fig.4 provides the exploration of only one aspect of the EA: the crossover. It was necessary because this feature was is-



Genotype	Evaluation	Pos.
593817	3.5000	5
593817	3.5000	6
3067464	4.2500	4
907721	3.5000	1
593817	3.5000	5
3744243	4.5000	1
593817	3.5000	5
593817	3.5000	3

Mom: 593817 = 3.5
Dad: 407399 = 2.0833
Best children: 3744243= 4.5
Best Parent's Evaluation: 3.5

☐ Parents substitution?
☐ Manual input Mother 0 Father 0
Path: D:\Downloads\testsmatlab\AD\
☐ Save in File
✓ Notification

Figure 4. Crossover tab in the EHW PSE. Users can export the simulation results in tables and graphics.

sue of several questions in the architecture implementation.

3.2. Phase 2: Evolvable Hardware Physical Prototyping Testbed

Arduino is a well-known hardware prototyping platform made for designers, architects, and professionals from areas of knowledge unrelated to Computing and Engineering. However, it has demonstrated to be a supportive tool for teaching programming, electronics, design concepts, both for school and college education.

As it is a low-cost hardware, easy to use and very popular, students have been using Arduinos in their DIY projects, as well as in academical studies. Areas like Internet of Things (IoT), Digital Control and Telecommunication are frequently approached by the microcontroller board.

Programming an Arduino is easy and there is an online repository of codes, and dozens of forums to assist beginners as well as expert developers. Beyond that, the communication between the Arduino and a Personal Computer was made easy through terminals, in a way students can monitor what is happening and also actuate in the experiments. Doing so, it is more feasible to verify the circuitry's functional characteristics.

When we simulate in a hardware development environment, we can only have a glimpse of the circuitry's behavior. Post-synthesis simulation, as the one presented in Fig. 1, helps validating the timing behavior of the signals, and it makes possible the visualization of the information flow through the waveforms. However, some behavior analysis are only possible when we perform in hardware, and it ensures the circuit's correct operation.

Nevertheless, when we implement a solution in hardware, there is no financially feasible way to monitor and inspect the signals that are inside the FPGA. Fig. 5

4. Results and Discussion

The input genotypes simulated to validate the MATLAB interface were generated randomly by Quartus II 16.1 IDE (Integrated Development Environment utilized to create the Architecture). The results presented in Table 1 indicates that the MATLAB algorithm has the same functional performance of the EHW Architecture implemented in the FGPA.

Another feature implemented in the PSE is a data visualization tool for the VRC. It is useful to discover if the possi-

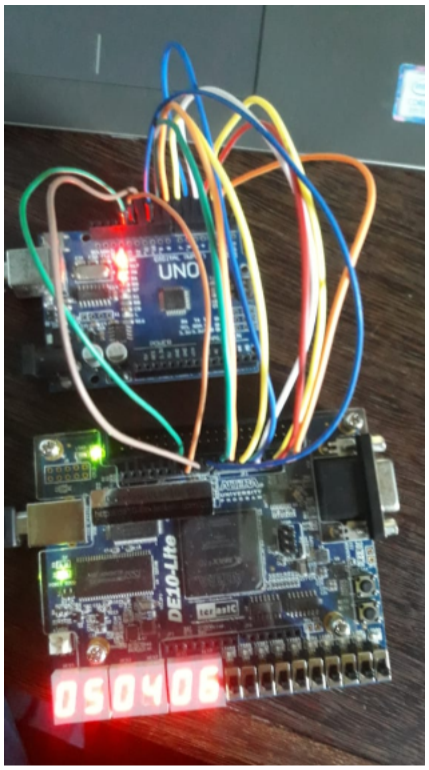


Figure 5. Evolvable Hardware FPGA and a testing Arduino Uno, emulating physical input stimula.

	Hardware			MATLAB		
Genotype	Phenot.	Fit.	NG	Phenot.	Fit.	NG
2AC4CC	0010	3	2	0010	3	2
00B129	0000	2	7	0000	2	7
1B639D	1111	2	1	1111	2	1
104125	0011	2	5	0011	2	5
54218	0010	3	4	0010	3	4
3BE17C	0000	2	1	0000	2	1
2E8CA1	0010	3	2	0010	3	2
21C96A	0100	3	5	0100	3	5
12D225	1100	2	7	1100	2	7
05F252	0100	3	4	0100	3	4

Table 1. Summary of the simulation results: post-synthesis hardware simulation and model verification in Matlab.

ble solutions implemented through the VRC correspond to the optima circuits or not, and also how the candidate solutions are distributed in the solution space. It also helps discerning whether a designed VRC is suitable for every circuit possible. In fact, a well designed VRC should have the representation of the optimum candidate solution in all the different problems. Fig. 6 shows an example of the XOR gate implemented by the VRC, using AND, OR and NOT gates. The points in red represent the optimal solutions (maximum functional fitness and minimum number of logic gates), while the blue ones represent the sub-optima solutions (maximum functional fitness). Each axis on Fig. 6 is a layer of the VRC, consisting of a different set of bits of the genotypes. So, one point in this three-dimensional space represents a genotype, a candidate solution for the aimed problem.

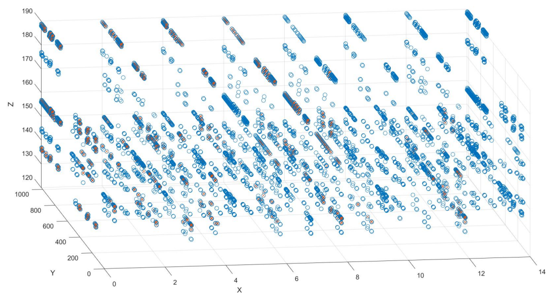


Figure 6. Genotype 3D visualization graphic exported by the PSE.

Table 2 compares two architectures: One developed and implemented before the creation and use of the PSE, and the other after it. The former was coded in VHDL and the exploration of the parameters of the EA and the architecture design process itself were made in Quartus II environment, in a time-consuming simulation process. The latter was also coded in VHDL, but the exploration of the parameters and the architecture design were made using the PSE. As the evolutionary algorithm was explored in the PSE, many optimization implementations were made, making the Architecture use less memory, registers, logic elements and IO elements. It means that the Arch.2 consumes less energy, and also it may work along other solutions inside of the FPGA, depending on a project's specification. It also operate at a higher clock frequency, which means that the design can work in contexts and applications where it is demanded more a faster processing power.

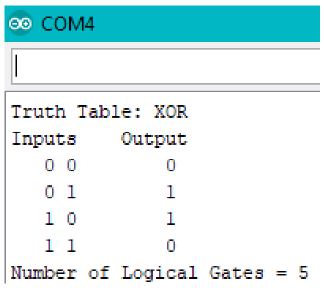
Furthermore, valuable conclusions were taken after the use of the PSE. Punctual misconceptions were highlighted, the process of collecting the results in the PSE is automatic, while the one in the Hardware IDE isn't.

Fig. 7 is a brief representation of the data visualization you can have on the Arduino platform. The information presented in the terminal must be the same coded and shown by the 7-segment displays of the FPGA. Fig. 5 attests the validation of the results, since the number of logic gates,

Parameter	Arch.1	Arch.2
Max Frequency (MHz)	84.53	133.37
Total Logic Elements	9285	537
Percentage	19	1
Total Registers	3342	200
Total Memory bits	2334	156
Percentage	0.1391	0.0093
IO Usage	239	74

Table 2. Comparison between the two proposed models of architectures for EHW

the phenotype and the problem correspond with the ones on Fig. 7. The only difference is that it is possible to insert more information on a terminal, in comparison with a 6-figured 7-segment display.



```
Truth Table: XOR
Inputs    Output
   0 0       0
   0 1       1
   1 0       1
   1 1       0
Number of Logical Gates = 5
```

Figure 7. PC-Arduino monitor interface: another way to verify the circuitry's functional correspondence and behavior.

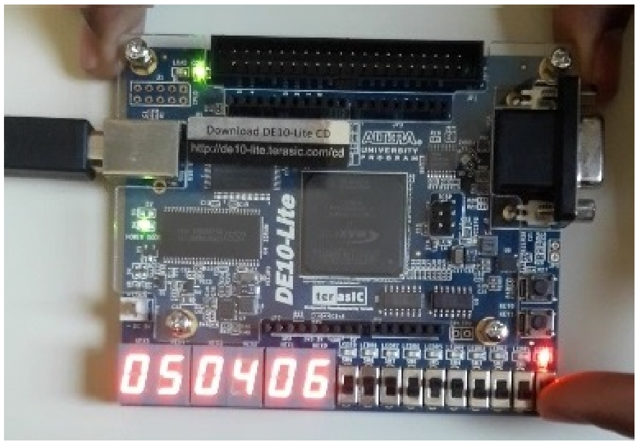


Figure 8. Evolvable Hardware set to evolve the XOR gate, in a MAX 10 FPGA, device 10M50DAF484C7G.

Another advantage seen in the use of an Arduino Uno to make the tests is that it was faster and more systematic than a human-made set of tests. For problems involving a low number of inputs, it is not a problem. But for a higher number, the quantity of combinations possible grows exponentially, so it would be harder to test all the possible combinations in a systematic way, and it would take much more time.

5. Conclusions

This article presents a learning platform for Evolvable Hardware Design, in a double-phase approach. EHW is a complex and interdisciplinary issue involving Reconfigurable Computing and Evolutionary Computing. The first phase approaches the simulation of the EHW architecture, and delivers a Problem-Solving Environment to assist the development process. The second phase gives support to the physical implementation of EHW architectures. It consists of a low-cost auxiliar hardware device, an Arduino UNO, to be a more accessible interface for project verification and FPGA testing.

The results indicates that a first phase is good for research and investigation before the EHW implementation. It helps the student or researcher to explore different approaches he/she could follow in designing the architecture because in the PSE the simulation time is lower than usual hardware development tools. In addition, it is less time consuming for research, comparing to traditional tools. Using the first phase approach the user can compare different EA tunning without worrying about hardware implementation issues, and is capable of improving the algorithm. The second phase is also a supporter for students and developers during the testing part of the project. Since hardware verification tools are usually expensive, a low-cost hardware interface like an Arduino UNO can perform the real-time analysis of what is happening inside the FPGA. This is also an extra way to validate the design, after using post-synthesis simulations in the IDE and the FPGA display interface, like in Fig.8.

Both phases are essential for complex hardware development designs, mostly in contexts where the students or developers cannot afford professional software acquisition or expensive hardware equipment. The presented results shows that a traditional approach, without the double-phase learning platform and only using the hardware IDE, can use 17 times the number of logic elements, around 16 times the number of total registers and almost 15 times the total memory bits, comparing with the implementation achieved through the double-phase learning platform using.

Future researches include the use of remote experimentation to deliver a space where electrical and computing engineering students and research can utilize FPGAs to develop their own EHW implementations, without the necessity to buy an FPGA to start developing their own projects.

References

[1] Pauline C Haddow and Andy M Tyrrell. Challenges of evolvable hardware: past, present and the path to a promising future. *Genetic Programming and Evolvable Machines*, 12:183–215, 2011.

[2] M A Almeida and E C Pedrino. Hybrid evolvable hardware for automatic generation of image filters. *INTEGRATED COMPUTER-AIDED ENGINEERING*, 25:289–303, 2018.

[3] Rui Yao, Ping Zhu, Junjie Du, Meiqun Wang, and Zhaihe Zhou. A general low-cost fast hybrid reconfiguration architecture for fpga-based self-adaptive system. *IEICE TRANSACTIONS ON INFORMATION AND SYSTEMS*, E101D:616–626, 3 2018.

[4] Lukas Sekanina. Evolutionary hardware design. volume 8067. SPIE-INT SOC OPTICAL ENGINEERING, 2011. Conference on VLSI Circuits and Systems V, Prague, CZECH REPUBLIC, APR¡br/¿18-20, 2011.

[5] Wang Jin and Chong-Ho Lee. Virtual reconfigurable architecture for evolving combinational logic circuits. *JOURNAL OF CENTRAL SOUTH UNIVERSITY*, 21:1862–1870, 2014.

[6] Derek Whitley, Jason Yoder, and Nicklas Carpenter. Intrinsic evolution of analog circuits using field programmable gate arrays. *Artificial Life*, 28(4):499–516, 2022.

[7] M Lovay, G Peretti, and E Romero. Implementation of an adaptive filter using an evolvable hardware strategy. *IEEE LATIN AMERICA TRANSACTIONS*, 13:927–934, 2015.

[8] D Grochol, L Sekanina, M Zadnik, J Korenek, and V Kosar. Evolutionary circuit design for fast fpga-based classification of network application protocols. *APPLIED SOFT COMPUTING*, 38:933–941, 1 2016.

[9] Phil Husbands, Yoonsik Shim, Michael Garvie, Alex Dewar, Norbert Domcsek, Paul Graham, James Knight, Thomas Nowotny, and Andrew Philippides. Recent advances in evolutionary and bio-inspired adaptive robotics: Exploiting embodied dynamics. *Applied Intelligence*, 51(9):6467–6496, 2021.

[10] William B Langdon. Genetic programming and evolvable machines at 20. *Genetic Programming and Evolvable Machines*, 21(1):205–217, 2020.

[11] F Cancare, S Bhandari, D B Bartolini, M Carminati, and M D Santambrogio. A bird's eye view of fpga-based evolvable hardware. pages 169–175, 2011.

[12] X Yao and T Higuchi. Promises and challenges of evolvable hardware. *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, 29:87–97, 1999.

[13] Lucas Augusto Müller de Souza, José Eduardo Henriques da Silva, Luciano Jerez Chaves, and Heder Soares Bernardino. A benchmark suite for designing combinational logic circuits via metaheuristics. *Applied Soft Computing*, 91:106246, 2020.

[14] Ju-Hwan Kim, Ho-Jun Lee, Sang-Ho Kim, and Jeong-Oog Lee. A problem solving environment portal for multidisciplinary design optimization. *ADVANCES IN ENGINEERING SOFTWARE*, 40:623–629, 2009.

[15] Gyongyver Molnar and Bello Csapo. Exploration and learning strategies in an interactive problem-solving environment at the beginning of higher education studies. pages 283–292. IKAM-INST KNOWLEDGE ASSET MANAGEMENT, 2017. 12th International Forum on Knowledge Asset Dynamics (IFKAD), St.¡br/¿Petersbur, RUSSIA, JUN 07-09, 2017.

[16] M H Eres, G E Pound, Z Jian, J L Wason, F L Xu, A J Keane, and S J Cox. Implementation and utilisation of a grid-enabled problem solving environment in matlab. *FUTURE GENERATION COMPUTER SYSTEMS*, 21:920–929, 6 2005.