JISSE
ISSN: 2636-4425

Journal of International Society for Science and Engineering
Vol. 4, No. 4, 92-98 (2022)

JISSE
E-ISSN:2682-3438

A Comparative Study between MATLAB HDL Coder and VHDL for FPGAs Design and implementation

Ghada ElSayed[1,*], Somaya Kayed[2],

[1]*Assistan Professor, Electrical Department, MTI University, Egypt*

[2]*Associate Professor, Acting Dean, Obour Higher Institute of Engineering and technology, Egypt*

A R T I C L E I N F O

Article history:
Received: 30-04-2022
Accepted:22-09-2022
Online:22-09-2022

Keywords:
FPGA
MATLAB HDL CODER
VHDL

A B S T R A C T

Nowadays, FPGA has become a very useful platform for multiple digital applications. Initially, the hardware programming languages like VHDL or Verilog were the only method for designing the FPGA. In this method, the designer should be able to transform the algorithm of the application into digital blocks. This consumes time and effort. Recently MATLAB realized the FPGA importance and decided to introduce a new tool for FPGA Design; this tool is MATLAB HDL Coder. The idea is to write a very easy MATLAB script and it will be converted to HDL using HDL Coder. Then this HDL code will go through the FPGA regular implementation path. This paper studies and compares, by example, the two methods. The comparison is done for Speed, FPGA utilization, and time for design/implementation. The digital unit under test was AES. The choice of this unit is based on having large input data, it makes many feedbacks, and It needs high speed. The test result doesn't recommend MATLAB HDL Coder for implementation but it recommends it to fast-proof ideas and fast prototypes. This is because the idea of just writing a simple script describing the algorithm results in a very complicated combinational circuit, which has a very low frequency. The recommended future research is to find a way to force the MATLAB script to be implemented in pipeline architecture. The expected result is to improve the performance in two directions utilization and frequency, but It'll lose the main advantage which is fast implementation.

1. Introduction

Field Programmable Gate Arrays (FPGAs) have played a very high important role in designing customized digital circuits. They introduce fast time to market, hardware security, and fast clock frequency in comparison with the Application-specific integrated circuit (ASIC) and software implemented using processors. Most software developers find difficulties in using the hardware description languages (HDL) in comparison with the software programming languages. MATLAB, as a very easy programming tool, decided to enter the FPGA field, by introducing an HDL coder. HDL Coder™ generates portable, synthesizable VHDL® and Verilog® code from MATLAB® functions, Simulink® models, and Stateflow® charts. Regardless of the PC specifications the generated HDL code can be used for FPGA programming or ASIC prototyping and design without affecting neither speed nor the utilization of the targeted hardware [1]. One of these three input methods can be used to generate the corresponding HDL code using an HDL coder. A very important question arises then, what are we trade-offs if we got the simplicity of designing the digital circuit? In another way, does the HDL coder method give a good utilization of the target FPGA? Does it give a good timing and frequency performance? To answer such questions, a case study is chosen. We choose the input to the HDL coder to be MATLAB functions, the target platform to be Digilent Cmod S6™ FPGA Board [2]. It has Xilinx Spartan 6 XC6SLX4-2CPG196 FPGA, and the digital circuit unit under test is to be the Advanced Encryption Standards (AES) [3]. This AES 128 was chosen as its VHDL code design was already obtained and published before [4]. We just used the Encryption part to use it in the VHDL path of comparison. The other path of the comparison is the MATLAB function for the same AES 128 encryption module; we just modified the tutorial of MathWorks [5]. In general, away from the HDL Coder, the AES implementation as MATLAB functions is available in [6]. We choose the AES not only because its VHDL code is ready with us but also for some main reasons in the digital design. Examples of these reasons are the required number of inputs and outputs are big and it should be entered or extracted using serial

[*] Ghada ElSayed, Obour Higher Institute of Engineering and technology, Cairo, Egypt, +201002337149, ghada.farouk@eng.mti.edu.eg

to parallel and parallel to serial Registers, it will be enjoyable to notice how MATLAB do it. Another reason is that AES needs to substitute bytes by bytes and perform some shifts that should be implemented either by block RAMS or Lookup tables (LUTs). In VHDL the designer optimize the timing by controlling the pipelines, especially in the feedback designs like AES, how can MATLAB HDL coder do this? Does it leaves all the work to the synthesizer?

In the following section, we discuss the previous work in the scope of evaluating the MATLAB HDL coder given MATLAB functions as input compared with the VHDL path. Section 3 highlights the VHDL path of designing the AES cipher unit, what are the top-level diagram, the main block diagram, the clock frequency, the critical paths, and the FPGA utilization. The second path is described and discussed with the same headings in Section 4. In section 5 we compare the results that we collected before and analyze the causes behind them. Finally, we reached the conclusion and the recommended research for the future.

2. Related Work

MathWorks has published HDL Coder™ Evaluation Reference Guide [7]. In which they discussed: HDL-supported blocks, Model setup, Design Under Test (DUT) and test bench partitioning, Accessing HDL settings and operations, SIMULINK modeling best practices for HDL designs, clock, sample rate and data flow control, fixed-point and floating-point settings, using a state-flow chart, code Generation tools, HDL Code Advisor, and HDL workflow Advisor. In many fields, they conclude their publications without comparing the default path, i.e. the hardware description languages. In 2015, a rapid prototyping from algorithm to FPGA prototype was studied in a master thesis by the University of Oulu, Department of Electrical Engineering. The thesis studied MathWorks High-Level Synthesis (HLS) workflow usage for rapid prototyping of wireless communication SoC Intellectual Property (IP) [8]. This thesis introduced the design and FPGA prototyping flow of the application-specific integrated circuit (ASIC). It presented targeted for HLS. It also studies MathWorks Hardware Description Language (HDL) generation flow with HDL Coder, This work concentrated on evaluating the usage and benefits of MathWorks HLS workflow targeted for rapid prototyping of System on Chips (SoCs). In 2018, the Evaluating Simulink HDL Coder as a framework for flexible and modular hardware description was published. It was found that the Simulink-generated hardware fell behind in performance, but that there were very few parts of the architecture that could not be replicated. Resets and enables are automatically generated by the software and thus could not easily be made to precisely match the functionality of the reference VHDL design. However, this did not impact the ability of the design to produce correct outputs, and the Simulink-generated code otherwise behaved as desired [9]. The most related work is the one that compares different design alternatives for hardware-in-the-Loop for power converters electronics [10]. While their conclusion was: that Automated MATLAB HDL code is not recommended as it involves more tools than other approaches, and also, it does not reach small latency and area. Many different designs were performed using HDL Coder like [11], [12], [13], [14], and [15].

In this paper, we compare based on the following hypothesis. First, the FPGA board is Cmod S6™ FPGA Board [2]. Regardless to the board and its FPGA size, as long as it is the same for the two paths then we compare the utilization percentage. Second, we chose the input to the MATLAB HDL Coder to be MATLAB function rather than SIMULINK, this is much easier as our target is to have a substitution to a hardware engineer with a software one achieving the same results with some advantages such as decreasing time to market, decreasing testing time and interfacing, decreasing the digital design knowledge and so on. The last hypothesis is that the design under test was chosen to be the encryption part of the advanced encryption standard of course with the required key expansion unit and the controlling.

In the following section, we discuss the VHDL design path. In section 4 the MATLAB HDL Coder design and implementation are discussed. After that in section 5, a comparison analysis between the two paths is emphasized. In section the testing and verification procedure is illustrated. Finally, we conclude the study and give some recommendations.

3. AES DESIGN USING VHDL

The VHDL code was already written within a previous publication [4]. Here we will focus on some important points to summarize the previous design and recall the important comparison axis.

3.1. Top-level and Interface

Figure 1 shows the top level of the unit under test, the AES Encryption module. Here there are thirty-one inputs and outputs pins. In AES 128 we expect a 128-bit key, 128 data input as plain text and 128-bit data output as cypher text. Due to the limited number of IOs in the used FPGA, we implement two serial to parallel registers for the input data and key. We also implemented one parallel to serial register for the output data, ciphertext. The serial to parallel is 8 bit to 128 bit and the parallel to serial is 128 bit to 8 bit.

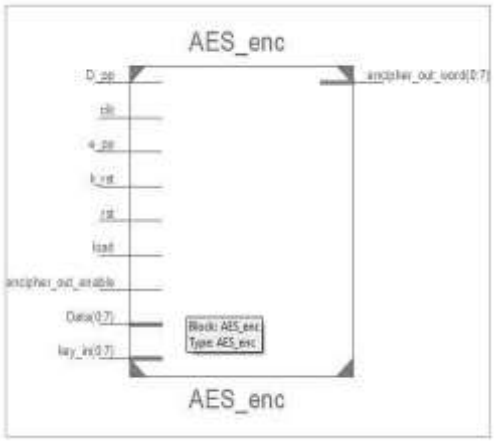

Figure 1 Top Level Symbol of the AES Encryption unit with VHDL implementation

This is illustrated in Figure 2, I1 is the serial to parallel for the cypher input, I2 is serial to parallel for the key input and I3 is parallel to serial for the output. We have seven other input pins one for the clock signal, CLK, the second for the reset signal, rst, and the third for the load signal if it is active I1 and I2 get the data and store it. The fourth one is a reset for the key expansion unit, krst, the fifth one is the output enable for the ciphertext output, and the sixth one and the seventh one enable for the encryption and decryption respectively. The other three components in the block diagram are: U1 is the Cipher unit, U2 is the Controller unit and U3 is the Key Expansion Unit. Here there is an advantage for the digital designer; the components are viewed in the RTL with a clear interconnection and routing.

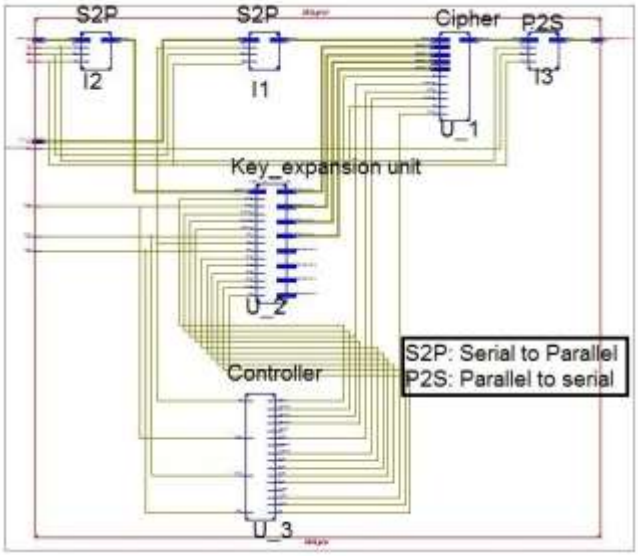

Figure 2: AES Encryption Module Block Diagram

4. AES DESIGN USING MATLAB HDL CODER

Figure 3 shows the top level of the unit under test (AES Encryption module). Like the one implemented by the VHDL, there are 3 eight-bit for key, plain text and ciphertext. In Figure 4, the head of the function of the AES Encryption module is focused on. Here we can see that the inputs are Plaintext, Cipherkey, We (write enable), and reset.

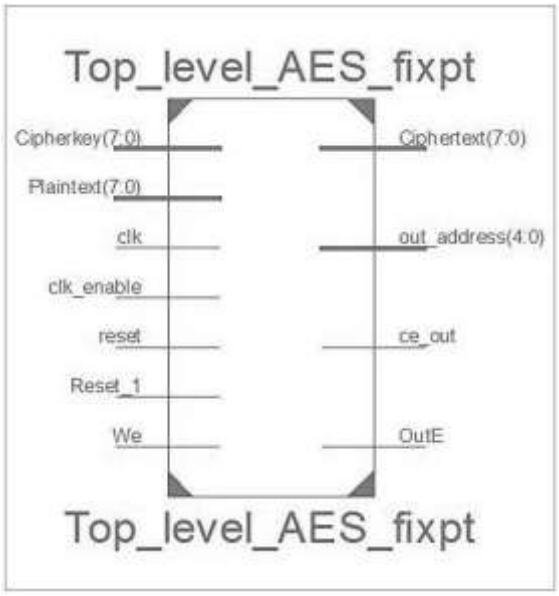

Figure 3: Top Level Symbol of the AES Encryption unit with MATLAB HDL Coder implementation

On the other hand, the outputs are the ciphertext, OutE (output enable to start to receive the output 8 bit by 8 bit) and out address; address to facilitate writing the output in an array. The sum of these input-output pins (IOs) is thirty-two bins the last four pins are added by the MATLAB HDL coder. There are: clock signal (clk), clock enable (clkenable), Reset1, ceout. This means that the developer has no access or control over the clock. This is a very important note; this means that controlling simultaneous action is only done using clock enables. Sometimes this method reduces the actual frequency to the frequency of the clock enables.

```
function [ Ciphertext,OutE,out_address]= Top_level_AES (Plaintext,Cipherkey,We,Reset)
```

Figure 4: MATLAB function illustrates the inputs and the outputs

In Figure 5 level one of the Register-transfer levels (RTL) is illustrated. As we notice, the function is just implemented without any ability to understand its visualization. In addition, by seeing the consumed LUTs (410%), we can understand that every count in for loops is implemented again without reuse. Generally, in FPGAs, we implement a module and control it by a counter. But here every count of the counter has a module that connects to the next module. It is similar to collecting on-shelf modules over a test board. This is more relative to the idea of hardware programming. Besides miss utilizing the area of the FPGA, it increases the propagation delay.

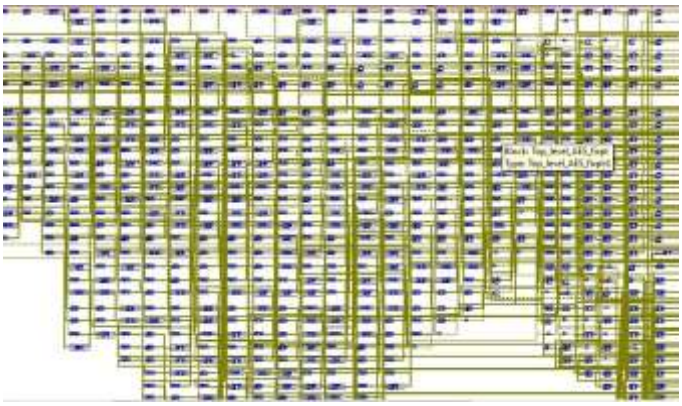

Figure 5: the RTL level 1 of the AES Encryption module implementation using MATLAB HDL Coder

On the other hand, MATLAB HDL Coder facilitates the data exchange between the PC and the FPGA. It's not the designer's responsibility to make this interface because MATLAB is in charge. This makes the importance of the handshaking protocol reduced. As a result, the signal required to do this task is not highlighted.

As a hardware engineer, there were some important questions for example: how do we determine the number of bits in inputs and outputs? How do we store new data in the same variable while it's well known in the software it will be overwritten? The answer was given by MathWorks [1]. They introduce the sentence "persistent" to work like the sentence "signal" in the VHDL code. Figure 6 shows how we could write the parts of serial to parallel and parallel to serial in MATLAB. We must first write the persistent sentence.

```
%define a signal to store the input plain text
persistent Plaintext_i;
% if it is the first time to be defined or active
% low reset was asserted then it will be 16 zeros
if (isempty(Plaintext_i)||Reset==false)
    Plaintext_i = uint8(zeros(BS*BS, 1));%zeros(16,1
end
%define a signal to store the input key
persistent Cipherkey_i;
% if it is the first time to be defined or active
% low reset was asserted then it will be 16 zeros
if( isempty(Cipherkey_i)||Reset==false)
    Cipherkey_i = uint8(zeros(BS*BS, 1));
end
%define a signal to store the output cipher text
persistent ciphertext_i;
% if it is the first time to be defined or active
% low reset was asserted then it will be 16 zeros
if( isempty(ciphertext_i)||Reset==false)
    ciphertext_i = uint8(zeros(BS*BS, 1));
end
%define a signal to store the address write
%for the memories
persistent ramWriteAddr;
% if it is the first time to be defined or active
% low reset was asserted then it will be 1
if (isempty(ramWriteAddr)||Reset==false)
    ramWriteAddr = 1;
end
```

Figure 6: Using the persistent variables

```
%define a signal to store the output enable
persistent OE;
% if it is the first time to be defined or active
% low reset was asserted then it will be false
if (isempty(OE)||Reset==false)
    OE = false;
end
%define a signal to store the output address write
persistent outaddress;
% if it is the first time to be defined or active
% low reset was asserted then it will be zero
if (isempty(outaddress)||Reset==false)
    outaddress = uint8(zeros(1, 1));
end
% Writting to RAM
if (We==true&&Reset==true && OE==false)
    if ramWriteAddr<=16
    % store 8 bits of data in an array in
    % a location detrmined by the address
    %write
    Plaintext_i(ramWriteAddr)=Plaintext;
    % and so for the key
    Cipherkey_i(ramWriteAddr)=Cipherkey;
    ramWriteAddr=ramWriteAddr+1;
    OE==false
    end
```

Figure 7: How to store the data in a serial to parallel and vice versa in MATLAB HDL Coder

As it is well known in software programming, the variable is updated on every line without any memory. So by adding the persistent keyword, the variable is kept memorized without updating till the next call to be reused with the last value, figure 8 illustrates this idea by flow chart.

By knowing the code of serializing the inputs and outputs besides the given HDL coder project [5] anyone can continue the recommended future research.

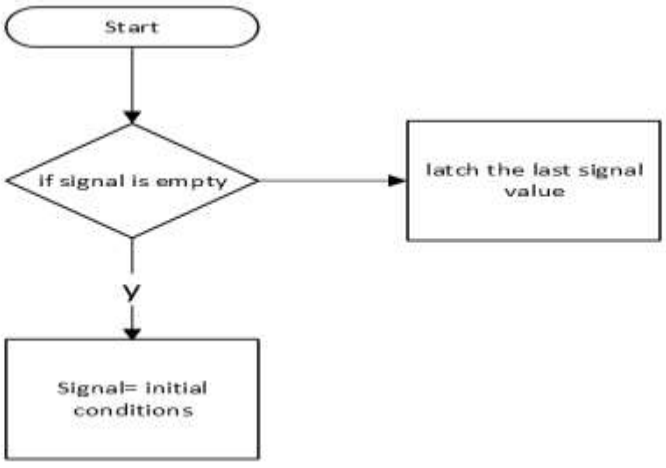

Figure 8: Flow chart for variables defined by “persistent” keyword.

5. TESTING AND VERIFICATION

The VHDL part was already tested and verified before in another publication [4]. In this section, we focus on MATLAB HDL Coder testing and verification because it is the most interesting part of this path. We invoked the MATLAB HDL Coder app and created a new project. It was asked to supply two files: MATLAB function and MATLAB test bench. The required MATLAB function is only the top-level, other functions that this file call should exist in the project directory. In the test bench file, we generate random plain text and cypher key. We call the MATLAB function based on these random inputs. The generated output is decrypted using the same key. The generated decrypted output is compared with the input. If they were identical then the design is working probably and prints "Decrypted plain text matches the original text." Otherwise, it prints "The decrypted plain text does not match the original plain text." As shown in Figure 9.

```
% this is the FPGA output
 Ciphertext_A=Ciphertext_A(2:end)
% this is the calculated decryption for the FPGA output
 Ciphertext_i = mlhdlc_aesd(Ciphertext_A, Cipherkey_A)
 % if the decrypted message is equal to the input plaint2xt then it prints
 % matches
 if(Ciphertext_i == Plaintext_A)
 fprintf('The decrypted plain text matches   the original plain text.\n');
 else
     % otherwise does not match
 fprintf('Decrypted plain text does not match  the original text.\n');
 end
```

Figure 9: MATLAB test bench file

Figure 10 illustrates the code by a flow chart.

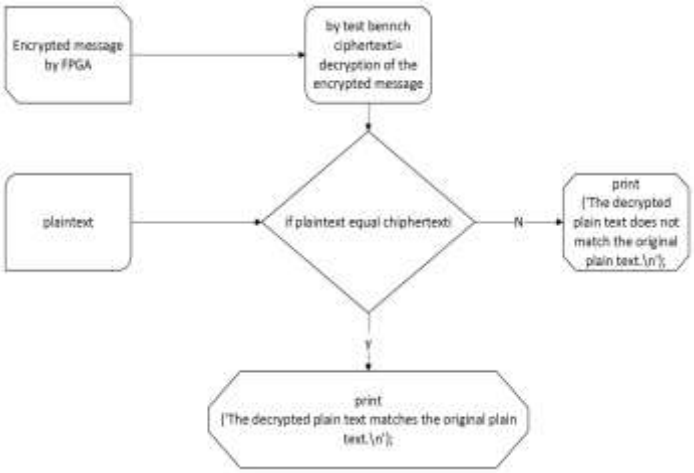

Figure 10: Flow chart illustrating the test bench procedural.

After passing the two files to the MATLAB HDL Coder project, we go through its own workflow as shown in Figure 11.

Figure 11: MATLAB HDL Coder Project

For the workflows, till the verifications step, we chose FPGA in the loop. We created our board specifications based on its manual [2] as shown in Figure 12 and Figure 13. We selected the General-Purpose Input/output (GPIO) for inputs and led for output.

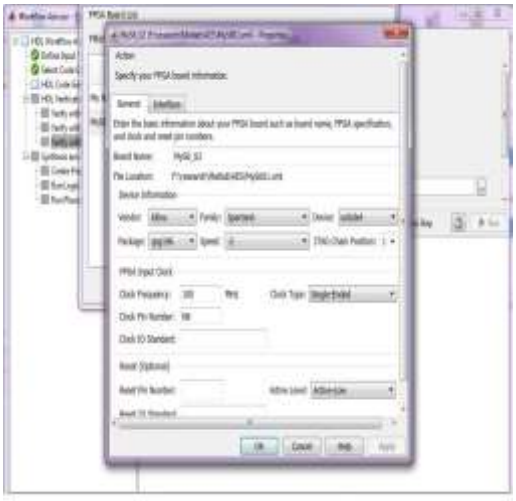

Figure 12: How to Specify the FPGA board Information

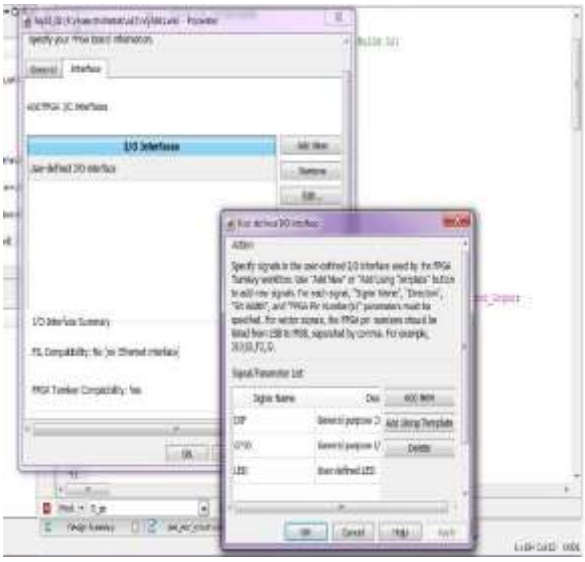

Figure 13: User-defined IO interfaces of Spartan 6 FPGA board

6. COMPARISON BETWEEN MATLAB HDL CODER AND VHDL IMPLEMENTATION

Utilization summary tables for target device xc6slx4-2cpg196 FPGA are illustrated in Table 1 for VHDL-based implementation and Table 2 for MATLAB HDL Coder-based implementation.

Table 1: Estimated Utilization Summary for VHDL-based implementation of AES encryption module

Number of	Used	Available	Utilization
Slice Registers	2395	4800	49%
Slice Look Up Tables (LUTs)	1886	2400	78%
Fully used LUT-FF pairs	1224	3057	40%
Bonded input-output blocks (IOBs)	31	106	29%
Block RAM/First In First Out (FIFO)	3	12	25%

Table 2 Estimated Utilization Summary for MATLAB HDL Coder-based implementation to AES encryption module

Number of	Used	Available	Utilization
Slice Registers	410	4800	8%
Slice Look Up Tables (LUTs)	9851	2400	410%
Fully used LUT-FF pairs	176	3057	6%
Bonded input-output blocks (IOBs)	36	106	33%

We can easily compare the two tables and then analyze the difference. As the bonded inputs/ outputs are considered the interface of the module then it will be a good starting point for comparison. We have 31 IOBs versus 36 IOBs. The number of Slice Registers in the VHDL path is 49% versus 8% in MATLAB coder which is much higher because of using the pipeline method. But the Number of Slice LUTs is 40% in VHDL while it exceeds the maximum in MATLAB which is 410 per cent and this was discussed in the previous section. This indicates an area problem; this is illustrated by figure 14.

According to timing, the Clock frequency in VHDL was Minimum period: 8.314ns (Maximum Frequency: 120.279MHz) while in MATLAB Coder Minimum period: 41.063ns (Maximum Frequency: 24.353MHz) and this is much important point that the speed using VHDL is almost five times faster than that using MATLAB Coder.

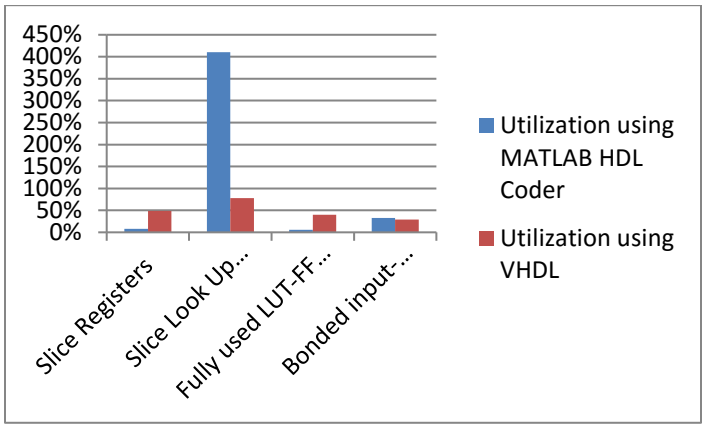

Figure 14: A column chart compares the FPGA utilization using VHDL and MATLAB HDL coder.

7. Conclusion and recommendations

The results of our studies indicate that the cost of speed and area utilization can't be paid for the ease of design and implementation. We recommend MATLAB Coder for fast proof of an idea but not for competition in a digital market.

8. FUTURE WORK

We recommend a research project that writes code to make the combinatorial circuits pipelined. This will both utilize the area and increase the speed.

On the other hand, this work to be completed should test all passes like Simulink as input. It should compare different designs under test with different FPGA boards.

References

[1] MathWorks "HDL Coder™ User's Guide©", [Online] available: https://www.mathworks.com/help/hdlcoder/index.html

[2] Cmod S6™ FPGA Board Reference Manual, Revised June 13, 2017. 1300 Henley CourtPullman, WA 99163 509.334.6306 [Online]. Available: www.digilentinc.com

[3] FIPS 197, "Advanced Encryption Standard (AES)", Nvlpubs.NIST.gov, [Online]. Available: https://nvlpubs.nist.gov › nist.PDF

[4] Ghada Farouk Naiem, Salwa Elramly, Bahaa Eldeen Hasan, Kaled Shehata," An efficient implementation of CBC mode Rijndael AES on an FPGA", IEEE 25th National Radio Science Conference (IEEE NRSC 2008), Tanta, D09, pp. 18, March 18-20, 2008. [Online]. Available: https://ieeexplore.ieee.org/document/454237

[5] Advanced Encryption Standard (AES)-HDL CODER Example [Online]. Available: https://www.mathworks.com/matlabcentral/mlc-downloads/downloads/submissions/50098/versions/3/previews/mlhdlc_tutorial_comms_aes.m/index.html. [Accessed: 30- Apr- 2022].

[6] David Hill (2022).” Advanced Encryption Standard (AES) 128,192, 256 “(https://www.mathworks.com/MATLABcentral/fileexchange/73412-advanced-encryption-standard-aes-128-192-256), MATLAB Central File Exchange. Retrieved January 19, 2022.

[7] HDL CoderTM Evaluation Reference Guide (R2020a-R2020b) © Copyright 2015-2020 by MathWorks, Inc.

[8] Oriol Font-Bach, Antonio Pascual-Iserte, Nikolaos Bartzoudis, and David López Bueno “MATLAB as a Design and Verification Tool for the Hardware Prototyping of Wireless Communication Systems”, [Online]. Available: https://www.researchgate.net/publication/228075597

[9] Valerie Youngmi Sarge, S.B., “Evaluating Simulink HDL Coder as a Framework for Flexible and Modular Hardware Description”, Master of Engineering in Electrical Engineering and Computer Science, 2018 Massachusetts of Technology. All rights reserved.

[10] Zamiri, E. Sanchez, A. Yushkova, M. Martínez-García, M.S.; de Castro, “A. Comparison of Different Design Alternatives for Hardware in- the-Loop of Power Converters. Electronics”, 2021, https://doi.org/10.3390/electronics10080926.

[11] Järviluoma J., “Rapid Prototyping from Algorithm to FPGA Prototype”, the University of Oulu, Department of Electrical Engineering, Degree Program in Electrical Engineering. Master’s Thesis, 59 p, (2015).

[12] Richard Carbone," A flexible hardware architecture for 2-D discrete wavelet transform: design and FPGA implementation” [Online]. Available: https://scholarworks.rit.edu/

[13] Po-Cheng Wu and Liang-Gee Chen, Fellow," An Efficient Architecture for Two-Dimensional Discrete Wavelet Transform", IEEE transactions on circuits and systems for video technology, vol. 11, no. 4, April 2001

[14] Donald G. Bailey, "Image Processing Using FPGAs", Department of Mechanical and Electrical Engineering, School of Food and Advanced Technology, Massey University, Palmerstone North 4442, New Zealand; D.G.Bailey@massey.ac.nz, Received: 6 May 2019; Accepted: 7 May 2019; Published: 10 May 2019

[15] Mohamed Ali Hajjaji ,1 Mohamed Gafsi, Abdessalem Ben Abdelali ,1 and AbdellatifMtibaa, "FPGA Implementation of Digital Images Watermarking System Based on Discrete Haar Wavelet Transform", Received 14 August 2018; Revised 11 November 2018; Accepted 9 December 2018; Published 3 January 2019

Acknowledgment

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

Bibliography

GHADA ELSAYED 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 then she shifted to an academic career, in which she worked for more than five years for MSA University then she travelled as a visiting researcher to Japan, Kyushu Institute of Technology. After that, she continued on the same path at MTI University. In 2014, Ghada published a book about securing satellites control link. She also published many scientific papers.

SOMAYA ISMAIL KAYED is an Associate Professor and head of Electrical Dept. (Electronics, communication, computer and control Engineering) at Obour Higher Institute for Engineering and technology. She graduated in 1987 from Ain Shams University with a BSc. in Electronics and Communications department, with a 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 in her class. In 1997, she finished her Masters of Science (MSc.) in the same department. Afterwards, in 2000, she was 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 Obour Higher Institute for Engineering and technology, she has published in local and international conferences many scientific papers as well as multiple books related to her research interests (Analog and digital VLSI design, current conveyor, Nano electronics).

At the same time, her passion for development is not limited to academia. She is also a volunteer member of an NGO. On one side, these diverse experiences enriched not only her teamwork skills but also her leadership competencies.