Secure IP Camera

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

Project Guide: Prof. Ramesh Kini M
Dept of E & C Engg

In collaboration with Prof. Oliver Bringmann & Sebastian Burg
University of Tuebingen, Germany

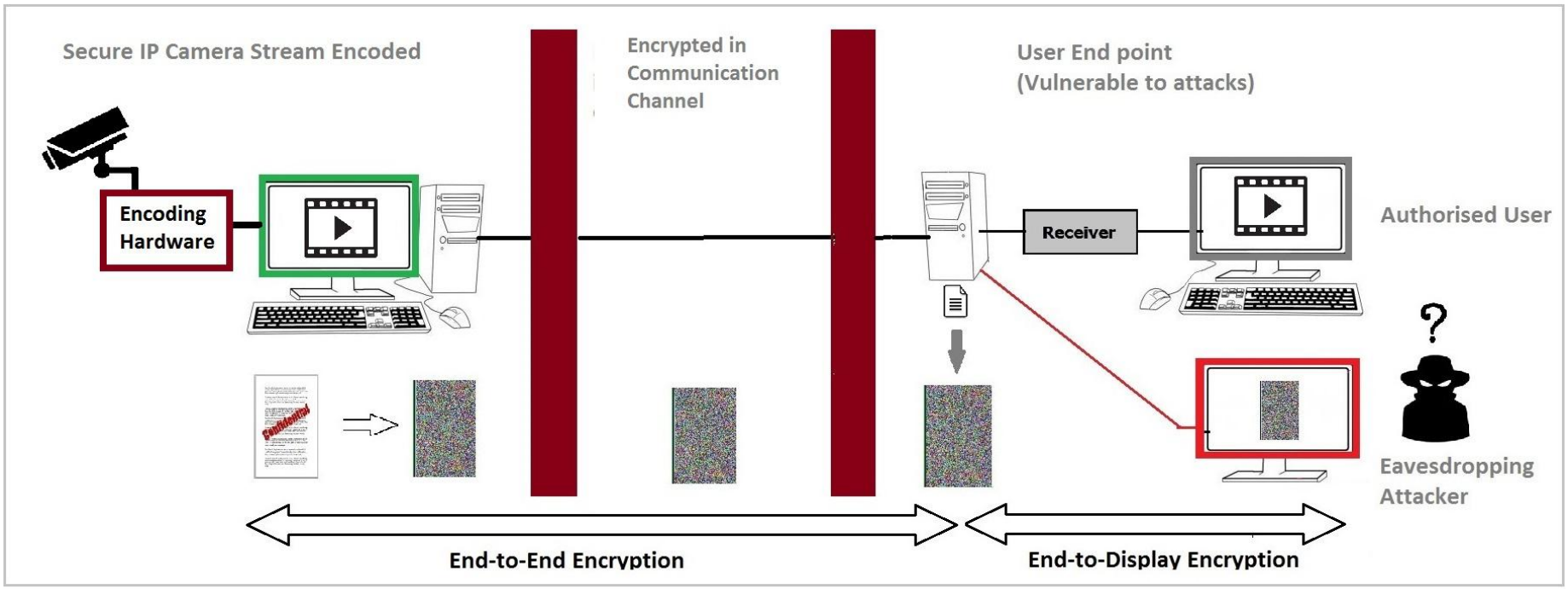
January - April 2016

Overview

- E2DE Overview
- Encoding Format
  - Compression
  - Encryption
- Implementation Results
- Verification
- Conclusion
- Learning Outcomes

E2DE Overview

- Concept of End-to-Display Encryption (E2DE)
- Extending protection of End-to-End encryption to beyond endpoint
- Leverage the strengths of E2DE for securing camera

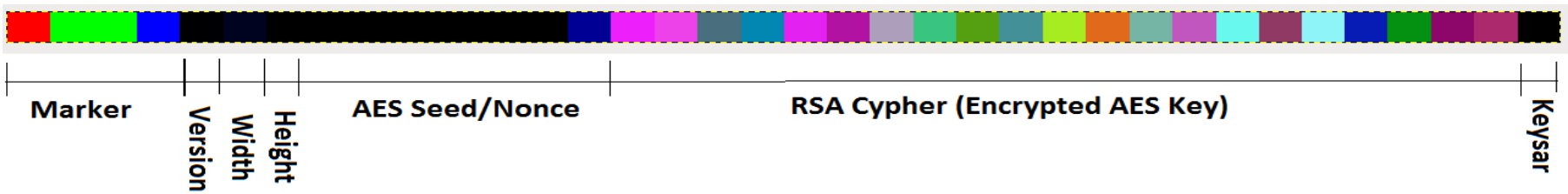


Encoding format

1. Pixel Domain Encryption using 128 bit AES
   - Image pixels encrypted using Advanced Encryption Standard(AES)
   - 128 bit AES vector XORed with 5 sets of 24bit RGB pixels
2. Compression of Image
   - Image compressed using Run length encoding
   - Ensures the image size remains the same
   - Compressed pixels represented in pairs {pixel colour, count}

Encoding format

3. Writing header pixels
   - Metadata for E2DE receiver to decode
   - AES seed for each line
   - AES key encrypted in RSA
   - Enables line - wise decoding
   - 36 pixels wide



Pixel Domain Encryption Overview

- AES seed increment for blocks of 5 pixels
- Open-source AES module used

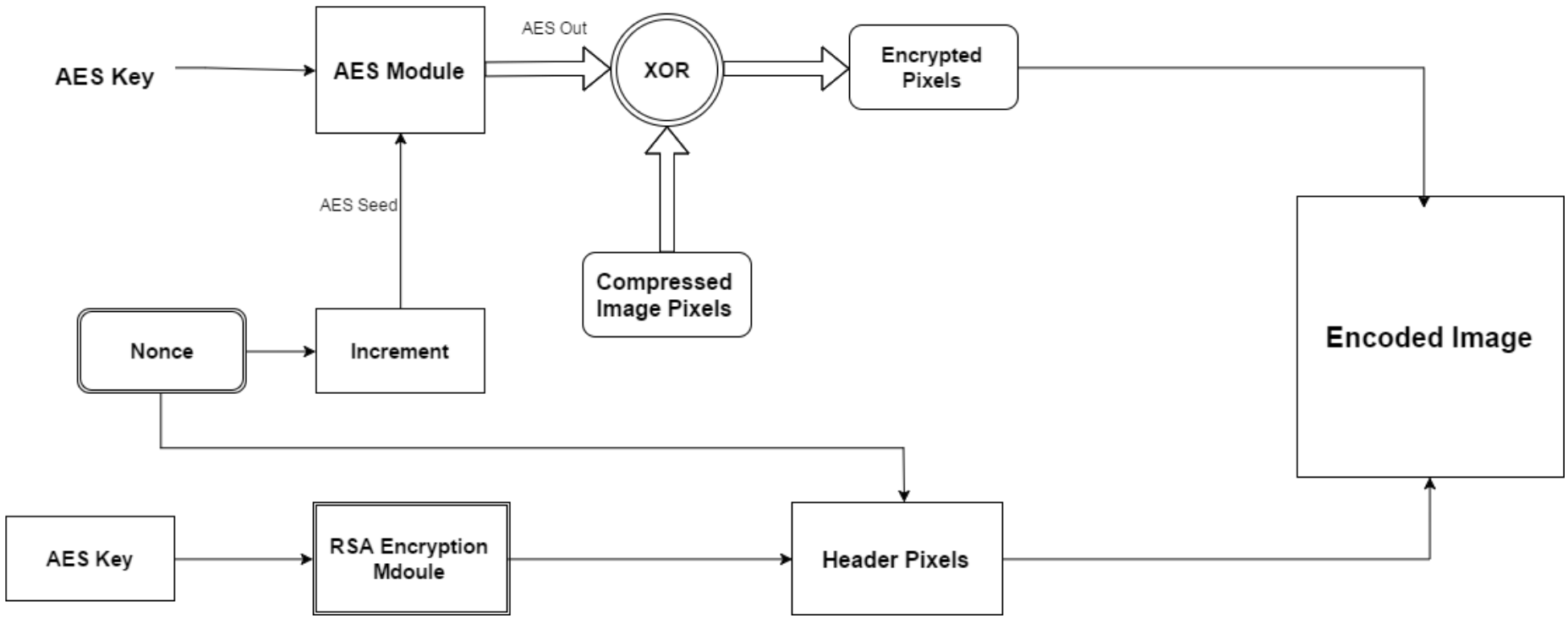


Figure: Data flow for encoding module

Pixel Compression Approaches

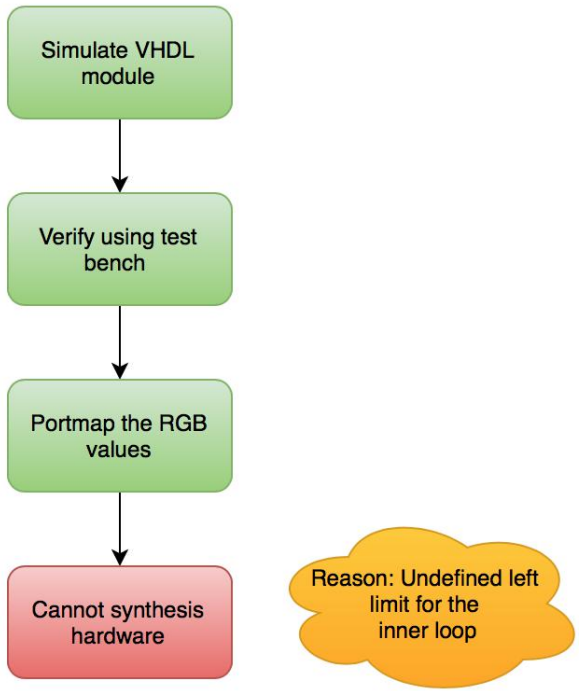
- Run length encoding
- 3 approaches for implementation
- First, simulation verified with testbenches
- Then, real-time implementation on FPGA

Pixel Compression - Approach 1

```
Initialization i=0;
for each i in the pixel buffer array do
    Initialization j=0;
    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
```

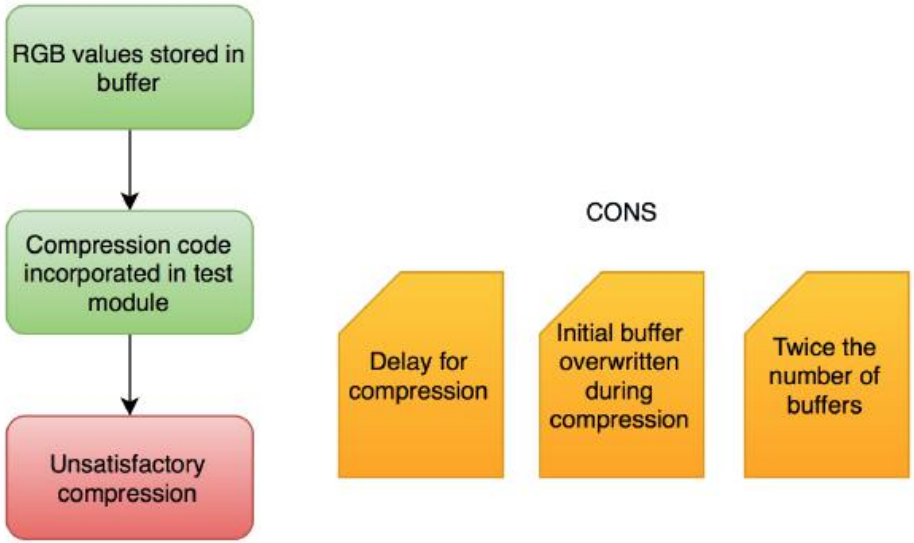
Approach 1

- VHDL module simulated
- Verified using testbench
- Error:Couldn't synthesise hardware
- Reason: Undefined left limit of for loop



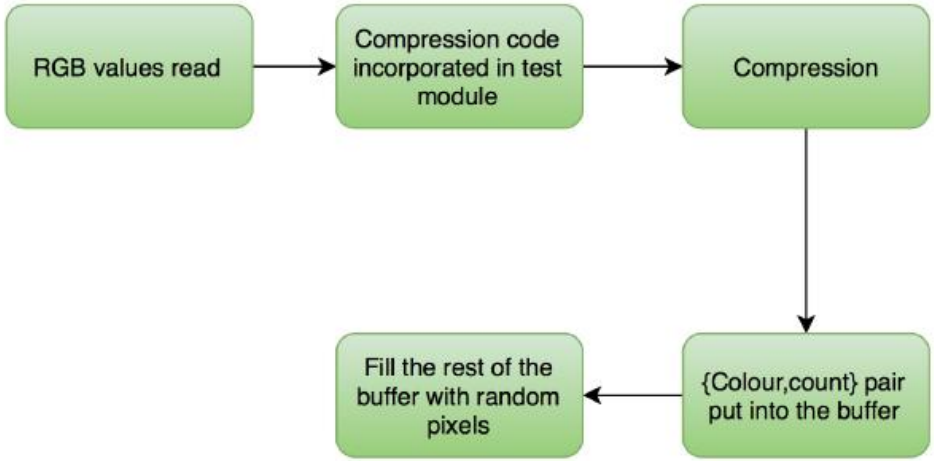
Approach 2- Traversal based compression

1. Traverse through data buffer
2. Compress pixels



Approach 3- Read based compression

- Compression performed as they are input to buffer
- {Colour, count} pixel pair put into buffer only after compression
- Optimised approach for compression



Implementation Results

Hardware Setup - Camera Configuration

- CMOS camera - OV7670.
  - Low voltage CMOS image sensor
  - Serial Camera Control Bus (SCCB) interface
  - Image array capable of operating at up to 30 frames per second.
- Development kit - Zedboard (Xilinx Zynq FPGA)
- Pmod connection between camera and Zedboard
- Camera output observed through VGA output

Camera Configuration

- Adopted interface code for Zedboard with VGA output
- Needed colour register correction as suggested by Hamsterworks[2]

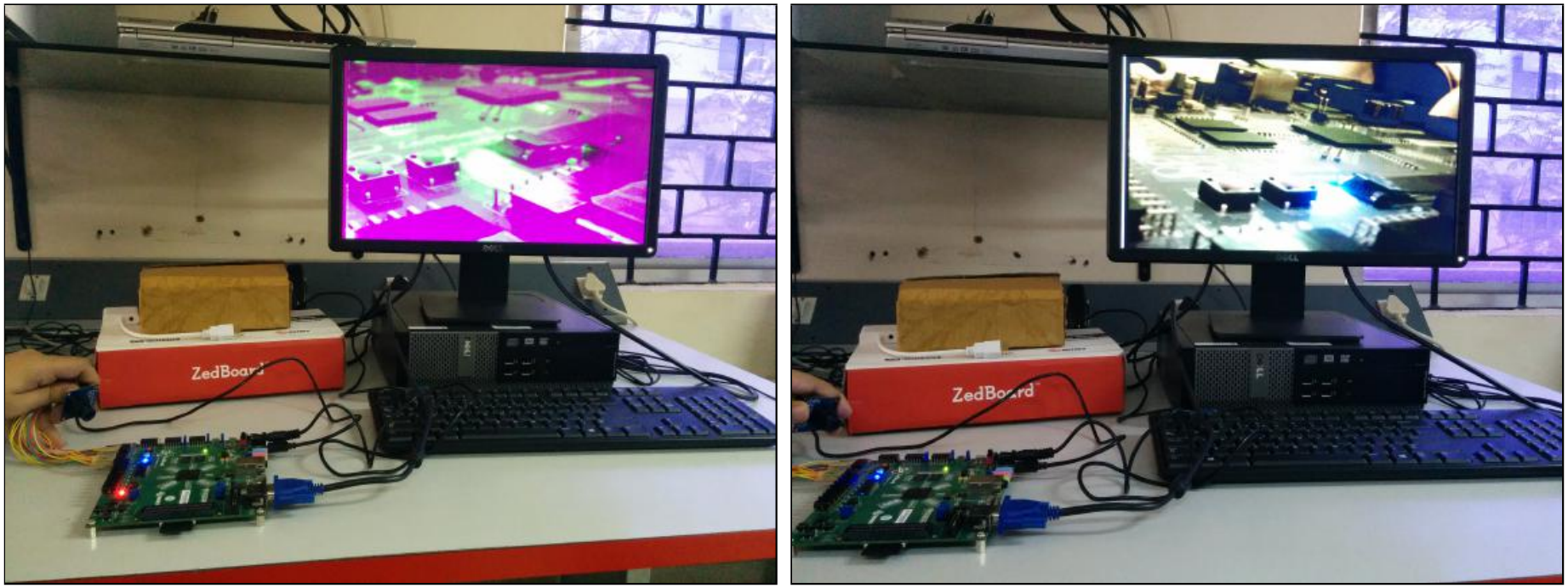


Figure: VGA output of the OV7670 without and with colour correction

Compression

- Read based compression
- Fill remaining parts of the line with black pixels
- Inherent repetition of data

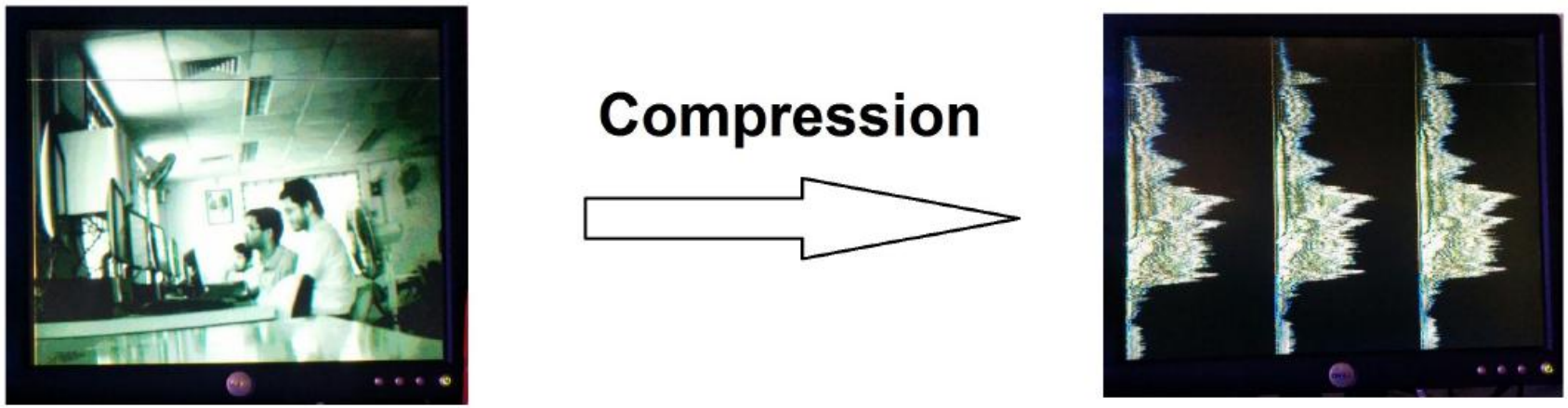


Figure: A snapshot of a frame compressed using RLE.

Compression

- Comparison of compression with different similarity indices
- Determines how lossy the compression is
- Higher the index, higher the compression and loss

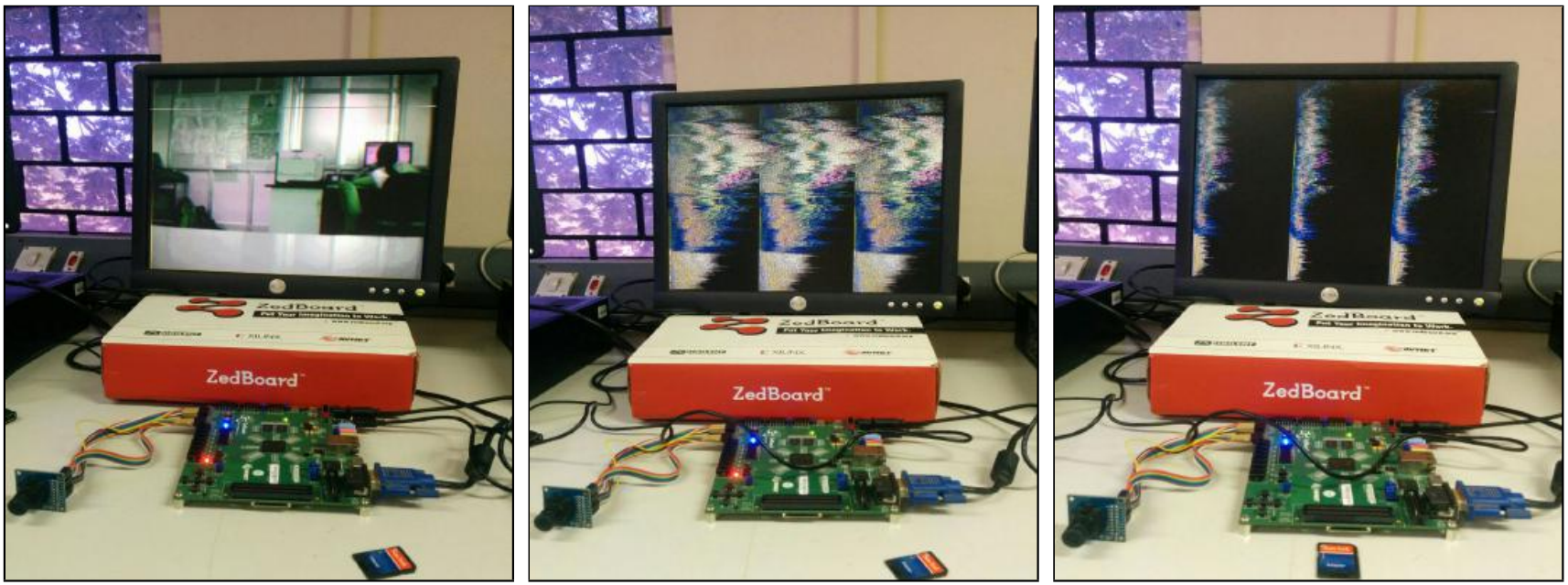


Figure: A comparison of the outputs for different extent of compression.

Pixel Domain Encryption

- Real-time encryption observed on screen
- Pixels stored in buffers for compression and header pixels
- Introduces delay

Pixel Domain Encryption

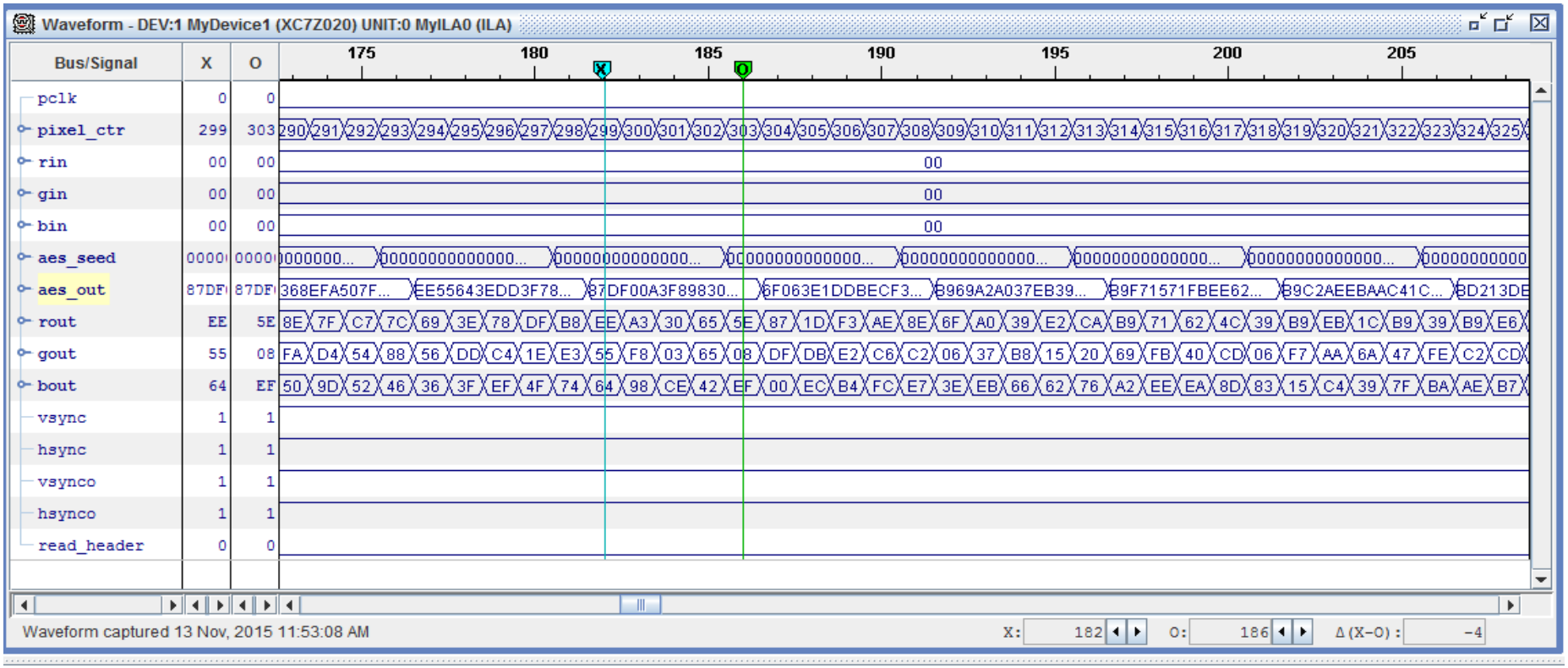


Figure: ChipScope graph of the pixel-domain encryption using 128 bit AES

Debugging and Verification

Debugging and Verification

1. Debugging using ChipScope Pro
   - Introduced ILA cores to observe changes in values in buffer and flags
   - Ensured correct timing of signals for compression and encryption
2. Export values for verification
   - Export RGB values to CSV file
   - Parse CSV file using Java applet
   - Recreate image from exported RGB value and save as PNG file
   - PNG - supports a lossless data compression

Compression Verification

- Java applet to perform decompression
- Recognise hybrid representation of Colour, Count
- Recreates runs of data accordingly

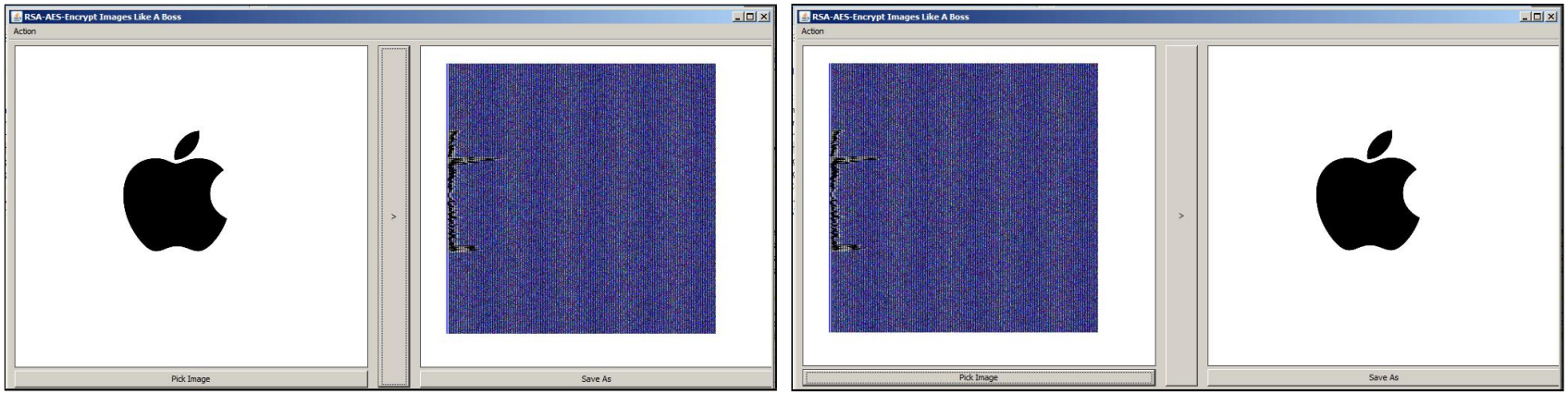


Figure: Compression and decompression demo in Java

Compression Verification

- Decompression performed on the the compressed image
- Part of the compressed image is recreated from exported ChipScope values

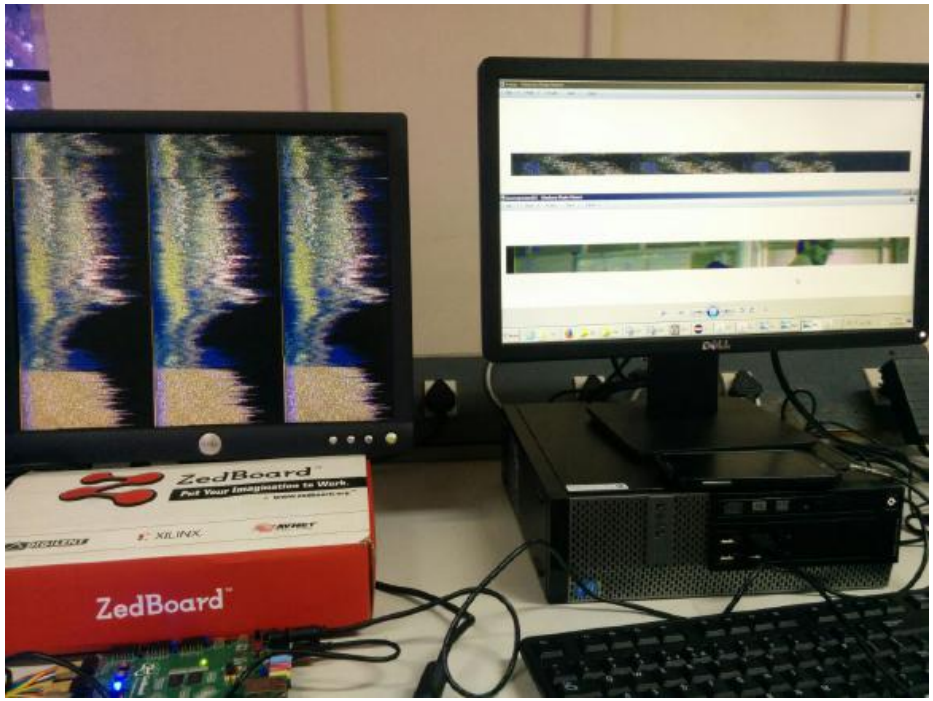


Figure: A snapshot of the decompression software, displayed on the monitor on the right, along with the compressed output seen on the monitor on the left.

Compression Verification

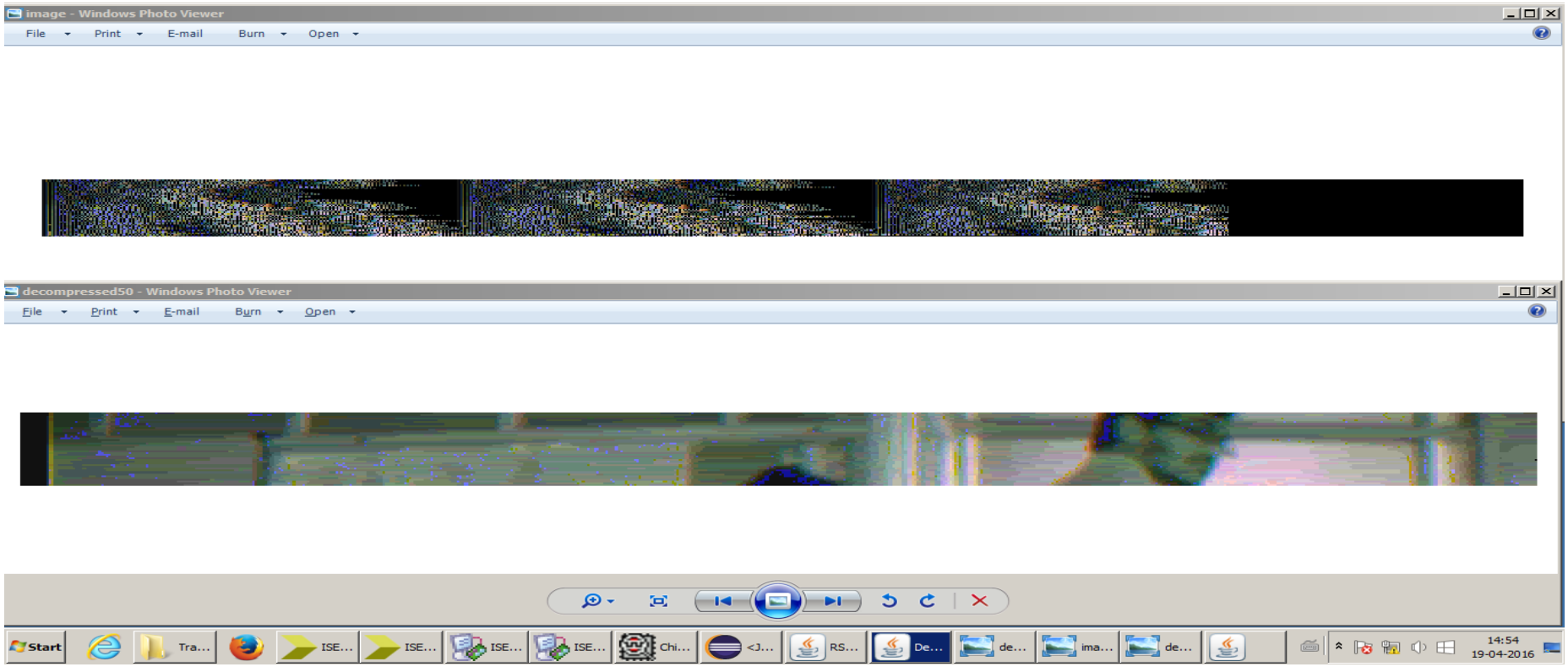
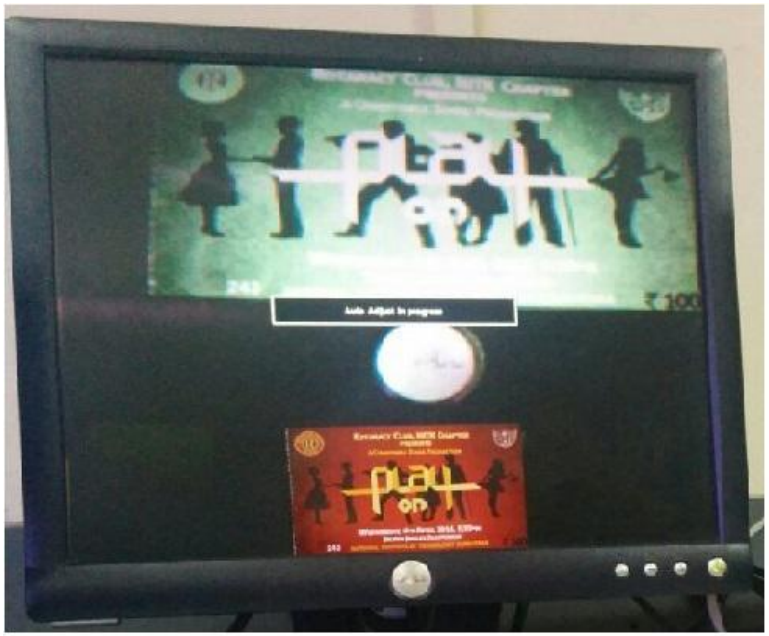


Figure: A zoomed in screenshot of the decompression result of the decompression shown in the previous figure.

Compression Verification

- Comparison of compression with sim index 10 and 100
- Image with sim index 100 more lossy than that with sim index 10



ORIGINAL IMAGE

Similarity Index = 100

Similarity Index = 10

DECOMPRESSED IMAGES

Encryption Verification

- 128-bit AES encryption
- AES seed increments affects the encrypted frame
- Moving object can be spotted if same seed is used in different frames

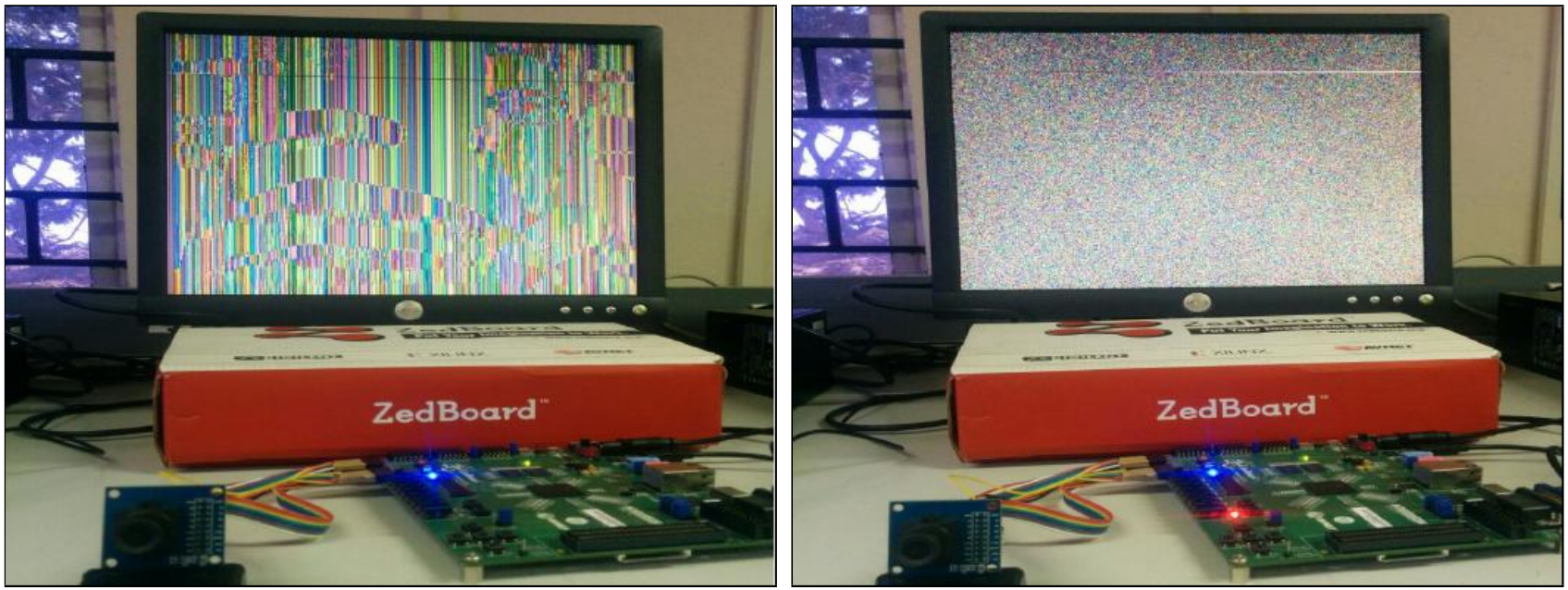


Figure: A comparison of the outputs for encryption with and without the seed reset for every line. With the seed reset for every line (left), the image has clear correlation to the outline of the original image while this is not the case for the latter.

Encryption Verification

- Decrypted using java applet
- Used Cipher class in Java Cryptographic Extension Framework
- Provides funstionality of a cryptographic cipher for encryption

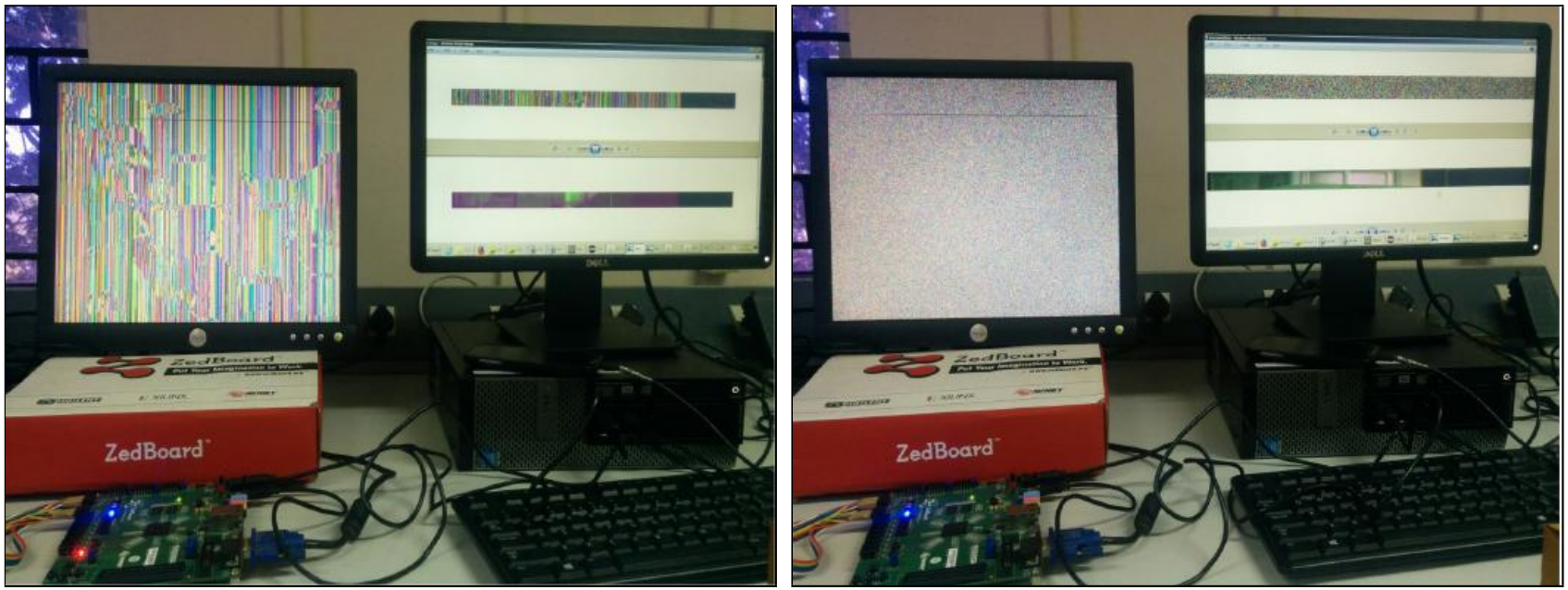
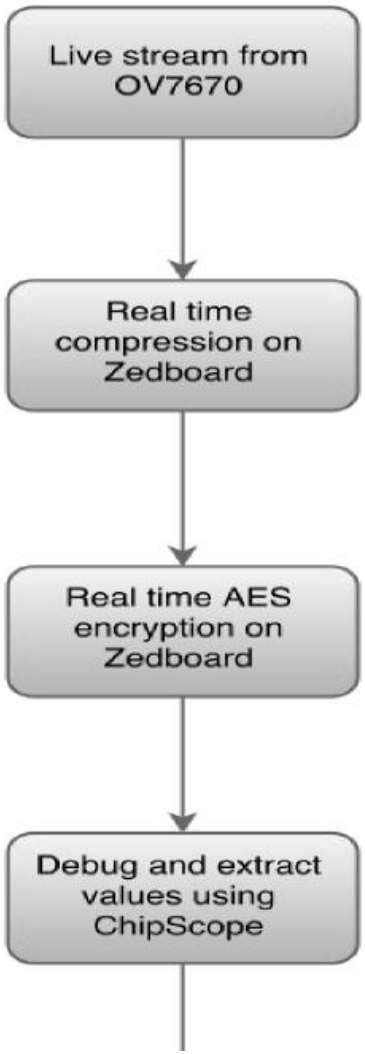
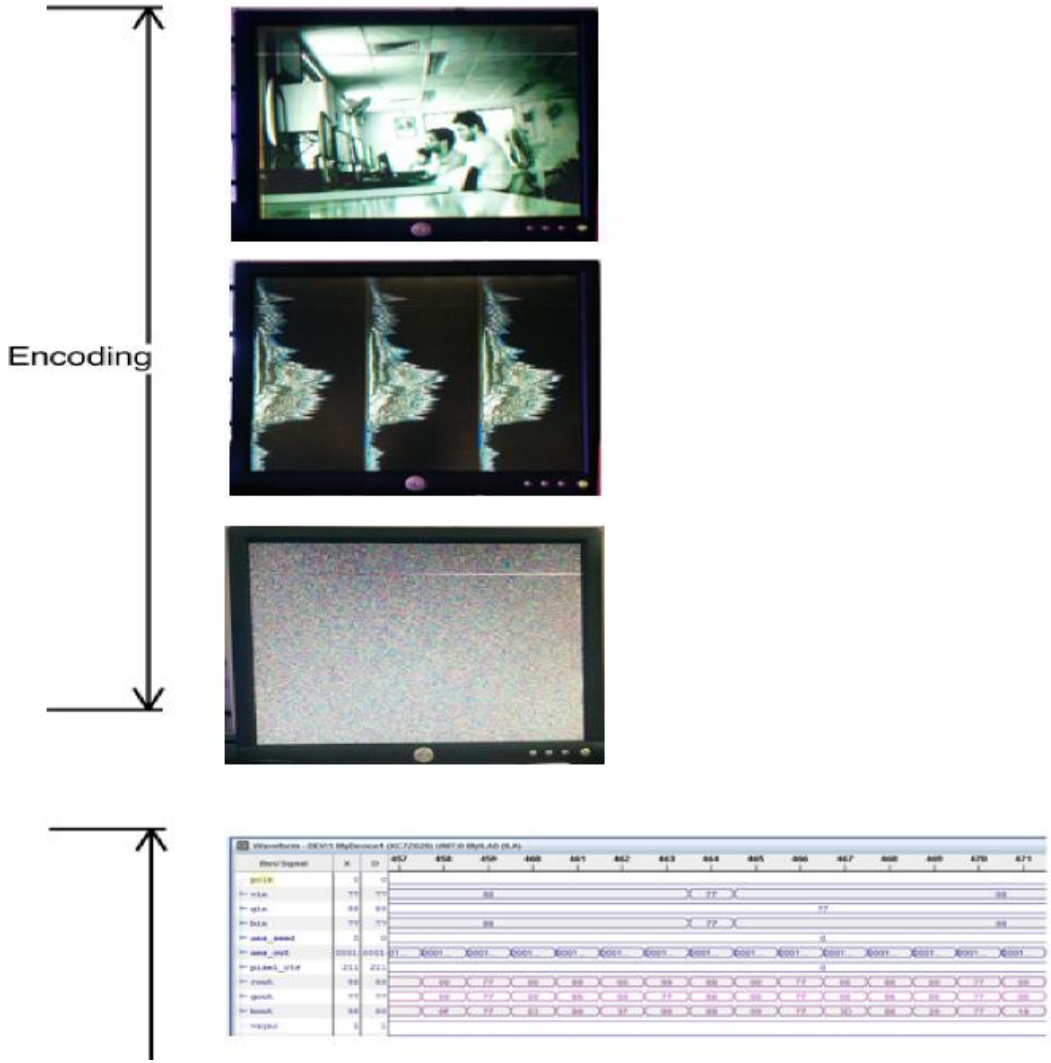


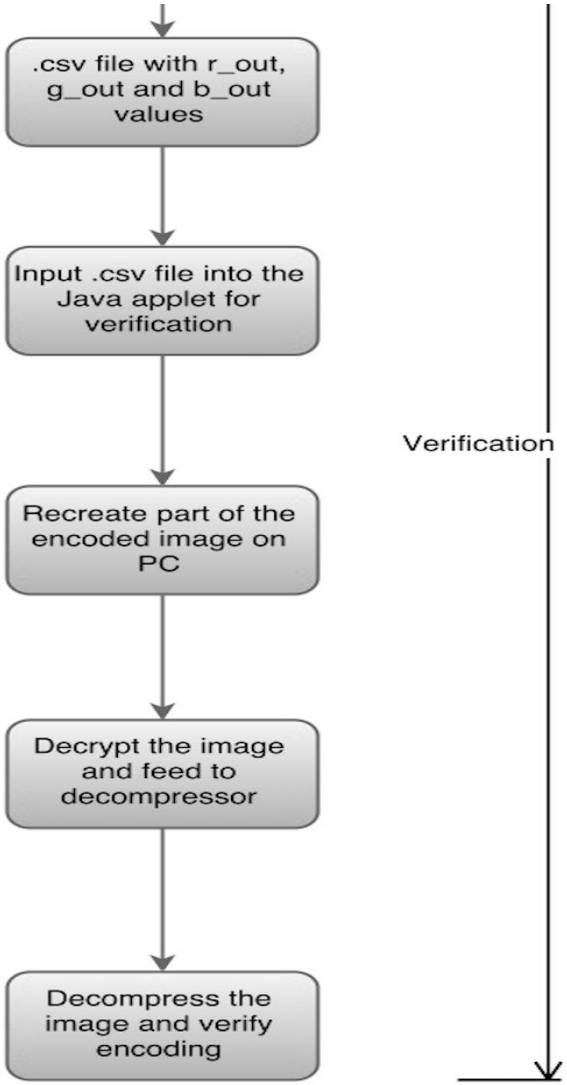
Figure: Decryption of the encrypted frame, with seed reset for every line (left) and without the reset (right), observed in the monitor on the right respectively for both pictures.

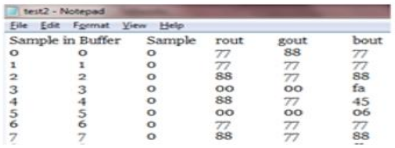
Complete Flow chart





Complete Flow chart




test2 - Notepad

File Edit Format View Help

Sample in Buffer	Sample	rout	gout	bout
0	0	77	88	77
1	0	77	77	77
2	0	88	77	88
3	0	00	00	fa
4	0	88	77	45
5	0	00	00	06
6	0	77	77	77
7	0	88	77	88

Conclusion

- 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
- 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

Learning outcomes

- 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

Thank You

References

1. Sebastian Burg, Pranav Channakeshava, Oliver Bringmann. *"Linebased End-to-Display Encryption for Secure Documents"*; In the proceedings of the 2nd IEEE Int'l 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. Mike Field, OV7670 Zedboard `http://hamsterworks.co.nz/mediawiki/index.php/Zedboard_OV7`
4. Marc Defossez, *D-PHY Solutions* Application Note: Spartan-6 and 7 Series FPGAs, August 25, 2014
5. H. Hsing. Tiny AES - Crypto Core. `http://opencores.org/project/tiny_aes` October 2013