Secure IP Camera

A Project Report
submitted by

Naveen John (12EC67)
Pranav Channakeshava (12EC74)
Rounak Sengupta (12EC80)

under the guidance of

Prof. Ramesh Kini

in partial fulfilment of the requirements
for the award of the degree of
BACHELOR OF TECHNOLOGY

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA
SURATHKAL, MANGALORE - 575025

April 20, 2016

0.1 DECLARATION

We hereby declare that the Project Work Report entitled **"Secure IP Camera"** which is being submitted to the National Institute of Technology Karnataka, Surathkal for the award of the Degree of **Bachelor of Technology in Electronics and Communication Engineering** is a bonafide report of the work carried out by us. The material contained in this Project Work Report has not been submitted to any University or Institution for the award of any degree

Name of the Student	Register No.	Signature with Date
Naveen John	12EC67	
Pranav Channakeshava	12EC74	
Rounak Sengupta	12EC80	

Department of **Electronics and Communication Engineering**

Place : National Institute of Technology Karnataka, Surathkal

Date : 21 April 2016

0.2 Certificate

This is to certify that the project entitled **"Secure IP Camera"** submitted by :

(1) Naveen John (12EC67),
(2) Pranav Channakeshava (12EC74),
(3) Rounak Sengupta (12EC80)

as the record of the work carried out by them, is *accepted as the B. Tech Project Work Report Submission* in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology in Electronics and Communication Engineering**.

Guide

Dr. Ramesh Kini
Associate Professor,
Department of Electronics and Communication Engineering.
National Institute of Technology Karnataka, Surathkal

Chairman-DUGC
(Signature with Date and Seal)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

Major Project -II

End Semester Evaluation Report, April 2016

Course code : EC498
Course Title: Major Project -II
Title: *Secure IP Camera*

Project Group:

Name of the Student	Register No.	Signature with Date
Naveen John	12EC67	
Pranav Channakeshava	12EC74	
Rounak Sengupta	12EC80	

Place: NITK Surathkal
Date: 21-04-2016

Name and Signature of Project Guide

ACKNOWLEDGEMENTS

We would like to thank Prof. Ramesh Kini, for all his support and for providing valuable insights leading to the successful completion the project. We are also grateful to Prof. Oliver Bringmann and Sebastian Burg, from the University of Tuebingen, who gave us the chance to work on a part of the End-to-Display Encryption project. We thank them for their support throughout the duration of the project.

We also sincerely thank our evaluators Dr. M.R. Arulalan, Dr. Deepu Vijayasenan and Dr. Raghavendra B S for their support and constructive advice along the way.

Further, we are very grateful to Mrs Prabha, lab-in-charge of DSP Lab along with the other lab personnel who have provided us with vital assistance for the lab resources along with software support over the 2 semesters. We also thank other professors, lab assistants, research scholars and colleagues whom we may have not mentioned, for all their support and advice.

ABSTRACT

The evolution of video surveillance technology towards network-based systems has marked the prominent trend in the security industry. It serves various functional and financial benefits to corporations in order to better protect their assets and personnel. The development of networked Internet Protocol (IP) surveillance has enabled increased accessibility and security while reducing the costs of traditional CCTV systems. However, IP cameras have been proven to be susceptible to hacks once the network integrity is compromised. This necessitates methods to secure IP cameras against such malicious attempts.

The project aims to develop a hardware module to secure IP cameras by adopting the concept of End-to-Display Encryption (E2DE), which has been developed and researched by the University of Tuebingen, Germany. E2DE enables data security at endpoints by extending the protection of End-to-End encryption to beyond the communication endpoint by using pixel-domain data instead of raw application data, through decryption done between the computer and the display.

The proposed module will encode the camera stream such that the stream can be decoded only by users who are granted access through the E2DE receiver of their display. Even if the stream is accessed by an eavesdropping attack, the hacker will only see the encoded stream and will not be able to decode it without having the required key for the E2DE receiver. As such, it enables the user to view sensitive data remotely without the risk of it being stolen or eavesdropped on.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
Introduction

1.1 Problem definition

Given the vulnerabilities of networked IP camera to attempts of eavesdropping by Trojan horses once the network integrity is compromised, there is a need to develop methods of securing the cameras. The project looks to adopt strengths of the concept of End-to-Display Encryption (E2DE) by encoding the camera output and enabling it to be viewed securely at remote locations.

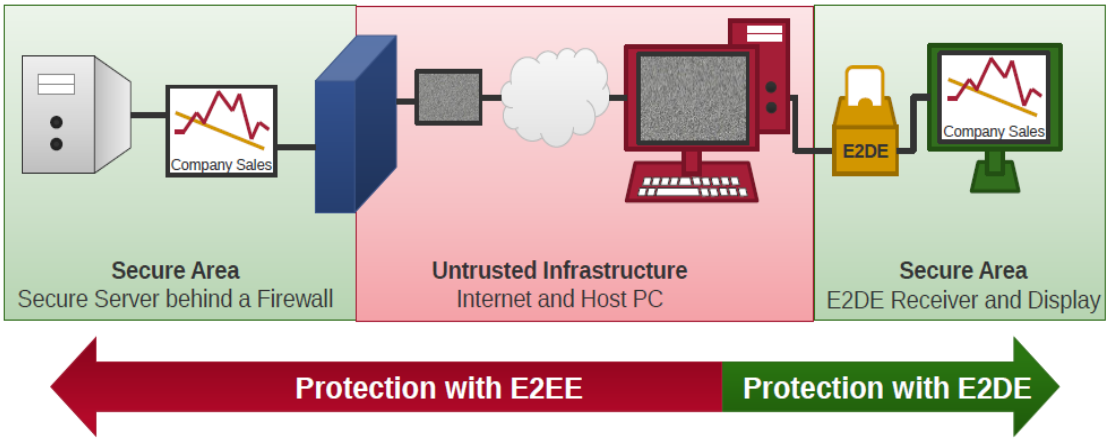


Figure 1.1: Comparison of End-to-end Encryption and End-to-Display Encryption [2]

E2DE adopts the encryption of images in the pixel-domain. When these images are displayed by arbitrary image viewers, the data stream is analyzed on the way to the display using the E2DE re-ceiver and the encrypted pixels are decrypted depending on a private key stored on a chip card inserted in the E2DE receiver. This creates a viewable representation of the confidential data on the display, without decrypting the information on the computer itself, making the data unreada-ble for any Trojan horse. As such, the video stream of an E2DE enabled camera can be viewed at weak endpoints without the risk of eavesdropping.

1.2 Literature Review

In this section, we review some of the top design vulnerabilities of E2EE IP cameras.

1. Cross Domain Image Streaming

   An attacker might be able to record the illegitimate motion pictures in the same format as accepted by the web interface and by exploiting the inherent remote file inclusion vulnerability to conduct successful CDIS attacks. These types of issues can hamper the surveillance by leveraging the attack to trick the victim to watch a similar illegitimate motion picture.

2. Insecure CGI Interface

   CGI scripts used in the IP surveillance platform are not secure and can be used to conduct potential attacks [9]. Remote command execution, authentication bypasses and authorization control manipulation are the common attacks that can be conducted easily on CGI interface.

3. Factory Defaults and Undocumented Credentials

   Factory default credentials are used to provide a first time administrative control of the online devices [8] [10]. The credentials are not interface specific but work unanimously across all the administrative interfaces. Factory default credentials can be found on the respective IP surveillance cameras documents and on the internet.

4. Insecure Front End Design

   Insecure front end design is one of the worst design practices that have been noticed in the web interfaces of IP surveillance cameras. Considering the general principle of security, a user should not be allowed to access any resource on the remote server until appropriate authentication is done and required authorization control is processed.

5. Unencrypted Parameters and Process Memory Dumping

   Process dumping becomes more rustic when memory dump is large. Administrative credentials (generic strings such as admin, guest etc.) can be found from the active installer process memory as a number of IP surveillance cameras do not use appropriate encryption modules which enable the credentials to flow in a clear text in the respective process memory.

1.3 Motivation

The presented design flaws in End-to-End Encryption (E2EE) IP surveillance cameras illustrates its various security vulnerabilities. By adopting the End-to-Display Encryption (E2DE) approach, it is possible to protect the information on not trustworthy endpoints and thus improving the capabilities of E2EE. As E2DE performs the decryption only in the wire which causes unencrypted data to be never accessible on the computer protects it against eavesdropping by Trojan horses. As E2DE performs the decryption only in the wire which causes unencrypted data to be never accessible on the computer protects

it against eavesdropping by Trojan horses. With corporate espionage on the rise, this technology can be a great benefit for corporations.

Most of the attempts at securing IP cameras have been software-based solutions for the network and few attempts to encrypt the stream using hardware have been made. In 2014, Kotel et al. [5] use AES to encrypt fullscreen VGA signal from a camera with the resolution of 720x480 pixels at 60Hz. But the implementation does not provide a complete system for secure access at the end-point, like that provided by E2DE. Thus, developing the proposed solution will provide a comprehensive framework for IP camera streams to be accessed at end-points securely.

1.4 Overview

The various stages of the project are as follows: -

1. Software Demonstrator to encode MIPI-CSI camera

   In order to demonstrate the concept of the E2DE-enable camera, a software version is done on the Raspberry Pi micro-computer, which has a MIPI CSI camera module. The software application will capture an image, encode it using the format used for E2DE and display the same on the webserver interface. In order to check the result, the image can be decrypted by the E2DE receiver on a computer with HDMI connection to the proof-of-concept FPGA implementation.

2. Developing the real-time encoding module

   The encoding module which implements the image compression using Run Length Encoding and subsequently, pixel-domain encryption using 128 bit AES. The AES key is encoded with 512 bit RSA encryption and written into the header pixels.

3. Decrypting the image

   The encoded images are decrypted using software or hardware like FPGA. Given that the current implementation of the receiver takes a HDMI input, necessary changes need to be realised to use the same design for verfication of the encoding process.

CHAPTER 2
Description

2.1 Encoding Format

Given that the E2DE receiver has already been developed, the encoding to be implemented by the hardware module proposed needs to adhere to the format adopted in the receiver. The captured image frame is first compressed and then encrypted. The header pixels are written at the start of each line.

The details of the different aspects of encoding follow:

2.1.1 Compression of image

In the encoding of the image, the image is compressed using Run Length Encoding. The need for this compression is due to the placement of the header pixels written in the im-age, which are used by the receiver module to decode the images. As the header is placed in every line of the image, the width of the encoded image will be wider than the width of the original image without any compression.

Width of uncompressed encoded image = Width of original image + Width of header

Hence, compression ensures that the width of the encrypted and the original image are the same. Correspondingly, decompression of the decrypted image is done by the receiver module. The compressed image, depending upon the extent of compression might be smaller than the original image, in which case, random pixels are added to ensure that the width of the encrypted image the same as that of the original image. This gives rise to an additional advantage that the encrypted image is now harder to decode, due to the intro-duction of random pixels.

Run length encoding is a simple data compression technique. It traverses runs of data or sequences which have several occurrences of the same data value in consecutive elements and stores them as a single value and count. In the image, compressed pixels are repre-sented as a pair of pixels with one pixel holding the colour and the other holding the num-ber of repetitions of the colour i.e. a pair of colour, count. This provides a simple representation making the corresponding decompression simple as well.

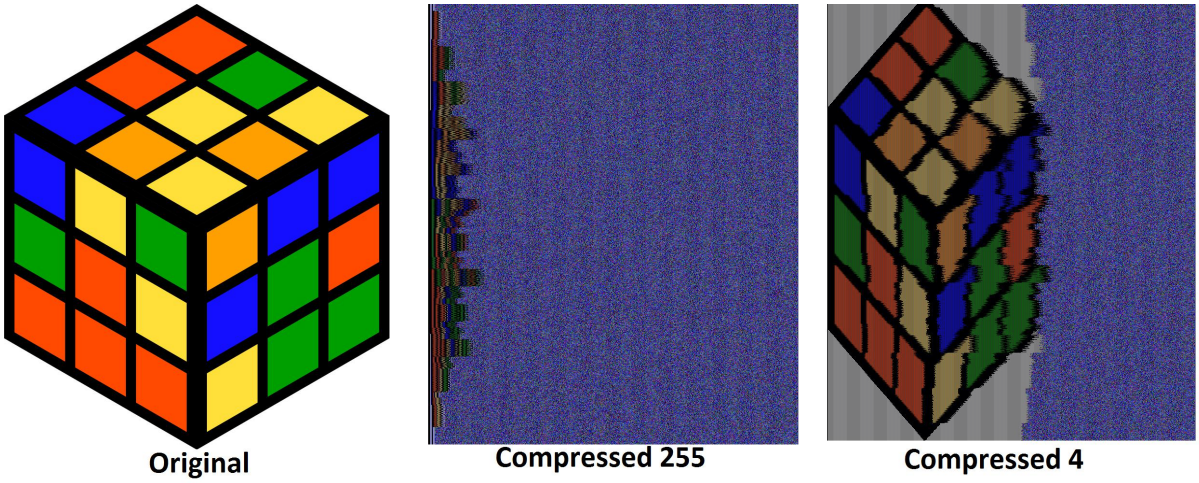


Figure 2.1: Run Length Encoding compression with different amounts of compression.

In the receiver side, the decompression of the image is done by the use of the colour and counter pointers. The decompression representation is recognized and the corresponding increments of the 2 pointers are done, to traverse the pixel array. When the count pixel is read, it is assigned to a register counter whose value is decremented after every clock cycle and pointers are not incremented until the counter becomes zero.

2.1.2 Pixel Domain Encryption using 128 bit AES

The individual pixels in the compressed image are encrypted by using the counter mode of 128 bit Advanced Encryption Standard (AES). This mode is a very common mode used in many secure applications where a block-sized count value is maintained, and encrypted, the result being XORed with the data to generate encrypted output. It is not necessary to use the full 128-bits of output, and so input data need not be padded up to the block-size of 128 bits. In this case, a 128 bit AES vector is used to XOR with a set of 5 24-bit RGB pixels.

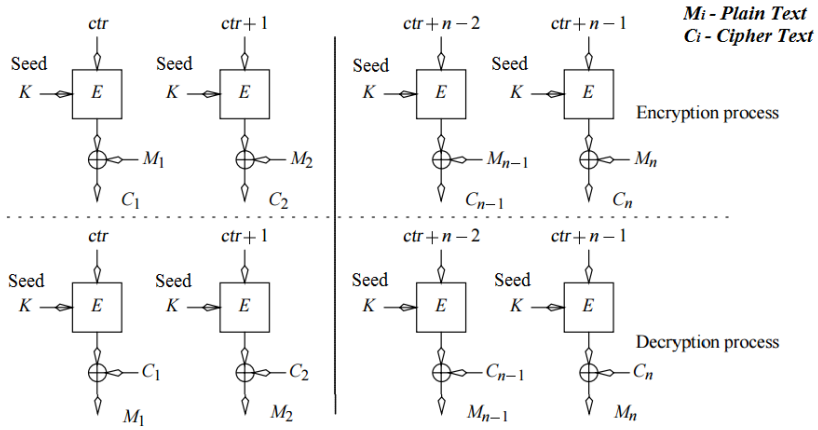


Figure 2.2: Counter Mode AES Encryption

In order to generate unique 128-bit per-block vectors, an initialization vector in the form of an incrementing nonce counter is used apart from the AES key. The generated 128 bit AES vector is XORed with the plaintext to give the ciphertext. One of the advantages of the counter mode of AES is that the latency through the AES module can be minimized, as the only processing applied to the data is an XOR function and the data itself does not go through the iterations of the AES module. Also, the mode can be parallelized and pipelined for better throughput.

2.1.3 Writing Header Pixels

The header has the AES nonce/seed which is written unencrypted in the header. This makes it easier to process the decryption of the image. The header pixel is 36 pixels wide in total. The contents of the header pixels in each line is specified below

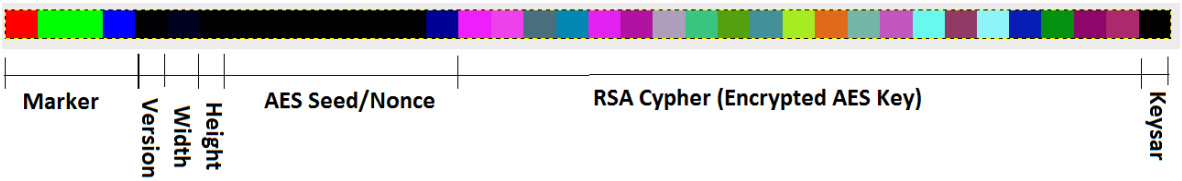


Figure 2.3: Header Pixels Definition

1. Marker (RGGB) 4 pixels
2. Version 1 pixel
3. Width 1 pixel
4. Height 1 pixel
5. AES Seed/Nonce - 6 pixels
6. RSA 22 pixels
7. Keysar - 1 pixel

2.2 Debugging and Verification using ChipScope Pro

In order to validate the encryption and compression implementation, the output from the encoding hardware is decrypted and decompressed to obtain the original data. The decoding hardware for the E2DE specifications has been developed at the University of Tuebingen. However, due to the lack of availability of hardware with similar I/O specifications, we develop a software approach to verification. Software applets which perform decryption and decompression are derived from the software demonstration mentioned earlier.

The LogiCORE IP ChipScope Integrated Logic Analyzer (ILA) core is a customizable logic analyzer core that can be used to monitor any internal signal of your design. The ILA core includes many advanced features of modern logic analyzers, including Boolean trigger equations, trigger sequences, and storage qualification. Because the ILA core is synchronous to the design being monitored, all design clock constraints that are applied to your design are also applied to the components inside the ILA core.

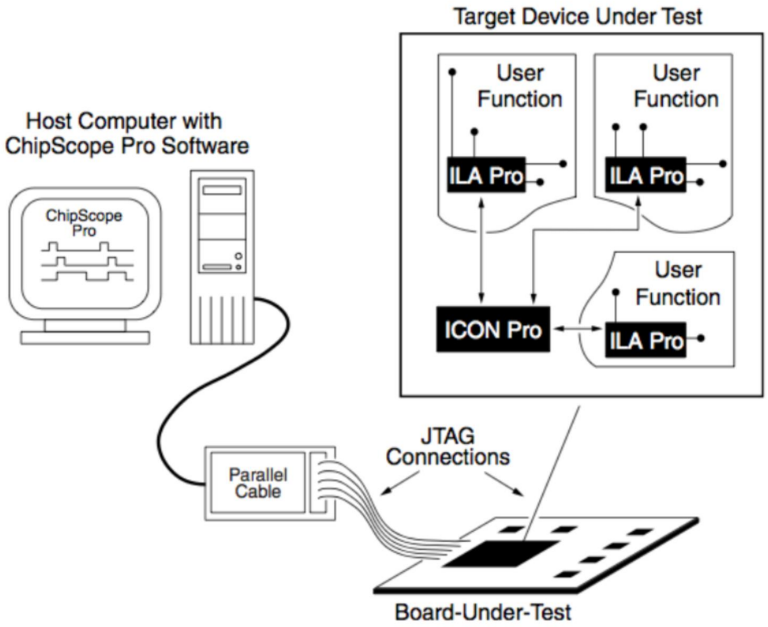


Figure 2.4: Overview of ChipScope Pro debugging by using the JTAG connection.

In this project, ChipScope was utilised initially to debug the compression process. Introducing the ILA cores into the test module enabled observing the changes in the values of various buffers and flags during the process of compression. The cores also can be triggered on a particular flag or an edge of the clock. On reaching a stage when the compression results were satisfactory, the purpose of ChipScope was to export the values necessary for the verification of the entire process of compression and encryption. At a time, around 32k values were exported in .csv format and was used to recreate the image and decompress it.

CHAPTER 3
Results

3.1 Software demonstrator

As a precursor to the hardware implementation, the concept is demonstrated through a software program to encode the camera. Given that the Raspberry Pi has a MIPI CSI camera module as-sociated with it, it was decided that the software version will be done on the Raspberry Pi micro-computer. The Java application available previously to encode images for the E2DE receiver was used for the encoding.

A shell script was written to do the following operations on the Raspberry Pi board with the camera module connected to it- i) Capture a picture using the camera module and store it in a specified location ii) Encode the captured image using the Java application, which also stores the encrypted image iii) Opens the encoded image using an installed image viewer

The software implementation demonstrates the encryption process for a single image. This mimics the process of encrypting a single frame by the real-time hardware, which is to be developed.

3.2 Real-time hardware encryption

The encryption hardware consists of 2 parts : 1. Encoding hardware 2. MIPI Interface. The former part of the hardware has been taken up. The encoding module is being developed by using a readily availlable CMOS camera which can be interfaced easily with a FPGA board. This would allow the focus to be on the real-time encoding design rather than on the I/O interface.

3.2.1 Camera Configuration

Hardware Description

The CMOS camera being used is the OV7670. It is a low voltage CMOS image sensor that provides the full functionality of a single chip VGA camera and image processor in a small footprint package. The OV7670 provides full-frame, sub-sampled or windowed 8 bit images in a wide range of formats controlled through the Serial Camera Control Bus (SCCB) interface.

It has an image array capable of operating at up to 30 frames per second (fps) in VGA with complete user control over image quality, formatting and output data transfer. All

required image processing functions, including exposure, gamma, white balance, colour saturation, hue control and more are also programmable through SCCB interface.

The Zedboard development kit is being used for designing the encoding module. It is based on the Xilinx Zynq-7000 Extensible Processing Platform(EPP). Combining a dual Corex-A9 Processing System (PS) with 85,000 Series-7 Programmable Logic (PL) cells, the Zynq-7000 EPP can be targeted for broad use in many applications. The ZedBoards robust mix of on-board peripherals and expansion capabilities make it an ideal platform for implementing the interface standards for the camera.

Some of the relevant features provided by the ZedBoard consist of:

1. Xilinx XC7Z020-1CSG484CES EPP
    - Primary configuration = QSPI Flash
    - Auxiliary configuration options: a) Cascaded JTAG b) SD Card
2. Memory
    - 512 MB DDR3 (128M x 32)
    - 256 Mb QSPI Flash
3. Interfaces
    - USB-JTAG Programming using Digilent SMT1-equivalent circuit
    - USB 2.0 FS USB-UART bridge
    - Five Digilent Pmod compatible headers (2x6) (1 PS, 4 PL)
4. On-board Oscillators
    - 33.333 MHz (PS)
    - 100 MHz (PL)
5. Display/Audio
    - HDMI Output
    - VGA (12-bit Color)
    - 128x32 OLED Display

The connection between the camera and the zedboard is established using Diligent Pmod headers. The Zedboard has five diligent Pmod compatible headers (2x6). Jumper wires are used to connect the pins from the camera to the Zynq EPP pin. We adopt the interface code developed by Hamsterworks for the OV7670 camera. The physical connections made are shown below.

Pmod	Signal Name	Zynq EPP pin	Pmod	Signal Name	Zynq EPP pin
JA1	JA1	Y11	JB1	JB1	W12
	JA2	AA11		JB2	W11
	JA3	Y19		JB3	V10
	JA4	AA9		JB4	W8
	JA7	AB11		JB7	V12
	JA8	AB10		JB8	W10
	JA9	AB9		JB9	V9
	JA10	AA8		JB10	V8

Figure 3.1: Jumper Wire connections obtained from the Zedboard hardware guide

The camera output has been observed through the VGA output onto a monitor. The colour conversion being used by the interfacing code required some correction in order to get the correct colours being displayed. As seen in the picture below, the without the correct register settings, the display shows incorrect colour representation. By implementing the required correction, we observe the colours working correctly.

Figure 3.2: VGA output of the OV7670 without and with colour correction

3.2.2 Pixel-domain Encryption

The encryption of the pixels using 128-bit AES has been implemented, by using the encryption modules which have been previously used in the E2DE receiver design. The flow of the encryption is shown in the figure below. A fixed 128 bit AES key stored in the register has been used for the encryption. The AES seed increments are done for each line so that it will act as a nonce for the AES module. The pixel encryption is done by XOR'ing the AES vector, which is generated from the encryption module, with a set of five 24-bit RGB pixels.

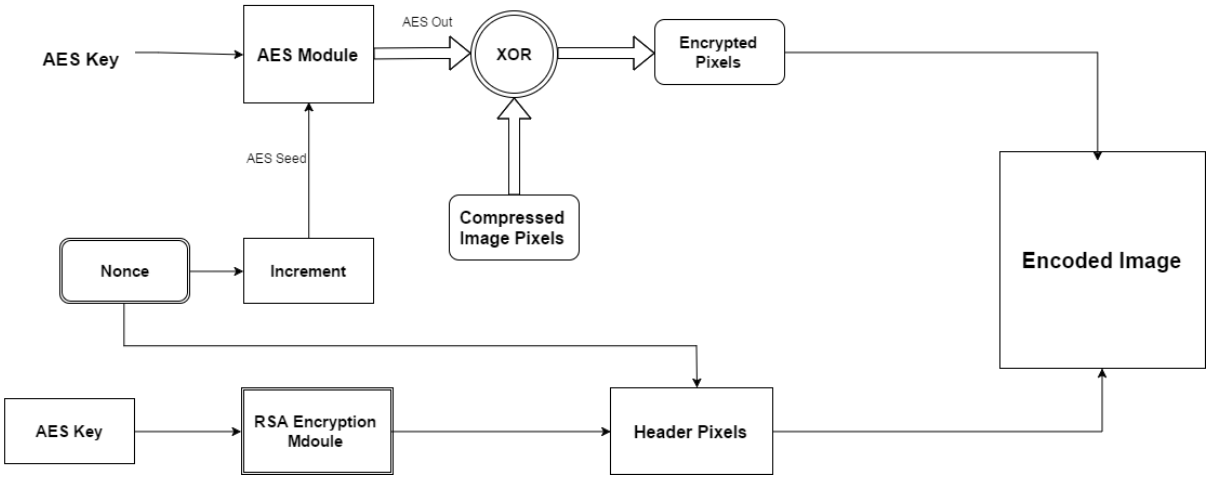


Figure 3.3: Data flow for encoding module

The input pixels are stored in a buffer as there is a need for the header pixels to be written along with the compression of pixels in a line. Thus, the output pixels are delayed when compared with the input. This delay has to be equal to the number of pixels in a line as the compression module considers the whole line for compression.

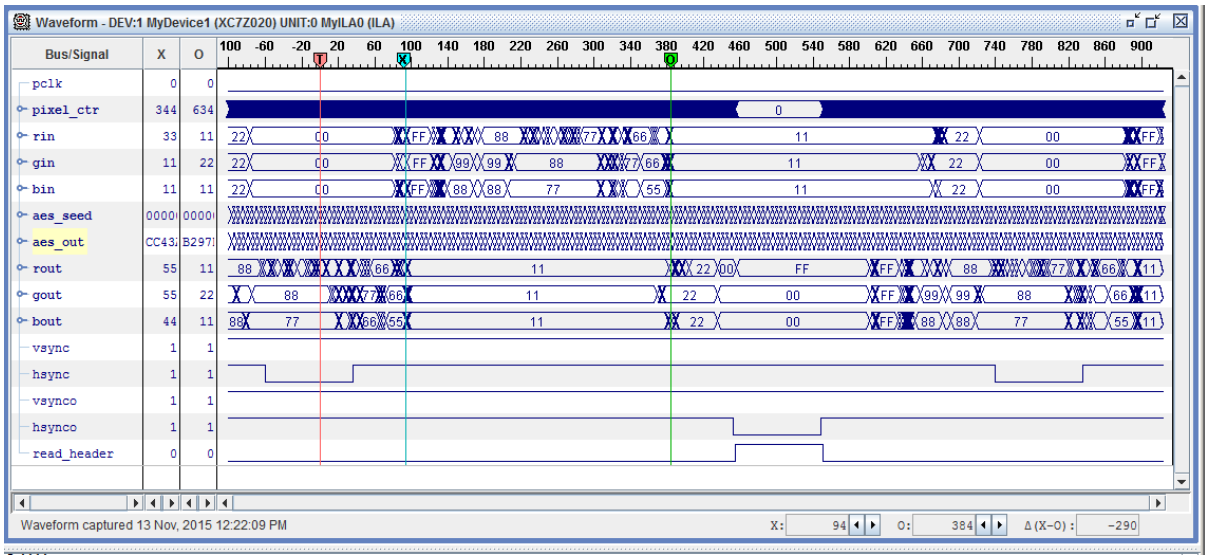


Figure 3.4: ChipScope graph of all the signals including the input pixels, vsync and hsync signals. The delay introduced by the encoding module is seen here.

The real-time encryption of the camera input can be seen below. ChipScope Pro has been used for real-time on-chip debugging to observe the signal values. As seen in the ChipScope graphs, we observe the delay in the output pixels(ro,go,bo) with respect to the input pixels (ri,gi,bi). Also, the header pixels are written for the first 37 pixels of every line. The header pixels which represent the metadata for each line have been inserted. As the real-time compression module is yet to be completed, the header pixels are currently written by cropping out the first 37 pixels of the line.

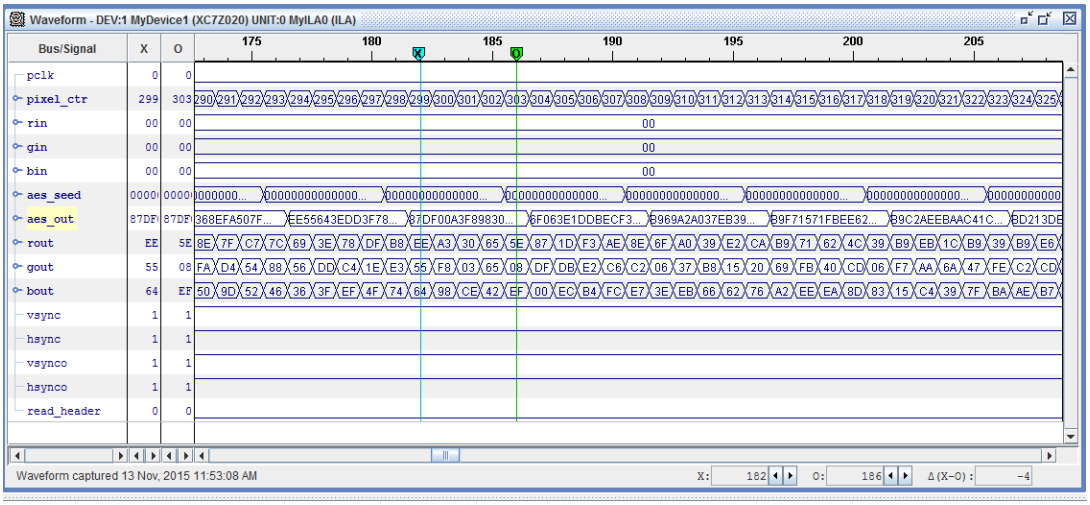


Figure 3.5: ChipScope graph of the pixel-domain encryption using 128 bit AES.

Figure 3.6: The real-time encrypted pixels as observed on the screen.

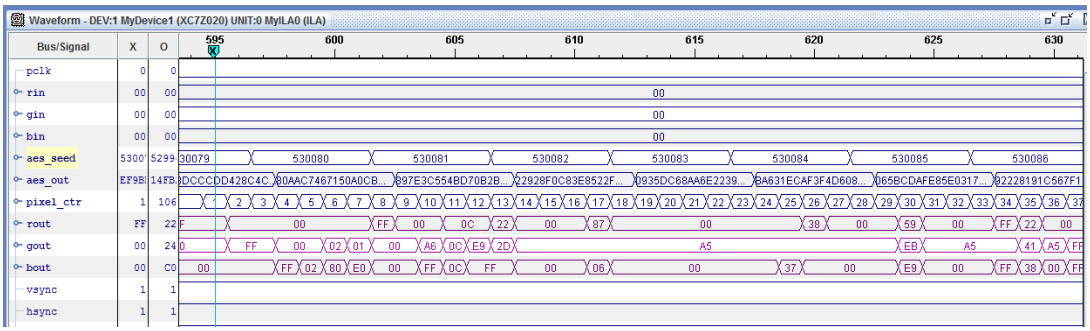


Figure 3.7: A ChipScope Pro graph illustrating the header pixels as metadata for a line.

3.2.3 Pixel Compression

Run length encoding is used for the compression of the image to accommodate the header pixels and maintain the size of the original image after encryption. The algorithm has been written in VHDL and the simulation of the same has been performed and verified to be correct.

We take each RGB pixel to be 24 bit wide with 8 bit individual components. These pixels are stored in a buffer. At any point of time the entire line of the image is stored so that compression of the entire line can be performed using run length encoding. The pseudo code being implemented is shown below.

Pseudo code:

```
Result: Compressed pixel array
initialization i=0;
for each i in the pixel buffer array do
    initialization j=1;
    for each i+j in the pixel buffer array do
        Check similarity index between buffer[i] and buffer[i+j];
        if sim index within range then
            increment counter,j;
            store pixel, count;
        end
        else
            store pixel, count;
            i = i+j;
            break;
        end
    end
end
```

Algorithm 1: Run Length encoding of pixel

The simulation performed has a buffer size of 10 instead of 640 as in the case of the original image. The buffer size has been taken as 10 for easier simulation purposes. This algorithm is being extending to implement the real-time compression of a line.

The extent of compression is decided by the similarity index. For example, allowing a difference of 2 per each colour of RGB, we can have similarity index as 6. If adjacent pixels are within this range, they are compressed. Similarity index also controls the loss of pixels due to compression.

3.2.4 Approaches to Pixel Compression

Approach 1

The initial approach to perform compression involved simulating a VHDL module with the above run-length encoding code.

The simulated module was verified using a testbench. The buffer with the rin, gin,

bin values were portmapped to the VHDL module. However, the hardware for the same could not be synthesised. The reason for not being able to synthesis the hardware was because the left limit for the inner loop is undefined.

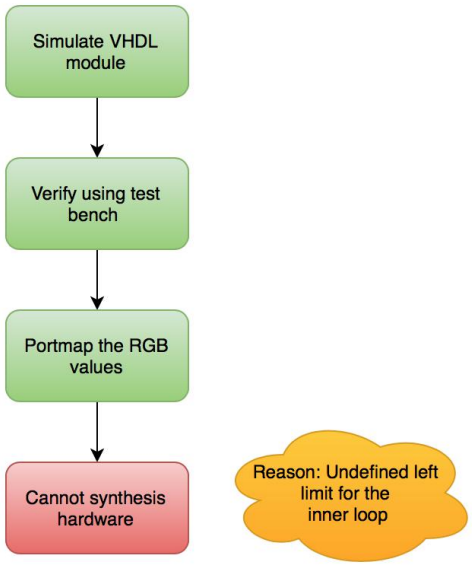


Figure 3.8: Approach 1

Approach 2- Traversal based compression

The RGB values stored in the buffer are traversed in a single clock cycle and the compression is performed on the pixels.

The compression code is incorporated in the testmodule directly. However, it was noticed that a number of pixels were either lost or overwritten when the compression was happening. This was mainly due to the delay in compression i.e by the time compression is performed on the entire buffer, some of the buffer values are overwritten with the RGB from the next line. Also this approach used twice the number of buffers, one for the original RGB values and one for the compressed pixels.

Hence, this approach to pixel compression did not give satisfactory results for implementation.

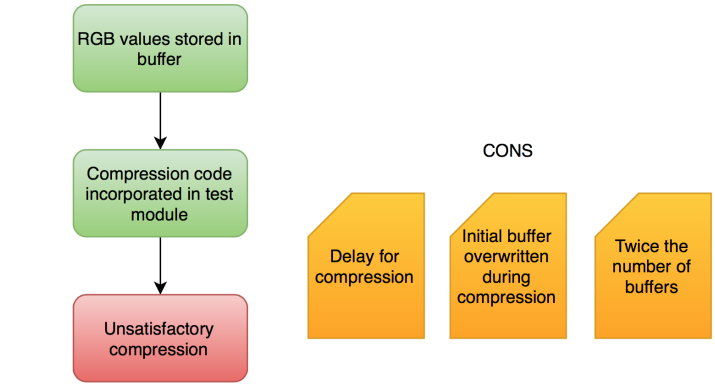


Figure 3.9: Traversal based compression

Approach 3- Read based compression

In this approach, like in the previous one, the compression code is incorporated in the test module directly. Instead of waiting for the entire buffer to get filled up by the RGB pixels, the compression is performed as the pixels are read. The colour, count pixel pair is put into the buffer only after the compression. This is by far, the most optimised approach for compression. The rest of the pixels in the buffer is filled with random pixels.

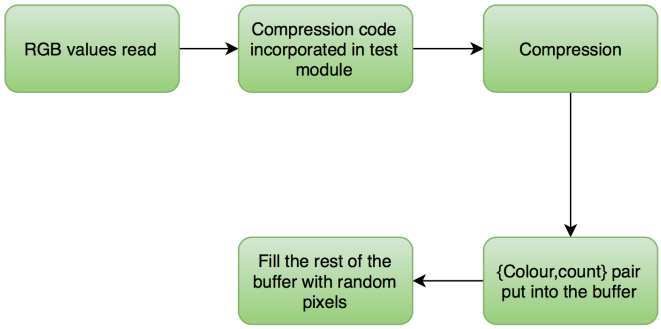


Figure 3.10: Read based compression

3.2.5 Real-time compression

As discussed in the previous section, read based compression is implemented to perform the compression of the image from OV7670 camera.

The working of read based compression is verified using a testbench using Xilinx ISE. For simulation purposes, a similarity index of 50 is chosen and count is to be represented as RGB = 0,0,255. Hence the maximum compression that can be performed is 255.

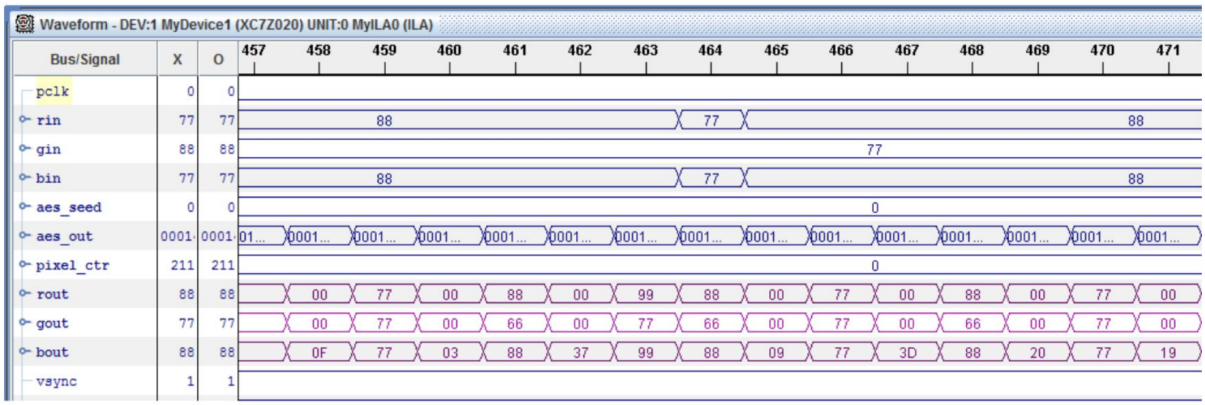


Figure 3.11: Simulation of compression with sim index =50

However, on synthesising the test module and burning it on the FPGA, a repetition of image is observed on the display. This is due to the fact that the freed up parts of the buffer due to the compression reads residue value and the same image repeats.

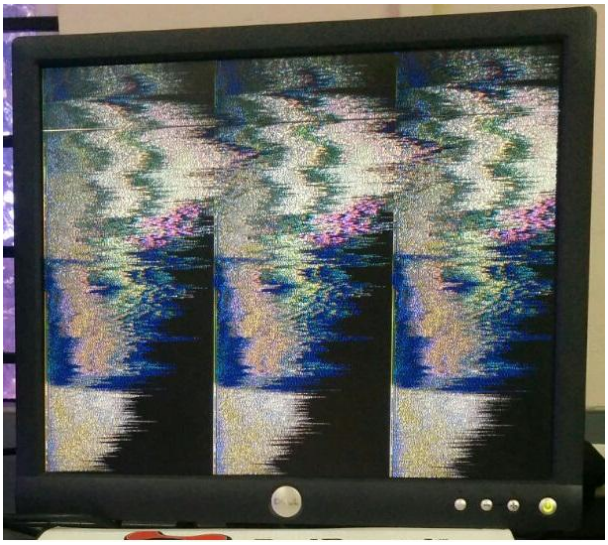


Figure 3.12: VGA output after compression

3.3 Verification

The verification process used to ascertain the accuracy of the encryption and compression done on hardware is summarised in sections that follow and illustrated in the picture below.

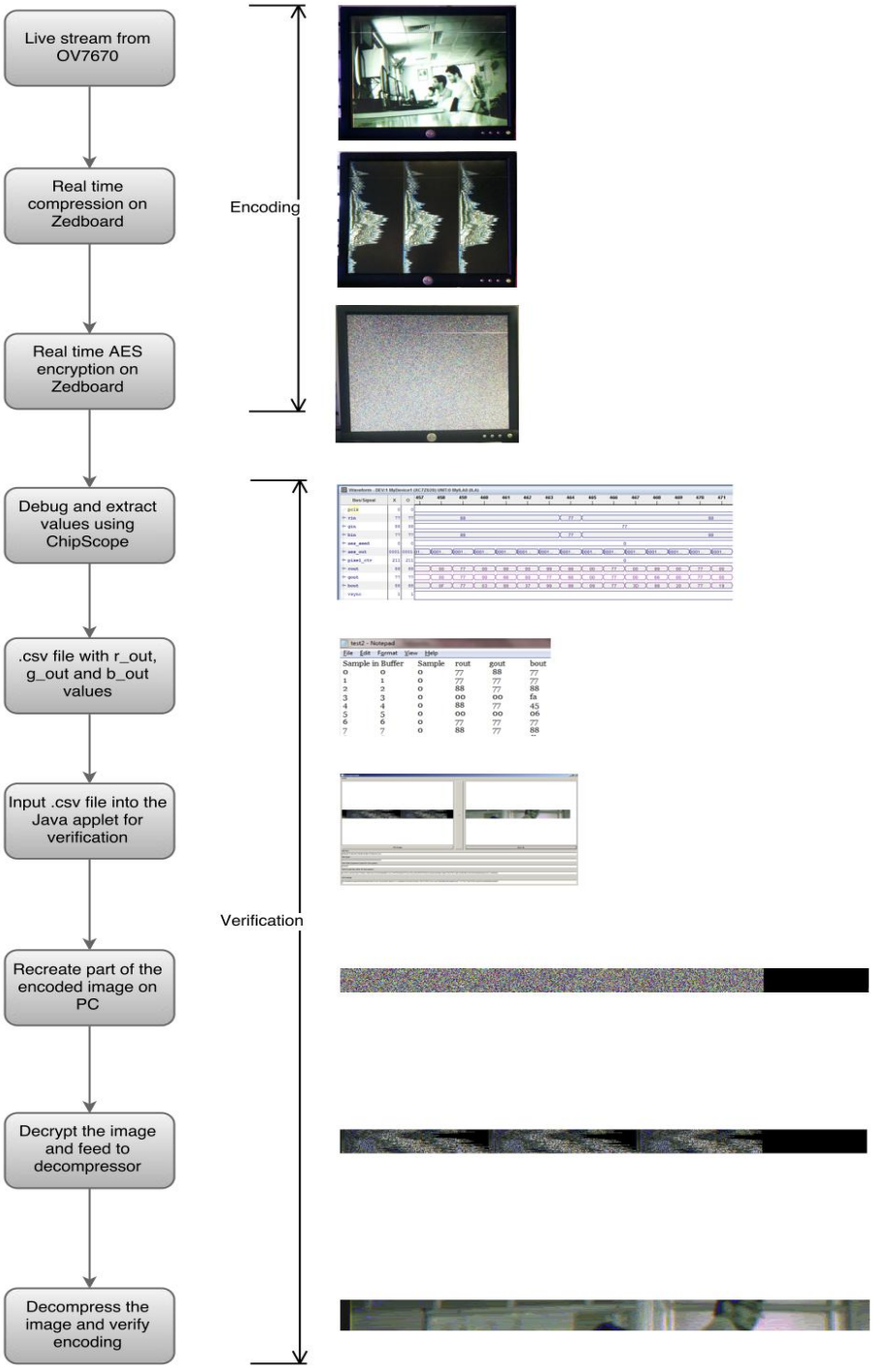
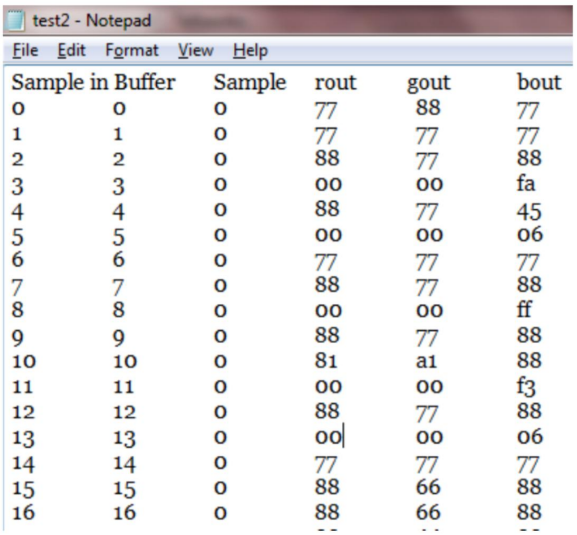


Figure 3.13: Summary of the encoding in hardware and decoding processes in software which are implemented in the project.

3.3.1 RGB Pixel Export

To verify the implementation of the encryption and compression, ChipScope Pro is used to extract parts of the frame. Given that the export utilises Block RAM (BRAM), the number of values that can be stored is restricted by the amount of BRAM available on the FPGA chip. In our case, the number of pixels exported is limited to 32768, which translates to about 40 lines of the frame which is 800 pixels wide.

The RGB data values are exported into a CSV file and then parsed using a Java applet to create a PNG image from the RGB pixel values. The encoded images are stored in the Portable Network Graphics (PNG) format as it supports a lossless data compression. This is an essential requirement of the decoding process, as any changes in the image may lead to incorrect decompression and decryption. A Java application has been derived from the original software demonstration that performs decryption and decompression individually.


test2 - Notepad

File Edit Format View Help

Sample in Buffer		Sample	rout	gout	bout
0	0	0	77	88	77
1	1	0	77	77	77
2	2	0	88	77	88
3	3	0	00	00	fa
4	4	0	88	77	45
5	5	0	00	00	06
6	6	0	77	77	77
7	7	0	88	77	88
8	8	0	00	00	ff
9	9	0	88	77	88
10	10	0	81	a1	88
11	11	0	00	00	f3
12	12	0	88	77	88
13	13	0	00	00	06
14	14	0	77	77	77
15	15	0	88	66	88
16	16	0	88	66	88

Figure 3.14: A snapshot of a part of the CSV file with RGB values, which are exported from ChipScope Pro.

3.3.2 Compression

The implementation of the compression module adopts the Read-based compression approach as discussed in Section 3.2.4. As the compressed output fills only a part of the buffer, we fill the remaining parts of the line with black pixels in order to maintain the sync of the lines. There is inherent repetition of the data as observed from the figure below.

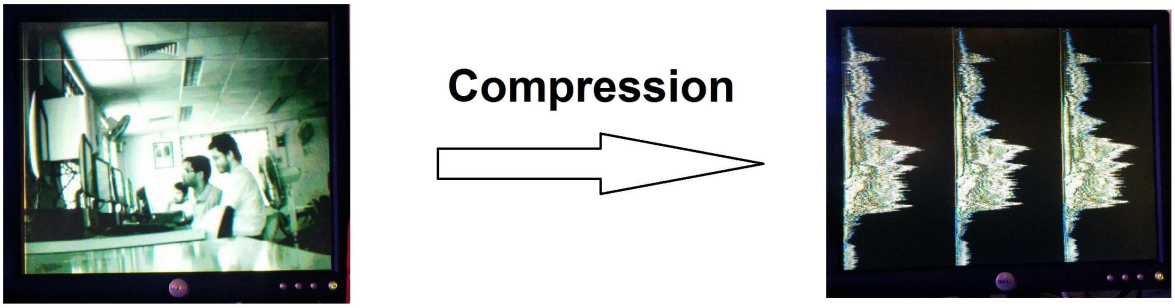


Figure 3.15: A snapshot of a frame compressed using RLE.

The decompression module recognises the hybrid compression representation of {Colour, Count} and recreates the runs of the data accordingly for single as well as multiple pixels.

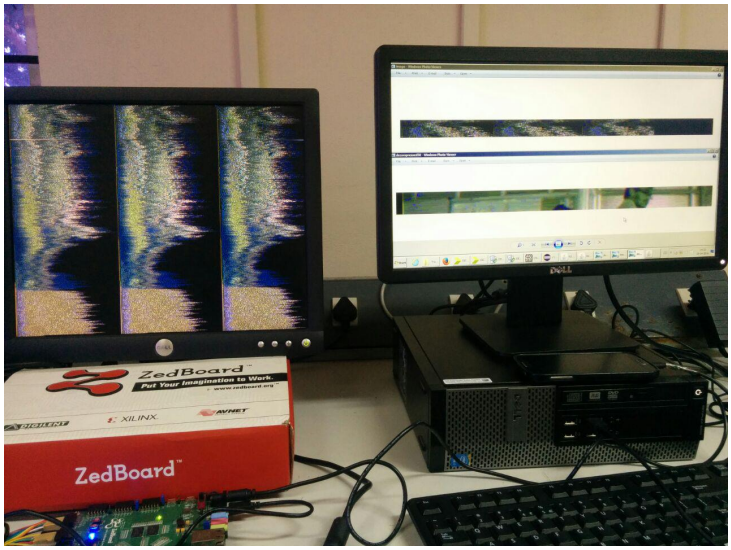


Figure 3.16: A snapshot of the decompression software, displayed on the monitor on the right, along with the compressed output seen on the monitor on the left.

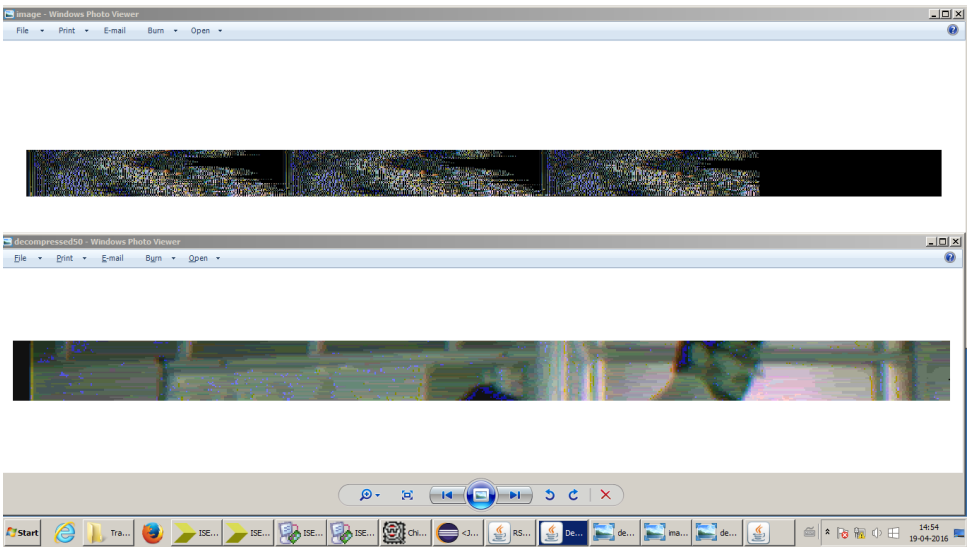


Figure 3.17: A zoomed in screenshot of the decompression result of the decompression shown in the previous figure.

The extent of the compression is set by the parameter similarity index as mentioned previously, which determines how lossy the compression is. As the value of the parameter is increased, the data is compressed more but there is more loss in the data. This can be observed in the figures below.

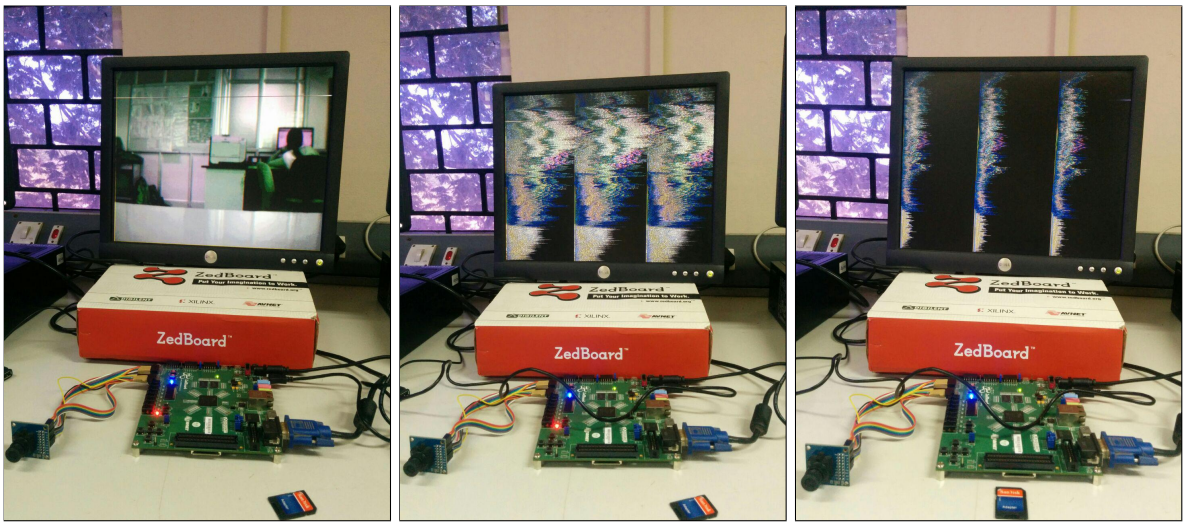


Figure 3.18: A comparison of the outputs for different extent of compression. The middle picture has similarity index of 50 while the right picture has 100 as similarity index.

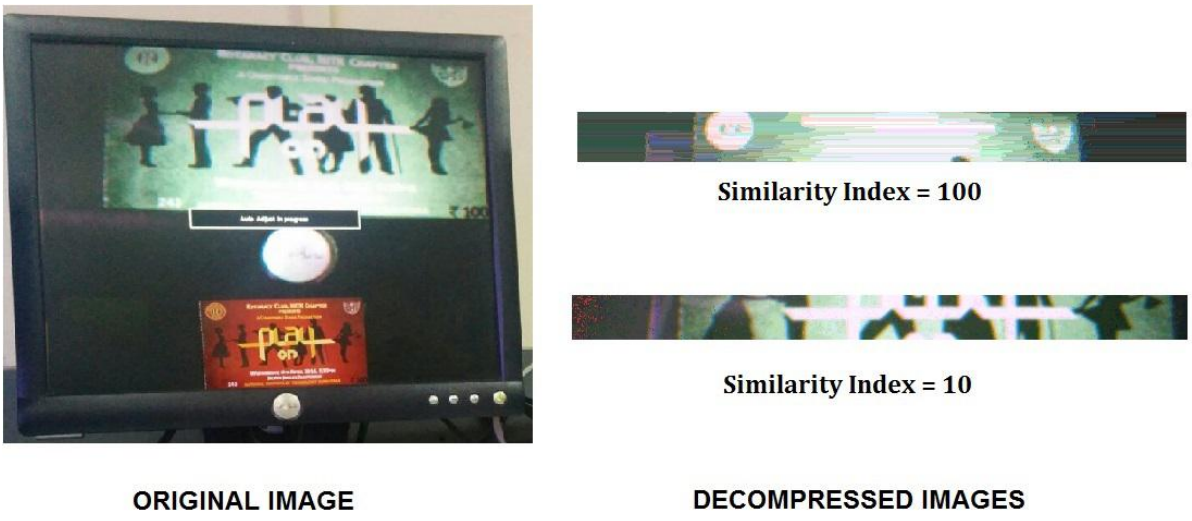


Figure 3.19: A comparison between the compression with different similarity index. The image used is shown on the left. The compression with similarity index 100 is quite lossy when compared to that with 10.

3.3.3 Encryption

The verification of the 128-bit AES encryption is done through the Java applet that derives the decryption part of the software demonstration. It utilises the Cipher class in Java Cryptographic Extension framework, which provides the functionality of a cryptographic cipher for encryption and decryption.

In the implementation, the impact of the AES seed increments on the appearance of the encrypted frame was observed. As suggested by Burg et. al. [1], having the same seed for encryption at the start of every line, can lead to potential vulnerabilities to hacking using simple differencing methods. This necessitates the need for different seed for every line of the frame. Further, having the same seed values for different frames means that a moving object can be spotted in the encrypted video stream. As such, the seed needs to be different for adjacent frames in order for the encrypted video stream to not give away any kind of information directly to an observer.

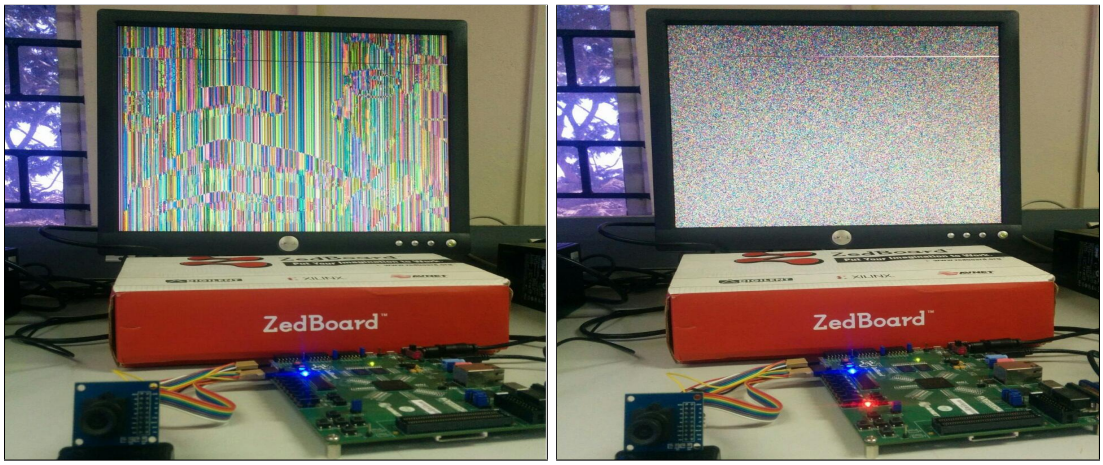


Figure 3.20: A comparison of the outputs for encryption with and without the seed reset for every line. With the seed reset for every line (left), the image has clear correlation to the outline of the original image while this is not the case for the latter.

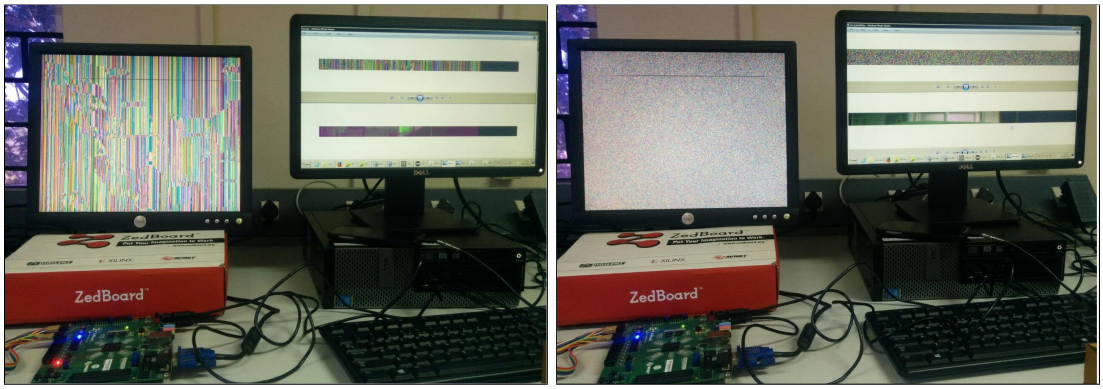


Figure 3.21: Decryption of the encrypted frame, with seed reset for every line (left) and without the reset (right), observed in the monitor on the right respectively for both pictures.

CHAPTER 4
Summary

In the project, the following tasks have been accomplished -

- Software demonstration on Java using Raspberry Pi
- Camera configuration using readily available OV7670 module with VGA output
- Encoding module for real-time encryption and compression using Run Length Encoding
- Exported RGB values using ChipScope Pro and created PNG image for verification
- Verified the working of the encoding module by decrypting and decompressing the recreated PNG image

Some of the main learning outcomes from the project were:

- Familiarising with Raspberry Pi 2 to run Java applet on it
- Interfacing OV7670 with the Zedboard
- Testing different run length compression techniques
- Debugging using ChipScope IP cores to sort out issues with the buffer write
- Real-time video processing challenges - Continuous I/O of pixels
- Debugging compression timing related issues to obtain properly compressed image
- Utilising the Java applet to perform decryption and decompression

CHAPTER 5
Conclusion

The project aims to develop an encoding hardware that will secure IP cameras through the End-to-Display Encryption (E2DE) paradigm. Given that today's IP cameras are vulnerable to hacks once the network integrity is compromised, the proposed module will provide data security by extending the protection of end-to-end encryption to beyond the communication endpoint. The encoding format employed is in view of the linebased E2DE that was proposed by Burg et. al. [1]. The encoding hardware, implemented on the Zedboard FPGA Kit, encompasses the compression using run length encoding and encryption using 128-bit Advanced Encryption Standard(AES).

In order to verify the working of the 2 modules, decryption and decompression of the encoded frames has been done. Due to the non-availability of FPGA boards with the specified I/O requirements, the verification is done through software. ChipScope Pro, which is an on-chip debugging tool, is used for the purpose. By inserting Integrated Logic Analyzer (ILA) cores in the design, a section of the output frame is captured by storing the RGB data in the block RAM. The obtained data is then exported to a CSV file, which is then parsed through to create a PNG image from the RGB exported data. This image is then decrypted and decompressed using a Java applet, which is developed as a part of the earlier software demonstration.

The working mechanism of the compression and encryption modules has been individually verified through the Java applet. The extent of loss induced due to the compression has been observed through adjusting the threshold for compression. As such, the encoding module has been developed as per the linebased E2DE format. It is hoped that this will aid to better secure IP camera for better data security.

REFERENCES

[1] Sebastian Burg, Pranav Channakeshava, Oliver Bringmann *Linebased End-to-Display Encryption for Secure Documents* In Proceedings of the 2nd IEEE International Conference on Identity, Security and Behaviour Analysis

[2] Sebastian Burg, Dustin Peterson, Oliver Bringmann *End-to-Display Encryption: A Pixel-Domain Encryption with Security Benefit* In Proceedings of the 3rd ACM Workshop on Information Hiding and Multimedia Security

[3] Marc Defossez, *D-PHY Solutions* Application Note: Spartan-6 and 7 Series FPGAs, August 25, 2014

[4] H. Hsing. Tiny AES - Crypto Core.
`http://opencores.org/project/tiny_aes` October 2013

[5] S. Kotel, M. Zeghid, A. Baganne, T. Saidani, Y. I. Daradkeh, and T. Rached *FPGA-Based Real-Time Implementation of AES Algorithm for Video Encryption* Recent Advances in Telecommunications, Informatics and Educational Technologies, pages 27 to 36, 2014.

[6] Robust Defenses for Cross-Site Request Forgery,
`http://seclab.stanford.edu/websec/csrf/csrf.pdf`

[7] Hacking CCTVs,
`http://events.ccc.de/congress/2005/fahrplan/attachments/692-slides_cctv_hacking.pdf`

[8] Security Camera Hack Conceals Heists Behind Dummy Video,
`http://www.wired.com/threatlevel/2009/07/video-hijack/`

[9] Dynamic Content with CGI,
`http://httpd.apache.org/docs/1.3/howto/cgi.html`

[10] Camtron CMNC-200 IP Camera Undocumented Default Accounts,
`http://www.exploit-db.com/exploits/15507/`

[11] Introduction to Cross Site Request Forgery,
`http://www.isecpartners.com/files/CSRF_Paper.pdf`