Kayed and Elsayed
Bulletin of the National Research Centre (2023) 47:94
https://doi.org/10.1186/s42269-023-01066-1

Bulletin of the National Research Centre

RESEARCH Open Access

An optimizing technique for using MATLAB HDL coder

Somaya Kayed[1] and Ghada Elsayed[2*]

Abstract

Background MathWorks has provided an invaluable tool for designing and implementing FPGAs. MATLAB HDL coder serves a dual purpose, providing a quick proof of concept on the one hand and providing the g an easy-to-use platform for testing and verification on the other. It has main drawbacks over these advantages; it generates a code that is not optimized for both area and frequency.

Results In this paper, we provide a technique for optimizing both area and frequency without losing the main advantages. The most affecting problem we found is loops. This paper classifies loop writing purposes into two types. The first one is preferable and introduces ease of writing a few lines instead of repeating the code. The second type is the problem that we intended to solve. Type II loop is appearing when the algorithm should perform these lines for several clock cycles. Writing it traditionally, force the synthesizer to implement all the repetitive clock cycles as repetitive hardware to be done in one clock cycle. This clock cycle is wide in time and is slow in frequency. This paper introduces an optimization technique for this problem. We compare before and after the implementation of our proposed technique.

Conclusions We used Xilinx Spartan 6 XC6SLX4-2CPG196 FPGA. Our proposed technique improves the number of slice LUTs (Look Up Tables) requirement from 366 to 72%. The frequency improved from: 26.574 to 185.355 MHz. Based on that, we now recommend using MATLAB HDL coder in FPGA Design.

Keywords FPGA, MATLAB HDL coder, AES

Background

In FPGA design, there are levels of abstraction. The register transfer level and behavior level are the most famous abstraction levels. Recently, MathWorks introduced a new level of higher abstraction. Processes are described semantically by their functionality, inputs, outputs, and preconditions needed for their execution. This, of course, makes the mind think about software, not hardware. Hardware designers should be aware of the level of each signal and its exact time. The invaluable addition is that one can write software and MathWorks by MATLAB HDL coder [1] generates the corresponding HDL code. This is a breakthrough in the field of digital design on FPGAs. It has many vital as it reduces time to market through ease of design. This is beside the ease of testing and verification. Not only writing MATLAB scripts but also from Simulink to HDL code is another option that was used for many applications [2–8]. There is another cooperation between the vendor of FPGAs and MathWorks. For example, Xilinx Company proposed an Integrator Design Environment (IDE) for FPGA under the MATLAB tool. This IDE is named XSG; it is a high-level design tool that allows the use of the MathWorks Simulink environment in the design of digital circuits dedicated to Xilinx FPGAs. It is used for hardware system generation, simulation,

*Correspondence:
Ghada Elsayed
Ghada.farouk@eng.mti.edu.eg
[1] Electrical Department, Obour High Institute for Engineering and Technology, Obour, Egypt
[2] Electrical Engineering Department, Modern University for Technology and Information, Cairo, Egypt
Full list of author information is available at the end of the article

Springer Open

© The Author(s) 2023. **Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

and validation throughout the hardware co-simulation technique. Examples of some implementations can be found in [9–14]. The performance of the HDL code generated by MATLAB HDL coder was studied many times [15, 16], for example. The result of these studies can be summarized by the generated HDL code having different IOs and utilizing more area with low speed. Research has shown that MATLAB currently consumes more resources and provides less speedy design [16, 17]. These researchers compared it to the traditional methods of designing FPGAs, i.e., VHDL/VERILOG. These vital advantages are a major motive for searching for methods to avoid these shortcomings. Not only we as researchers, but also MathWorks has begun to answer these questions and provided some solutions. There exist optimization methods in MATLAB HDL coder. It dealt with the same points that we propose the loop problem and pipeline. In loops problem, MATLAB HDL coder provides two options: one called stream option and the other called unroll option. For area optimization, the stream option is used to generate one instance plus some logic to maintain the functionality in the HDL CODE. For speed optimization, unroll option is used to make multiple instances of the loop body in the HDL code. But there is no control by the developer on which loop on the code to be optimized by which option. MathWorks assumes one loop all over the script code. However, they use the making between the embedded FPGA memory and matrices [18] which is very useful. MathWorks also introduces a pipeline optimization option. Distributed pipeline is a subsystem-wide optimization supported by HDL coder for reducing the critical path and achieving high-clock speed hardware. By turning on the distributed pipeline, HDL coder redistributes the input pipeline registers, output pipeline registers of the subsystem, and the registers in the subsystem to positions that minimize the combinatorial logic between registers and maximize the clock speed of the chip synthesized from the generated HDL code. In this paper, we aim to present a technique for writing codes to reach the closest and best use of resources and to improve the speed of design as well. In This paper, we focus on improving the performance of a DUT (Design Under Test) unit. We choose the AES (Advanced Encryption Standard) [19] to be our DUT unit. The AES was written by MathWorks as an example for MATLAB HDL coder [20, 21]. In a previous scope, we compared it with the VHDL implementation [16]. We aim to introduce a layer between digital design engineers and software developers. Digital designers know the timing values of each signal in their design. They use timing simulation to review the functionality of the design from the top level to the lowest level of interconnection. This takes time. This spent time is for the cost of optimization of area and frequency. However, software developers know about transforming the algorithm into a script code. They do not care about how this script is performed and at what time. They just care about output validation. We intend to put a procedure for them to reach the optimized output of the FPGA implementation without having to learn digital design.

Digital designers know some facts; first, pipeline improves the frequency performance. In digital circuits, the pipeline is to insert registers after combinatorial circuits and between combinatorial circuits [22]. Figure 1 illustrates the pipeline of a combinatorial logic with output feedback to the input. This gives the chance for the clock's periodic time to be reduced. In consequence, the frequency increases. The last and most important fact is that digital circuits are controlled by a clock. The looping function is done by the existence of that clock with a clock enabled. The synthesizer of the HDL code is not able to understand loops. But instead, if you replace it with an if statement you force the synthesizer to know that here there is a clock enable signal and this clock is under the control of the if statement conditions. The loop is automatically performed. In "Methods" section, we introduce a technique to improve the performance by taking some functions of the AES by MathWorks as examples we compare the area optimization before and after the implementation of the proposed technique. In

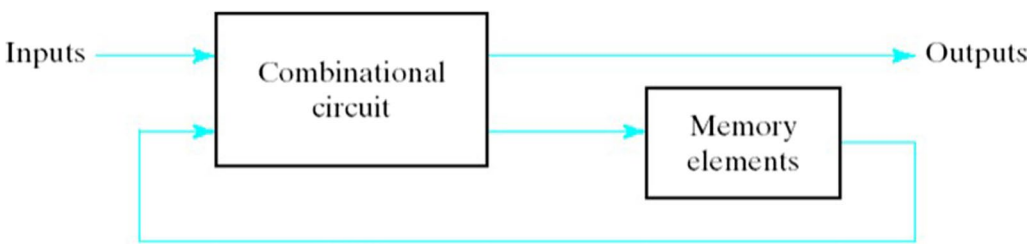

Fig. 1 Simplified description of digital circuits

"Results" section, we represent the findings and results. Next, we discuss the results. Finally, we conduct the conclusion.

Methods

There is a main problem according to our research forces MATLAB HDL Coder to consume more resources than coding by VHDL/VERILOG languages. This problem is writing loops. Two types of loops could be found. The first one is meant to create multiple instants to be used all in one clock cycle, i.e., multiple repetitive blocks with different inputs. The second one is meant to force the synthesizer to implement on the FPGA one instant to be repetitively, used each clock cycle. This for-loop type is our concern here. HDL synthesizers understand loops by creating multiple instants of the loop's body. If we keep this type, the synthesizer will synthesize it as multiple repetitive blocks without needs. This reduces the performances in both area and frequency. In the next two subsections, we will discuss two examples: one for the type I loops and the other for type II loops. We take the Design Under Test (DUT) as AES encryption.

Example of the type I loop

The type I loop has no problem with the performance it just facilitates the coding. Figure 2 illustrates the extension of the AES key script. Each round of the ten rounds needs a key that depends on the previous key. Each key has four columns, BS = 4. Each round key has sixteen bytes (128-bit) structured as columns such that column 1 has k_out array numbers 1, 5, 9, and 13, column 2 has k_out array numbers 2, 6, 10, and 14, column 3 has k_out array numbers 3, 7, 11, and 15, and column 4 has k_out array numbers 4, 8, 12, and 16 [19]. The loop in Fig. 2 is run over the number of columns BS. This loop ends by creating a 128-bit round key "*k*." The whole round key is needed each round/ cycle. So let the synthesizer repeat the implementation of the instant, knowing that this matches the requirements without affecting the performance, so this is an example of loop type I, in which each round from the ten rounds the function should output 128 bits of the key. The loop here is just to generate the 128-bit key, not the ten 128-bit keys. This key is necessary for each round. So we need this loop to be unrolled while synthesizing which is the default. Then, there is no need for any optimization just keep it as it is.

Example of the type II loop

The AES 128 encryption algorithm [19] has 10 rounds to process one plain text. Initially, the input plain text and cipher key are used in the first round. The output of the first round is the temporary cipher text "*s*." Each round needs a round key "*k*," this "*k*" is generated from the extension of the AES key script. This means that the next rounds process both "*k*" and the feedback "*s*" by the "mainroundsstate" script to get new "*k*" and "*s*." The last round is different in using the final round script instead of the "mainroundsstate" as shown in Fig. 3. As we can see, we can process the round in a cycle. The output is feedback to the input. This should be stored in registers. However, if we keep this loop as it is and proceed to the synthesizer, then we will obtain a bad utilization of the FPGA resources. We need to find a way for storing the previous value of the output and feedback on the input with it. There exists a keyword in MATLAB that do so. This keyword is persistent. A persistent variable is a local

```
% Calculate the first column of the new round key
k_out((1-1)*BS + 1) = bitxor(k_in((1-1)*BS + 1), k_temp2(1));% kout(1)
k_out((2-1)*BS + 1) = bitxor(k_in((2-1)*BS + 1), k_temp2(2));% kout(5)
k_out((3-1)*BS + 1) = bitxor(k_in((3-1)*BS + 1), k_temp2(3));% kout(9)
k_out((4-1)*BS + 1) = bitxor(k_in((4-1)*BS + 1), k_temp2(4));% kout(13)

% Calculated the rest columns of the new round key
for i = 2:BS
 k_out((1-1)*BS + i) = bitxor(k_in((1-1)*BS + i), k_out((1-1)*BS + i - 1));
 % kout(2),kout(3),kout(4)
 k_out((2-1)*BS + i) = bitxor(k_in((2-1)*BS + i), k_out((2-1)*BS + i - 1));
 % kout(6),kout(7),kout(8)
 k_out((3-1)*BS + i) = bitxor(k_in((3-1)*BS + i), k_out((3-1)*BS + i - 1));
 % kout(10),kout(11),kout(12)
 k_out((4-1)*BS + i) = bitxor(k_in((4-1)*BS + i), k_out((4-1)*BS + i - 1));
 % kout(14),kout(15),kout(16)
end
```

Fig. 2 Type I loops example

```
function ciphertext = mlhdlc_aes(plaintext, cipherkey)

    BS = 4;
    RS = 10;
    s = uint8(zeros(BS*BS, RS));
    k = uint8(zeros(BS*BS, RS));

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % First round
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    s(:,1) = bitxor(plaintext, cipherkey);
    k(:,1) = ExtendKeys(cipherkey, 1);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Second to tenth round
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i = 1:RS-1
        s(:, i+1) = mainroundstate(s(:, i), k(:, i));
        k(:, i+1) = ExtendKeys(k(:, i), i+1);
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Final round
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    ciphertext = finalround(s(:, RS), k(:, RS));
end
```

Fig. 3 Type II loops example

variable in a MATLAB® function that retains its value in memory between calls to the function. If one generates code from a model [23]. It generates a storage device with either registers or memory blocks. Use the persistent keyword to store the output of the loop that will be feedback to the input in the next call. This storing of the feedback acts as a pipeline. Pipeline improves both area and frequency performance. Let the tester call the top level in a loop. This loop in the tester replaces the clock effect in the hardware (FPGA). Figure 4 illustrates the code modifications. In Fig. 4, the loop is replaced with an if statement to solve the looping problem and so improve the area utilization. We also added a register to the output by using the keyword persistent. These registers both store the last value and pipeline the combinatorial logic. This improves the frequency. In the next section, results will be presented not only for the proposed method/technique but also for the built-in MATLAB HDL coder methods.

Results

In this section, we present the synthesizer results for non-optimized code, unroll option, stream option, and our proposed optimization technique. The FPGA platform is the target platform to be Digilent Cmod S6™ FPGA Board [24]. It has Xilinx Spartan 6 XC6SLX4-2CPG196 FPGA based on a previous AES design and implementation using VHDL, [25]. Table 1 shows the utilization of Xilinx Spartan 6 XC6SLX4-2CPG196. Table 2 shows the utilization under the unroll optimization option. Table 3

```
function [ciphertext] = AES_Enc(plaintext_in_aes,cipherkey_in_aes,i)
persistent  s;
persistent  k;
BS = 4;
RS = 10;
ciphertext = uint8(ones(BS*BS, 1));

if (isempty( s))|| (isempty( k))
   s = bitxor(plaintext_in_aes, cipherkey_in_aes);
   k = ExtendKeys(cipherkey_in_aes, 1);
elseif  i  == RS
    ciphertext = finalround(s, k);
else
     s = mainroundstate(s, k);
     k = ExtendKeys(k, i+1);

end
end
```

Fig. 4 Applying the proposed optimization for loop problem

Table 1 Estimated utilization summary for MATLAB HDL coder-based implementation to AES encryption module without loop optimization

Logic utilization	Utilization (%)
Number of slice registers	5
Number of slice LUTs	366
Number of fully used LUT-FF pairs	2
Number of block RAM/FIFO	8
Number of BUFG/BUFGCTRL/BUFHCEs	6

Table 2 Device utilization summary (estimated values) for unroll option by MathWorks

Logic utilization	Utilization (%)
Number of slice registers	58
Number of slice LUTs	357
Number of fully used LUT-FF pairs	11
Number of block RAM/FIFO	25
Number of BUFG/BUFGCTRL/BUFHCEs	6

Table 3 Device utilization summary (estimated values) for stream option by MathWorks

Logic utilization	Utilization (%)
Number of slice registers	132
Number of slice LUTs	638
Number of fully used LUT-FF pairs	19
Number of block RAM/FIFO	8
Number of BUFG/BUFGCTRL/BUFHCEs	6

Table 4 Device utilization summary (estimated values) for the proposed technique

Logic utilization	Utilization (%)
Number of slice registers	20
Number of slice LUTs	72
Number of fully used LUT-FF pairs	35
Number of block RAM/FIFO	16
Number of BUFG/BUFGCTRL/BUFHCEs	6

shows the utilization under the stream optimization option. Table 4 shows the utilization under the proposed optimization technique. In the next section, we will discuss these results in detail

Discussion

To illustrate the numbers, we draw a histogram for each table utilization percentages. Figure 5 shows the non-optimization options compared to the unroll optimization. The effect of the unroll optimization area is the huge increases of the used number of slice registers from 5 to 58%, the number of fully used LUT-FF pairs from 2 to 11%, and the number of block RAMs/FIFO from 8 to 25%. The only thing that slightly decreased is the number of slices LUTs 9%. This means that the unroll optimization option has an undesired effect. Figure 6 shows the comparison between the non-optimization options and the stream optimization. The figure illustrates that the stream optimization option in the

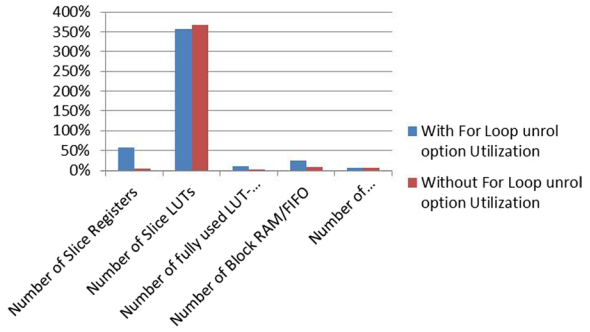

Fig. 5 A comparison between the FPGA utilization with and without the unroll option

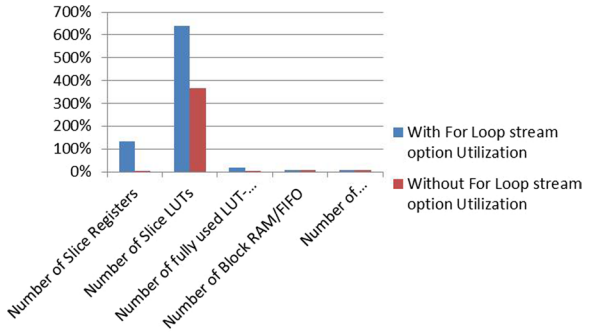

Fig. 6 A comparison between the FPGA utilization with and without the stream option

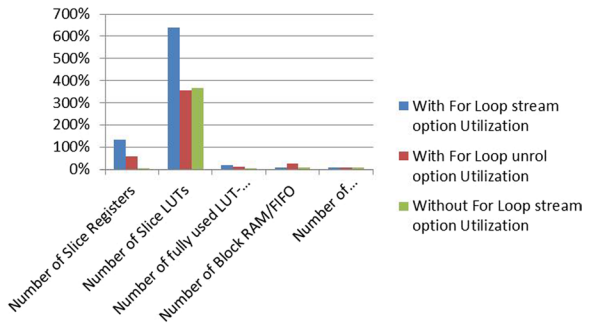

Fig. 7 A comparison between the stream, unroll, and non-optimization options

HDL coder increased the number of slice registers from 5 to 132%. The number of slice LUTs is increased from 366 to 638%. The number of fully used LUT-FF pairs is increased from 2 to 19%. This illustration leads us not to recommend the stream optimization option. Figure 7 compares the three tables together. The comparison shows that without HDL coder, optimization option is better than both options.

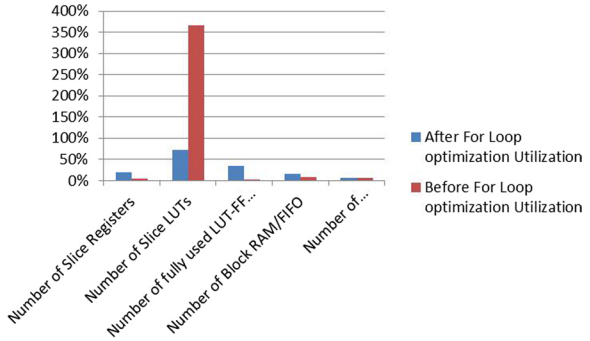

Fig. 8 A comparison between the proposed technique optimization and non-optimization utilization

In Fig. 8, a histogram comparison between before and after optimization is introduced. This figure shows that our optimization has freed up 294% number of slice LUTs, at the expense of 15% number of slice registers, 33% number of fully used LUT-FF pairs, and 8% of the number of block RAM/FIFO. The maximum frequency of our proposed technique is 185.355 MHz, while the frequency of the non-optimization method is 26.574 MHz. That we improve the speed approximately seven times faster. The proposed optimization technique is limited to the script path and not applicable to the Simulink path. This technique depends on loop problem. In the next section, the conclusion of the paper will be conducted.

Conclusions

In this paper, we prove that the code writing affects the synthesizing too much. We classified the loops into two categories. One is used to reduce the time of repeating the same body. This type should be kept without optimization. The other is the equivalent of the clock and clock enable signal in the hardware design. We introduced two-step procedure to optimize type II loop. The first step is to replace the for statement by if statement with a condition over a counter. The second step is to pipeline the output with a register using the persistent keyword. Our proposed technique improved both area and frequency. Using the proposed technique freed up 294% of the critical resources. This area optimization enabled the unfit FPGA to be fit. At the same time, the frequency was enhanced to be seven times faster than without our proposed optimization solution. The target was Xilinx Spartan 6 XC6SLX4-2CPG196 FPGA. Based on these results, we recommend designing the FPGA using MATLAB HDL coder on one condition; following the newly proposed technique that helps in overcoming its shortcuts.

Abbreviations

AES: Advanced Encryption Standard
FPGA: Field Programmable Gate Array
LUT: Look Up Table

Acknowledgements
The authors would like to acknowledge Electronics Research Institute (ERI), Egypt, for supporting us with the simulation tool (MATLAB).

Author contributions
SK and GE have contributed equally to the manuscript. They have read and approved the manuscript.

Availability of data and materials
We provide the code and the results in this manuscript. Any required data or materials are available if needed.

Declarations

Competing interests
The authors declare that they have no competing interests.

Author details
[1]Electrical Department, Obour High Institute for Engineering and Technology, Obour, Egypt. [2]Electrical Engineering Department, Modern University for Technology and Information, Cairo, Egypt.

Received: 11 March 2023 Accepted: 11 June 2023
Published online: 30 June 2023

References

1. MathWorks, inc: HDL Coder™ User's GuideCOPYRIGHT 2012-2015 (2012) https://www.mathworks.com/help/hdlcoder/ Accessed 20 Feb 2023
2. Km A, Duttagupta S (2023) Hdl-ready mac layer implementation for multi-node Li-Fi communications. Int J Inf Technol. https://doi.org/10.1007/s41870-023-01255-1
3. Sankar D, Lakshmi S, Babu C, Mathew K (2023) Rapid prototyping of predictive direct current control in a low-cost fpga using hdl coder. Int J Power Energy Syst 43(10):1–9. https://doi.org/10.2316/J.2023.203-0437
4. Bendahane B, Jenkal W, Laaboubi M, Latif R (2023) Hdl coder tool for ecg signal denoising. In: Digital Technologies and Applications: Proceedings of ICDTA'23, Fez, Morocco, Vol 1, pp 753– 760. Springer
5. Wallace NL (2023) Developing firmware for space weather probes 2 using HDL coder. Thesis on Master of Science (MS), Electrical and Computer Engineering Commons, Utah State University, Date of Award 8-2023. https://doi.org/10.26076/b311-3847
6. Nandakrishnan R, Arjun S, Nandakumar CN, Sajesh S, Harikrishanan K, Devi PA (2023) Adaptive beamforming using minimum variance distortionless response. Int J Res Eng Sci Manag 6(4):27–30
7. Havinga T, Jiao X, Liu W, Moerman I (2023) Accelerating fpga-based wi-fi transceiver design and prototyping by high-level synthesis. arXiv preprint arXiv:2305.13351
8. Rajaby E, Sayedi SM, Yazdian E (2023) Hardware design and implementation of high-efficiency cube-root of complex numbers. Microprocess Microsyst. https://doi.org/10.1016/j.micpro.2023.104847
9. Loganathan P, Karthikeyan R (2023) Combination of wavelet transform and sobel operator using xilinx system generator for edge detection in medical plant leaf. In: Computational Vision and Bio-Inspired Computing: Proceedings of ICCVBIC 2022, Springer, pp 333– 341
10. Neelima K, Meruva KR, Subhas C (2023) Image fusion using xilinx system generator for mri and ct medical image modalities. In: 2023 International Conference on Emerging Smart Computing and Informatics (ESCI), pp 1– 5
11. Khudair NA, Shujaa MI, Zghair EM (2023) Iot based image processing filters. In: AIP Conference Proceedings, vol 2591, AIP Publishing LLC, p 020007
12. Semmad A, Bahoura M (2023) Scalable serial hardware architecture of multilayer perceptron neural network for automatic wheezing detection. Microprocess Microsyst 99:104844
13. Hamdaoui F, Sakly A (2023) Automatic diagnostic system for segmentation of 3d/2d brain mri images based on a hardware architecture. Microprocess Microsyst 98:104814
14. Lopez-Ramirez M, Ledesma-Carrillo LM, Rodriguez-Donate C, Miranda-Vidales H, Mata-Chavez RI, Cabal-Yepez E (2023) Fpga-based online voltage/current swell segmentation and measurement. Comput Electr Eng 107:108620
15. Zafar A (2022) Performance of FPGA-based implementations of data classification techniques using HDL coder. California State University, Long Beach
16. Elsayed G, Kayed S (2022) A comparative study between matlab hdl coder and vhdl for FPGAs design and implementation. J Int Soc Sci Eng 4(4):92–98. https://doi.org/10.21608/jisse.2022.136645.1056
17. Zamiri E, Sanchez A, Yushkova M, Martínez-García MS, de Castro A (2021) Comparison of different design alternatives for hardware-in-the-loop of power converters. Electronics 10(8):926
18. MATHWORKS: Speed and Area Optimization (2022) Improvements through resource sharing, streaming, pipelining, RAM mapping. https://www.mathworks.com/help/hdlcoder/speed-and-area-optimization-1.html Accessed 20 Feb 2023
19. FIPS 197: Advanced Encryption Standard (AES) (2001). https://csrc.nist.gov/publications/detail/fips/197/final Accessed 20 Feb 2023
20. MATHWORKS Copyright 2011-2015 The MathWorks, Inc: Advanced Encryption System (AES) (2023) https://www.mathworks.com/matlabcentral/mlc-downloads/downloads/submissions/50098/versions/3/previews/mlhdlc_tutorial_comms_aes.m/index.html Accessed 20 Feb 2023
21. David Hill: Advanced Encryption Standard (AES)-128,192, 256 (2021) https://www.mathworks.com/matlabcentral/fileexchange/73412-advanced-encryption-standard-aes-128-192-256 Accessed 20 Feb 2023
22. Dasgupta S (1989) Computer architecture: a modern synthesis. Vol 1, John Wiley & Sons
23. Mathworks: Initialize Persistent Variables in MATLAB Functions (2022) https://www.mathworks.com/help/simulink/ug/initialize-persistent-variables.html Accessed 20 Feb 2023
24. Digilent: Cmod S6 FPGA Board Reference Manual (2023) https://digilent.com/reference/_media/reference/programmable-logic/cmod-s6/cmods6_rm.pdf Accessed 20 Feb 2023
25. Naiem GF, Elramly S, Hasan BE, Shehata K (2008) An efficient implementation of cbc mode rijndeal aes on an fpga. In: 2008 National Radio Science Conference, pp 1– 8

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Somaya Kayed SK is an Associate Professor and a head of Electrical Dept. (Electronics, communication, computer, and control engineering) at Obour Higher institute for Engineering and technology. She is graduated in 1987 from Ain Shams University with a BSc in Electronics and Communications department, with general grade (very Good) 80% her order of merit 13 of the successful students totaling (75). Her graduation project tackled Distinction and she was top ranked as the 13th on her class. In 1997, she finished her Masters of Science (MSc.) in the same department. Afterward, in 2000, she awarded her PhD also from Ain Shams University under the supervision of Prof. Hani Fikry Ragaie. She was an acting dean for the 2019 first term at Oubor Higher institute for Engineering and technology, she has published in local and international conference many scientific papers as well as multiple books related to her research interests (analog and digital VLSI design, current conveyor, nanoelectronics). At the same time, her passion for development is not limited to academia. She is also a volunteer member in NGO. On one side, these diverse experiences enriched not only her team work skills but also her leadership competencies.

Ghada Elsayed GE has received her Bsc, Msc, and PhD degrees in electronic and communication engineering, 2001, 2005, and 2008, respectively. She worked for the Egyptian Space Program for the first seven years and then she shifted to academic career, in which she worked for more than five years for MSA University and then she traveled as a visitor researcher to Japan, Kyushu Institute of Technology. After that, she continued for the same path in MTI University. In 2014, GE published a book about securing satellites control link. She also published many scientific papers.

Submit your manuscript to a SpringerOpen journal and benefit from:

- ► Convenient online submission
- ► Rigorous peer review
- ► Open access: articles freely available online
- ► High visibility within the field
- ► Retaining the copyright to your article

Submit your next manuscript at ► springeropen.com