Palestine Polytechnic University

Department of
Computer Systems Engineering

Implementing a hybrid Compression Encryption Technique for Securing WMSN on FPGA

Team Members:

Tasneem Zahdeh	161011@ppu.edu.ps
Lama Jawabreh	161003@ppu.edu.ps

Supervisor:
Dr.Mousa Farajallah

Tue, Jan 26th, 2021

Abstract

Wireless Multimedia Sensor Network (WMSN) is an environment for transferring images, audios, and videos. However, it has many restrictions and limitations that affect the security level as well as the required power consumption. This project is to implement the protection of sensitive data transmission through wireless networks with high performance and low power consumption. The implementation is based on Field-Programmable Gate Array (FPGA) using the Verilog HDL hardware language for the hardware programming and C/C++ high-level languages for the software programming of the FPGA. The project implementation is proposed for the solution that was presented by Hraini.[1] In the hardware implementation, RGB images are included in addition to the grayscale images in the software solution. Images are compressed by Set Partitioning in Hierarchic Tree (SPIHT) algorithm[2] and encrypted selectively during the compression process (i.e. jointly). The importance of this project lies in being useful in making a secure environment for transferring images through the most unsecured networks which are wireless networks such as Bluetooth as was applied in this project. The expected results are a hardware implementation that corresponds to the software solution in Hraini's thesis. Also, the system is expected to present secure transmission for RGB and grayscale images through wireless networks in addition to perform the proposed security solution in a way that guarantees a minimum use of power consumption and high performance. Also, it is expected to satisfy the data confidentiality and privacy the software solution aimed to apply besides having a very compact output bitstream and a constant bitrate in both parties (i.e., sender party and receiver party) as has been achieved in the software solution.

Table of Contents

List of Tables

List of Figures

Chapter 1
Introduction

1.1 Overview of the Project

WMSNs contain small devices that collect data by their sensors as shown in Fig1.1. They work by sensing multimedia data like audio, videos, and images. However, these networks have few restrictions regarding power consumption, and they also have security issues. So, to solve these problems, a software solution has been proposed in Hraini's thesis[1] that guarantees a secure transmission for images.

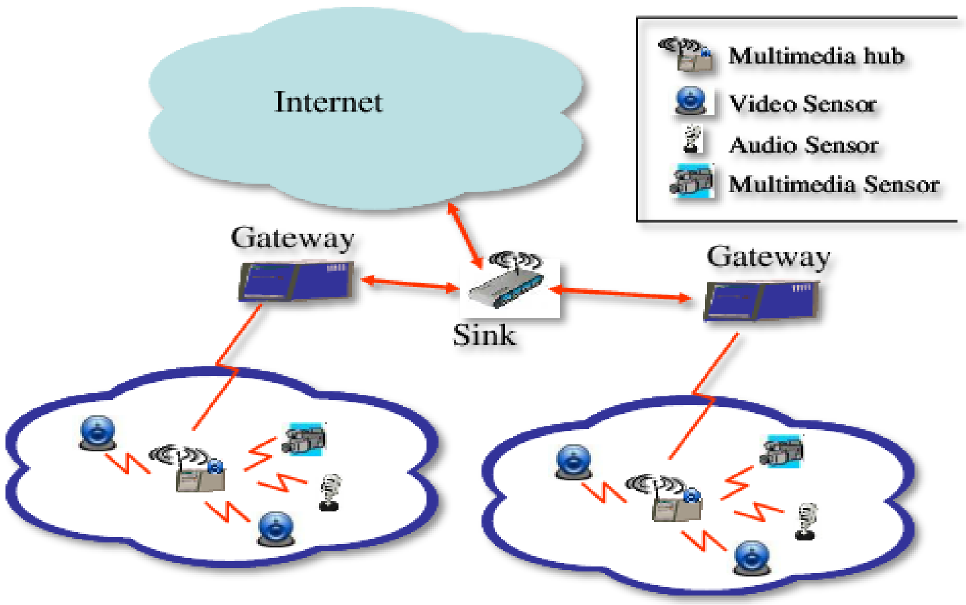


Fig.1.1: Wireless Multimedia Sensor Networks

In this project, the hardware implementation for that solution has been determined to be on FPGA so to handle the power consumption issue by selecting specifically Cora Z7 FPGA. The project deals with any image in the BMP extension and of any size. It can also process RGB and grayscale images.

1.2 Motivation

Implemented applications in hardware circuits are the trend nowadays for the easiness of reprogrammable them and for their adavantage of detecting and fixing errors with nearly no cost at all. However, the hardware circuits face two major problems which are are power and performance. Therefore, it is preferred to use the type of processor with excellent performance and low power consumption. FPGAs stand out with these characteristics and so have been chosen to be the hardware tool for the project implementation.

To the best of our knowledge, graduation projects in Palestine Polytechnic University (PPU) of computer engineering major have been dedicated to Arduino and Raspberry Pi in general and were away from FPGA specifically. Thus, this approach is the first of its kind adding to that the challengeable uniqueness of FPGAs themselves because of being programmed with Hardware Description Languages (HDLs) for making the hardware definition files of the FPGA board. As for the software programming of the FPGA processor, C and C++ are the languages used.

1.3 Importance

Converting the proposed software solution to secure WMSNs into real-time and hardware solution is important for business companies to be able to make integration of the implemented solution.

The project is useful in implementing a secure environment for transferring images through the most unsecured networks (i.e., wireless networks such as Bluetooth).

1.4 Objectives

The goals of this project are:

- Securing the transition of RGB and grayscale images in WMSNs which don't protect the transmitted data from attacks and that results in losing its integrity and confidentiality.
- Achieving a high performance by using the excellent SPHT for compression and Hraini's embedded encryption technique with just a 0.2914% overhead and a constant bitrate.
- Having a minimum power consumption by using the Cora Z7 FPGA hardware component in implementing the software solution. Cora Z7 can either takes power input from 4.5V to 5.5V external source or through USB.

1.5 Description of the Project

Implementing the SPIHT algorithm with the solution presented in Hraini's thesis[1] by encrypting selectively the pixels with the most amount of information simultaneously with compressing the image.

The implementation is carried out using FPGA based on the Verilog hardware language for hardware programming and C/C++ for software programming. The hardware programming programs the FPGA by the definitions file of the FPGA board (i.e., HDF extension file) in addition to a bitstream that is created based on the FPGA block design which is translated to Verilog codes (i.e., V extension files). The software programming inserts the C/C++ code of the SPIHT and selective encryption into the FPGA processor.

Images are not collected by sensors; they are stored in the computer locally after being brought from the Internet. FPGA gets the DWT images from the MATLAB program on the sender side. The output of the FPGA after it processes the image is a binary file that has the image header and bitstream. The binary file gets transferred through WMSN to the receiver who gives it as an input to the FPGA. After the FPGA processes the bitstream, an inverse DWT image is generated and taken as an input to the receiver MATLAB program so to reconstruct the image to its original version

Although Hraini's thesis covers grayscale images only, the hardware implementation deals with both RGB and grayscale images of different sizes.

1.6 Design Options

Many available options are suitable to be applied in the system, and each option has its advantages and disadvantages. Here is a comparison between options of the same category with pointing out to the selected choice for this project.

FPGA Hardware Programming Language:

- VHDL: It is an abbreviation of VHSIC Hardware Description Language. VHSIC is yet another abbreviation of Very High-Speed Integrated Circuits.
  - VHDL advantages:

It has roots in Ada programming language in the syntax and concept. It includes a lot of features and thus has a great number of uses. It is a strongly typed language, [3] excellently typed, deterministic, and more natural language than Verilog HDL, and its designs are self-documenting. VHDL can also define more errors that are detected but left undefined in Verilog HDL. Moreover, it doesn’t have any unambiguity in its semantic as its keywords are expressive and meaningful. Furthermore, it provides flexible and smoothing switching between tools.

  - VHDL disadvantages:

    It hasn’t been learned in university so there is no assurance of self-learning it fully, deeply, and correctly in such a short time. Also, there is no guarantee that it would become practiced and professionally typed even after learning it. Moreover, it needs more programming when converting from one data type to another.

- Verilog HDL (Has been chosen):
  - Verilog HDL advantages:

    Its roots refer to an early HDL, Hilo, and that makes it a more low-level hardware language than VHDL. It also shares a strong language like C its syntax and structure, and the advanced version of C, which is C++, has been learned in university. It is a deterministic language as VHDL, and all data types are predefined so there is no extra coding when converting from one data type to another as in VHDL. Moreover, it is a bit-level representation which means it is very concise and can be looked at in-depth. Furthermore, it has been taught in university and thus the knowledge and experience exist.
  - Verilog HDL disadvantages:

    It is a weakly typed language,[3] and it doesn’t have the flexibility needed.

FPGA Software Programming Language:

- C Language:
  - C language advantages:

    For embedded systems such as FPGA, C language is a more suitable choice since it requires less runtime support. It is also faster in compiling since it has more low-language similarity than C++. Moreover, it can be easily interfaced with other languages compared to C++ which has a high level of complexity in its libraries.

    - C language disadvantages:
      It is harder to understand as it is a bit far from being English-like language. Also, it doesn’t have inline functions and doesn’t support functions with default arguments.

- C++ Language (Has been chosen):
    - C++ language advantages:
      It can work with the C language in the same program code. It also improves performance and reserves memory slots efficiently. Moreover, it has a very large set of in-built functions and user-defined functions.
    - C++ language disadvantages:
      It isn’t secure, and it works on a specific set of platforms. Also, it can become very complex in large programs.

Host Device:

- FPGA: FPGA has slots for one SD card, Ethernet, and power connectivity. Those three components allow FPGA to work independently of a host computer. If there is no need for the Internet, the Ethernet can be excluded. However, there would still be a need for an LCD screen to display output on.
- USB Host (Has been chosen): USB can work as a host by connecting any PC with an FPGA. The UART communication protocol happens by transmitting and receiving the transferred bytes to and from the FPGA’s two channels. Each channel has a light indicator that lights in the transmission process. The left light, from the USB viewpoint, is for transferring the data from FPGA to the other side of the USB connection which is the PC. The right light is for transferring the data to the FPGA from the other side of the USB connection which is again the PC.

Serial Communication Software:

- Putty:
    - Putty advantages:
      It is the most common software that is used for serial communication and has a higher review than TeraTerm.
    - Putty disadvantages:

It lacks the simplicity than TeraTerm and can come across printing unknown characters without a clear reason.[4]

- TeraTerm (Has been chosen):
    - TeraTerm advantages:

      It works best for beginners and serves their needs well by providing the necessary properties and characteristics.
    - TeraTerm disadvantages:

      It is not as highly configurable as Putty and connecting remotely makes a real issue as it sometimes works and other times not under the same environmental conditions.

Hardware Component:

- Arduino: It can be used in digital image processing. It is an easy kit for use, but it is too simple to deal with the complexity of image processing. Also, it is not powerful enough to deal with images. Therefore, high resolution and great image quality are not the results to get from Arduino.
- Raspberry pi: It is characterized by its low-cost and ability to do digital image processing by providing a simple interface so to reduce the complexity of real-time applications.[6] However, Raspberry pi can't be used for real-time applications.
- FPGA (Has been chosen): It is often used for image processing and that is because of FPGA structure which gives it an advantage for doing spatial and temporal parallelism. Besides, FPGA size can contain image sizes when they grow larger in processing due to their nature.[6]

1.7 Overview of the Rest of the project

The analysis of the problem statement, definition, list of requirements, and expected results are provided in chapter 2. A short description of the parts used in the system, why they are chosen, specification, and design constraints are presented in chapter 3. Information about the software developed to drive the system is shown in chapter 4. Implementation description, issues, and challenges are included in chapter 5. Validation and analysis of the project results are in chapter 6, and finally, the conclusion of the project as well as future work.

Chapter 2
Problem Statement

2.1 Problem Analysis

Transferring information between different parties is a daily process in cyberspace, and there is a need for securing the transferred content especially when the confidentiality and privacy of the data are important. This means that the only legal destination can read and understand the transferred information.

Also, the integrity of the data should be guaranteed when they arrive at their destination. Therefore, any modification to the data must be identified. To achieve that, selective encryption alongside compression by the SPIHT algorithm is presented by Hraini in [1] to get the security level needed with high performance and minimum power consumption.

2.2 Definition

This project is proposed to implement the presented solution in the thesis[1] so to convert the theoretical solution to a real-life application.

SPIHT is the compression algorithm used, and it contains four stages which are initialization, sorting, refinement, and quantization. Selective encryption is the best choice to be used when processing time is the most important criteria in addition to many limitations on power and memory. The selective encryption is achieved jointly with the compression in the second stage of the SPIHT (i.e., the sorting stage).

This implementation is performed on an FPGA board connected to a PC via a USB cable. The PC is mainly used for displaying the images before and after processing while the USB is for transferring and receiving the binary file. The images are jointly compressed and encrypted, and the images are jointly decompressed and decrypted.

2.3 List of Requirements

The system requirements in the project can be summarized as the following:

1- The system must be able to achieve the confidentiality and integrity of the images as it was achieved in the software solution in [1]. Therefore, they don't get exposed to passive and active attacks and thus preserving the secrecy and reliability of data.

2- The average time of the joint compression-encryption with SPIHT and Hraini's encryption technique is approximately the same as the time of compression alone with SPIHT. This should be reflected in the hardware implementation.

3- Getting a duplicate image, to the bare eye, of the original one in the destination by using decompression by the SPIHT which is lossy compression. An example of lossy compression is shown in Fig.2.1.

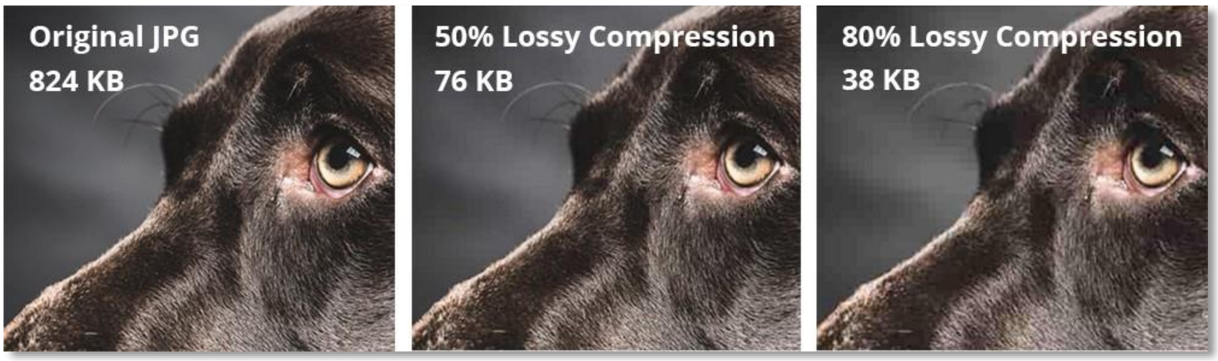


Fig.2.1: Different rates of the lossy compression in comparison to the original image

2.4 Expected Results

The system is expected to have some certain results, and they are:

1- The hardware implementation is expected to correspond to the software solution.

2- The system is expected to present the secure transmission proposed in [1] for RGB and grayscale images through wireless networks.

3- The system is expected to perform the proposed security solution in a way that guarantees a minimum use of power consumption and high performance.

4- The hardware implementation is expected to satisfy the data confidentiality and privacy the software solution aimed to apply.

5- The hardware implementation is expected to have a very compact output bitstream and constant bitrate as has been achieved in the software solution.

Chapter 3
Background

3.1 Theoretical Background and Literature Review

For the past few years, security problems make a big issue concerning data transmission especially through wireless networks. Thus, the software solution approach focuses mostly on securing data through WMSNs.

The main aspect of the proposed solution is the joint encryption of the images while getting processed with the SPIHT algorithm before getting transmitted. The SPIHT algorithm was developed from the strengths of the powerful compression algorithm — the Embedded Zero tree Wavelet (EZW) which is created based on Discrete Wavelet Transform (DWT). A popular technique, that was proposed by Shapiro to achieve compression, has been used to encode DWT. DWT states that any image can be divided into four subbands as shown in Fig.3.1.

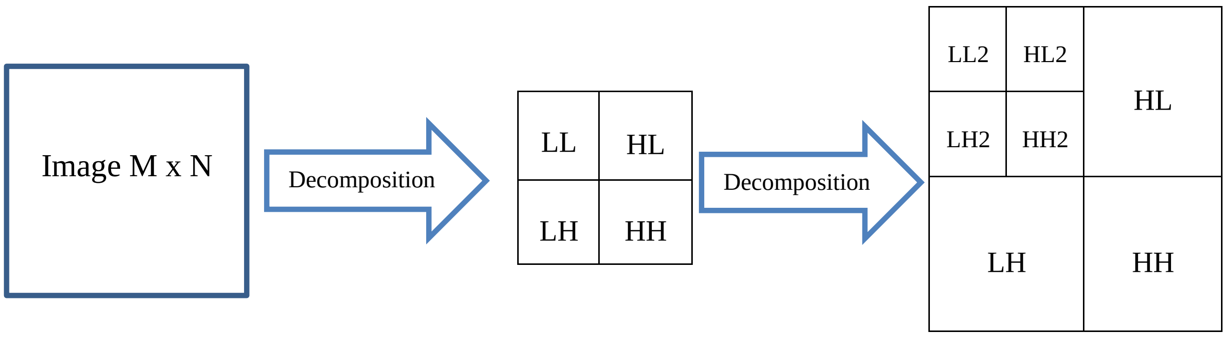


Fig.3.1: Any image can be divided into four subbands

In each LL subband, divisions take place and continue until a tree is formed. The order of the four coefficients from the topmost subband are Low Low (LL), High Low (HL), Low High (LH), and High High (HH). They all share the same resolution.

The prime purpose of the tree is to form the coefficients and generate the bitstream in prioritized order such as the most significant coefficients are placed in the LL subband of the first level (it is LL2 in Fig.3.1), less significant coefficients are placed in HL subband of the higher levels, and less significant coefficients than the ones in HL are placed on LH subband of the same level and so on. The least significant coefficients are placed in the HH subband of the last level which would be the highest. In other words, the most important information of the image lay in the subbands of the lower levels starting from LL. Pixels that contain information of little importance lay in the furthest subbands such as HL, LH, and HH of the last levels.

The software solution in the thesis[1] that was proposed to satisfy the secure transmission and data confidentiality in WMSNs is shown in Fig.3.2.

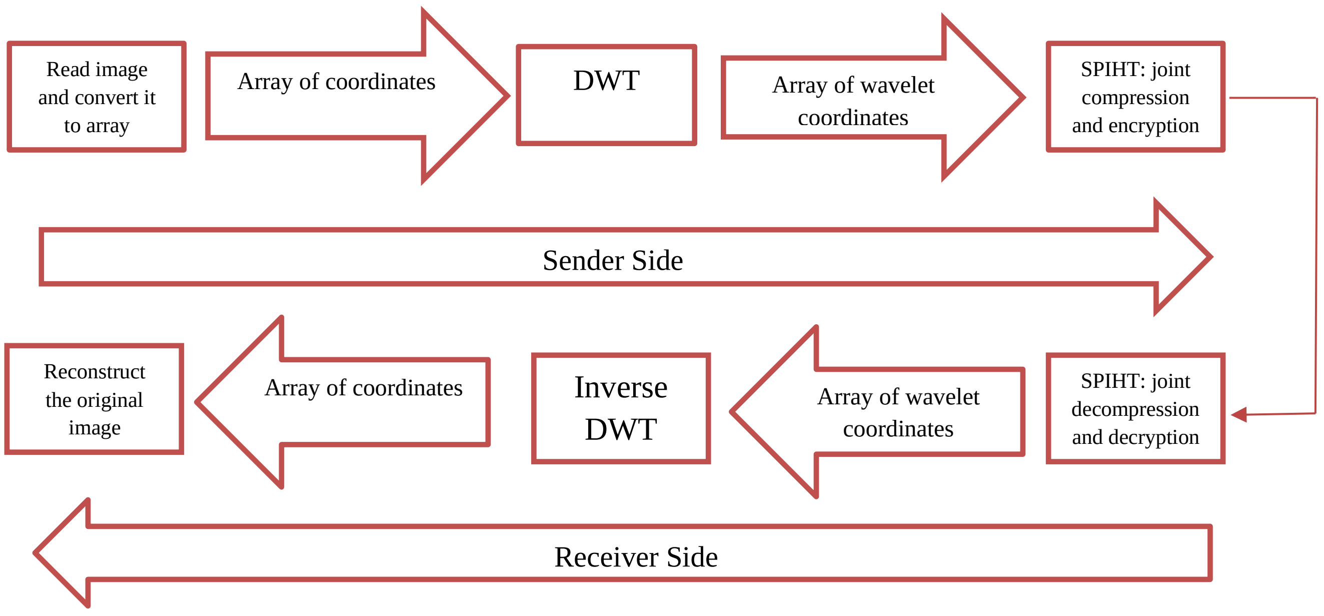

Read image and convert it to array
Array of coordinates
DWT
Array of wavelet coordinates
SPIHT: joint compression and encryption
Sender Side
Reconstruct the original image
Array of coordinates
Inverse DWT
Array of wavelet coordinates
SPIHT: joint decompression and decryption
Receiver Side

Fig.3.2: The solution proposed in Hraini's thesis

The SPIHT is used for compression in four stages for any image on the sender side and decompression in the same four stages but vise versa on the receiver side. A very important notation is that the encoder and decoder are identical, and a few lists are consistently updated. They are:

- LIP: is an abbreviation of List of Insignificant Pixels and is the first list.
- LSP: is an abbreviation of List of Significant Pixels and is the second list.
- LIS: is an abbreviation of List of Insignificant Sets and is the last list.

The coordinates for the first and second lists are (i, j) which represents individual pixels. However, the last list takes sets as its entry.

The key step added to the SPIHT is the selective encryption that happens jointly with the compression. The encryption occurs in the second stage of the SPIHT which is the sorting stage. This is shown in Fig.3.3.

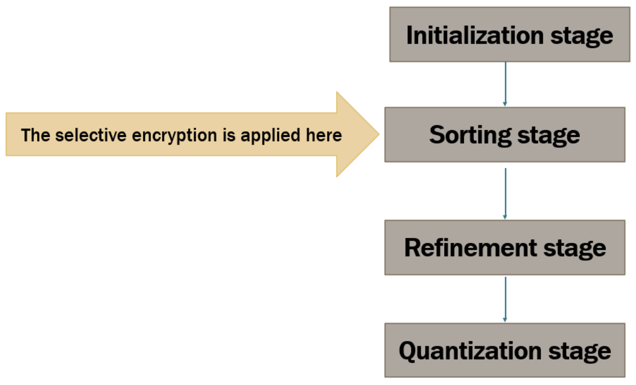


Fig.3.3: The encryption place in the SPIHT algorithm

The encryption is applied with a simple XOR equation. The XORing happens with an array of secret keys which are generated with the generator in [7]. However, the encryption can be applied with any secure Pseudo-Random Number Generator (PRNG). The encryption process is explained in detail in Chapter 6 — section 6.4.3.1.

Several projects and researches of image processing have been made using FPGA. Research about FPGA-based image processor architecture for WMSN in [8] has focused on achieving high-speed processing with minimum power consumption, and its approach is to compress and transfer images through WMSN to examine WMSN issues such as power management and system synchronization. Another research about FPGA architecture for object extraction in WMSN in [9]. It is for detecting and extracting captured images in real-time. The system is optimized to achieve high-speed processing with minimum power consumption. Both reasearches share the same goals as in this project.

3.2 System Hardware Components

The hardware components that are needed for the project are explained here with a short description of each one. The integration between them is followed immediately.

The hardware components:

1- FPGA board:

FPGAs are semiconductor devices that deal with configurable logic blocks (CLBs) in the form of matrices, and they are all connected by programmable interconnects. They can also be programmed to be used for a wide range of available purposes. The choice has been set on Cora Z7 in Fig.3.4 which is a Zynq-7000 SingleCore for ARM/FPGA[10] SoC Development.

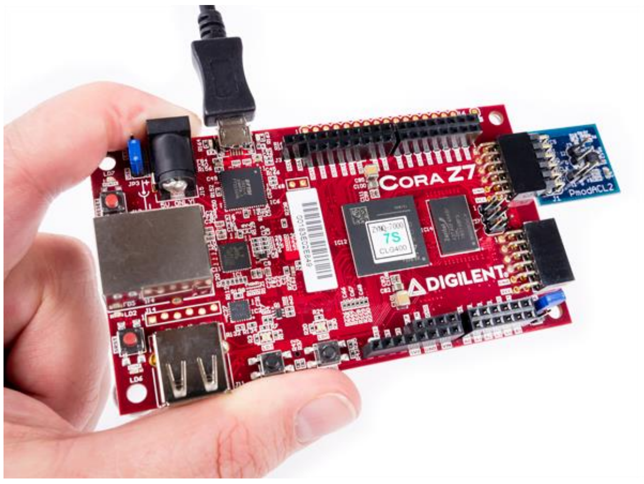


Fig.3.4: Cora Z7 Zynq-7000 SingleCore for ARM/FPGA

The platform's design was done by Zynq-7000, and the board architecture is from ARM. It contains a single-core with a speed that equals 667 MHz with a Cortex-A9 processor with a Xilinx 7-series. The Cora Z7 makes this FPGA a great platform for developing software as it has a wide range of hardware interfaces such as the 1 Gbps Ethernet PHY, general-purpose input/output pins, and analog-to-digital converters. It acts as a one-component containing a large set of solutions. The board has its independent SD card slot, Ethernet slot, and power slot.

Cora Z7 is the one used in the project, not Zybo Z7[11] nor ZedBoard[12] for a few reasons, and they are:

- Zybo Z7 in Fig.3.5 has a larger memory than Cora Z7, but Cora Z7's memory is enough for the required processing in this project besides being cheaper.
- Cora Z7 and ZedBoard in Fig.3.6 have the same memory size and both have Ethernet slots but Cora Z7 is much cheaper than ZedBoard.

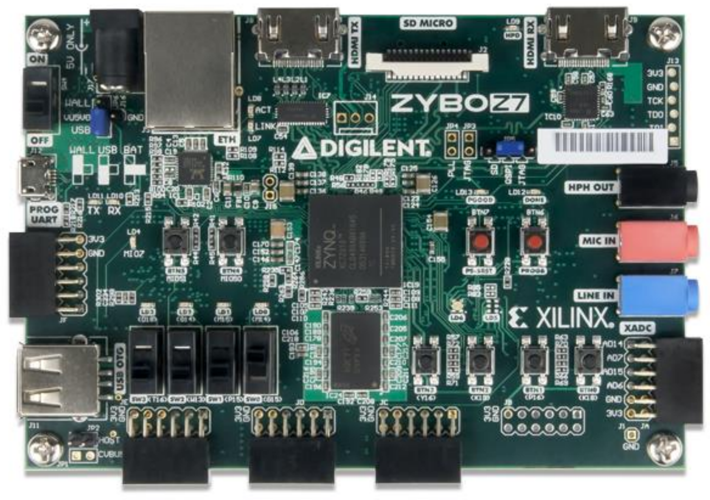


Fig.3.5: Zybo Z7 — Zynq-7000 ARM/FPGA SoC Development Board

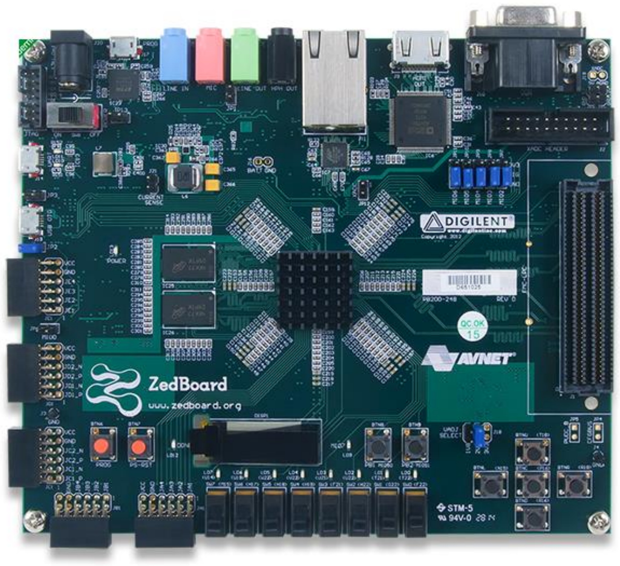


Fig.3.6: ZedBoard Zynq — 7000 ARM/FPGA SoC Development Board

2- Personal Computer:

The project needs one Personal Computer (PC) for each party. The user enters the necessary inputs such as an image for the sender and a bitstream for the receiver. The user gets the outputs on the PC such as a bitstream for the sender and a reconstructed image for the receiver. Also, for determining the number of secret keys and the length of the sequence by the keyboard. The recommended operating system is Windows 10.

3- USB cable:

A mobile USB cable in Fig.3.7 is needed to connect the FPGA board with the PC. The USB will work as a host and establishes the UART protocol.

Fig.3.7: A Mobile USB Cable

4- Micro SD card and Micro SD card USB Adapter:

To insert the micro SD card which is the non-volatile memory of the FPGA into the PC, there must be a third-party to connect those two, and it is the adapter. The SD memory card is needed to store and retrieve data to and from the FPGA.

3.4 Specification and Design Constraints

The processor of the FPGA is programmed with a bitstream that is different from the Hraini's bitstream. The FPGA's bitstream contains the used logic gates defined by the Verilog language and constraint files for each Intellectual Property (IP) that exist in the hardware block design.

3.4.1 System Specifications

Certain features must be presented in the system to convey the wanted results correctly and precisely. Such features are:

1- The system must be able to deal with all images whether they are RGB or grayscale and of different sizes or unequal axes in the BMP extension. The accepted minimum size is 2x2 px because of the DWT function allowable range in MATLAB and because of SPIHT since an image of 1x1 px would have zero levels.

2- Two forms of the images must be presented on the PC screen. They are the image in its original state and the reconstructed image after getting decompressed and decrypted in the destination. The MATLAB programs do the displaying task.

3- The hardware code of the system is written in Verilog and embedded in the FPGA processor core initially before programming the FPGA again with the software code written in C/C++.

3.4.2 Design Constraints

There exist some limitations regarding the size of the hardware components so they should be taken into consideration when using the system. They are:

- The FPGA RAM is 512 MB. It is Double Data Rate 3 (DDR3) — Synchronous Dynamic Random-Access Memory (SDRAM).
- The FPGA 32-GB SD memory card is a non-volatile memory.
- FPGA can take input power that ranges between 4.5 V to 5.5 V from an external source and get connected to a PC wirelessly or take power from a PC through a USB cable. In this project, the USB approach has been chosen to get with, and that creates a limitation on the possible distance between the FPGA and PC. That is, the distance can't be more than the USB cable length which is 1 meter.

Chapter 4
System Design

4.1 Detailed Description of the System

The following steps describe the system in detail:

1- The sender on his PC chooses an image from any external source such as from the Internet. An example is shown in Fig.4.1.

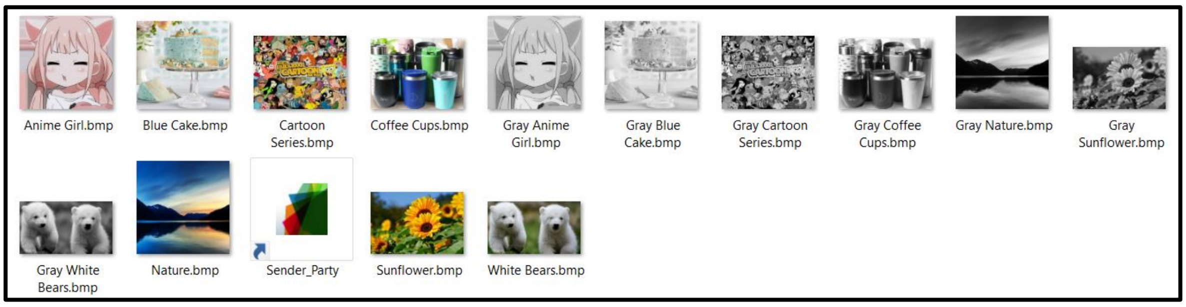


Fig.4.1: A set of random images from the Internet

2- The sender runs the MATLAB program "Sender Party" on his PC as shown in Fig.4.2. The program shortcut can detect any image that resides in its path.

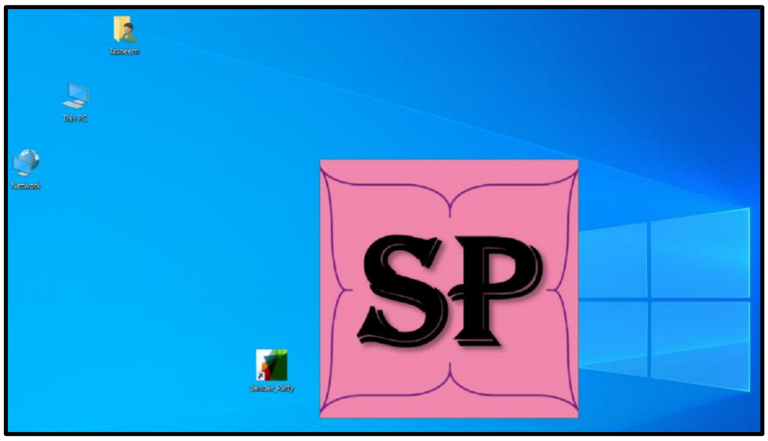
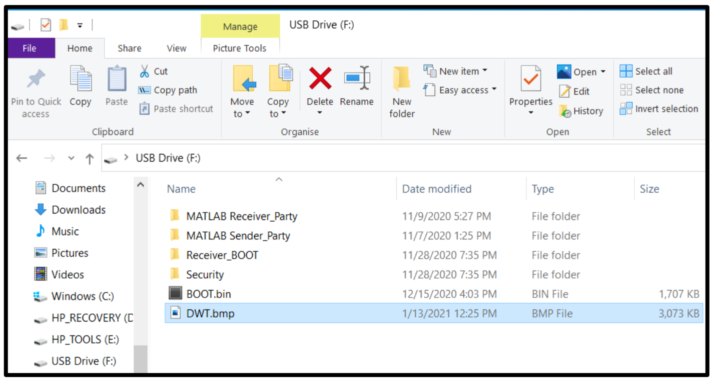


Fig.4.2: Running the MATLAB program "Sender Party"

3- The MATLAB program converts the image to its DWT equivalent as shown in Fig.4.3. The image gets stretched in its width four times because of the function imwrite() in MATLAB which rearranges the image pixels in the range 0-255. So, to preserve the DWT image exact values, each pixel gets 4 bytes in the new DWT image instead of one byte.



Name	Date modified	Type	Size
MATLAB Receiver_Party	11/9/2020 5:27 PM	File folder	
MATLAB Sender_Party	11/7/2020 1:25 PM	File folder	
Receiver_BOOT	11/28/2020 7:35 PM	File folder	
Security	11/28/2020 7:35 PM	File folder	
BOOT.bin	12/15/2020 4:03 PM	BIN File	1,707 KB
DWT.bmp	1/13/2021 12:25 PM	BMP File	3,073 KB

Fig.4.3: The DWT image in the micro SD card

4- The MATLAB program sends the DWT as a BMP image to the SD memory card which would be inserted via a USB adapter into the sender’s PC as shown in Fig.4.4. The SD memory card will act as a hard drive after being formatted once with the FAT32 system on Windows OS.

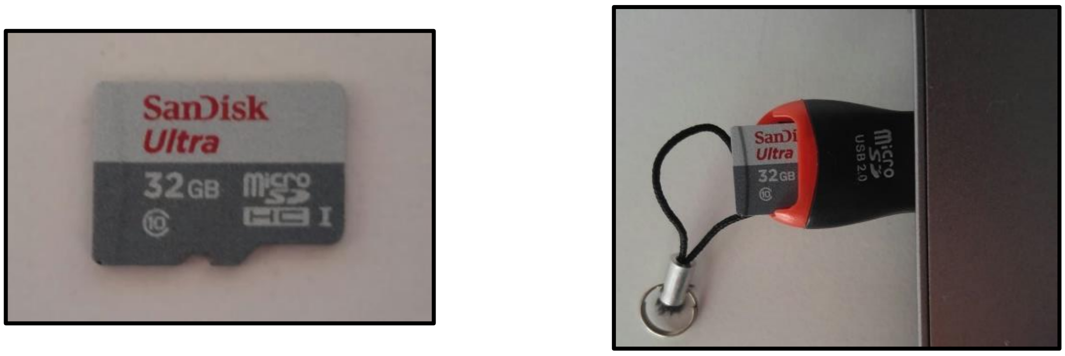


Fig.4.4: The micro SD card inserted inside the USB adapter

5- The MATLAB program displays the chosen image in its original form in a new window on the sender’s PC as shown in Fig.4.5.

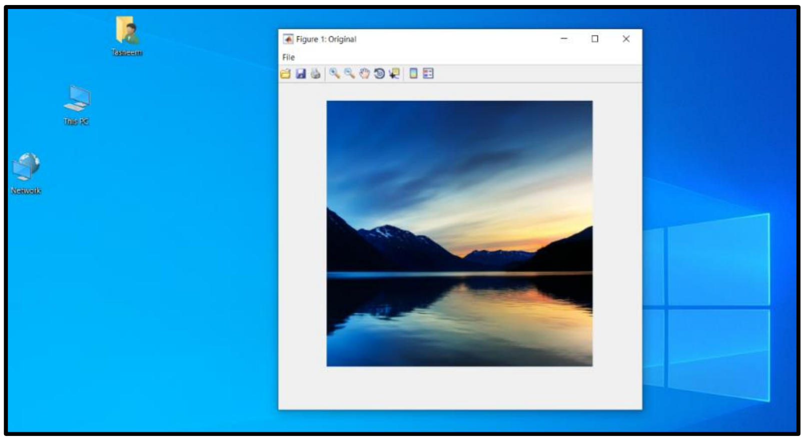


Fig.4.5: The image is displayed in its original form

6- The sender takes the SD memory card from his PC and inserts it into the FPGA onboard SD Card slot as shown in Fig.4.6. The FPGA would be connected to the PC through a USB cable which provides the power for the FPGA from the PC and works as a host. Through the USB, a UART protocol gets established and thus transfer data through FPGA two channels; one of them is for sending and the other is for receiving.

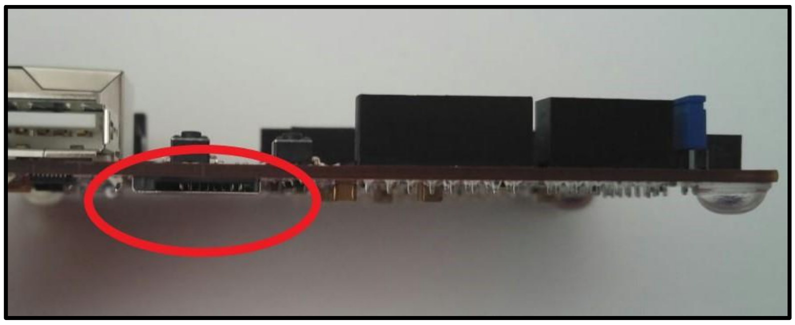


Fig.4.6: The slot in FPGA for the micro SD memory card

7- The sender clicks on the reset button on the FPGA for the FPGA to read the inserted memory as shown in Fig.4.7, then clicks on the soft reset button to run the FPGA program.

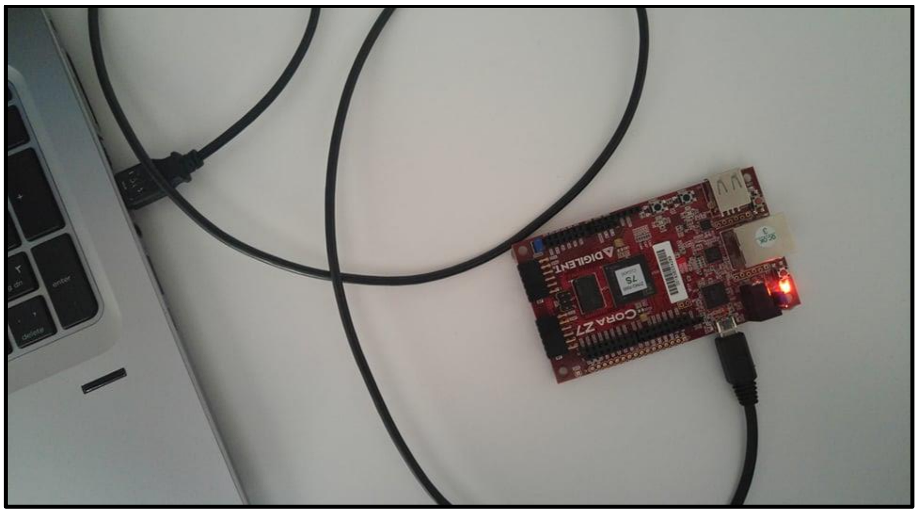


Fig.4.7: The FPGA is connected to the PC through a USB cable

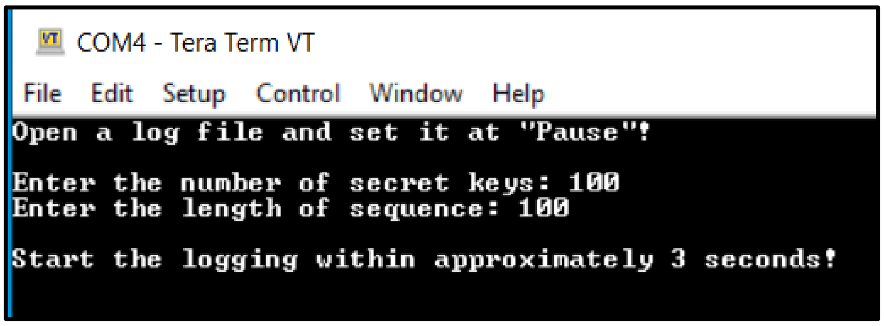
8- The FPGA program runs and outputs several messages on TeraTerm, which is a virtual hardware screen, on the sender's PC as shown in Fig.4.8:

- 1st message prompts the sender to open a binary file, which takes zeros and ones only, by the logging option that is available in TeraTerm, and set the logging on pause.
- 2nd message prompts the sender to enter the number of secret keys that he would like to use as long as they are less than the number of secret key files

stored in the SD memory card. However, the sender can enter an enormous number of keys since the memory is 32 GB.

- 3rd message prompts the sender to enter the sequence length.
- 4th message prompts the sender to start the logging in 3 seconds as with the header of the image followed by the bitstream will get printed.

The TeraTerm is the platform in which the user gives inputs to it and takes outputs from it. Since FPGA can't connect to the user's PC directly, the USB cable and TeraTerm do the job in establishing a connection between the two (i.e., the user's PC and FPGA board).



```
Open a log file and set it at "Pause"!

Enter the number of secret keys: 100
Enter the length of sequence: 100

Start the logging within approximately 3 seconds!
```

Fig.4.8: The messages get printed on TeraTerm when FPGA runs

9- The DWT image gets compressed and encrypted inside the FPGA as shown in Fig.4.9.

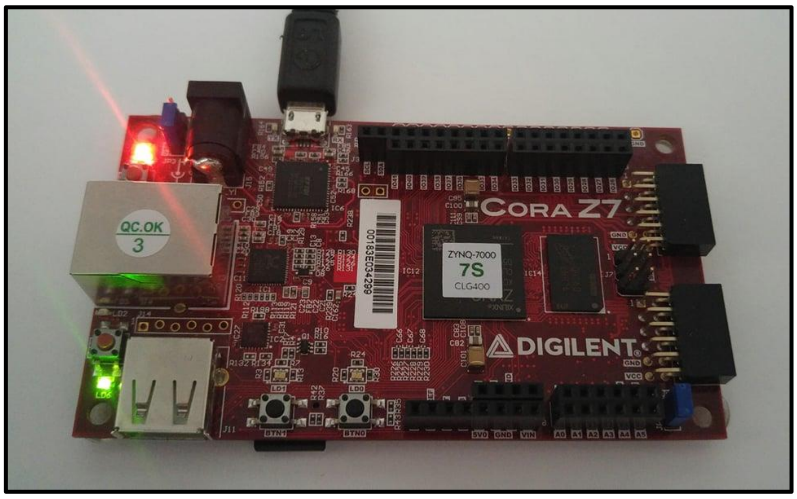
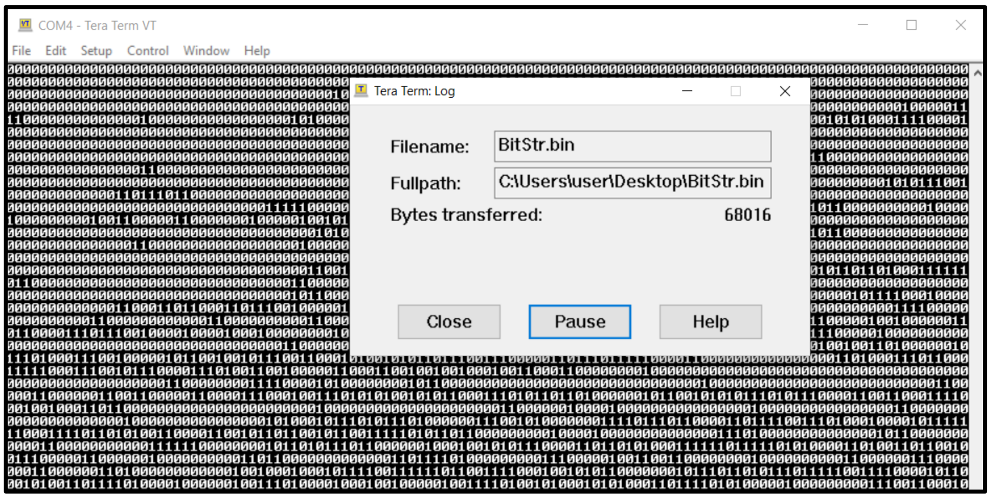


Fig.4.9: The FPGA is running and processing the user's image

10- FPGA outputs the image header and bitstream on TeraTerm in the form of zeros and ones as shown in Fig.4.10, and FPGA prints the time calculations for generating the keys array and the time calculations for compressing and encrypting the image in a text file with the name 'Time'. It would have the processing time, average generation time, mean computation time, sample rate, bit rate, the number of clock cycles, and the time consumed in printing the header and bitstream on the terminal.



```
Filename:            BitStr.bin
Fullpath:            C:\Users\user\Desktop\BitStr.bin
Bytes transferred:   68016
```

Fig.4.10: The header and bitstream are printed on TeraTerm

11- The header and bitstream are logged into the binary file chosen in the logging step as shown in Fig.4.11.

C:\Users\user\Desktop\BitStr.bin - Sublime Text 1.4 (UNREGISTERED)
File Edit Selection Find View Tools Project Preferences Help
BitStr.bin
Line 1, Column 1
Plain Text
Windows

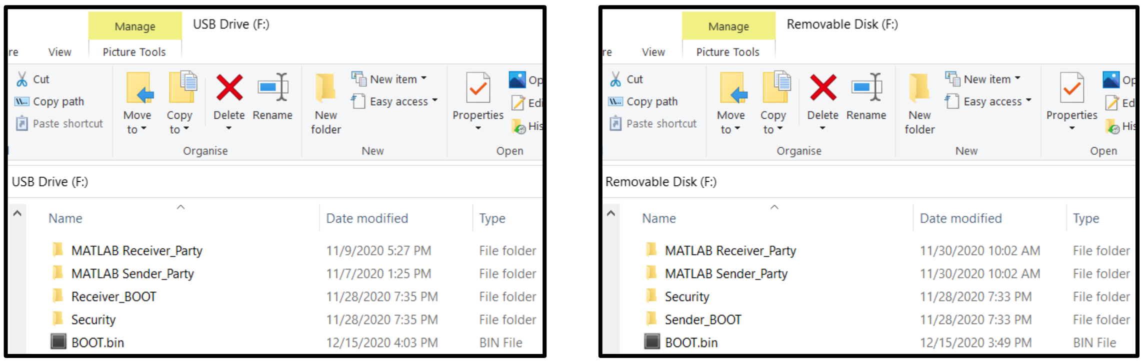
Fig.4.11: The output is stored in the binary file

12- The binary file gets transferred through the Bluetooth in Fig.4.12 from the sender's PC to the desired destination which is the receiver's PC.

Bluetooth™

Fig.4.12: The Bluetooth icon

13- The receiver uses his FPGA board and inserts his SD memory card into the FPGA onboard SD Card slot, then clicks on the reset button on the FPGA. The memory card has the same content in both parties except for the boot image which automatically runs the compression-encryption program in the FPGA on the sender side and the decompression-decryption program on the receiver side. However, both bootable images exist in both memories, and the user can choose between them. The memories' content is shown in Fig.4.13.



USB Drive (F:)

Name	Date modified	Type
MATLAB Receiver_Party	11/9/2020 5:27 PM	File folder
MATLAB Sender_Party	11/7/2020 1:25 PM	File folder
Receiver_BOOT	11/28/2020 7:35 PM	File folder
Security	11/28/2020 7:35 PM	File folder
BOOT.bin	12/15/2020 4:03 PM	BIN File

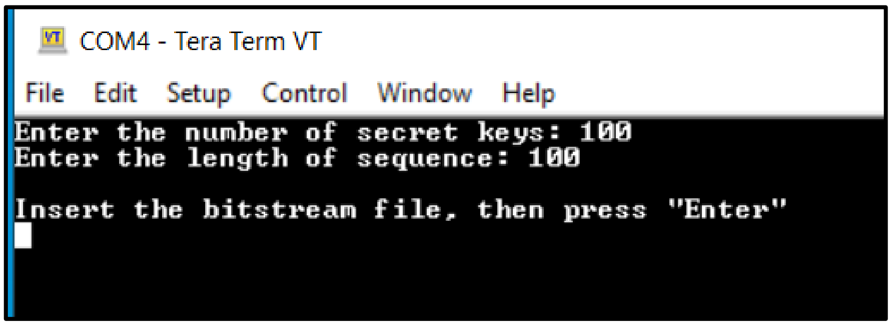
Removable Disk (F:)

Name	Date modified	Type
MATLAB Receiver_Party	11/30/2020 10:02 AM	File folder
MATLAB Sender_Party	11/30/2020 10:02 AM	File folder
Security	11/28/2020 7:33 PM	File folder
Sender_BOOT	11/28/2020 7:33 PM	File folder
BOOT.bin	12/15/2020 3:49 PM	BIN File

Fig.4.13: The SD memory card in the sender and receiver

14- FPGA program runs and outputs several messages on TeraTerm on the receiver's PC as shown in Fig.4.14:

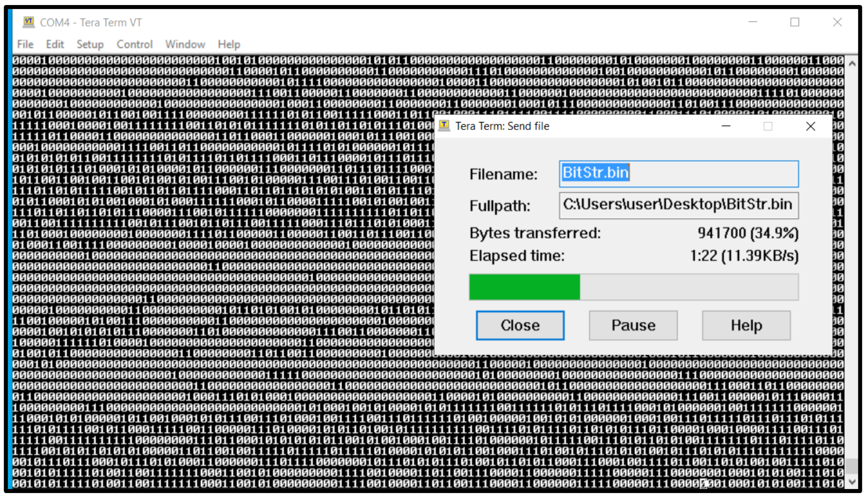
- 1st message prompts the receiver to enter the number of secret keys that are used.
- 2nd message prompts the receiver to enter the sequence length.
- 3rd message prompts the receiver to send the received binary file from the PC desktop to TeraTerm as shown in Fig.4.15.



```
Enter the number of secret keys: 100
Enter the length of sequence: 100

Insert the bitstream file, then press "Enter"
```

Fig.4.14: The messages get printed on TeraTerm when FPGA runs

The inputs for the first and second message must be identical to the sender's inputs.



Filename: BitStr.bin
Fullpath: C:\Users\user\Desktop\BitStr.bin
Bytes transferred: 941700 (34.9%)
Elapsed time: 1:22 (11.39KB/s)

Fig.4.15: The transferring process for the header and bitstream

15- The bitstream gets decompressed and decrypted in the FPGA, and FPGA stores the inverse DWT output in its SD memory card that is inserted inside it. This is shown in Fig.4.16. The FPGA also prints the time calculations for generating the keys array and the time calculations for compressing and encrypting the image in a text file with the name ‘Time’. It would have the processing time, average generation time, mean computation time, sample rate, bit rate, and the number of clock cycles.

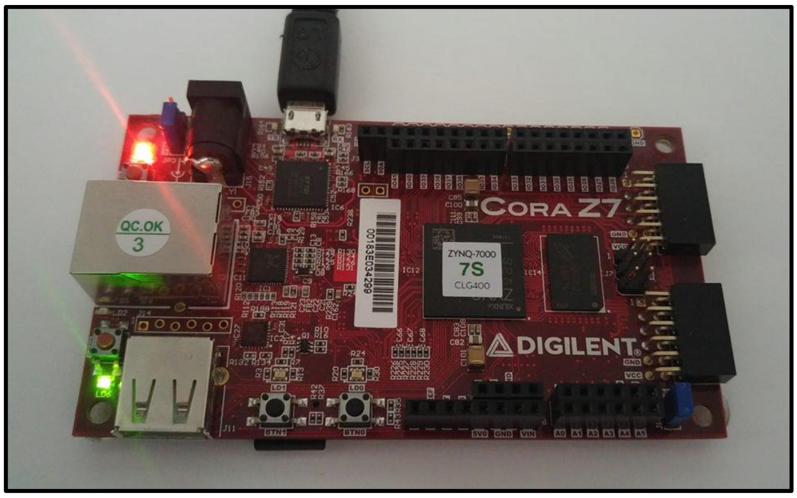
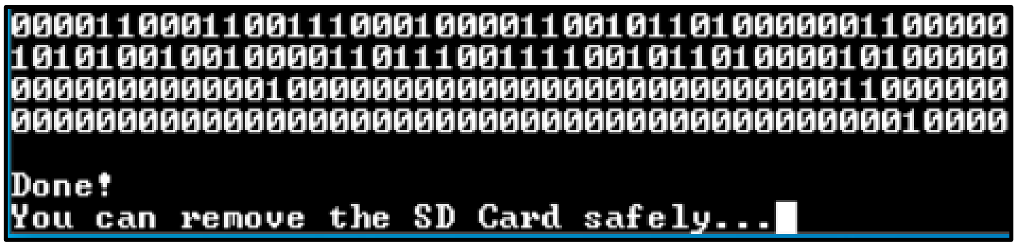


Fig.4.16: The FPGA is running and processing the image’s bitstream

16- The receiver takes the SD memory card from the FPGA after the FPGA finishes the processing by printing a message on TeraTerm as shown in Fig.4.17. Then the receiver inserts the SD card into the PC via a USB adapter.



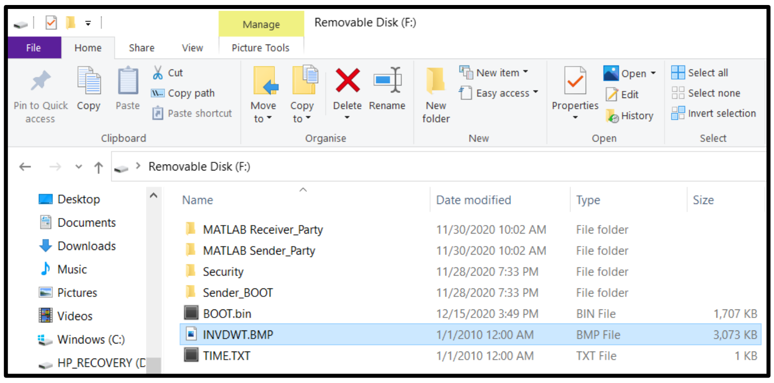
```
00001100011001110001000011001011010000001100000
10101001001000011011100111100101101000010100000
00000000000010000000000000000000000000001100000 0
00000000000000000000000000000000000000000010000

Done!
You can remove the SD Card safely...
```

Fig.4.17: A message is printed on TeraTerm after FPGA finishes processing

17- The receiver runs the MATLAB program “Receiver Party” on his PC as shown in Fig.4.18.

Fig.4.18: Running the MATLAB program “Receiver Party”

18- The MATLAB program receives the inverse DWT BMP image from the SD memory card as shown in Fig.4.19.



Removable Disk (F:)

Name	Date modified	Type	Size
MATLAB Receiver_Party	11/30/2020 10:02 AM	File folder	
MATLAB Sender_Party	11/30/2020 10:02 AM	File folder	
Security	11/28/2020 7:33 PM	File folder	
Sender_BOOT	11/28/2020 7:33 PM	File folder	
BOOT.bin	12/15/2020 3:49 PM	BIN File	1,707 KB
INVDWT.BMP	1/1/2010 12:00 AM	BMP File	3,073 KB
TIME.TXT	1/1/2010 12:00 AM	TXT File	1 KB

Fig.4.19: The inverse DWT image in the micro SD card

19- The MATLAB program reconstructs the original image from its inverse DWT.

20- The MATLAB program displays the reconstructed image in a new window on the receiver's PC and sends a copy of it in the BMP extension to the receiver’s desktop. This is shown in Fig.4.20.

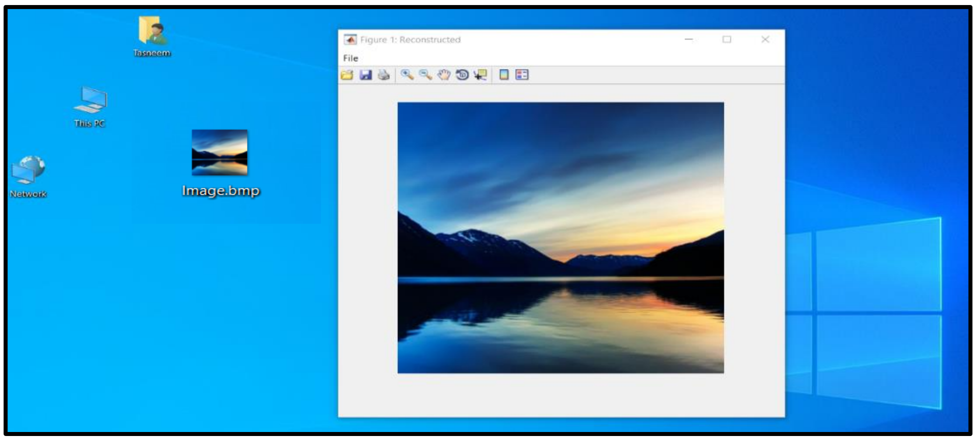


Fig.4.20: The reconstructed image is displayed and sent to the user's desktop

4.2 Block Design

Creating a block design, which is shown in Fig.4.21, is necessary for the FPGA to run. It is the hardware programming part of the FPGA board used. The tool provided in Vivado software allows the user to create a block design where all the IPs that would be used in the software programming (i.e., the C/C++ code) are inserted and connected. The block design is a user-friendly interface to create the equivalent Verilog files where the logic gates would be defined and wired in detail. The Verilog files pass a Hardware Definition File (HDF) wrapping so Vivado can deal with it as one package and in the way it understands. Constraint files are very important to be attached either manually by importing the files from the FPGA manufaturing company to Vivado or automatically by checking the "Apply Board Preset" button in "Zynq Processing System 7" IP properties in Vivado.

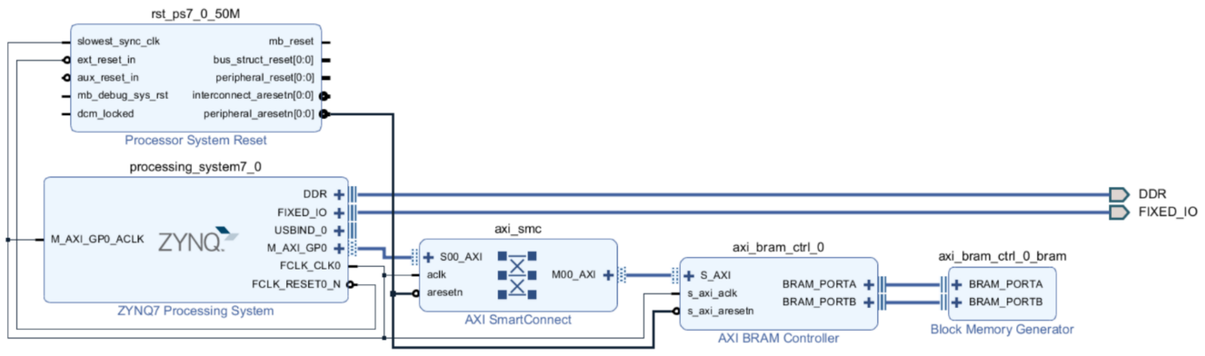


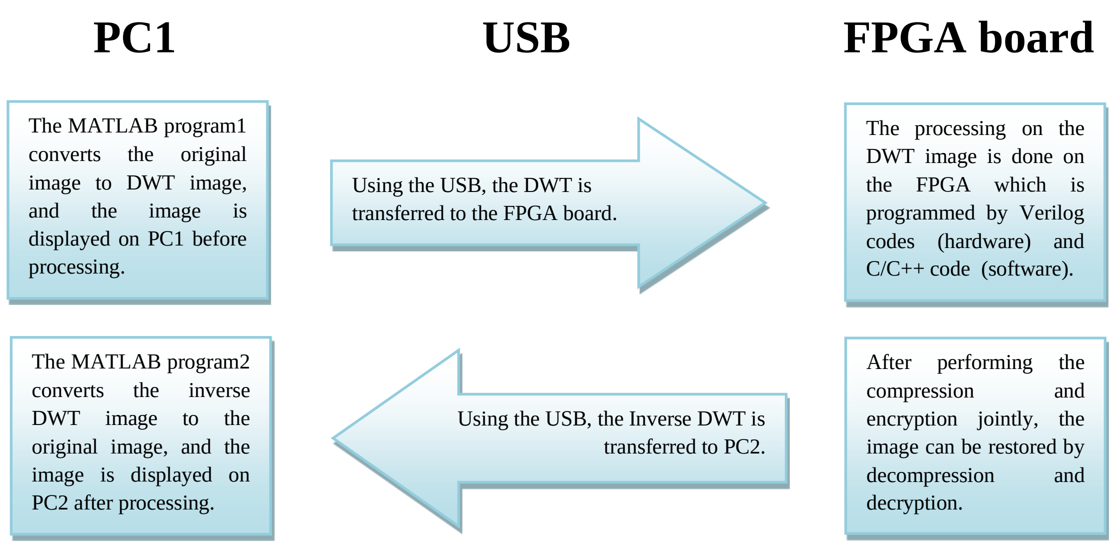
Fig.4.21: FPGA Block Design with the needed IPs

The HDF wrapping gets simulated, synthesized, and implemented in order. If it passes the three stages, then a bitstream is generated based on the Verilog files prescribed. The design bitstream alongside the HDF file are exported to the software development tool which is the

Software Development Kit (SDK). In the SDK, the C/C++ codes are written and run in the FPGA after programming it with the exported bitstream.

Notice that there are two bitstreams in this project, a bitstream for the hardware programming of the FPGA, and another bitstream for the user's image after getting compressed and encrypted in the FPGA.

4.3 Block Diagram


PC1
USB
FPGA board
The MATLAB program1 converts the original image to DWT image, and the image is displayed on PC1 before processing.
Using the USB, the DWT is transferred to the FPGA board.
The processing on the DWT image is done on the FPGA which is programmed by Verilog codes (hardware) and C/C++ code (software).
The MATLAB program2 converts the inverse DWT image to the original image, and the image is displayed on PC2 after processing.
Using the USB, the Inverse DWT is transferred to PC2.
After performing the compression and encryption jointly, the image can be restored by decompression and decryption.

Chapter 5
Software

5.1 Description of the Software

Xilinx SDK 2017.2 is the software tool used for writing the C/C++ code of this system. It can be opened directly or through Vivado, which is another software tool but for writing the hardware Verilog code of the FPGA. In this system, the software code has been written with C and C++ together. SDK can compile C and C++ libraries as any other compiler. It uses FPGA libraries better than C and C++ libraries. FPGA libraries are added to the project for use based on the IPs chosen in Vivado first. Therefore, it is best to make the FPGA libraries the first choice when there are two in-built functions for the same purpose.

SDK and Vivado work as one unit, but SDK can’t work without the HDF file which is made in Vivado after generating the design bitstream successfully. It also needs the design bitstream to make the C/C++ code work in FPGA by programming the FPGA processor with it each time the C/C++ code runs on FPGA.

SDK gives the user serval choices to debug and run the C/C++ code such as launching it on hardware (i.e., Hardware Debugger) and locally as a C/C++ application. Moreover, SDK can be connected via a USB cable with a terminal emulator or use the SDK terminal inside the SDK to take user-inputs and print outputs. However, the SDK terminal is just limited to taking user-inputs via a keyboard and printing outputs on it; it can’t, for example, log output or take local files like terminal emulators.

The user can deal directly with the DDRAM3 memory and other RAMs available in the FPGA. There are two options for dumping and restoring memory content in binary file format, and they work as a connector between FPGA and the local PC used since FPGA can’t see local user files even if they were added to the path of the project itself. Therefore, another option is the micro SD memory card which can send and receive files to and from the FPGA if it is inserted into its slot in the board. However, some FPGAs don’t have an SD card slot. Instead, they have a steady non-volatile memory on their boards themselves. Cora Z7 follows the SD memory card approach.

5.2 Flowchart

The system flowchart is shown in Fig.5.1 in Appendix A.

5.3 Pseudocode

The pseudocodes are shown in Fig.5.2, Fig.5.3, Fig.5.4, and Fig.5.5 in Appendix B.

5.4 MATLAB Padding

Padding is applied on images of unequal sizes in rows and columns and is applied on images that aren't in the allowable range which is 2 to the power of any positive integer. What makes padding an important step to do is because the DWT function used in MATLAB which doesn't accept any image of unequal sizes and not in the range. Also, the SPIHT doesn't deal with images except for the same number of rows and columns.

There are two padding techniques in the project's MATLAB two programs. The first technique is applied on all the images except black images (i.e., its pixels values are zeros) which gets the second padding technique. The techniques are:

1- Zero-padding:

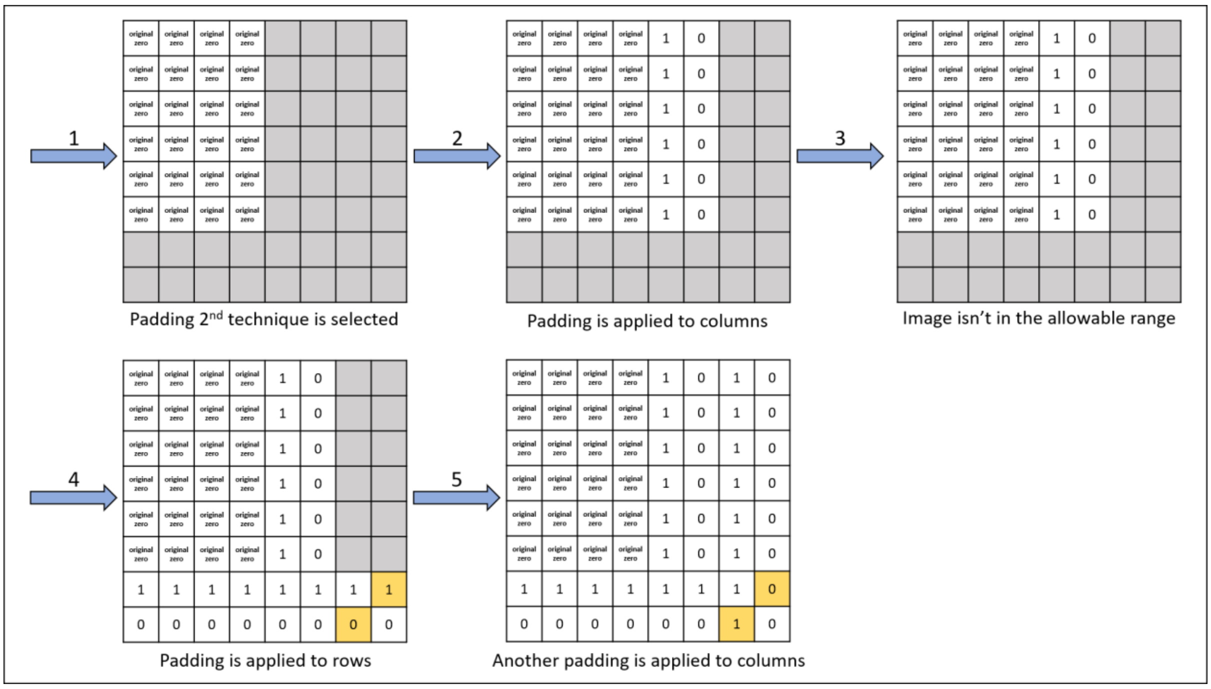
   This has been chosen instead of the NIST padding which is much better and secure because after the image gets processed in the FPGA, some bits interfere in the pixels of where the zero-padding resides, and they are all ones. Thus, a ratio of 87% is the best proportion for checking the number of zeros compared to the ones for the same row or column. The interfered bits exist because the padding gets processed as the image original pixels.

   NIST padding is not beneficial here since it adds unnecessary weight to the total size of the image. The bit of the NIST padding will not indicate the start of the padding as should do since there would be other bits of the same value in the padding.

   One possible issue that might arise is to have identical image paddings for two different images.

2- Ones and zeros padding:

   This method is applied for any image of zero pixels such as black images. It is about starting the row and column of the padding with ones. The rows get padded first, and if the columns need padding then the zero-padding is applied even on the ones generated in the columns because of the rows padding previously. This is illustrated in Fig.5.6.



1
Padding 2nd technique is selected
2
Padding is applied to columns
3
Image isn't in the allowable range
4
Padding is applied to rows
5
Another padding is applied to columns
original zero

Fig.5.6: Ones and Zeros Padding technique

The problem of the NIST padding arises when the bit has a high probability of getting removed after the image gets processed in the FPGA since the SPIHT is a lossy compression.

Also, zero-padding won't work as it will be a pure zero-padding and thus can't be differentiated from the original zeros of the image.

The disadvantage in this technique is to have one row or column or one row and column padding. The line of ones that precede the zero-padding won't have a place in the image; therefore, it will become zero-padding and thus won't be differentiated and removed on the receiver side. Increasing the padding to the next power of 2 is not a solution since it will add a great unnecessary weight compared to restore the image with one additional row or column or one row and column of zero pixels.

5.4.1 Padding Test Cases

The cases for the tested images of the padding are:

1) The number of rows and columns is identical, and they are not in the range. (Test succeded)

2) The number of rows is less than the columns, and the columns are in the range. (Test succeded)
3) The number of columns is less than the rows, and the rows are in the range. (Test succeded)
4) The number of rows and columns is not identical, and they are not in the range. (Test succeded)

The meaning of the range here is the allowable size for the DWT functions in the MATLAB programs, and it is any positive number to the power 2.

5.5 Header Overhead

Since the header of the images is transferred with the bitstream, its overhead has been calculated in Table 5.1 for Lena images.

Image size	Image type	Transferred bytes	Bitstream bytes	Header overhead
16x16 px	grayscale	10,642	2,018	81.037%
	RGB	6,358	5,926	6.795%
32x32 px	grayscale	16,760	8,136	51.456%
	RGB	24,476	24,044	1.765%
64x64 px	grayscale	39,440	30,816	21.866%
	RGB	91,221	90,789	0.005%
128x128 px	grayscale	126,589	117,965	6.813%
	RGB	347,692	347,260	0.124%
256x256 px	grayscale	434,034	425,410	1.987%
	RGB	1,319,505	1,319,073	0.033%
512x512 px	grayscale	1,473,208	1,464,584	0.585%
	RGB	4,771,081	4,770,649	0.009%

Table 5.1: The overhead percentage for Lena tested images

In the software C/C++ code, each byte in the 1078 bytes for grayscale headers and 54 bytes for RGB headers is converted to 8 bits in ASCII. However, each bit whether it is 0 or 1 is dealt with as a byte in the terminal. This isn’t applied to the bits in the header only but the bitstream too. Therefore, there are 8624 bytes for the grayscale header and 432 bytes for the RGB header.

5.6 Discrete Wavelet Transform

The DWT is used in this project to analyze the discrete images. The original image enters two analysis filter banks which are Low Pass Filter (LPF) and High Pass Filter (HPF). They produce the forward DWT coefficients after applying downsampling with a factor of 2 as shown in Fig.5.7.

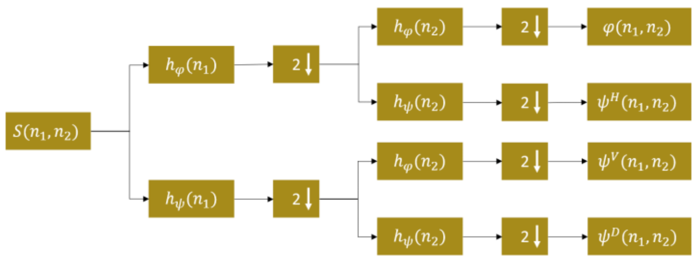


Fig.5.7: Forward DWT

The downsampling divides the bandwidth in half so to preserve the bandwidth size 'n' such that $n = 0, 1, 2, \ldots, N-1$. After that, the N samples enter the LPF and HPF again. Thus, the number of samples increase. In Fig.5.7, $\varphi(n_1,n_2) = \varphi(n_1)\varphi(n_2)$ represents the LL band, $\psi^H(n_1,n_2) = \psi(n_1)\varphi(n_2)$ represents the HL band, $\psi^V(n_1,n_2) = \varphi(n_1)\psi(n_2)$ represents the LH band, and $\psi^D(n_1,n_2) = \psi(n_1)\psi(n_2)$ represents the HH band, where φ is produced from the LPF, and ψ is produced from the HPF.

The image can be reconstructed through inverse DWT as the equation below shows, and synthesis filters are applied after applying upsampling by a factor of 2 as Fig.5.8 shows.

$$S^*(n_1,n_2) = \frac{1}{\sqrt{N_1 \times N_2}} \sum_{k_2} \sum_{k_1} W_\varphi(j_0,k_1,k_2)\, \varphi_{j_0,k_1,k_2}(n_1,n_2) + \frac{1}{\sqrt{N_1 \times N_2}} \sum_{j=j_0}^{L} \sum_{k_1} \sum_{k_2} W_\psi(j_0,k_1,k_2) \psi_{j_0,k_1,k_2}(n_1,n_2) \text{ , where}$$

- $S^*(n_1,n_2)$ represents the reconstructed image. It might not be 100% the same as the original image except for the bare eye.
- L represents the level of partitions.

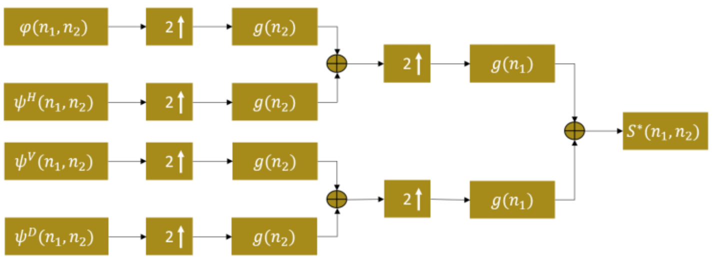


Fig.5.8: Inverse DWT

For more information about the DWT, refer to the book in [13].

Chapter 6
Validation and Discussion

6.1 Implementation Issues

To the best of our knowledge, MATLAB paths for writing and reading images should be fixed, and thus, they have been set manually in the two MATLAB programs in the project.

6.2 Implementation Challenges

The challenges that have been faced in this project are:

1. The lack of resources about FPGA on the Internet: Xilinx Community Forums is one of the main resources but doesn’t provide solutions for beginners' understanding. Answers there are very few, too short, and generalized.
2. No previous knowledge of the FPGA: Dealing with the FPGA for the first-time required reading, searching, discovering, trying, testing, understanding, and inquiring before implementing at the end something correct and right.
3. Memory restrictions: Dealing with 512 MB RAM and dividing it into sections wasn’t easy and simple.
4. BMP Images: A deep understanding has been necessary to deal with BMP images. So many images have been tested and checked to manually compare the differences in each image in the BMP format according to the pixels’ values.
5. BMP headers: The RGB and grayscale headers differ in the initial set of bytes in images of different sizes. It has been necessary to understand where the difference occurs and when it occurs to convert and reconstruct them correctly.

6.3 Description of the method used to validate the System

The system validation has been done by several tools, and they are:

1) SDK Performance Analysis tool: It has been used to get the CPU usage percentage, data cache miss rate, and the number of CPU instructions per cycle.
2) Vivado Implementation statistics: It has been used to get the power report of this system's hardware design.
3) Microsoft Excel application: It has been used to create the graphs, plots, and tables for the validation results that are shown in figures.
4) Xilinx functions are in-built functions in SDK: They have been used for calculating the time consumption and clock cycles precisely of the compression-encryption and decompression-decryption procedures.
5) MATLAB application: It has been used in evaluating the correlation results by displaying and comparing the pixel values in the encryption images.

6.4 Results Validation and Analysis

6.4.1 Power Validation and Analysis

One of these project goals is to achieve as minimum power consumption as possible. It is worth mentioning several factors that affect power. They are:

1- Performance: Whenever the performance is high, the power consumption increses, and FPGAs are known for processing tasks with high performance using parallel procedures.

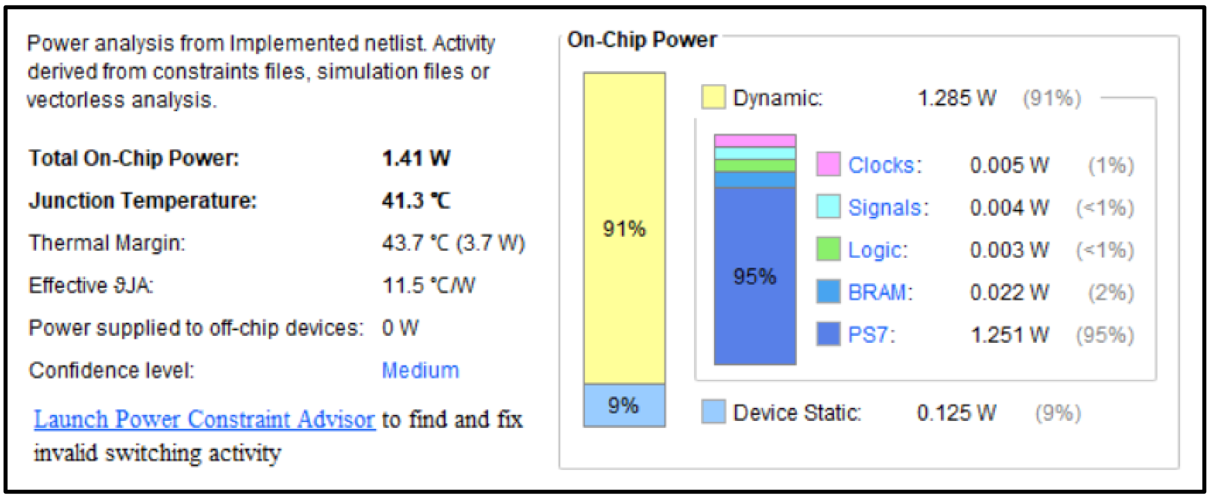
2- Time: Whenever the processed data is large, the time increase and thus the power consumption increases.

3- Utilization: Low utilization in FPGA components would result in an unnecessary power consumption. The power utilization is shown in Table 6.1.

On-Chip	Power (W)	Used	Available	Utilization (%)
Clocks	0.005	3	-	-
Slice Logic	0.003	4490	-	-
LUT as Logic	0.002	1690	14400	11.74
LUT as Distributed RAM	<0.001	112	6000	1.87
Register	<0.001	1869	28800	6.49
LUT as Shift Register	<0.001	134	6000	2.23
CARRY4	<0.001	4	4400	0.09
F7/F8 Muxes	<0.001	1	17600	<0.01
Others	0.000	356	-	-
Signals	0.004	3036	-	-
Block RAM	0.022	32	50	64.00
PS7	1.251	1	-	-

Table 6.1: Design components with power consumption and utilization percentages

Table 6.1 is also shown graphically in Fig.6.1. It is a screenshot taken from Vivado.


Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**1.41 W**
Junction Temperature:	**41.3 ℃**
Thermal Margin:	43.7 ℃ (3.7 W)
Effective ϑJA:	11.5 ℃/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

Dynamic:	1.285 W	(91%)
Clocks:	0.005 W	(1%)
Signals:	0.004 W	(<1%)
Logic:	0.003 W	(<1%)
BRAM:	0.022 W	(2%)
PS7:	1.251 W	(95%)
Device Static:	0.125 W	(9%)

Fig.6.1: The power chart of Cora Z7 in Vivado

6.4.2 Time Validation and Analysis

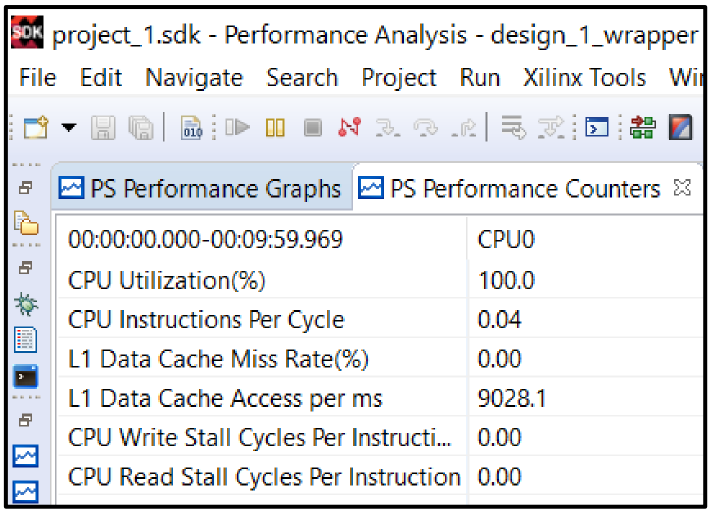
Lena images of different sizes have been used to validate the time consumed for processing in the FPGA. Lena images of different sizes are shown in Fig.6.2, and the time calculations for two Lena images is shown in Table 6.2.

Fig.6.2: Lena images of different sizes

For **100** keys & **100** sequence-long	512x512 px lena image	512x512 px lena image	512x512x3 px lena image	512x512x3 px lena image
	Compression & Encryption	Decompression & Decryption	Compression & Encryption	Decompression & Decryption
Image Size	263,222 bytes	263,222 bytes	786,486 bytes	786,486 bytes
Stretched Image Size	1,049,654 bytes	1,049,654 bytes	3,145,782 bytes	3,145,782 bytes
Transferred Bytes	1,473,208 bytes	1,473,208 bytes	4,771,081 bytes	4,771,081 bytes
Elapsed Time	127,868.824243076 ms	127,868.824243076 ms	414,124.201969230 ms	414,124.201969230 ms
Transfer Speed	11.56 KB/s	11.56 KB/s	11.12 KB/s	11.12 KB/s
Clock Cycles	916,114,054 cycles	1,195,227,788 cycles	3,068,890,984 cycles	4,129,715,757 cycles
Processing Time	2,818.812473846 ms	3,677.623963076 ms	9,442.741489230 ms	12,706.817713846 ms
Average Generation Time	11.10986328 ms	14.365718841 ms	12.295236587 ms	16.545335769 ms
Mean Computation Time	0.344093322 ms	0.448928713 ms	0.384226143 ms	0.517041742 ms
Sample Rate	2.906188249 mHz	2.227525234 mHz	2.602633953 mHz	1.934079766 mHz
Bit Rate	92.998023986 ms	71.280807495 ms	83.284286499 ms	61.890552520 ms

Table 6.2: Time calculations for 512x512 px image

Tables 6.2 has a 99% CPU usage and 0% data cache miss rate with 25 cycles for processing one instruction. These statistics have been taken from the SDK Performance Analysis tool which is shown in Fig.6.3.

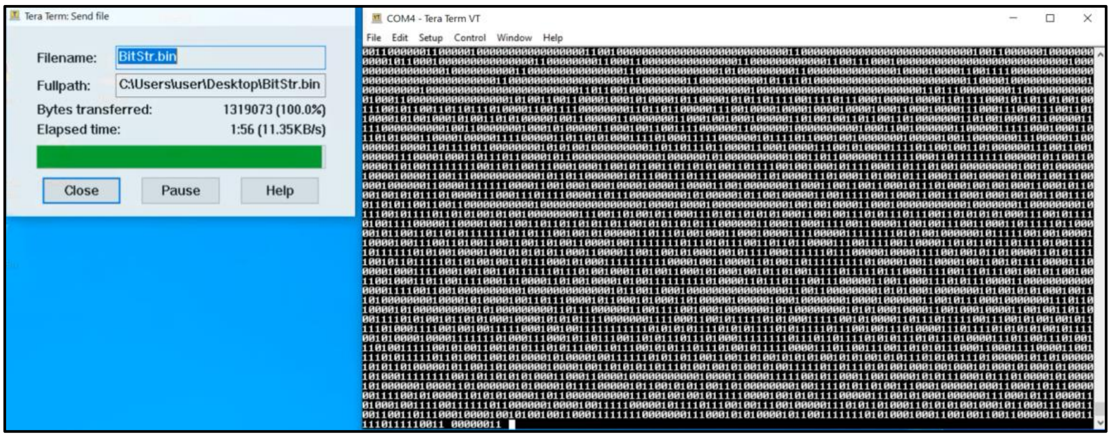

project_1.sdk - Performance Analysis - design_1_wrapper
File Edit Navigate Search Project Run Xilinx Tools Wi
PS Performance Graphs
PS Performance Counters

00:00:00.000-00:09:59.969	CPU0
CPU Utilization(%)	100.0
CPU Instructions Per Cycle	0.04
L1 Data Cache Miss Rate(%)	0.00
L1 Data Cache Access per ms	9028.1
CPU Write Stall Cycles Per Instructi...	0.00
CPU Read Stall Cycles Per Instruction	0.00

Fig.6.3: CPU utilization and miss rate in SDK

In this section, each category in Table 6.2 is discussed:

1- Image size: $Image\ Sample\ Size\ = 2^k$ such that $k\ = 1,2,3,\ldots$
2- Stretched image size: The Matlab program stretches the image in the sender and returns it to its original structure in the receiver.
3- Transferred bytes: The bytes intended here are the header and bitstream bytes.
4- Elapsed time: It is the time consumed in transferring the bytes to and from the FPGA. TeraTerm elapsed time calculations are shown in Fig.6.4.



Filename: BitStr.bin
Fullpath: C:\Users\user\Desktop\BitStr.bin
Bytes transferred: 1319073 (100.0%)
Elapsed time: 1:56 (11.35KB/s)

```
[illegible]
```

Fig.6.4: Elapsed time of 256x256x3 px in TeraTerm approximate measurements

5- Transfer speed: The speed is measured in TeraTerm automatically on the receiver side.
6- Clock cycles: $Clock\ Cycles\ = Ending\ Time - Starting\ Time$, where
   - Starting Time is calculated with the number of clock cycles in SDK by `XTime_SetTime()` and `XTime_GetTime()` functions and has units of cycles.

- Ending time is calculated with the number of clock cycles in SDK by `XTime_GetTime()` function and has units of cycles.

7- Processing time: $Processing\ Time = \frac{Clock\ Cycles}{\frac{650{,}000{,}000 \times 1000}{2}}$, where

- *650 million* is a macro in C language with the name COUNTS_PER_SECOND (it is divided by *2* because the clock cycles are in two directions).
- The clock cycles are divided by *1000* to get the processing time in milliseconds (ms) instead of seconds (s).

8- Average generation time: $Average\ Generation\ Time\ = \frac{Processing\ Time}{Image\ Parts}$, where

- *Image Parts* are the outer loop in image processing in the SDK. The image gets partitioned if its size is over 32x32 px for memory efficiency.

9- Mean computation time: $Mean\ Time\ = \frac{Average\ Time}{Part\ Size}$, where

- *Part Size* represents the size of one image part after the image gets partitioned.

10- Sample rate: $Sample\ Rate\ = \frac{1}{Mean\ Time}$

11- Bit rate: $Bit\ Rate\ = \ Sample\ Rate\ \times 32$

6.4.3 Security Validation and Analysis

The encryption and decryption techniques are the ones used in Hraini's thesis while the security tests are from the Ph.D. thesis in [14] in addition to a comparison between the hardware implementation of the system and the software solution in Hraini's thesis is at the end of this section.

6.4.3.1 Encryption/Decryption technique

The equation used in encryption is:

$B' \ = \ B\ \oplus\ K[i]$, where

- *B'* is the bit after getting encrypted.
- *B* is the bit before getting encrypted.
- *K* is the keys array.
- *i* is the index that starts at 0 and increases by 1 with each invocation.

Notes:

- K[i] is converted to binary from hex, and its bits get XORed from the rightmost bit to the leftmost one. Then it becomes ready to get XORed with the desired bit in the bitstream.
- If the keys in the array aren't enough to cover all the bits, the array starts over from i equals 0 and continues to encrypt each key with one bit in the bitstream.

The equation used in decryption is:

$B = B' \oplus K[i]$, where

- B is the recovered bit after getting decrypted.
- B' is the encrypted bit.

If the keys are larger or equal to the bits that should be encrypted, the cipher is secure as it would be one-time pad encryption. However, if the keys are reused because they weren't enough to cover all the bits, the cipher would become exposed to two-time pad attacks. Therefore, a recommended range of the array length is shown in Table 6.3.

Image size	**Key range**	
	grayscale	**RGB**
4x4 px	20 keys at most	60 keys at most
16x16 px	260 keys at most	780 keys at most
32x32 px	1,028 keys at most	3,084 keys at most
64x64 px	4,100 keys at most	12,300 keys at most
128x128 px	16,388 keys at most	49,164 keys at most
256x256 px	65,540 keys at most	196,164 keys at most
512x512 px	262,148 keys at most	786,444 keys at most

Table 6.3: Array recommended length for each image size

Notice that if the image is an RGB image, the maximum number of keys is tripled.

The calculations that have been done to get the results in Table 6.3 are explained in the following example:

> "A 4x4 px image has 16 pixels. It gets converted to DWT and divided as a result into the LL, HL, LH, and HH bands. Each band have 4 pixels. The sign of significant coefficients in HL, LH, and HH bands will be encrypted at some point in the sorting pass, so 12 keys are needed. In the LL band, the magnitude and sign of the significant bits will be encrypted in the first pass but magnitude only for insignificant bits, so 8

encryption procedure at most would happen here. Therefore, the total keys are approximately 20 keys to ensure one-time pad encryption."

The encryption equation in the software code is:

```
bitstream[pointer] = encrypt.xoring(bitstream[pointer]);
```

where:

- *bitstream[pointer]* is an array that has all the output bits of the DWT image.
- *encrypt* is an object of a class where the XORing process takes place.
- *xoring* is a function in the class in which the XORing process happens.
- *pointer* is an index of the bitstream array, and it increases by 1 after each invocation.

The encryption occurs in four different places in the software code, and they are:

1. Encrypting the magnitude of LL coefficients that reside in the LIP and satisfy the threshold condition in the first pass.
2. Encrypting the sign for LL coefficients that reside in the LIP and satisfy the threshold condition in the first pass.
3. Encrypting the output for LL coefficients that reside in the LIP and doesn't satisfy the threshold condition in the first pass.
4. Encrypting the sign of significant coefficients in the LIS.

The decryption equation in the software code is:

```
bit = decrypt.xoring(bit);
```

where:

- *bit* is the recovered bit after decryption.
- *decrypt* is an object of a class where the XORing process takes place.
- *xoring* is a function in the class in which the XORing process happens.

The decryption occurs in four different places in the software code, and they are:

1. Decrypting the magnitude of LL coefficients that reside in the LIP and satisfy the threshold condition in the first pass.
2. Decrypting the sign of LL coefficients that reside in the LIP and satisfy the threshold condition in the first pass.
3. Decrypting the sign of significant coefficients in the LIS.
4. Decrypting the sign of insignificant coefficients in the LIP if they satisfy the threshold condition and are not in the first pass.

6.4.3.2 Security Tests

Tests		***Results***	***Optimal Results***
Plain-text sensitivity			
Number of Pixel Change Rate (NCPR)		89.36%	99.61%
Unified Average Changing Intensity (UACI)		9.54%	33.46%
Hamming Distance (HD)		38.53%	close to 50%
Key sensitivity			
Number of Pixel Change Rate (NCPR)		88.01%	99.61%
Unified Average Changing Intensity (UACI)		19.03%	33.46%
Hamming Distance (HD)		39.37%	close to 50%
Histogram Analysis			
chi-square		242.5	293 (σ is 0.05)
Correlation Analysis			
1	-0.0738	(0,0)	(0,1)
-0.0738	1	(1,0)	(1,1)
Information entropy			
Information entropy (H)		7.51	8
Measurement of encryption quality			
Encryption Quality (EQ)		37.12	127.5
Time performance			
Encryption Throughput (ET)		676,186.55	-
CPU speed		667 MHz	-
Number of cycles per Byte (NCpB)		0.99 cycles	-

Table 6.4: Security results for 128x128 px Lena image

In Table 6.4, the first column is for the tests applied, the second column is for the test results according to the hardware implementation calculations, and the third column is for the optimal results according to the Ph.D. thesis in [14]. All the tests are well-explained in the Ph.D. thesis. As for the plain-text sensitivity test, the first pixel's most-significant bit has been changed, and in the key sensitivity test, the second bit in a 100 keys array has been changed.

6.4.3.3 A Comparision with the Thesis Results

Several tests have been made on the software solution[1] and thus, in this section, the hardware implementation results are compared with Hraini's thesis results to show if this project's results are like them or even better.

- Consumed Time:

  In the software solution, the processing time has been calculated for Lena image of 512x512 px, and it equaled 12.66 s. In the hardware implementation, the time consumed for the initial steps of processing the image in MATLAB is 4.66 s, and the compression-encryption procedure in FPGA is 1.50 s. Therefore, the total processing time is 6.16 s which is half the time in Hraini's result. However, the FPGA processing alone corresponds to 8 s in the software solution and thus is twenty times better. That result shows how much FPGA is fast in processing (i.e., providing high performance) as required.

- Time Overhead:

  The time overhead according to Hraini's thesis for a 512x512 px Lena image is 0.002914 s and for a key generation time that equals 0.0000722789 s. The time overhead in the hardware implementation for the same key generation time is 0.21441 s as shown in the following equation:

$$\begin{aligned} Overhead &= \frac{Encryption\ Time}{Compression\ Time} + Generating\ Keys\ Time \\ &= \frac{1.31967}{(4.65772 \text{ MATLAB} + 1.49914 \text{ FPGA})} + 0.00007 = 0.21441\,s \end{aligned}$$

  The overhead percentage is 21.4% which is larger than 1% as in the software solution by 20 times. It is still small although not negligible. The reason for this difference is because, for each pixel in the image, one bit will be dedicated for the sign and is encrypted; thus, it isn't a small amount of data getting encrypted. Also, the encryption process is not a simple XORing equation; it is getting the key from the secret keys array, then increasing the pointer by one, then checking if the array came to an end, then converting the key's digits to binary, then XORing the bits with each other till the result is one bit, and finally, XORing that resulted bit with the desired bit in the bitstream.

In Fig.6.5 and Fig.6.6 appears the calculated time for FPGA processing only with and without the encryption/decryption time. The MATLAB application time is not added in the graph as it doesn’t affect the comparison here. The two curves in each figure indicates the time overhead for each Lena image sample.

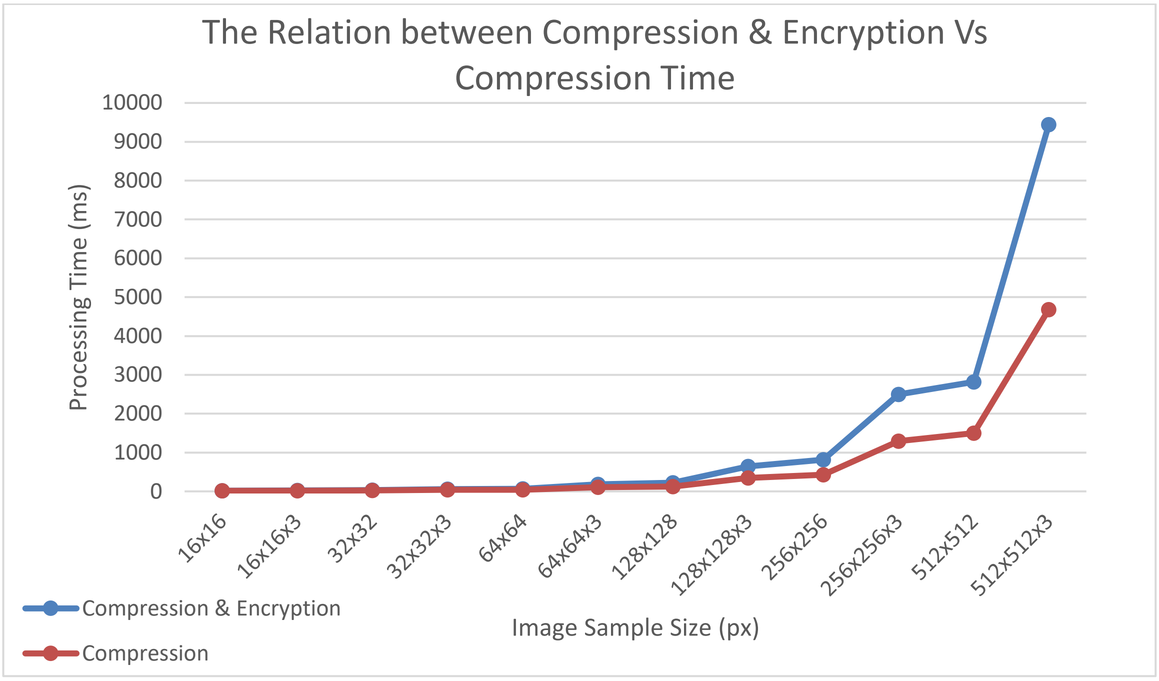


Fig.6.5: A comparison between compression & encryption Vs compression time

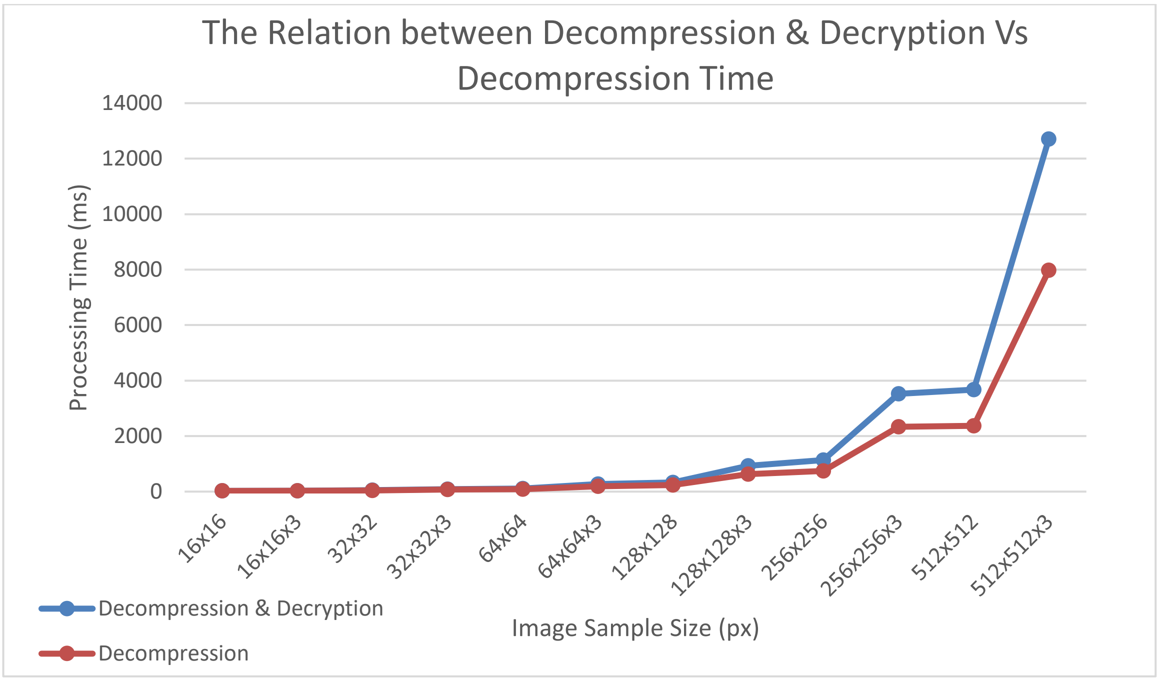


Fig.6.6: A comparison between decompression & decryption Vs decompression time

- Constant Bitrate:

The number of bits in the compression is the same as the number of bits in the compression and encryption in the software solution and hardware implementation.

- Histogram:

The histogram in the software solution is uniform in the encrypted image as opposed to the plain image for a 512x512 px Lena image. Fig.6.7 shows Hraini's results.

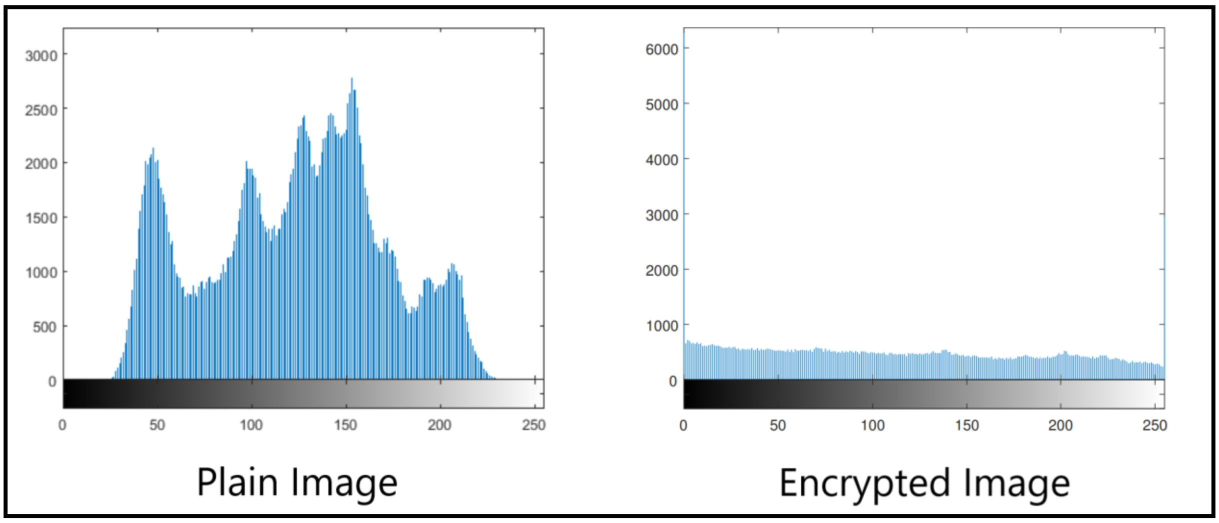


Fig.6.7: Software solution histogram results

It is necessary for the histogram in the encrypted image to be equally distributed as to not leak any information about the image and would indicate that the image is secured from any gaps for starting attacks. This is what the results show in Fig 6.8 and thus the hardware implementation corresponds to Hraini's results and is greatly secured.

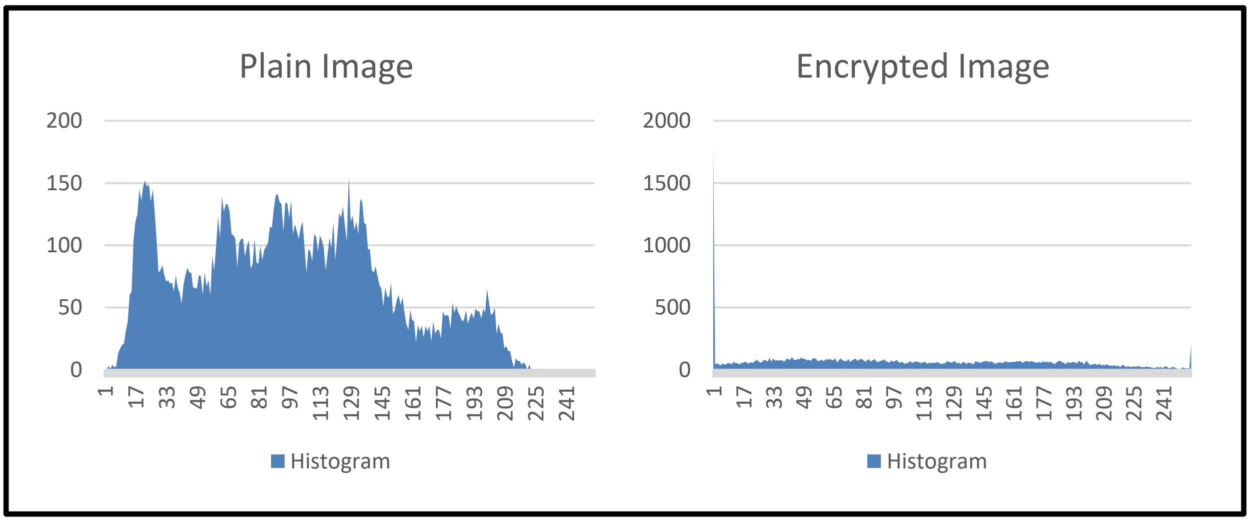


Fig.6.8: Hardware implemenation histogram results

6.5 Recommendations based on the Results

It is recommended to increase the security technique used with less time. Also, to optimize the C/C++ code as highly as possible after writing it professionally. Moreover, to make use of all the FPGA components and features to not go to waste and to achieve the best level of utilization which has a huge effect on power consumption. Furthermore, fully understanding the FPGA board in use to attain the best performance and results such as the results related to time and security.

Conclusion

It can be summarized from this document that the FPGA board is the best choice for image processing. Also, Hraini’s encryption technique with the SPIHT algorithm can deliver a secure transmission with a constant bitrate.

The system has been expected to perform the proposed security solution in the thesis in a way that guarantees the minimum use of power consumption and high performance. Adding to that, satisfying data confidentiality and privacy which the software solution aimed to apply. In the hardware implementation, these goals have been achieved in addition to the inclusion of RGB images. It is considered as an improvement to the software solution which dealt with grayscale images only.

Appendix A
System Flowchart

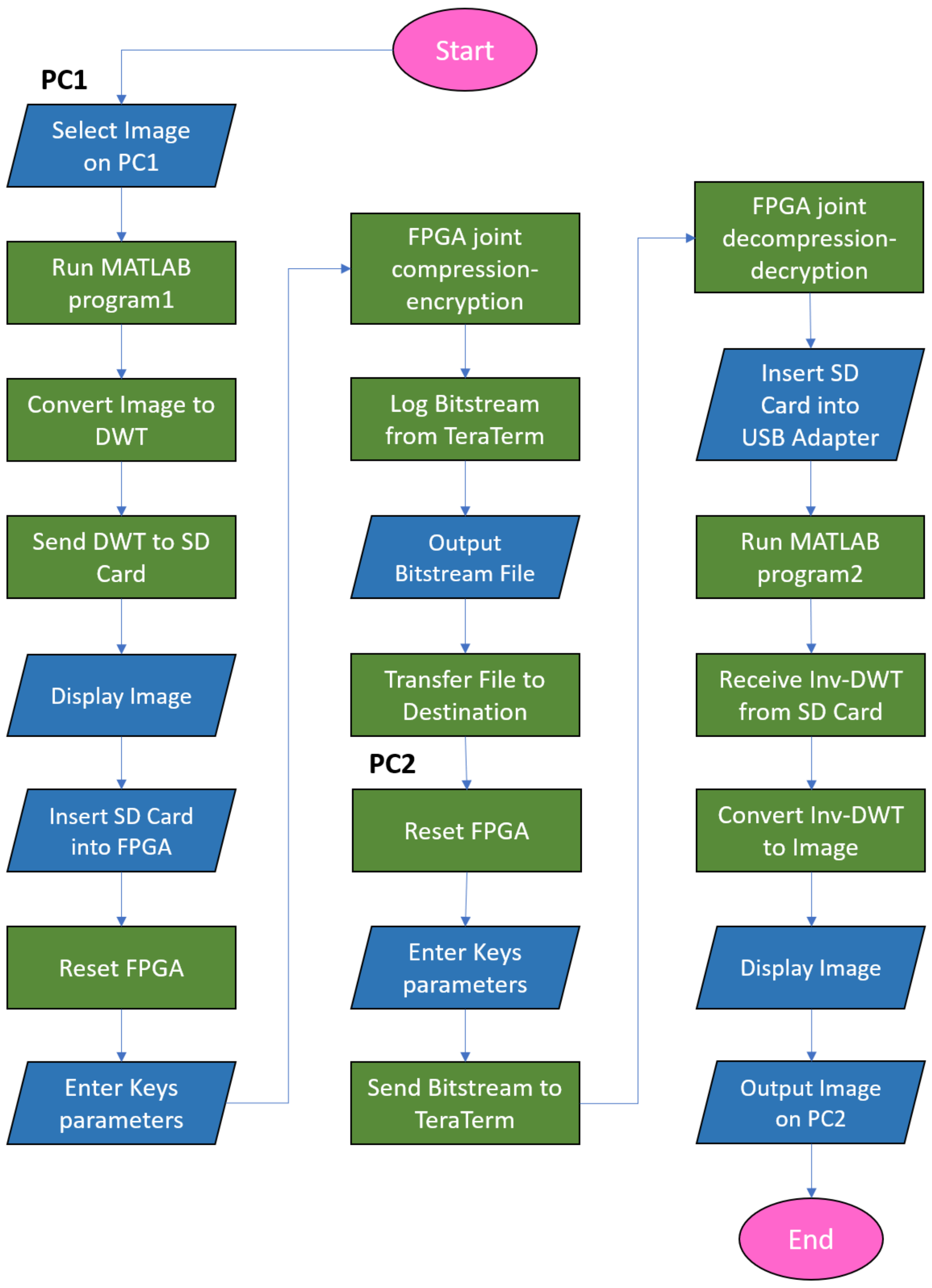


Fig.5.1: System Flowchart

Appendix B
System Pseudocode

```
//DWT in the MATLAB program "Sender Party"


Read the image entered by the user in the program path
Calculate the image size
Add zero padding if the image dimentions are not equal
Add zero padding if the image size is not in the DWT function range
Create a DWT image of double type
Zero the DWT image
Determine the DWT filter to be "bior4.4"
Get the lowpass value from the filter
Get the highpass value from the filter
Calculate the levels through the equation "log2(number of rows/cols)"

For each channel in the image, do {
         Enter the DWT function parameters (the image, levels, lowpass, highpass)
         Get the DWT image
}

Check the number of channels
If 3
         Concatenate the three DWT images into one DWT image
Else
         Do nothing
Covert the DWT image to integer type
Create an image of the same dimentions as the DWT image but four times in width
Determine the image type to be unsigned byte

For each channel, do {
         For each row, do {
                  For each column, do {
                           Check the pixel sign
                           If negative
                                    Set 1 in the first byte
                                    Remove the negative sign from the value
                           Else
                                    Do nothing
                           For each byte of the rest three bytes in the image, do {
                                    Assign two digits of the pixel value in the byte of the image
                           }
                  }
         }
}
```

Fig.5.2: Sender's MATLAB program

```
//The SPIHT compression with the selective-encryption technique
Print the prompt message “Open a log file and set it at Pause!”
Generate the keys array {
          Print the prompt message “Enter the number of secret keys:”
          Print the prompt message “Enter the length of sequence:”
          Calculate the consumed time and its variable
          Print the time calculations in Time.txt file
}
Print the prompt message “Start the logging within approximately 3 seconds!”
Wait for 3 seconds
Start the timer
Initialize the keys array pointer to point at the beginning of the array
Define the user’s image data
Determine the image size for a single image part
Calculate the number of levels of the image part
For each part of the image parts, do {
          Organize the data of the image part {
                    Take the pixels from the DWT image
                    Orginze the pixels in a vector
          }
          //Pass 1 in Hraini’s thesis: Initialization
          Calculate the maximum pixel value based on magnitude only
          Determine number of cycles for the upcoming three passes based on the maximum pixel
          Zero the list of insignificant sets (LIS)
          Fill LIS with the pixels in the LH, HL, and HH subbands
          Zero the list of significant pixels (LSP)
          Zero the list of insignificant pixels (LIP)
          Fill LIP with the four pixels in the LL subband
          While the number of cycles is larger or equals zero, do {
                    //Pass 2 in Hraini’s thesis: Sorting
                    Calculate the threshold
                    For each pixel in LIP, do {
                              Split it from its sign
                              Check if the it is larger than the threshold:
                              If yes
                                        Move the pixel to LSP
                                        Add 1 to the bitstream
                                        Check if we are in the first pass:
                                        If yes
                                                  Encrypt the bit
                                        Else
                                                  Do nothing
                                        Check the sign of the pixel:
                                        If positive
                                                  Add 1 to the bitstream
                                        Else
                                                  Add 0 to the bitstream
                                        Encrypt the sign
                              Else
                                        Add 0 to the bitsream
                                        Check if we are in the first pass:
                                        If yes
                                                  Encrypt the bit
                                        Else
                                                  Do nothing
                              Return the sign for the pixel magnitude
                    }
                    For each subband in LIS, do {
                              For each pixel in the subband, do {
                              Split it from its sign
                              Check if it is larger than the threshold:
                              If yes
                                        Add 1 to the bitstream
                                        For each pixel in the subband, do {
                                                  Split it from its sign
                                                  Check if it is larger than the threshold:
                                                  If yes
                                                            Moves the pixel to LSP
                                                            Add 1 to the bitstream
                                                            Check the sign of the pixel:
                                                            If positive
                                                                      Add 1 to the bitstream
                                                            Else
                                                                      Add 0 to the bitstream
                                                            Encrypt the sign
                                                  Else
                                                            Moves the pixel to LIP
                                                            Add 0 to the bitsream
                                                  Return the sign for the pixel magnitude
                                        }
                              Else
                                        Do nothing
                              Check if no pixel in the subband was larger than the threshold:
                              If yes
                                        Add 0 to the bitsream
                              Else
                                        Do nothing
                              }
                    }
                    //Pass 3 in Hraini’s thesis: Refinement
                    Check if we are in the first pass:
                    If yes
                              Do nothing
                    Else
                              For each element magnitude in LSP except LSP elements for this cycle, do {
                                        Convert the element to binary
                                        Check the bit that corresponds the number of this cycle:
                                        If 1
                                                  Add 1 to the bitstream
                                        Else
                                                  Add 0 to the bitstream
                              }
                    //Pass 4 in Hraini’s thesis: Quantization
                    Decrease the number of cycles by one
          }
}
Erase the keys array content in memory
Stop the timer
Calculate the consumed time and its variable
Append the time calculations in Time.txt file
Print the bitsream on the terminal
```

Fig.5.3: Sender's SDK program

```
//The SPIHT decompression with the selective-decryption technique
Generate the keys array {
    Print the prompt message “Enter the number of secret keys:”
    Print the prompt message “Enter the length of sequence:”
    Calculate the consumed time and its variable
    Print the time calculations in Time.txt file
}
Read the bitsream from the terminal {
    Print the prompt message “Insert the bitstream, then press Enter”
    Transfer the bitsream from the terminal to FPGA
}
Start the timer
Initialize the keys array pointer to point at the beginning of the array
Get the number of levels of the image from the bitstream
Determine the number of channels of the image
Calculate the image size for a single image part
Determine the final image file size in the BMP format
Write the header of the image into a new BMP file
While the bitstream didn’t reach its end, do {
    Get an image part based on the precalculated size
    Get the number of cycles for this image part
    Determine the size of the list of significant pixels (LSP) vector based on the number of cyles minus 1
    //Pass 1 in Hraini’s thesis: Initialization
    Zero the image vector
    Zero the list of insignifiant pixels (LIP) vector
    Zero the list of insignifiant sets (LIS) vector
    Zero the list of significant pixels (LSP) vector
    Set the first four pixels in LIP to 1
    While the number of cycles is larger or equals zero, do {
        //Pass 2 in Hraini’s thesis: Sorting
        Calculate the threshold
        For the first four pixels in LL subband, do {
            Get a bit from the bitstream
            Check if we are in the first pass:
            If yes
                Decrypt the bit
            Else
                Do nothing
            Check the value of the bit:
            If 1
                Get a bit from the bitstream
                Decrypt the bit
                Check the value of the bit:
                If 1
                    Set the image vector with the threshold value
                Else
                    Set the image vector with the negative threshold value
                Zero the corresponding LIP pixel
                Store 1 in the LSP
            Else
                Do nothing
        }
        For each element in the subband, do {
            Check its equivalent in LIP:
            If 1
                Get a bit from the bitstream
                Check the value of the bit:
                If 1
                    Get a bit from the bitstream
                    Decrypt the bit
                    Check the value of the bit:
                    If 1
                        Set the image vector with the threshold value
                    Else
                        Set the image vector with the negative threshold value
                Else
                    Do nothing
                Zero the corresponding LIP pixel
                Store 1 in the LSP
            Else
                Do nothing
        }
        For the three subbands in each level, do {
            For each subband elements, do {
                Check if the subband has been checked before:
                If yes
                    Do nothing
                Else
                    Get a bit from the bitstream
                    Check the value of the bit:
                    If 1
                        Mark the subband as checked
                        For each element in the subband, do {
                            Get a bit from the bitstream
                            Check the value of the bit:
                            If 1
                                Get a bit from the bitstream
                                Decrypt the bit
                                Check the value of the bit”
                                If 1
                                    Set the image vector with the threshold
value
                                    Else
                                    Set the image vector with the negative
threshold value                                 Increase LSP value
                                Store LSP value in the corresponding pixel
                                Else
                                Set the corresponding LIP pixel to 1
                        }
                    Else
                        Do nothing
            }
        }
        //Pass 3 in Hraini’s thesis: Refinement
        Check if we are in the first pass:
        If yes
            Do nothing
        Else
            While LSP value minus 1 doesn't equal 0, do {
                Get a bit from the bitstream
                Check the value of the bit:
                If 1
                    Check the sign of the corresponding pixel in the image vector:
                    If negative
                        Split it from its sign
                        Add the threshold value to the array pixel magnitude
                        Return the sign
                    Else
                        Add the threshold value to the array pixel
                Else
                    Do nothing
                Decrease the LSP value by 1
```

Fig.5.4: Receiver's SDK program

```
//Inverse DWT in the MATLAB program "Receiver Party"


Read the image from the user's desktop
Calculate the image size
Convert the image to integer type
Create an array of the same dimentions as the Inverse DWT image but the width is divided by four
Zero the array
For each channel, do {
        For each row, do {
                For each column, do {
                        Assign the last three bytes of the Inverse DWT to the array element
                        Check the first byte
                        If 1
                                Multiply the array element with -1
                        Else
                                Do nothing
                }
        }
}
Determine the Inverse DWT filter to be "bior4.4"
Get the lowpass value from the filter in the reverse mode
Get the highpass value from the filter in the reverse mode
Prepare an array of the Inverse DWT coefficients
Calculate the levels through the equation "log2(number of rows/cols)"

For each channel in the image, do {
        Enter the Inverse DWT function parameters (the array, lowpass, highpass, levels)
        Restore the original image
}

Check the number of channels
If 3
        Concatenate the three reconstructed images into one DWT image
Else
        Do nothing
Remove the zero padding from the image rows
Remove the zero padding from the image columns
Display the image
Write the image oo the user's desktop
```

Fig.5.5: Receiver's MATLAB program

References

[1] Hraini, I. (2019). *Joint Crypto-Compression Based on Selective Encryption for WMSNs* [online]. Available from: DSpace at PPU (Palestine Polytechnic University) [accessed 1 April 2020].

[2] Luigi Dragotti, P. ; Poggi, G. ; Ragozini, A.R.P. (2000). *Compression of multispectral images by three-dimensional SPIHT algorithm* [online]. Available from: IEEE Explore, Digital Library [accessed 4 April 2020].

[3] Vaggalis, N. (2012). *Weakly Typed Languages* [online]. Available from: I Programmer [accessed 24 April 2020].

[4] (2019). *Putty vs TeraTerm* [online]. Netscylla Cyber Security Ltd [accessed 17 November 2020].

[5] Shakir, N. (2018). *Learn how to use your Raspberry Pi to alter images and videos with this basic image processing tutorial* [online]. Available from: EETech Media, LLC [accessed 29 April 2020].

[6] AlAli, M. ; Mhaidat, K. ; Aljarrah, I. (2014). *Implementing image processing algorithms in FPGA hardware* [online]. Available from: IEEE Explore, Digital Library [accessed 29 April 2020].

[7] Salhab, O. ; Joodeh, M. ; Abutaha, M. ; Jweihan, N. (2018). *Survey paper: Pseudo random number generators and security tests* [online]. Hebron: Palestine Polytechnic University [accessed 17 November 2020].

[8] Pham, M. ; Aziz, S. (2011). *FPGA-Based Image Processor Architecture for Wireless Multimedia Sensor Network* [online]. Available from: ResearchGate [accessed 26 January 2021].

[9] Pham, D. ; Aziz, S. (2011). *FPGA architecture for object extraction in Wireless Multimedia Sensor Network* [online]. Available from: ResearchGate [accessed 26 January 2021].

[10] *Cora Z7: Zynq-7000 Single Core and Dual Core Options for ARM/FPGA SoC Development* [online]. 1300 Henley Ct #3, Pullman, WA 99163: Digilent Inc. [accessed 11 April 2020].

[11] *Zybo Z7: Zynq-7000 ARM/FPGA SoC Development Board* [online]. 1300 Henley Ct #3, Pullman, WA 99163: Digilent Inc. [accessed 21 June 2020].

[12] *ZedBoard Zynq-7000 ARM/FPGA SoC Development Board* [online]. 1300 Henley Ct #3, Pullman, WA 99163: Digilent Inc. [accessed 21 June 2020].

[13] Gonzalez, R. ; Woods, R. *Digital Image Processing Third Edition* [online]. P[488-512] [accessed 20 December 2020].

[14] Farajallah, M. (2015) *Chaos-based crypto and joint crypto-compression systems for images and videos* [online]. Engineering Sciences (physics). UNIVERSITE DE NANTES.