Northumbria University
NEWCASTLE

Dissertation

EN0765 MSc Engineering Project

Student Name: Alexandr Kuzmenko

Supervisor Name: Hoa Le-Minh

Implementation of Camera Based Visible Light Communication on FPGA

2018/2019

Declaration

I declare the following:

(1) that the material contained in this dissertation is the end result of my own work and that due acknowledgement has been given in the bibliography and references to **ALL** sources be they printed, electronic or personal.

(2) the Word Count of this Dissertation (without appendixes) is 6568 (21692 with).

(3) that unless this dissertation has been confirmed as confidential, I agree to an entire electronic copy or sections of the dissertation to being placed on the eLearning Portal if deemed appropriate, to allow future students the opportunity to see examples of past dissertations. I understand that if displayed on eLearning Portal it would be made available for no longer than five years and that students would be able to print off copies or download.

(4) I agree to my dissertation being submitted to a plagiarism detection service, where it will be stored in a database and compared against work submitted from this or any other School or from other institutions using the service.

In the event of the service detecting a high degree of similarity between content within the service this will be reported back to my supervisor and second marker, who may decide to undertake further investigation that may ultimately lead to disciplinary actions, should instances of plagiarism be detected.

(5) I have read the Northumbria University Policy Statement on Ethics in Research and Consultancy and I confirm that ethical issues have been considered, evaluated and appropriately addressed in this research.

SIGNED: A.Kuzmenko

DATE: 08/09/2019

Abstract

Today, ongoing growth and development of wireless communications requires higher and higher bandwidths requirements, that leads to a problem of spectrum saturation. An alternative that can help with this capacity issue is the visible light communication.

This project targets to extend existing Visible Light Communication system to FPGA domain. It aims to merge together two different technologies: digital image processing and visible light communication. Previous studies presented a concept of smartphone-to-smartphone communication link in which screen of smartphone works as a transmitter, showing data encoded into images on screen. By designing an image processing algorithm for FPGA, receiver for this short-range VLC link can be created. It would allow to establish VLC communication with whole new family of devices. Design of this algorithm is a main point of this project.

The algorithm was designed and implemented on Spartan-3e FPGA. A range of experiments was taken to test algorithm's performance. Despite its limited abilities to extract data from rotated or tilted screen, algorithm can recover data from image in less than 2ms, which, considering achieved data density of 960 bits per frame, can provide a link with 480kbps data transmission capability.

Contents

Declaration **1**

Abstract **2**

1 Introduction **6**

1.1 Aims 6

1.2 Background 6

1.2.1 Theory and importance of visible light communication 6

1.2.2 Literature review 6

1.2.3 Reasons for the work 7

1.3 Objectives 8

1.4 Work Done and Results 10

1.4.1 Conclusion of the research analysis 10

1.4.2 Created algorithm and its performance 10

1.5 Structure of the report 12

2 Theory **13**

2.1 Visible Light Communication 13

2.1.1 Digital camera 13

2.1.2 Image recognition concept 13

2.2 FPGA 14

2.2.1 Overview 14

2.2.2 Internal structure 15

2.2.3 Hardware Description Languages 18

2.2.4 Comparison with ASICs 19

3 Project plan **20**

3.1 Work Breakdown Structure 20

3.2 Task List 21

3.3 Gantt Chart 22

4 Implementation **23**

4.1 Hardware equipment 23
4.2 Algorithm structure 24
4.2.1 System Block Diagram and overall design 24
4.2.2 Schematic symbols 26
4.2.3 Input block and data compression 28
4.2.4 Data saving 30
4.2.5 Data analysing (Haar functions) 32
4.2.6 Rows borders searching 34
4.2.7 Cells borders searching 37
4.2.8 Data extraction 40
4.3 Tests 42
4.3.1 Tests' design and experimental set up 42
4.3.2 Overall performance tests 44
4.3.3 Grayscale image test 46
4.3.4 Cell amount test 48
4.3.5 Sequence test 51
4.3.6 Screen rotation test 54
4.3.7 Screen tilting test 55
4.3.8 Maximum distance test 56
5 Conclusions 57
5.1 Overall achieved results 57
5.2 Suggestions for further study 57
5.2.1 Hardware update and external memory usage 57
5.2.2 Rotation compensation and perspective correction 58
Bibliography 59
Appendix A. Research Proposal 61
Abstract 63
Introduction and General Overview 63
Project Aim and Objectives 67
Project Plan 70
Financial Justification 73
Risks, ethical and legal assessment 74
Conclusion 75

References 76
Appendix B. Verilog code 77
Algorithm main module 77
Testbench file for simulation 123
User Constrain File for implementation 128
Matlab code for bitmap-to-hex converter 136

1 Introduction

1.1 Aims

This project aims to expand existing Smartphone-to-Smartphone VLC system for the case of Smartphone-to-Terminal and Machine-to-Machine VLC, particularly design and implementation of receiver domain on FPGA. The main focus is a digital image processing algorithm that will recover transmitted data.

1.2 Background

1.2.1 Theory and importance of visible light communication

Nowadays, the continuous growth of telecommunication network leads to higher and higher bandwidths requirements, facing a problem of spectrum saturation. One of the possible alternatives to help with this capacity issue is the visible light communication. The core of the VLC technology is the intensity modulation of LEDs that can be switched on and off at a very high rate, enabling data communications. LEDs are widely used in everyday infrastructures including homes, offices, street and traffic lights and smartphones. In display devices such as smartphones or computers, the individual elements of the pixel arrays can be independently modulated and captured with a camera in order to recover the transmitted information (Boubezari *et al.*, 2016).

1.2.2 Literature review

In her PhD thesis R.Boubezari (2017) combined VLC and image processing technologies, creating a camera-based smartphone-to-smartphone communication. It uses a sequence of grayscale images that are sent from smartphone screen to other smartphone's camera at certain frame rate and then detected and analyzed at real time using SURF descriptor. Block diagram and picture of data frames of this system are shown below:

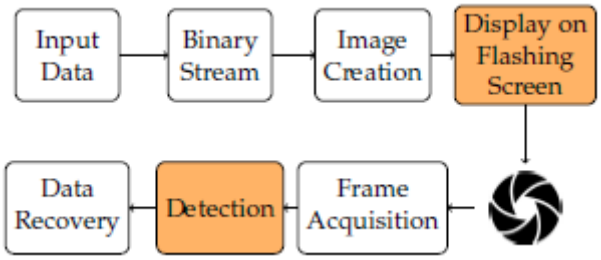


Figure 1. VLC block diagram.

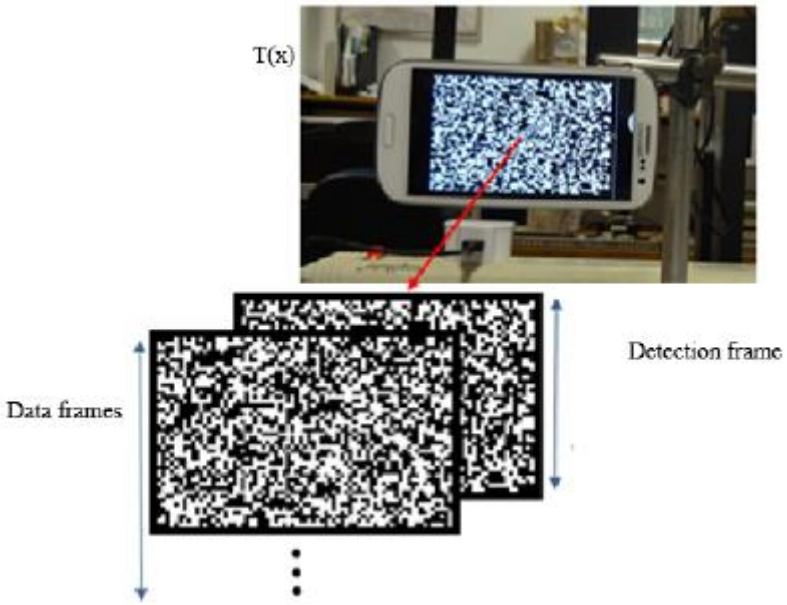


Figure 2. Data frames.

Discussing the results, it was pointed out that an extension to other devices would be very useful. A vast variety of devices can be covered by creating a VLC receiver for embedded systems.

In Sledevich & Serackis conference paper (2012) it was shown that real-time image processing algorithm can be implemented at FPGA with achievable recognition frame rate higher than average digital cameras' capturing framerate.

Combining these researches, it might be assumed that it is possible to use FPGA to design a camera based VLC receiver.

1.2.3 Reasons for the work

This project aims to extend previous development in camera based visible light communication for FPGA domain. Digital image processing algorithm will be

designed and implemented on FPGA, which opens a possibility to communicate with whole new family of devices.

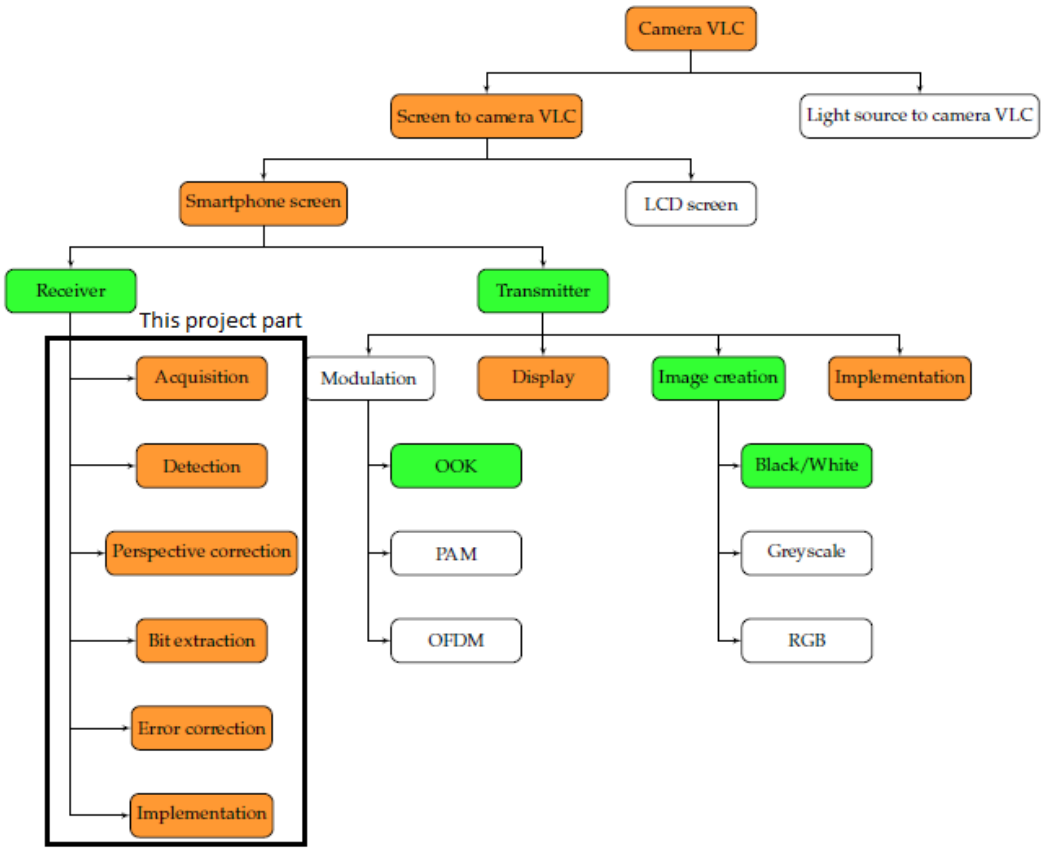


Figure 3. A place of the project in camera based visible light communications domain.

1.3 Objectives

To achieve the aim of the project, the following key objectives should be completed:

	Objective	Done?
1	To obtain theory of different image processing algorithms using Boubezari's PhD thesis and its references as well as Northumbria University Library Search	Yes
2	To understand how image processing can be done on FPGA by finding researches about implementation of image processing on hardware	Yes
2.1	To write 'Theory' chapter of the report	Yes
3	To borrow hardware equipment from University and evaluate its possibilities	Yes

4	To choose digital image processing algorithm that would fit requirements of this project	No
4.1	To design new algorithm that *can* be implemented on provided FPGA	Yes
5	To write an algorithm's code in Verilog HDL	Yes
6	To run behavioural simulation of algorithm and check whether it works correctly	Yes
7	To load designed algorithm on FPGA, assemble VLC system and run tests to check its performance	Yes
8	To redesign algorithm and improve its performance	No
9	To write implementation report	Yes

Figure 4. Project objectives.

Not all of these objectives were completed. Objective 4 was found to be unachievable, because all existing image processing algorithms works with whole saved image, while provided FPGA didn't have enough internal recourses to store whole image. Therefore, new algorithm had to be designed from scratch, which is a new objective 4.1.

Objective 8 was not finished due to the limited time of the project.

1.4 Work Done and Results

During this project, a new digital image processing algorithm was designed to extract data from captured image of transmitter. The main limitation of the algorithm was a size of hardware. Possibilities and suggestions for performance improvements are discussed in the last chapter of this report.

1.4.1 Conclusion of the research analysis

After obtaining required equipment from university and after analysis of researches in digital image processing field it was concluded that none of the existing algorithms can possibly be implemented on provided FPGA. Therefore, it was decided to create new digital image processing algorithm, implement it on FPGA and test created system.

1.4.2 Created algorithm and its performance

System block diagram is shown on figure 5:

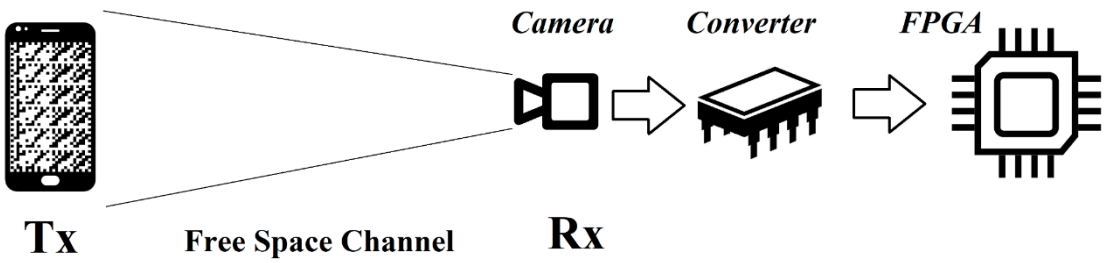


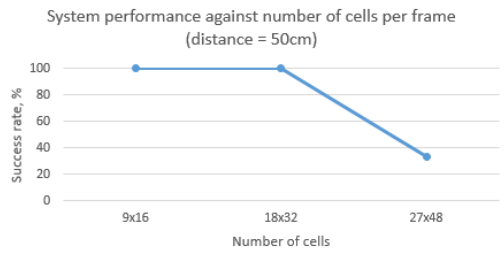
Figure 5. System block diagram.

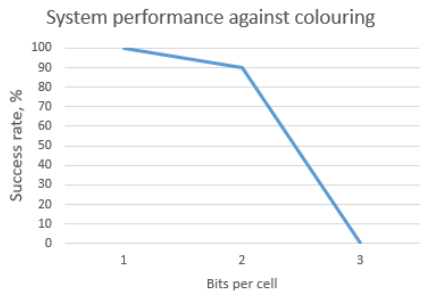
VLC system consist of transmitter, which is in this case a smartphone that shows information encoded in grayscale cells on screen; and receiver, that captures data by capturing image of the Tx screen with camera and then send this image to FPGA to analyse it and extract data.

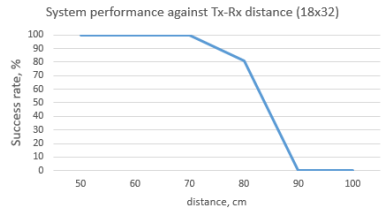
Developed algorithm for FPGA consists of 6 main blocks to compress, store and analyse incoming image. Algorithm uses Haar functions to detect borders of cells on captured image of Tx screen and compares saturation of cells with thresholds. See Chapter 4 for details. Performance of the system was tested, results are shown in figure below:

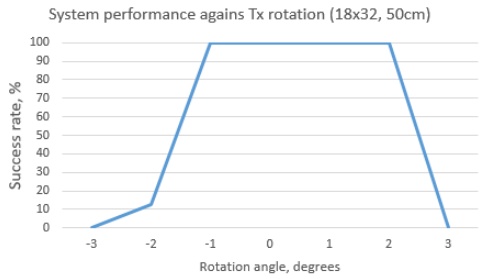
Input images: 350x180 px

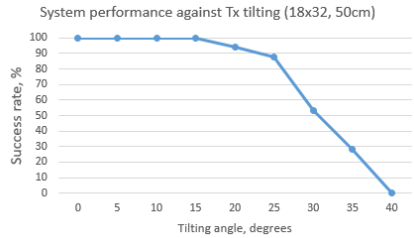
Speed: 2ms per picture at 100MHz clock.











Optimal parameters	
Parameter	Value
Optimal transmitting distance, cm	50-70
Optimal number of cells	18x32
Max number of colours per cell	4
Data per frame	960
Max rotation angle, degrees	~2
Max tilting angle, degrees	~20

Figure 6. Achieved results.

1.5 Structure of the report

The report is organized in five chapters as follows:

- Chapter 1 – Introduction

The first chapter provides a brief overview of the project, its motivation, aims, objectives and results.

- Chapter 2 – Theory

This chapter describes a theory of two main areas related to this project: Visible light communication and FPGAs. First part provides an overview of image recognition concept and digital image processing algorithms, while second part dedicated to the theory of FPGA, their internal structure and how they are used.

- Chapter 3 – Project plan

The third chapter shows plan of the project with work breakdown structure, task list and Gantt Chart with timings for these tasks.

- Chapter 4 – Implementation

This is the main and the largest part of the report. It contains of two major parts. One of them is dedicated to explanation of designed algorithm's structure with in-depth details about all its blocks and their functions. The other part of the chapter shows results of experiments with assembled system.

- Chapter 5 – Conclusions

The last chapter concludes the work done by presenting achieved results. Suggestions for improvements and future studies are also shown in this chapter.

2 Theory

2.1 Visible Light Communication

2.1.1 Digital camera

The key component in digital camera is an array of image sensors that adsorbs photons that travelled through camera lenses. Photons are converted into electrical signal with an amplitude proportional to captured photon's energy (i.e. wavelength). The resolution of digital image depends on the number of camera's sensors. Two types of image sensors which are used in digital cameras are complementary metal-oxide semiconductor sensor (CMOS) and charge-coupled device sensors (SSD).

2.1.2 Image recognition concept

Digital image processing is multidimensional subcategory of the digital signal processing. It is a challenging task due to the dynamic environment that changes illumination, position and other target properties from frame to frame.

Object detection usually uses the concept of keypoints that describe specific parameters of the image such as intensity and orientation of pixels, corners, etc. Object detection, therefore, consists of three steps: keypoints selection, keypoints description and keypoints matching. Keypoints detection is usually done by image analysing using wavelets.

For this project, while real-time recognition is required, SURF (speed up robust features) algorithm was planned to be used, however, it was found that it's impossible due to the hardware limitations. See 'Implementation' chapter for details. The SURF algorithm uses Haar wavelet and a set of 64-elements descriptors to detect keypoints of the received image (Bay *et al.*, 2008).

There are some other existing VLC systems that ought to be mentioned. QR code is a well succeeded technology which uses the same main principle of edges and orientation detection and is usually used as a "physical hyperlink" to connect items to websites (Belussi & Hirata, 2011). COBRA is a VLC system that uses colour barecodes transmitting them in a similar way (Hao *et al.*, 2012).

2.2 FPGA

2.2.1 Overview

FPGAs (field-programmable gate arrays) are an integrated circuits that designed to be configured (programmed) *after* manufacturing. FPGA consists of configurable logic blocks with programmable connections between them, input/output blocks and global routing clock network. This provides an obvious advantage comparing with other chips: flexibility and reconfiguration, which leads to less cost, shorter design process (not include manufacturing of chip) and reduce negative impact of design errors, because it can be fixed at any stage (Dorta *et al, 2009).*

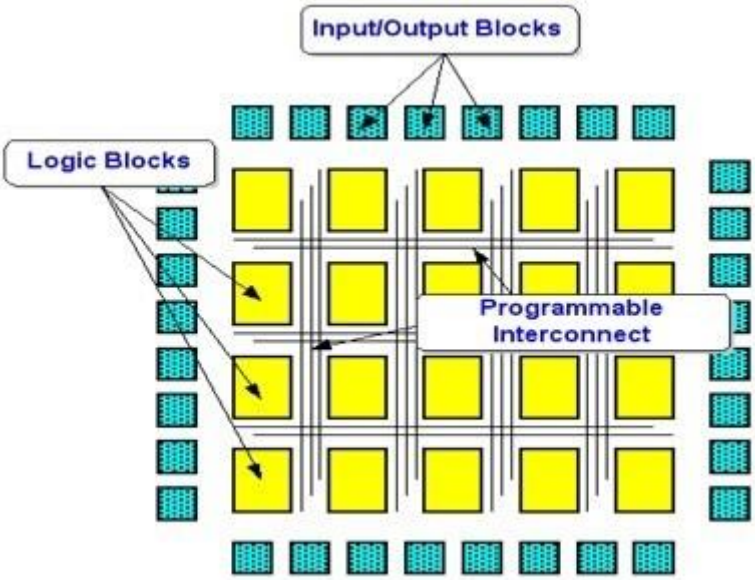


Figure 7. FPGA block diagram structure.

For this particular project, FPGA Spartan-3 by Xilinx was chosen. This FPGA is comparably small and simple (and therefore low-cost) and has a version optimized specifically for Digital Signal Processing (xilinx.com, 2019).

2.2.2 Internal structure

- Configurable Logic Blocks (CLBs)

Configurable logic blocks are the main part of FPGA. They can be programmed to make flip flops, memory, shift registers, logic, etc. Structure of CLB is shown on figure below. It consists of several 'slices' and connects with other CLBs through Switch Matrix.

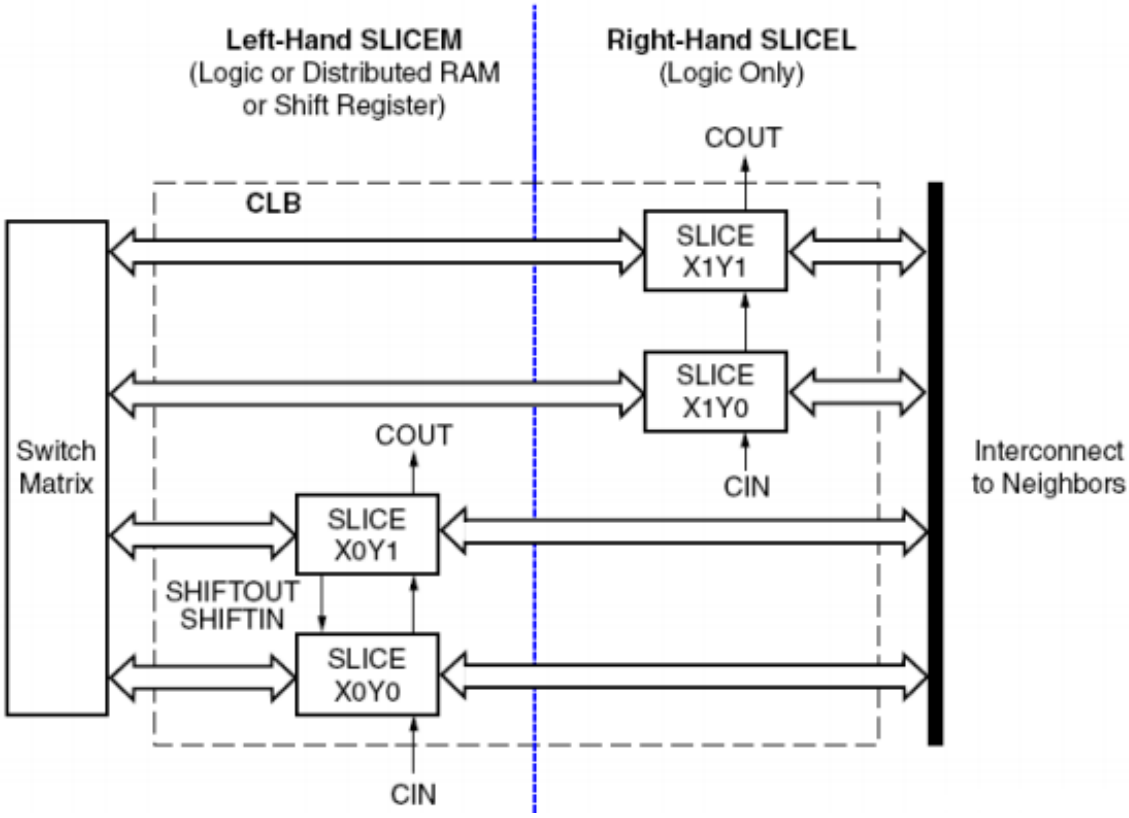


Figure 8. Internal structure of CLB.

Slices contain LUTs – look-up tables that are basic building blocks that can implement any logic functions of up to 4 inputs. Figure below shows structure of a slice.

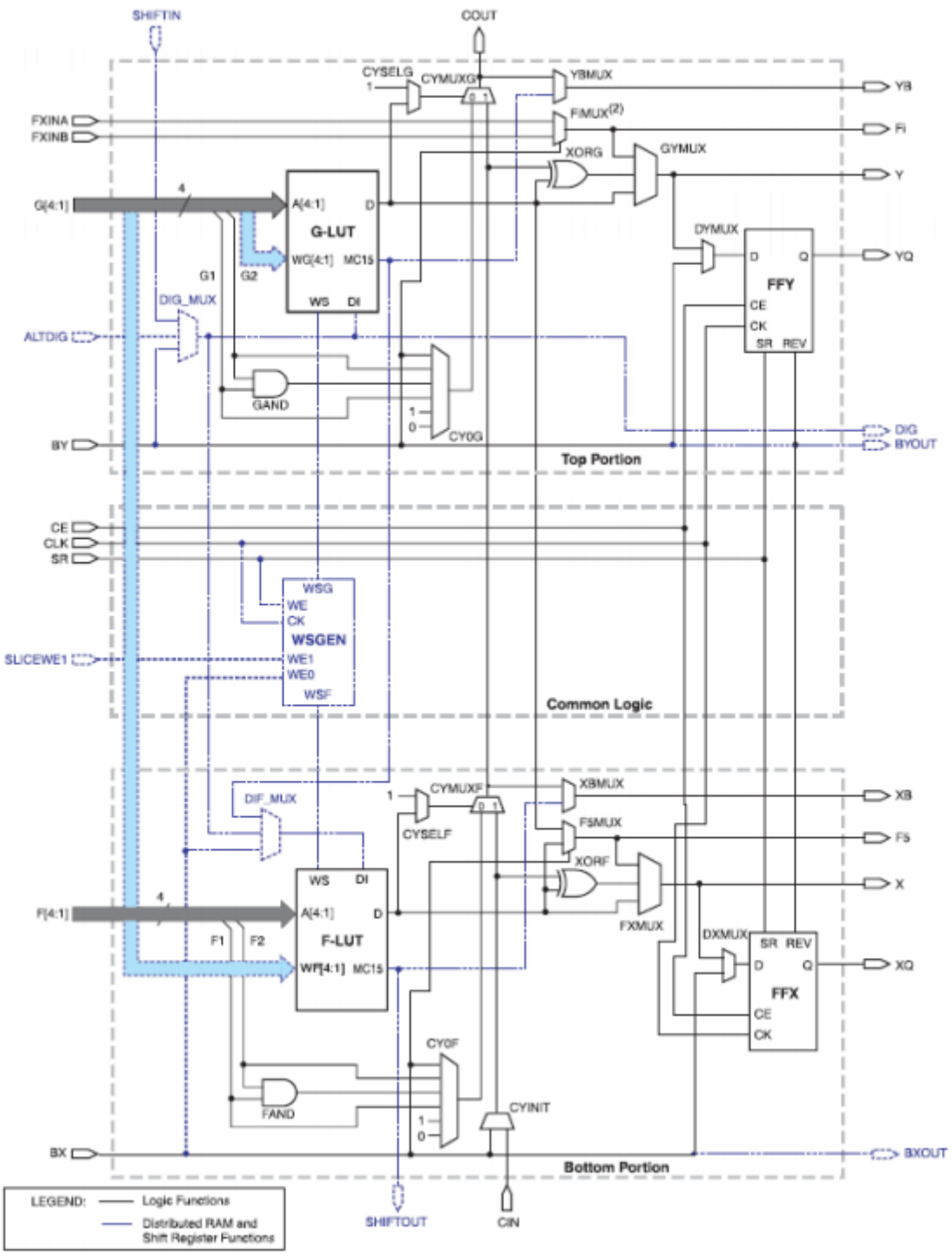


Figure 9. Internal structure of a Slice.

- IO Blocks

Input/output blocks connects FPGA to the outside world. Usually there are different types of IOBs with different signalling voltages and driving currents. Internal structure of input/output block is shown on figure below.

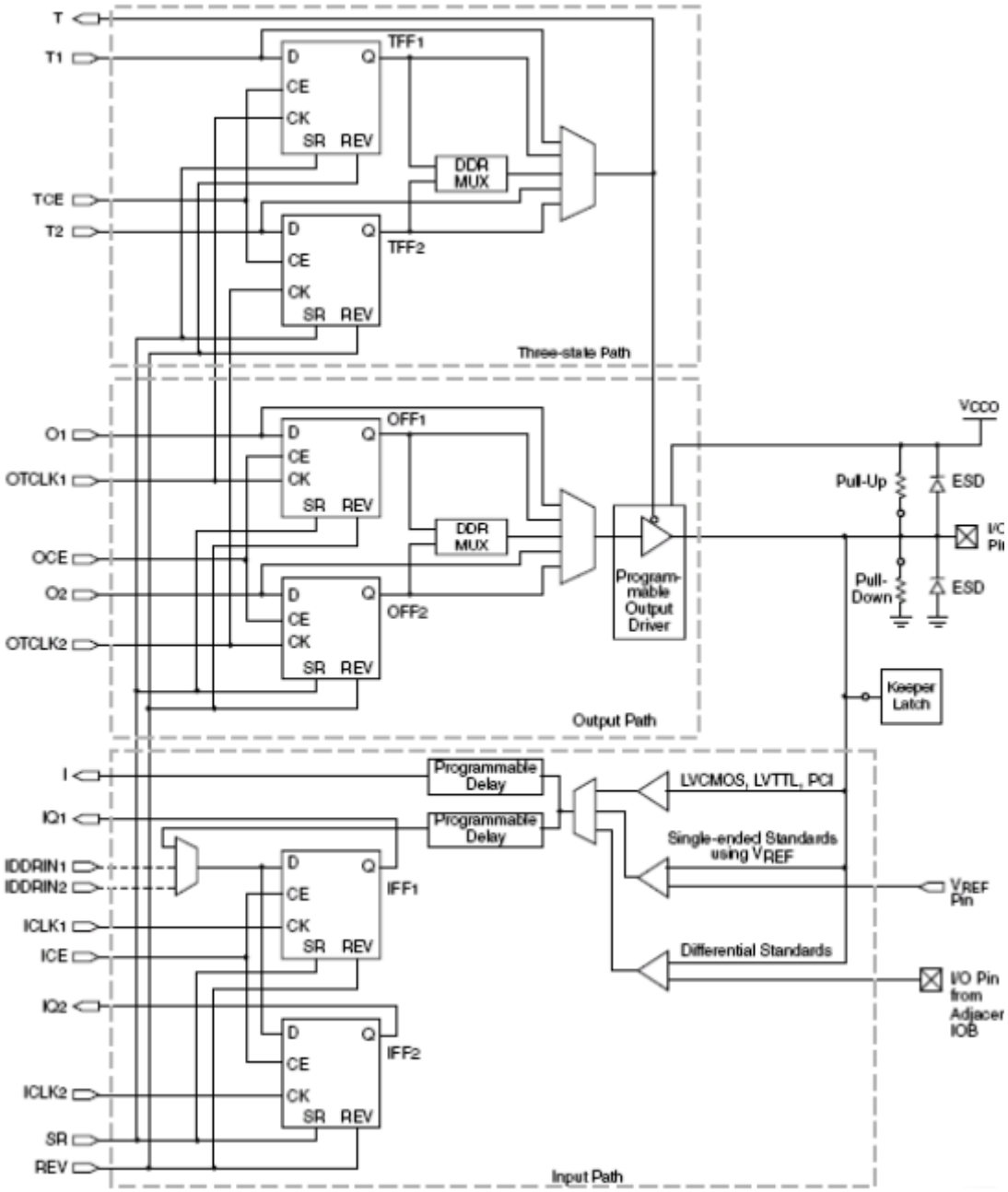


Figure 10. Internal structure of a I/O Block.

- Digital Clock Manager

DCMs integrate advanced clocking capabilities into FPGA's global clock distribution network. Its most common use is to multiply or divide clock frequency and/or to shift phase of a clock signal. Internal structure of DCM is shown on figure below.

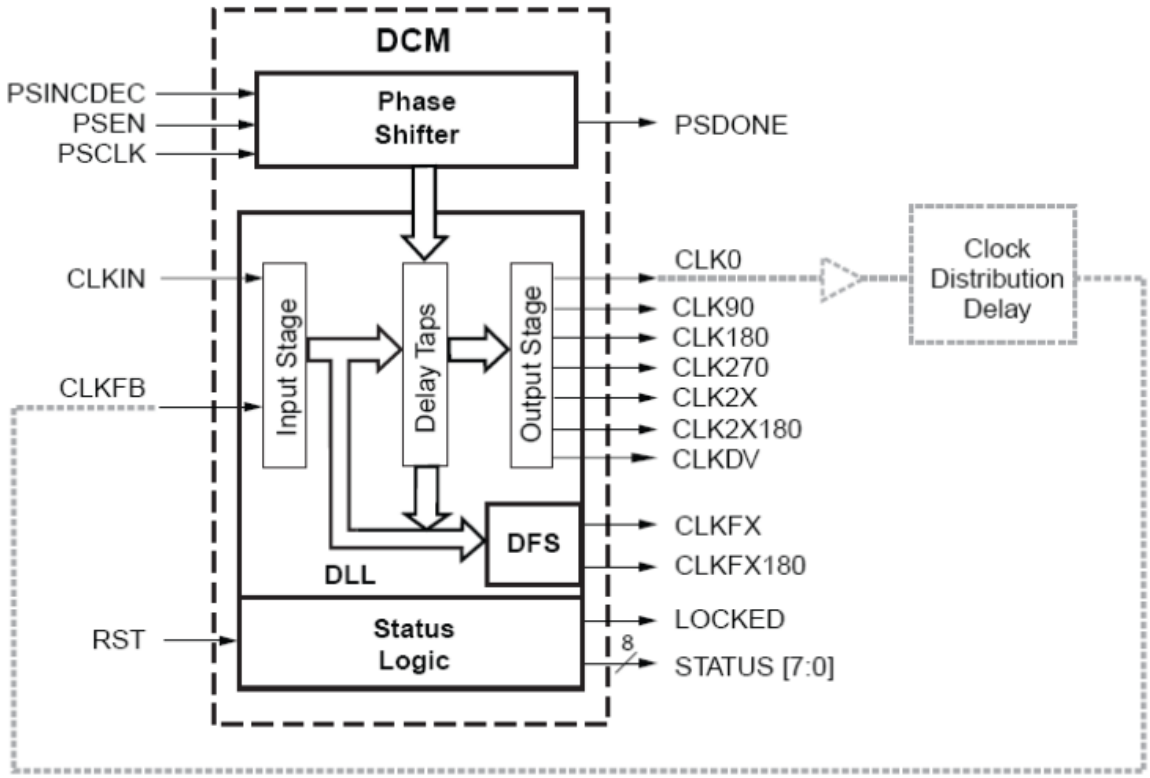


Figure 11. Internal structure of DCM.

2.2.3 Hardware Description Languages

To program FPGA and define its behaviour, a hardware description languages are used, the most common of them are Verilog and VHDL (Elliott, 2018). Verilog was created by Gateway Design Automation private company in 1985 with syntaxes based on C language; while Ada-based VHDL was created in 1987 by US Defence Department.

The main difference between HDLs and software programming languages is that software languages are single-threaded, without or with a very limited ability for parallel programming. Hardware description languages, in other hand, model parallel processes (registers, latches, etc.) that run independently. While software compilers convert source code into program to be executed on the target microcontroller, HDL compiler creates a *netlist* which describes connection between circuit's components

For this project, Verilog will be used as it has more syntax similarities with C and therefore is easier to learn.

2.2.4 Comparison with ASICs

ASICs are integrated circuits designed for one specific application that allow to achieve high efficiency. So that, general and flexible FPGAs usually much slower, bigger and consume more energy for the same function; but, in other hand, ASICs are much more expensive to design due to their non-recurring engineering. Because of that, FPGAs are often used to design an ASIC (Brunvand, 2018).

FPGA & ASIC Design Advantages	
FPGA Design Advantages	**ASIC Design Advantages**
Faster time-to-market - no layout, masks or other manufacturing steps are needed	**Full custom capability** - for design since device is manufactured to design specs
No upfront NRE (non recurring expenses) - costs typically associated with an ASIC design	**Lower unit costs** - for very high volume designs
Simpler design cycle - due to software that handles much of the routing, placement, and timing	**Smaller form factor** - since device is manufactured to design specs
More predictable project cycle - due to elimination of potential re-spins, wafer capacities, etc.	**Higher raw internal clock speeds**
Field reprogramability - a new bitstream can be uploaded remotely	

Figure 12. Comparison of FPGA and ASIC.

3 Project plan

3.1 Work Breakdown Structure

To fulfil objectives of the project and to fit within limited given time, work structure was reconsidered since Project Proposal. Project timeline was separated in three stages with fourth being parallel to them.

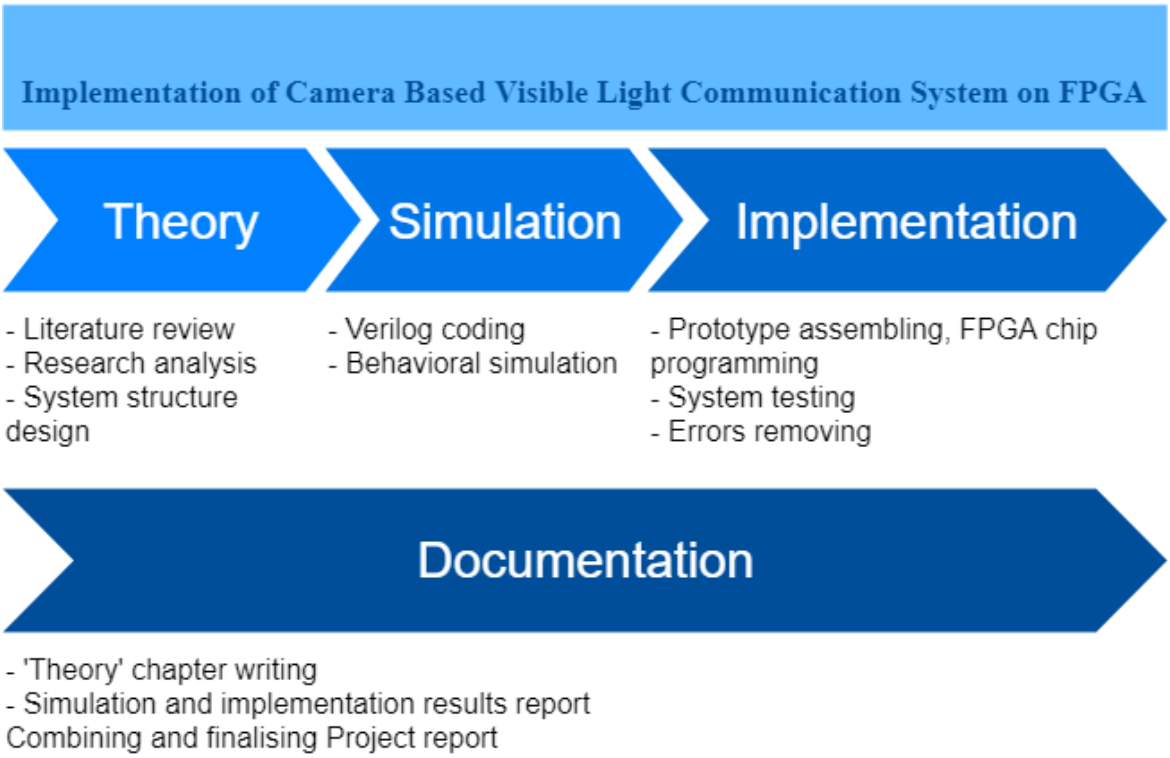


Figure 13. Work Breakdown Structure.

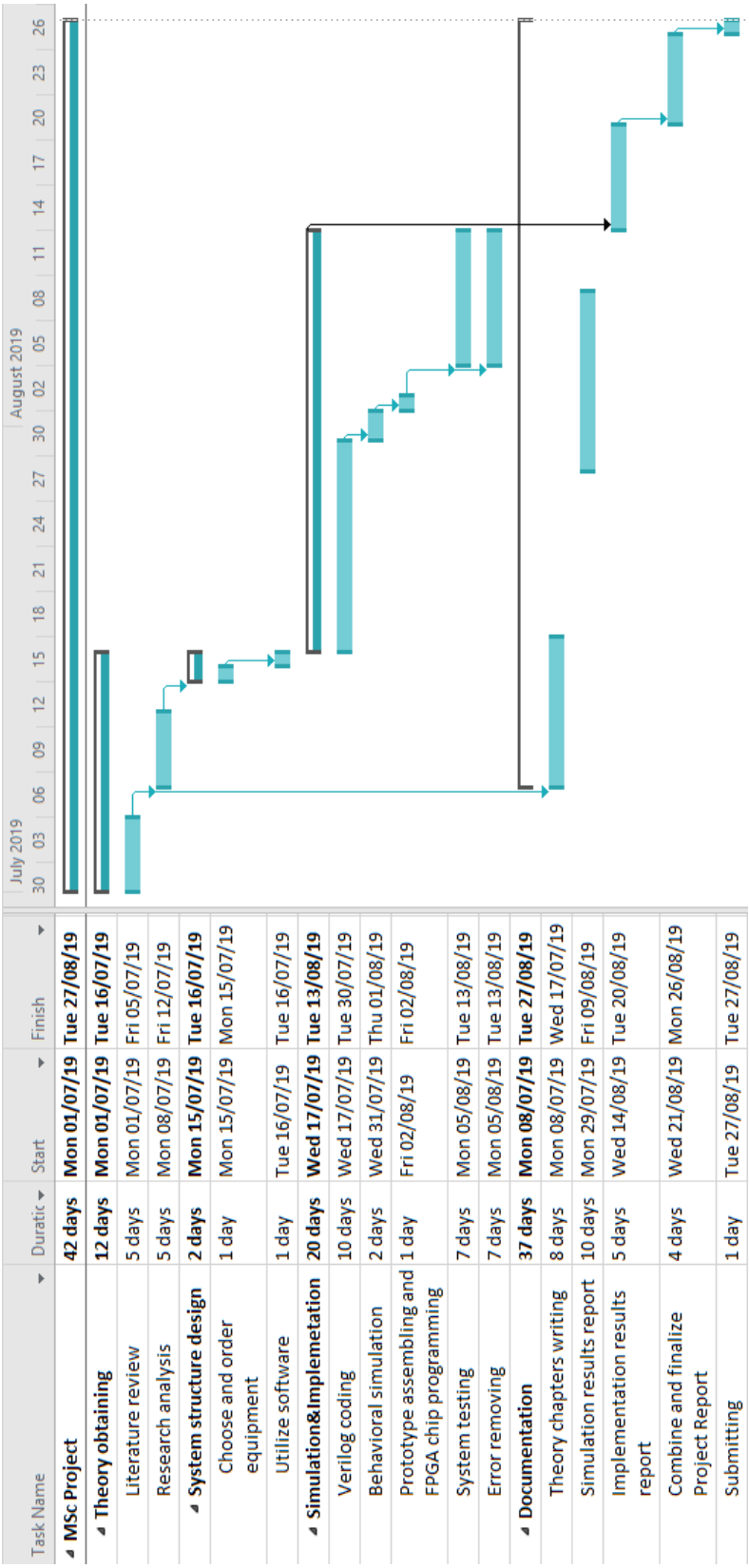
3.2 Task List

No	Task name	Duration, working days
1	Theory obtaining	12
1.1	Literature review	5
1.2	Research analysis	5
1.3	System structure design	2
2	Simulation	12
2.1	Verilog coding	10
2.2	Behavioural simulation	2
3	Implementation	8
3.1	Prototype assembling and FPGA chip programming	1
3.2	System testing	7
3.3	Removing errors	7*
4	Documentation	37*
4.1	Writing theory chapters	8*
4.2	Create simulation& implementation results report	15*
4.3	Combine and finalize Project Report	5
	Total	42

* - in parallel with other tasks

Figure 14. Task list.

3.3 Gantt Chart



Task Name	Duratic	Start	Finish
MSc Project	**42 days**	**Mon 01/07/19**	**Tue 27/08/19**
Theory obtaining	**12 days**	**Mon 01/07/19**	**Tue 16/07/19**
Literature review	5 days	Mon 01/07/19	Fri 05/07/19
Research analysis	5 days	Mon 08/07/19	Fri 12/07/19
System structure design	**2 days**	**Mon 15/07/19**	**Tue 16/07/19**
Choose and order equipment	1 day	Mon 15/07/19	Mon 15/07/19
Utilize software	1 day	Tue 16/07/19	Tue 16/07/19
Simulation&Implemetation	**20 days**	**Wed 17/07/19**	**Tue 13/08/19**
Verilog coding	10 days	Wed 17/07/19	Tue 30/07/19
Behavioral simulation	2 days	Wed 31/07/19	Thu 01/08/19
Prototype assembling and FPGA chip programming	1 day	Fri 02/08/19	Fri 02/08/19
System testing	7 days	Mon 05/08/19	Tue 13/08/19
Error removing	7 days	Mon 05/08/19	Tue 13/08/19
Documentation	**37 days**	**Mon 08/07/19**	**Tue 27/08/19**
Theory chapters writing	8 days	Mon 08/07/19	Wed 17/07/19
Simulation results report	10 days	Mon 29/07/19	Fri 09/08/19
Implementation results report	5 days	Wed 14/08/19	Tue 20/08/19
Combine and finalize Project Report	4 days	Wed 21/08/19	Mon 26/08/19
Submitting	1 day	Tue 27/08/19	Tue 27/08/19

Figure 15. Gantt chart.

4 Implementation

4.1 Hardware equipment

Algorithm design was limited by the ability of provided FPGA (mostly by internal memory capacity). Table below shows comparison between FPGA Spartan-3e chip that was used for this project and modern Spartan-6:

	Spartan-3e	Spartan-6
Year designed	2005	2011
Logic cells	10’476	101’261
DSP Slices (specialised digital signal processing blocks)	none	180

Source – Xilinx.com

Figure 16. Comparison of modern and provided FPGAs.

It is notable that even Spartan-6 do not have enough space to store whole pictures.

As a result, designed algorithm works only with few rows of the picture at the same time. As an advantage, image can be processed using very limited resources. Also, because the algorithm does not wait for whole input picture to load inside FPGA, processing goes along with receiving data and finishes quickly: if use single [7:0] input wire bus at 100MHz clock signal, for 160x90 picture it’s 360us from start of receiving until all data is extracted, or 40 pixels per us, or 19.29 frames per second for the sequence of 1920x1080 pictures (just an estimation; this result for picture of this size can be achieved on modern FPGA, but the capacity of Spartan-3e is not enough to work even with several rows of picture this big).

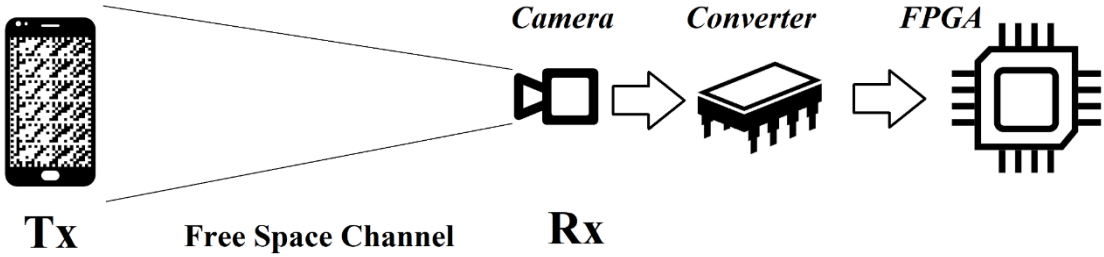
The major disadvantage of this algorithm is that, since whole picture can not be observed, all 4 corners of frame can not be detected at the same time. Therefore, if the screen of Tx smartphone is angled left or right from vertical position, picture can not be rotated to fix it. However, if smartphone is tilted slightly backwards of forward (and as a result projection of the screen looks trapezial), data still can be extracted

without errors – as long as rows of cells are parallel to the rows of pixels (see test results below).

4.2 Algorithm structure

4.2.1 System Block Diagram and overall design

The diagram below shows the layout of VLC system. Main focus of this project is to develop an FPGA algorithm to extract data from incoming images, however, due to limitations of camera and FPGA communication abilities, a converter from camera output format to FPGA-friendly data format also had to be created (see Appendix for matlab code).



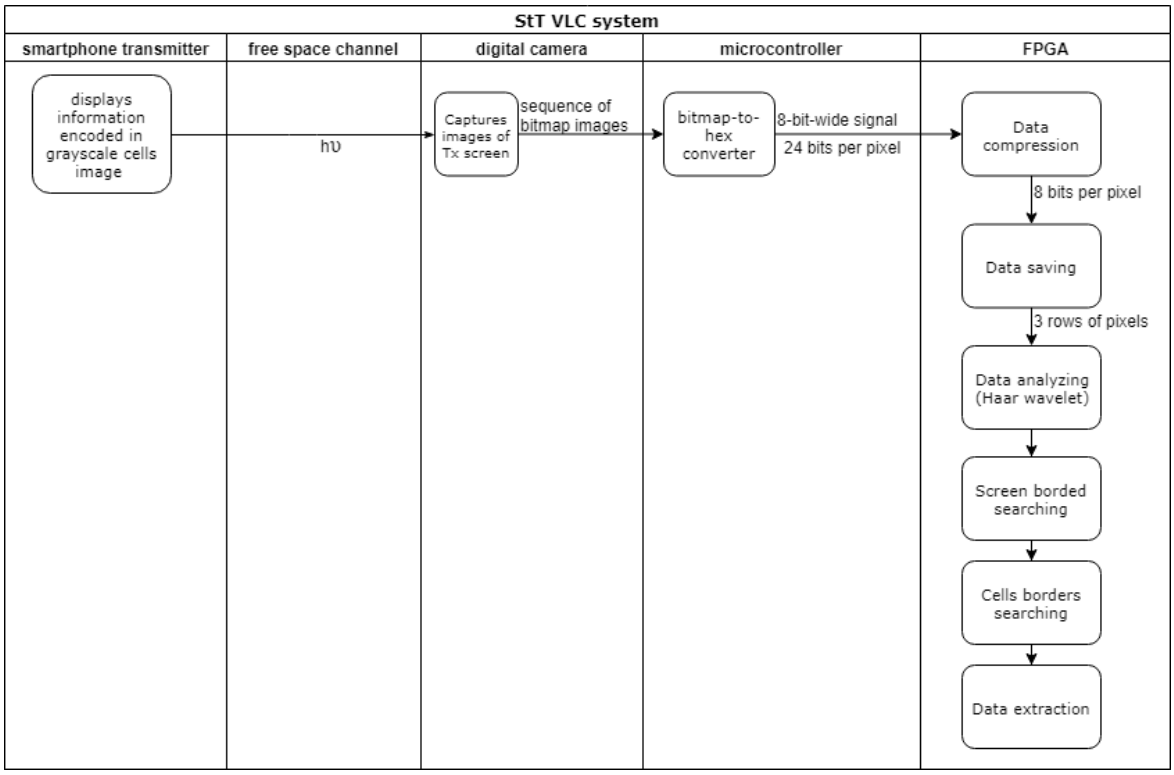


Figure 17. System block diagram.

Because of the limited capacity of FPGA, none of existing digital image processing algorithm was not suitable for this project. New algorithm was developed from scratch. It has some similarities with SURF algorithm (see 'Theory' chapter above): it uses the same idea of finding points of interest via calculating Haar functions, but overall structure is completely different.

The algorithm itself consists of 6 blocks, and incoming data is pipelining through them to be analysed. Structure and functions of each block are explained below.

4.2.2 Schematic symbols

Internal structure of some complex components used in the algorithm is briefly explained in this part:

- Demultiplexer

Demultiplexer is a logic device that forwards signal from input to one of outputs. It is controlled by *selector* signal (A and B on circuit below), and can be seen as a single-input to multiple-output switch (*Ugrumov, 2007*).

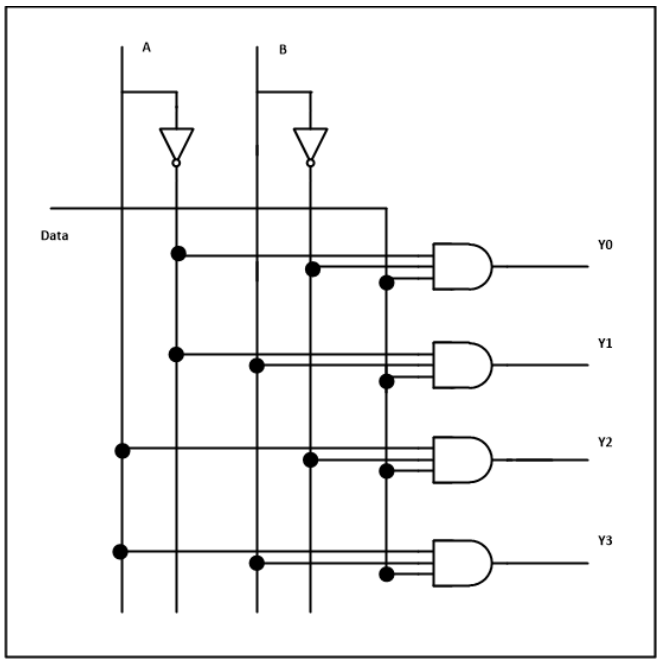


Figure 18. Demultiplexer circuit.

In this algorithm, the main purpose of demultiplexers is to sort what part of image data is being worked now – what colour channel of pixel, which row of pixels, etc. Its blackbox representations is shown at figure below.

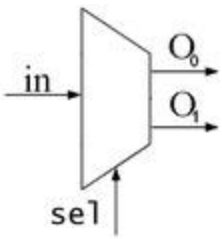


Figure 19. Schematic representation of demultiplexer.

- Comparator

Comparator is an electronic device with two inputs and 3 outputs. Outputs are assigned values of inputs' comparison: logic '1' if value of first input greater, equal or less that value of the second one, respectively to first, second and third output (*Ugrumov, 2007*). Internal structure of comparator is shown at figure below.

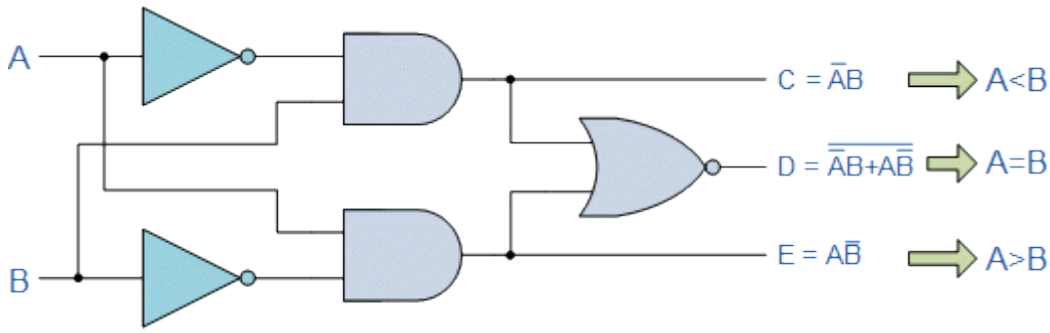


Figure 20. Comparator circuit.

In this algorithm, comparators are used for two primary situations: to detect when counters, that control the behaviour of circuit, reach certain value; and to check whether analysed parameters greater or less than their thresholds. For the sake of simplicity, in all schematics below comparators are shown in blackbox representations, as at figure below.

Figure 21. Schematic representation of comparator.

4.2.3 Input block and data compression

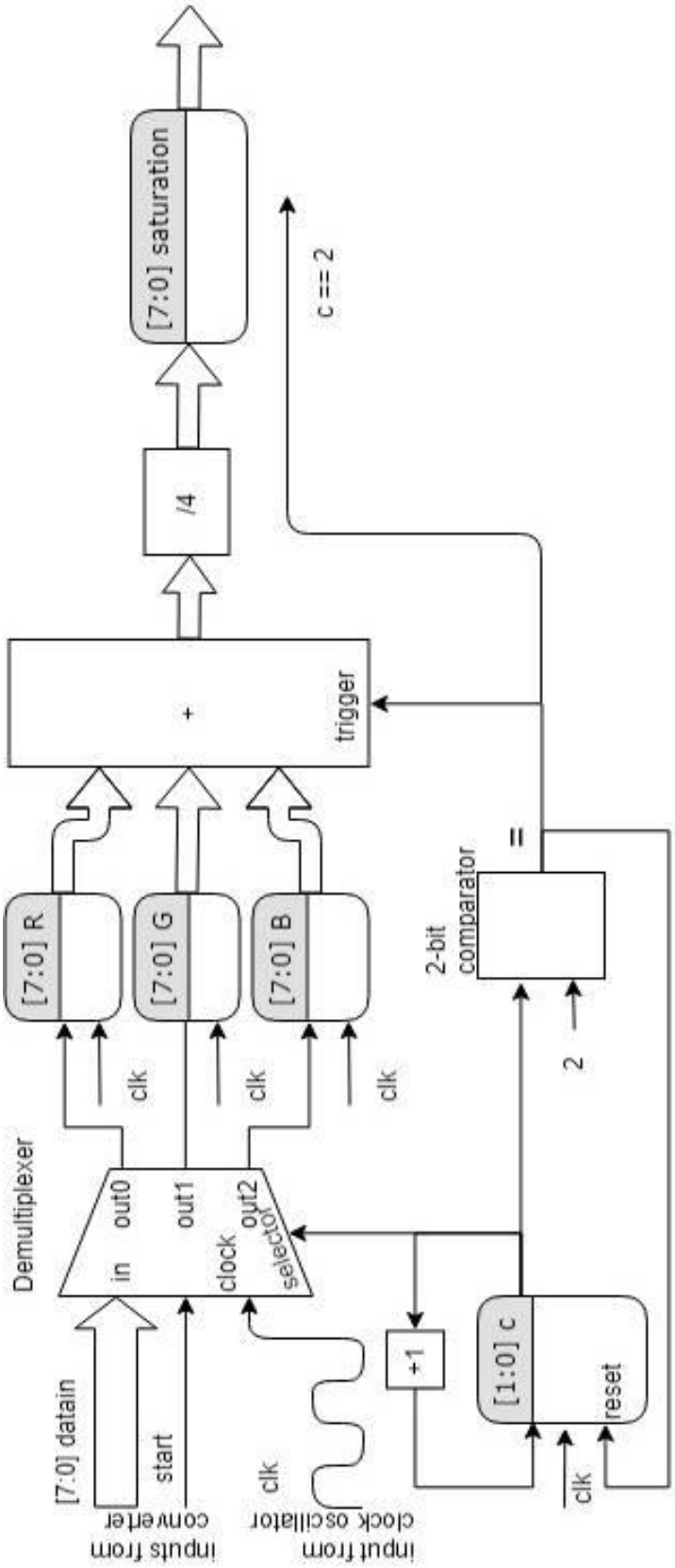


Figure 22. Input and compression block schematic.

The first block in the algorithm is intended to deal with inputs of circuit and compress incoming data.

Inputs of this block comes from outside of the chip, they are 8-bit width data input plus single 'start' wire. 'Start' signal is used to establish synchronization with outside world and not to miss the start of data transmission.

Also, clock signal comes into FPGA from on-board 50MHz Oscillator and through Digital Clock Manager (see chapter 2.2) to make it 100MHz. All LUTs, registers and flip-flops in algorithm are driven by this 100MHz clock (often not shown on schematic).

Via 'datain' bus information comes in a RGBRGBRGB… sequence with all 3 colour channels goes in series. A counter 'c' is added to control which channel is at input now. Each clock cycle 'c' changes its state and 'datain' is written in respective temporary R, G or B registers. Once per 3 clock cycles, when 'c' counter's value is equal 2, R, G and B registers are merged together into one 8-bit register 'saturation', and 'c' counter resets to 0. The reason for this process is to save resources by using only 8 bits per pixel instead of 24.

4.2.4 Data saving

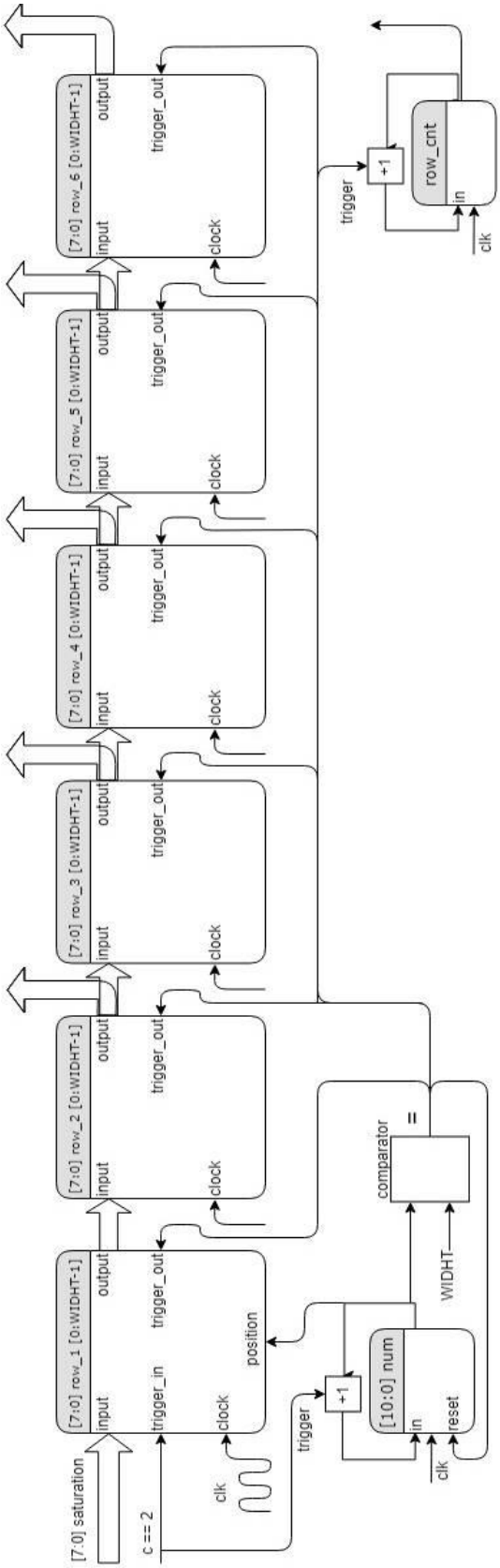


Figure 23. Data saving block schematic.

6 large two-dimensional registers stores data of 6 rows of pixels. 6 rows with length of 180 pixels each and 8 bits per pixel fill almost entire space of Spartan-3e chip (8.5k out of 10k logic cells), that's why this image size is a limit of Spartan-3e capabilities. Counter 'num' controls which position in a row is targeted now. Each 3 clock cycles (synchronised with previous block using the same 'c' counter) position number [num] of row_1 stored the value of 'saturation' register, and 'num' increments to target new position. When 'num' reaches the end of the row (that is controlled by comparator checking whether 'num' equals constant parameter 'WIDHT'), signal from comparator resets counter and shifts all stored data: from row_1 to row_2, from row_2 to row_3 and so on. Data from row_6 is eliminated. Therefore, at any moment of time we have one row being filled with incoming data and 5 previous rows of pixels ready to be analysed.

When row 6 is completely filled, signal 'h' is being sent to initialize next step of algorithm.

4.2.5 Data analysing (Haar functions)

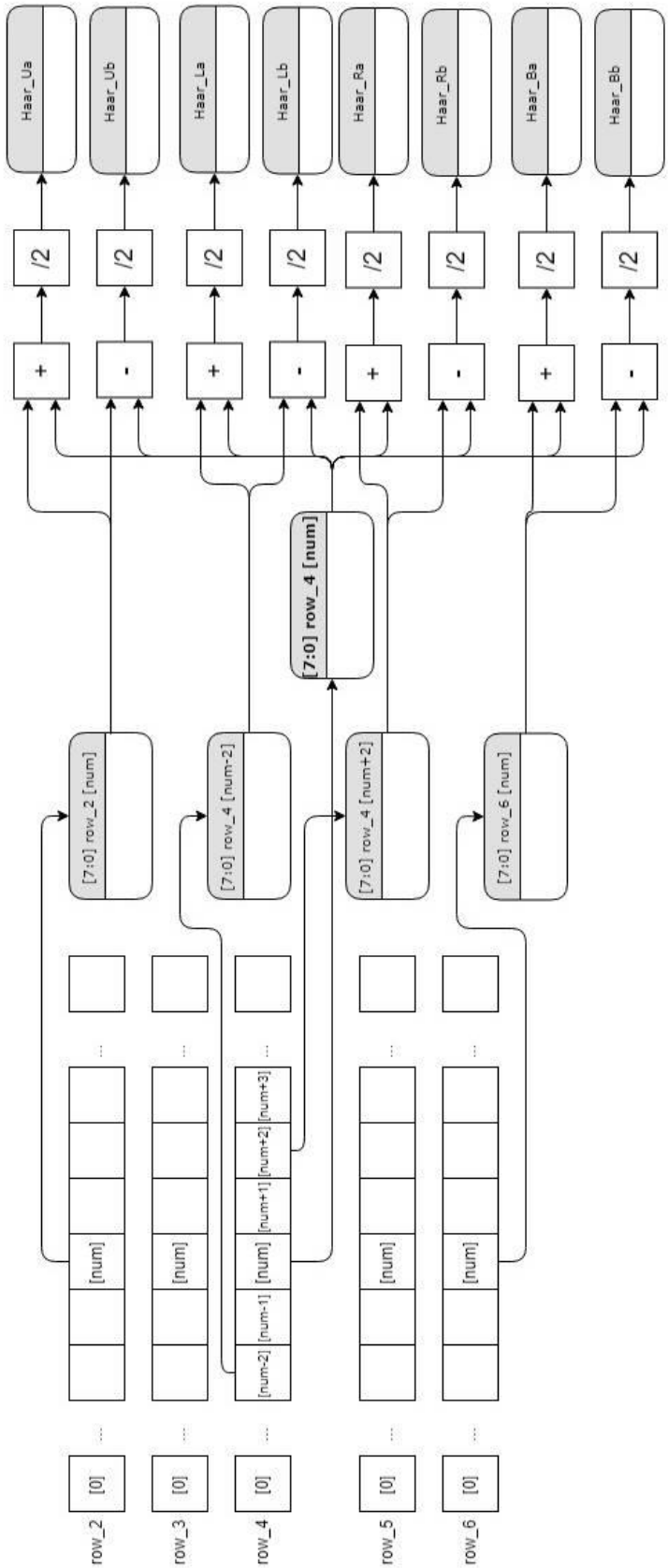


Figure 24. Haar block schematic.

Having five rows of pixels to analyse, the middle one can be targeted. For each pixel in a row discrete Haar wavelet transformation (Haar, 1910) can be used to calculate Haar_a and _b values for each pair 'targeted pixel-neighbouring pixel' (not direct neighbour, Haar radius of 2 found to be optimal for this size of image), Where Haar_a will be approximation and Haar_b is detalisation (Ruch *et al.*, 2009). Visualization and equations for Haar transformation are shown at figure below. Haar_a functions are calculated, but not used in this algorithm.

At each 3 clks for pixel number 'num' in a row_4 eight Haar values are calculated (which means that analysing in row_4 goes synchronically with saving new data in a row_1).

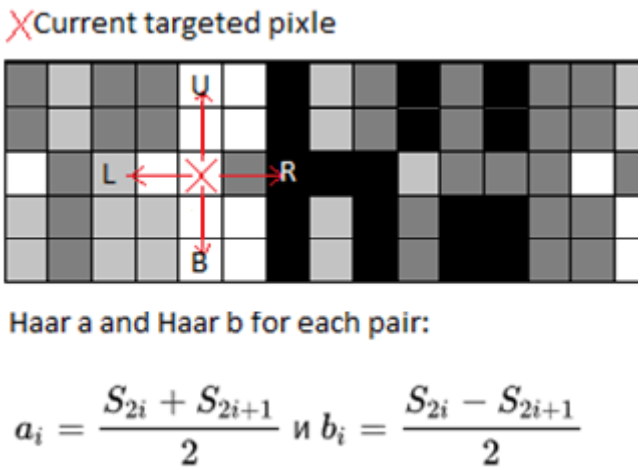


Figure 25. Haar functions explanation.

The values of Haar functions are not being permanently saved and overwrites with new data each cycle.

4.2.6 Rows borders searching

While previous blocks works permanently, the next 3 blocks have more complex behaviour and works in series with synchronising signals between each other. When rows borders searching block detects the edge of the row, it sends initialising signal to cell borders searching block. Conversely, when cell borders searching block is running, it generates blocking 'busy' signal that prevents previous block from running. When cell borders searching is done, it sends signal to data extraction block and sets 'busy' signal low, so row borders searching block starts to work again. Timing diagram of these processes is shown below.

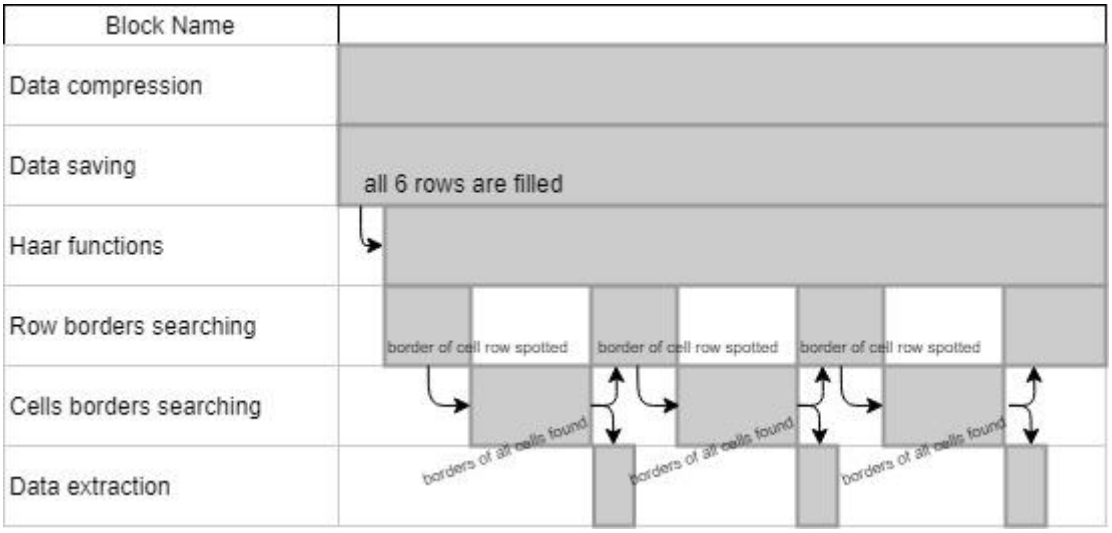


Figure 26. Algorithm's blocks timing diagram.

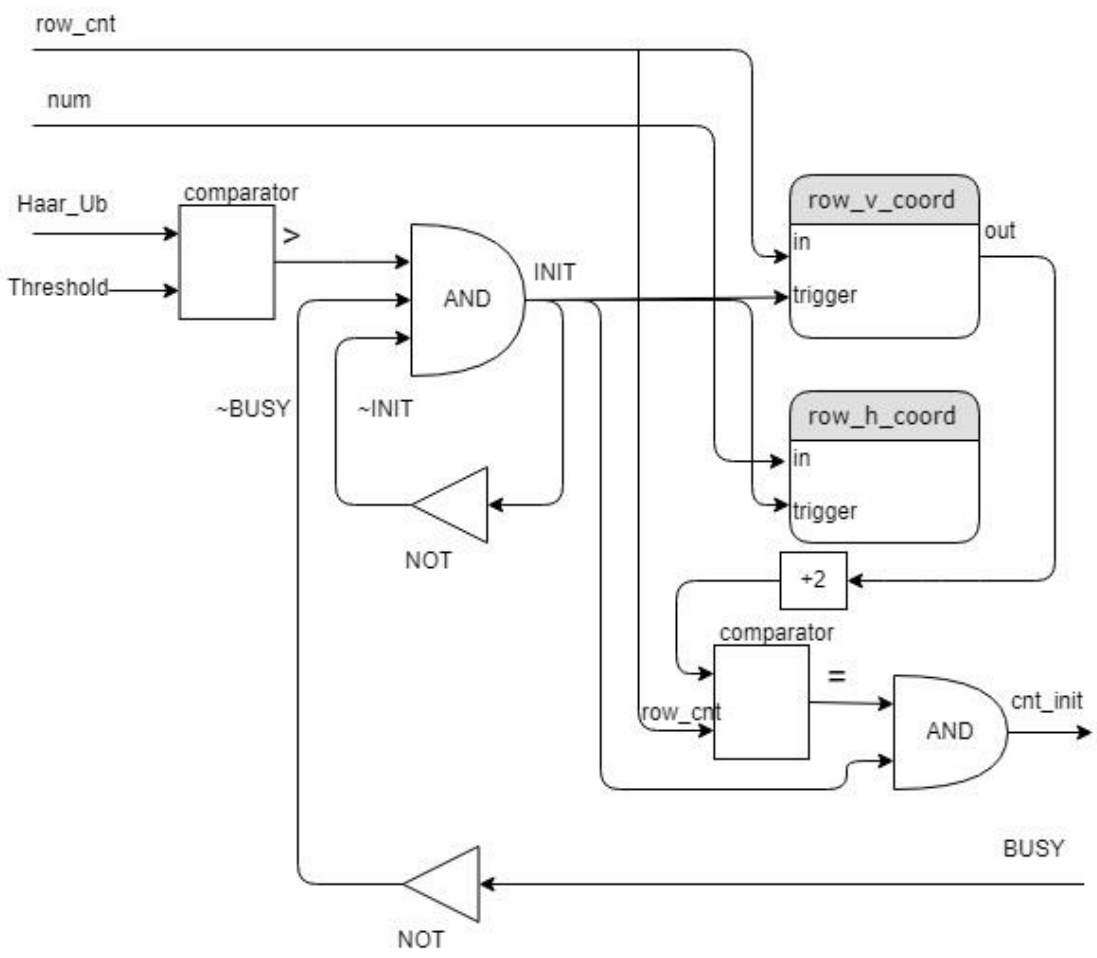


Figure 27. Rows borders searching block schematic.

Haar_b value, calculated in previous block, represents the difference in saturation between targeted pixel and it neighbours. Therefore, if there will be a rapid change of saturation (more than threshold which optimal value is found during algorithm testing), it should be a point of interest. Rows goes from bottom to top, pixels in a row are scanned from left to right, therefore, Haar_b for upper neighbour and for right neighbour are tracked to find the edge of the screen.

The picture below is a part of one tested image. Pixel where value of Haar_b function first raised above threshold, is highlighted. It is noticeable how edges of cells are blurred. That distortion makes Haar function with a radius of 1 ineffective. Pictures with higher resolution may require larger Haar radius.

Figure 28. Row border detection.

Once the edge of the cell row is spotted, current values of row and column counters ('row'_cnt' and 'num') are stored to a separate registers to save the coordinates of the cell row's border. Then algorithm waits for two full rows, so that analysed row_4 will be not at the bottom edge of cell row but closer to its centre. Then initiating signal is sent to the next block to start count number of cells in a row and their sizes.

4.2.7 Cells borders searching

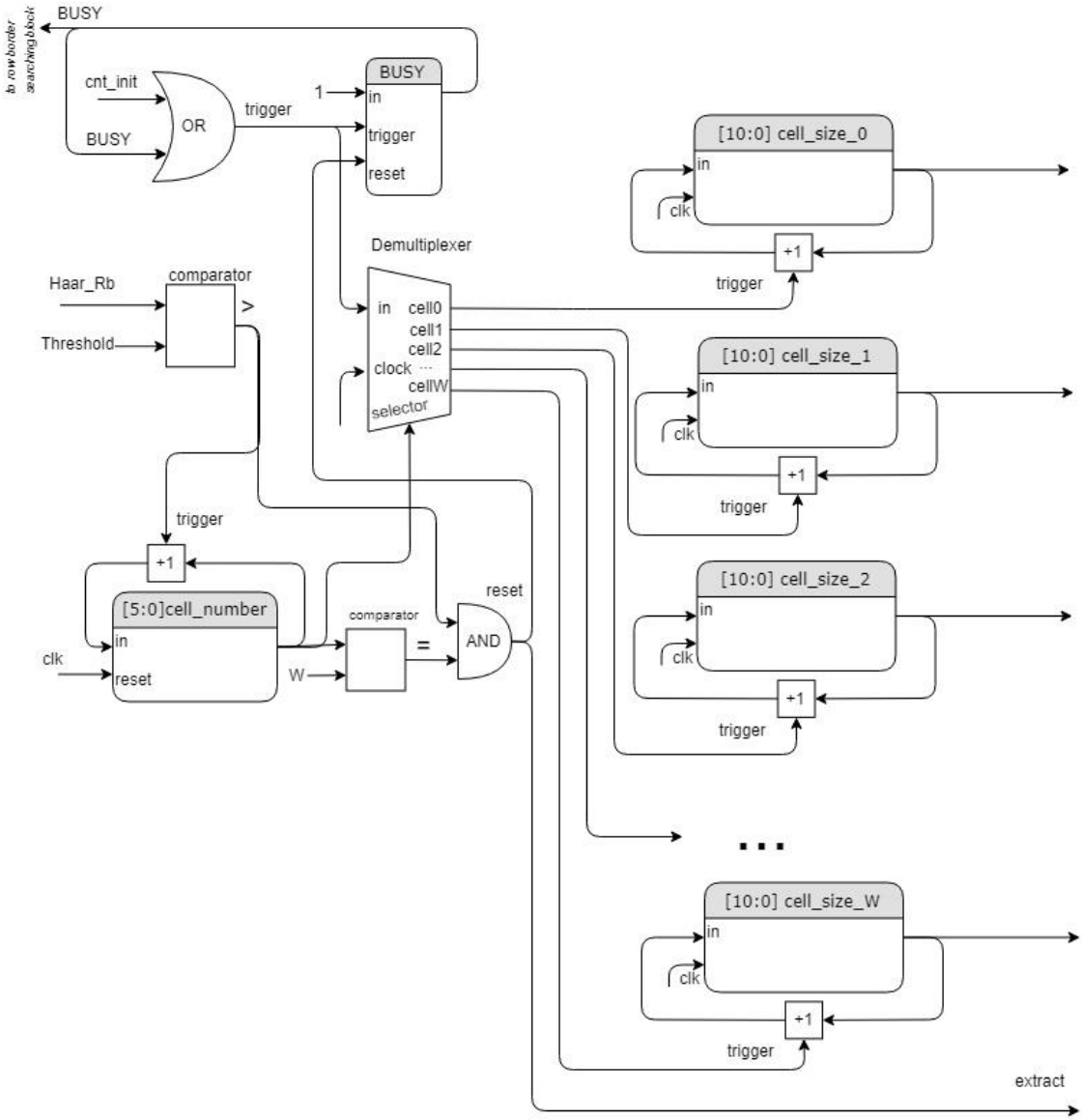


Figure 29. Cells borders searching block schematic.

Using Haar_b values for the right neighbour calculated in Block 3, algorithm is spotting borders between cells. For this purpose, cells bordering the edge of the screen (first row of cells) are not representing any encoded information but coloured black and white, so the borders between each of them is clearly visible (see picture below).

Figure 30. Transmitter data and captured image. Cells along the border are coloured in chessboard order to detect their edges.

Number of current analysed cell is stored in 'cell_number' counter, that increments each time when comparator spots a border (rapid change of saturation between cells). This counter controls, size of which cell is being measured right now, working as a selector in demultiplexer for counter trigger. At each given moment of time 'cell_number' selects respective 'cell_size_n' counter to be triggered for incrementation. When Haar comparator sports cell border, 'cell_number' increments by 1 and now targets 'cell_size_n+1' counter.

Distance between each border is being measured and saved. As a result, with information about corner's coordinates and size of each cell, the centre of each cell can be calculated and targeted for data extraction in the next block.

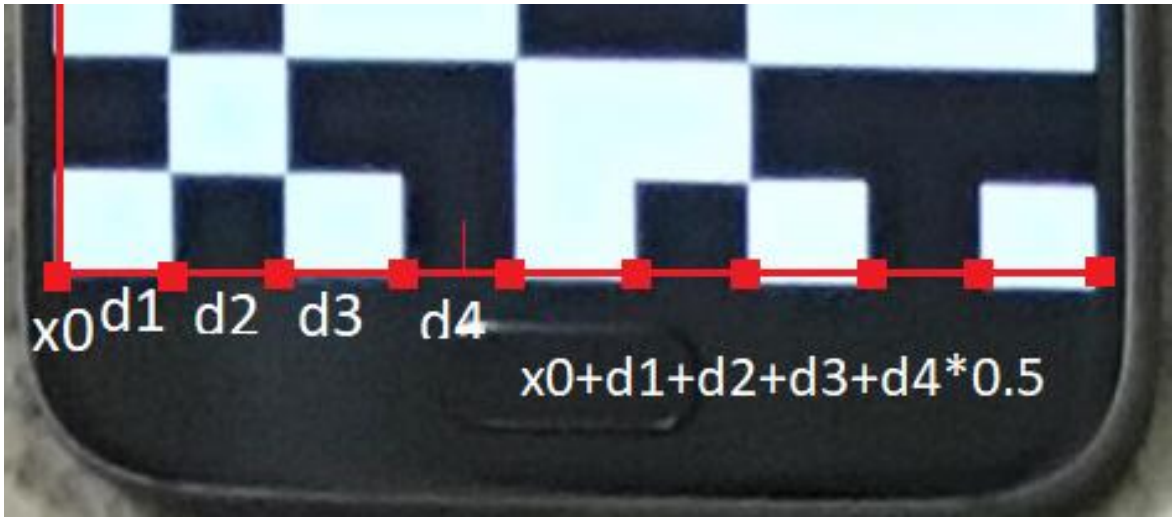


Figure 31. Calculation of cell's centre position.

At the photo above cell borders along the red row are highlighted, expression for horizontal coordinate of 4th cell is shown. Notice that phone is slightly rotated from straight horizontal line. If the last cell of the row stays on the line, data still can be recovered. Therefore, estimated critical angle is atan(1/horizontal_number_of_cells). See test results below where this estimation was checked,

When all cell borders are found and coordinates of the centre of each cell in a row is found, signal from comparator (which checks whether number of found cells equals constant 'W' parameter) resets 'busy' signal to 0 and send coordinating signal 'extract' to the next block.

4.2.8 Data extraction

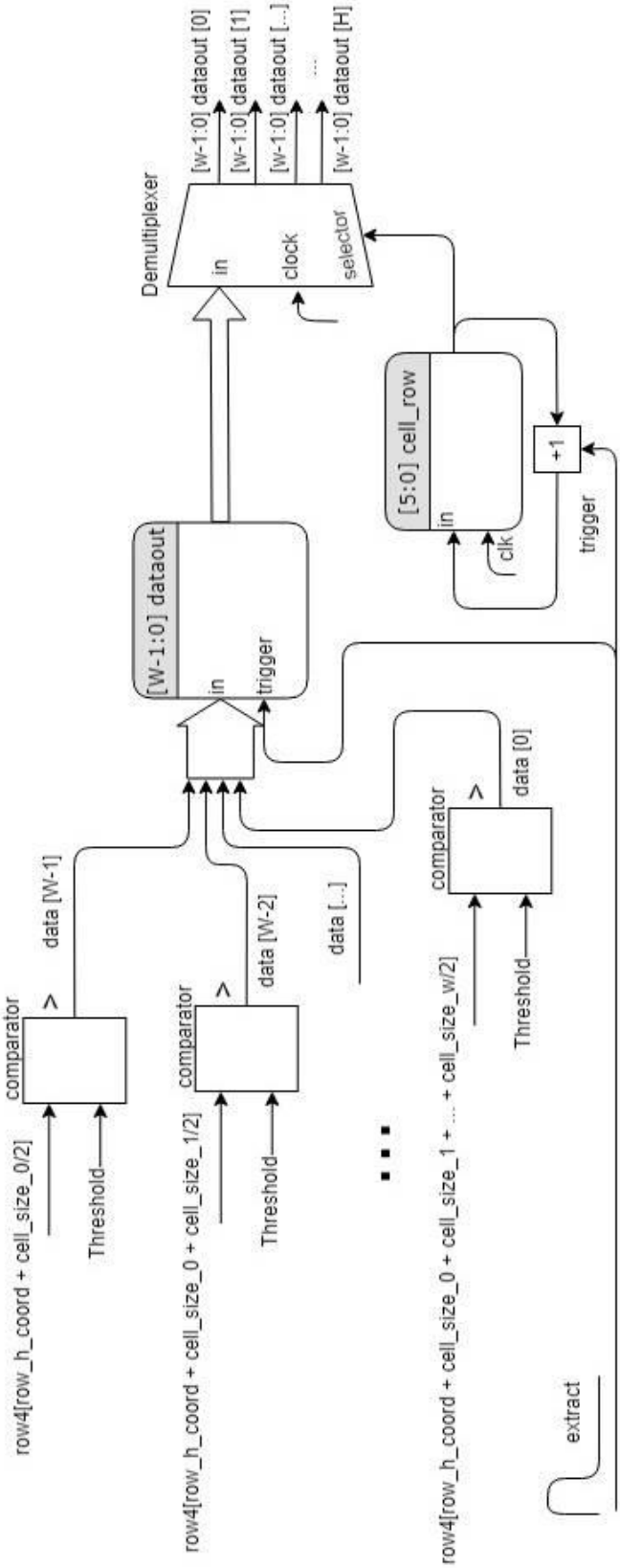


Figure 32. Data extraction block schematic.

For each cell in a row a register (2 registers in case of grayscale image) is created to temporary store data. For each row of pixels saturation in a middle of each cell is compared with a thresholds (one threshold to separate 0 and 1 in black&white case and 3 thresholds to separate 00, 01, 10 and 11 in grayscale case, for the sake of simplicity the first case is shown on schematic).

Permanent 2-dimentional array saves these temporary data when 'extract' trigger signal arrives from previous block. Amount of these extractions represents a number of cell rows analysed, this data is being saved in 'cell_row' counter. When data from all cells on screen is extracted (i.e. when 'cell_row' counter equals 'H' parameter), 'finish' signal is generated at comparator's output to be sent to fpga's output (this comparator is *not* shown on schematic above).

After generation of 'finish' signal, 'dataout' array represents extracted data from analysed picture. It can be compared with transmitted information to check if algorithm works correctly. Experiments that tests algorithm in different conditions with different transmitted data and their results can be found in the next chapter.

4.3 Tests

4.3.1 Tests' design and experimental set up

After algorithm was designed and its description was written in Verilog HDL, it was implemented on provided Spartan-3e FPGA and its performance was tested. Photo of experimental setup is shown below:

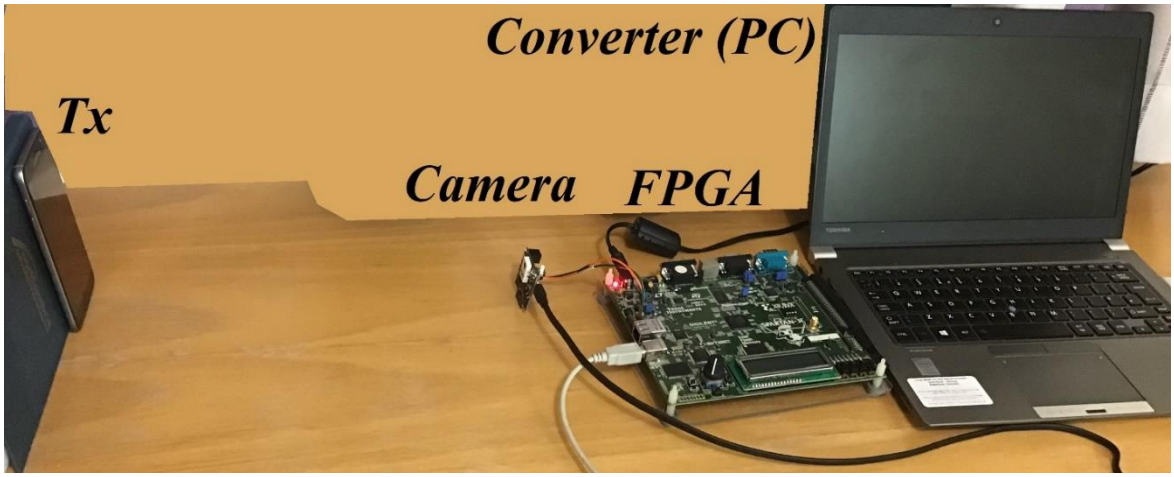


Figure 33. Physical experimental setup.

Due to the absence of any microcontroller to use it as a converter, data from camera was converted on PC via matlab (see code attached in Appendix). Different sets of data were encoded and shown on Tx screen, screen was captured with camera, captured data was sent to PC, converted to hexadecimal string and sent to FPGA to be analysed. Table below shows how data looks like at different stages of transmission/conversion.

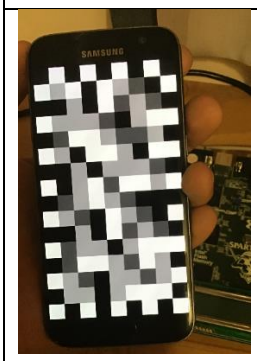
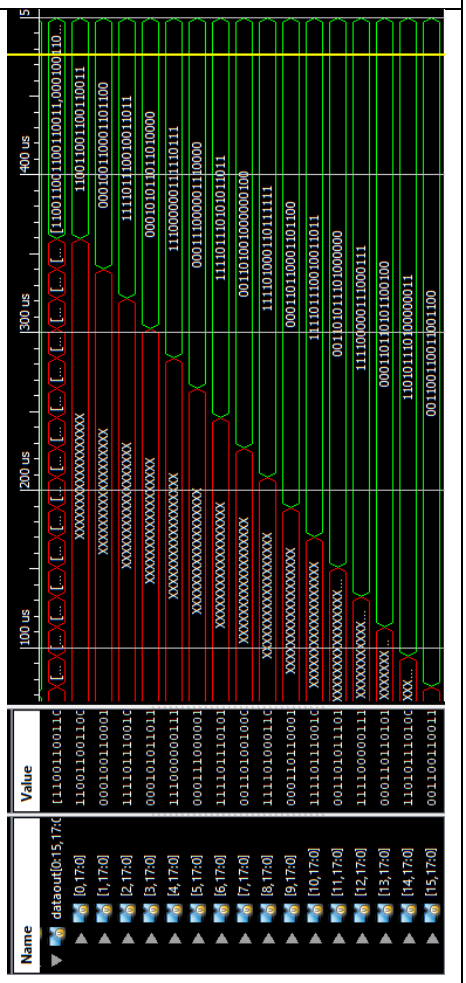
Information on display	Images captured by camera	Image encoded in hex format	Processed image output
		5e 55 44 5f 56 45 60 58 45 5f ...	

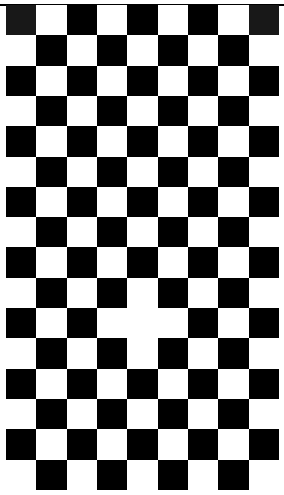
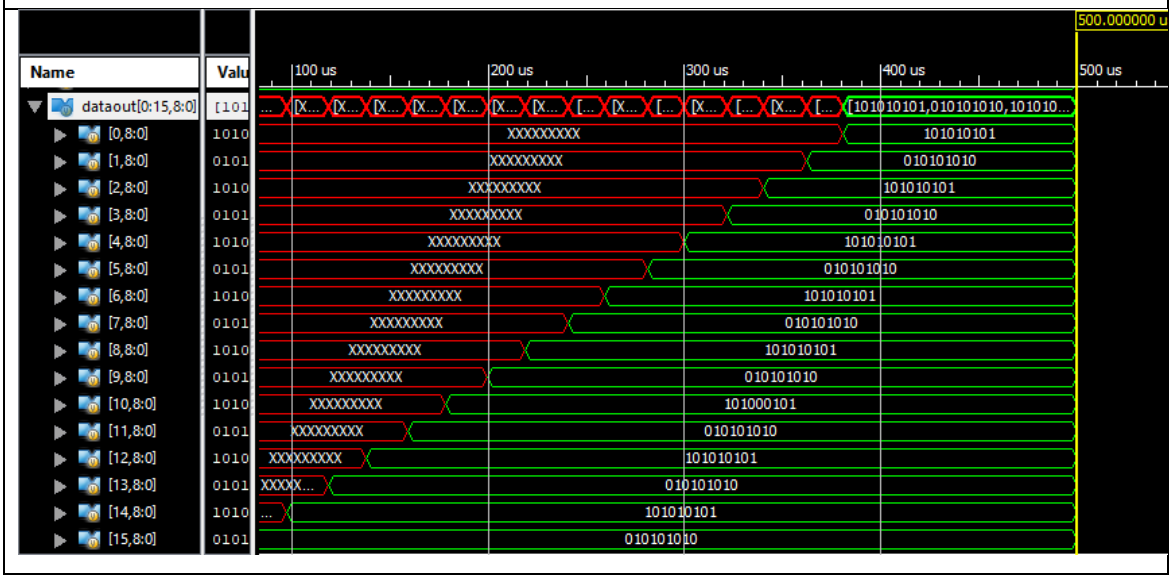
Figure 34. Data representation at different stages.

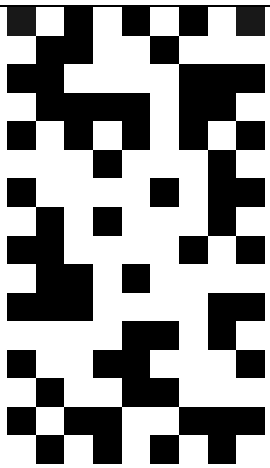
Different conditions was tested, e.g. Tx-Rx distance, number of cells per screen and colours per cell, angles of rotation and tilting, etc. Below is a report about these tests and their results.

4.3.2 Overall performance tests

This series of tests was done first to check whether algorithm works at all. Conditions of testing were close to ideal, because it was not intended to test algorithm's possibilities and limitations. For example, estimated maximum width of image was 180 pixels, but for these tests images 160x90 were taken. Lightning and distance conditions were chosen to capture transmitter's screen with minimal distortions. Rotation and tilting of transmitter were minimised.

Table below shows results of two testing sessions. More tests were done, but test reports were not saved due to the consistency of results and absence of new information.

Information shown on screen	Image of the screen captured by camera
 "Chessboard"	
Algorithm output	
	

Information shown on screen	Image of the screen captured by camera
 Binary representation of pi. Sequence A004601, from Encyclopedia of Integer Sequences (Sloane, 1964; Delahaye, 1997)	

Algorithm output

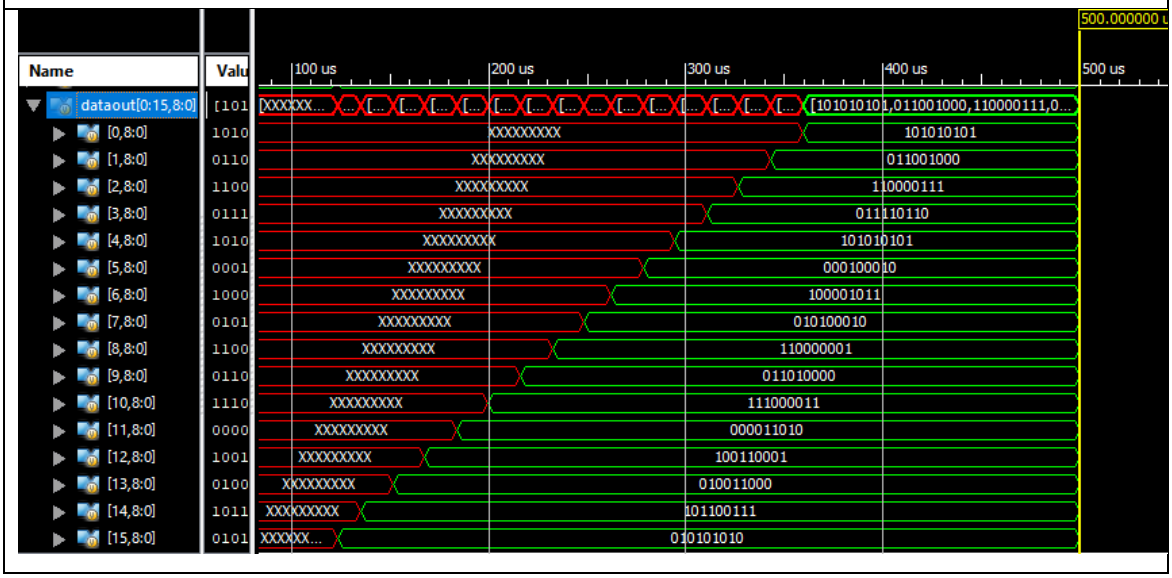
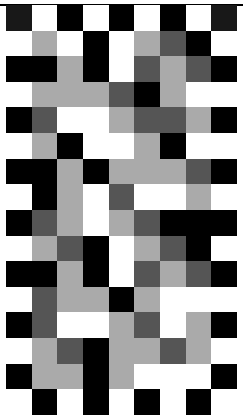


Figure 35. Overall performance tests.

After comparison of extracted data with original sent data it was concluded that algorithm works as supposed to. Next tests were done to find limits of its performance.

4.3.3 Grayscale image test

Information shown on screen	Image of the screen captured by camera
 Lorem ipsum text in ASCII (Mackenzie, 1980)	

Algorithm output

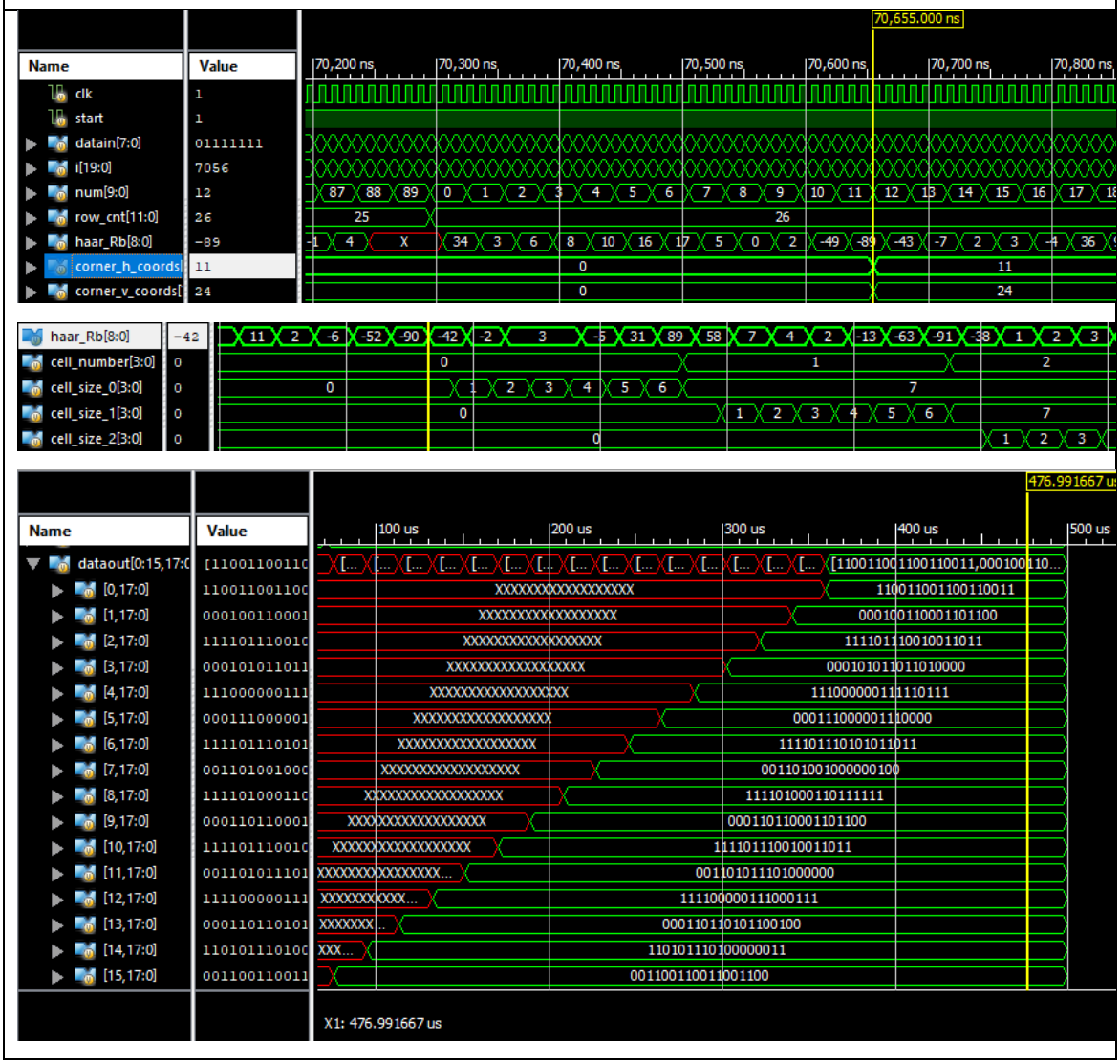


Figure 36. Grayscale image test.

The reason for this test was to find out how much data can be encoded in a single cell by using different shades of grey colour instead of just black/white cells. Results were consistently successful only with 2 and 4 colours per cell, algorithm was not able to recognize 8 different shades of grey scale. Therefore, max amount of data encoded in a single cell is 2 bits. Graph below shows system performance with different amount of colours per cell.

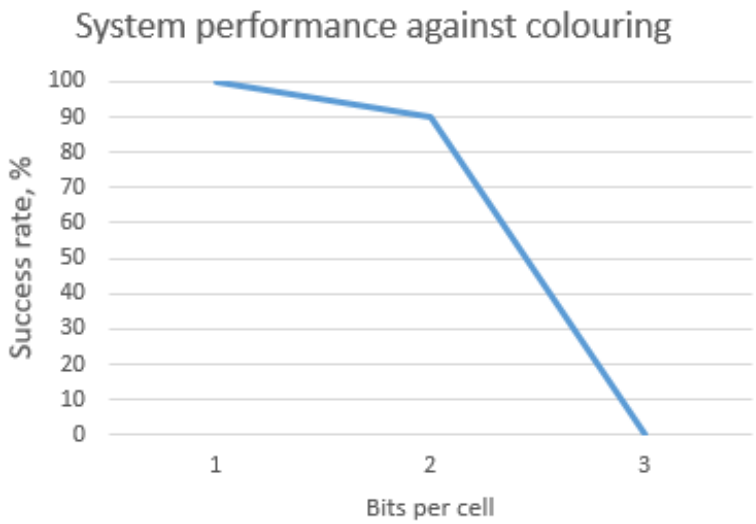
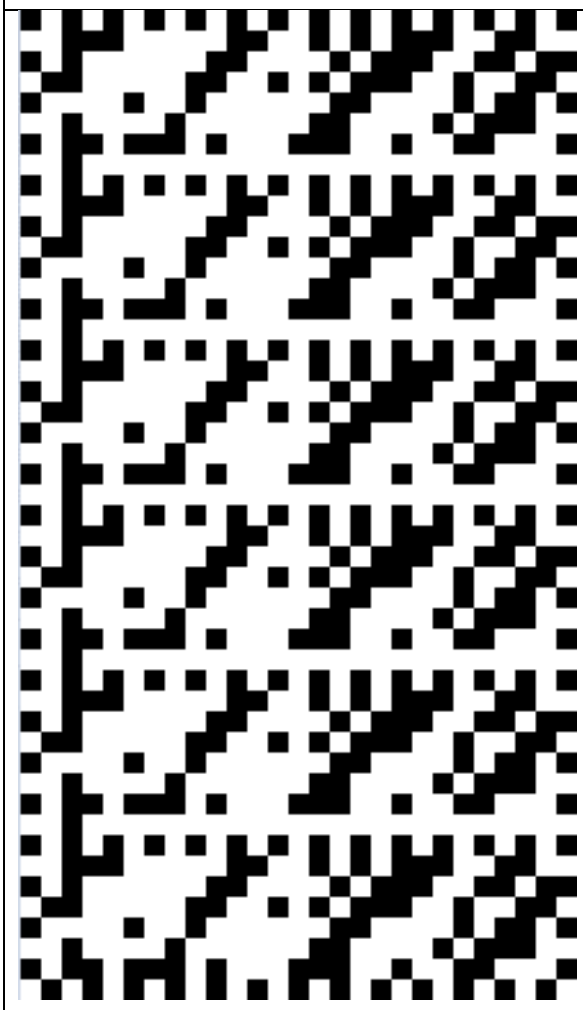


Figure 37. Success rate vs colours per cell.

4.3.4 Cell amount test

This test was done to find out how many cells can be shown on screen in one frame. By increasing number of cells per frame, data rate can be increased. Therefore, max number of cells that can be recognized by algorithm is an important parameter.

Modern smartphones has a screen aspect ratio of 9/16. So that, number of cells also follows that ratio. Tests were run with 9x16=144 cells, 18x32=576 cells and 27x48=1296 cells per frame.

Information shown on screen	Image of the screen captured by camera
 (latin phrase 'per aspera ad astra' encoded in ASCII (Mackenzie, 1980) and repeated)	 Notice that reflection at top right corner of the screen and how the last rows of extracted data are incorrect because of it

Algorithm output

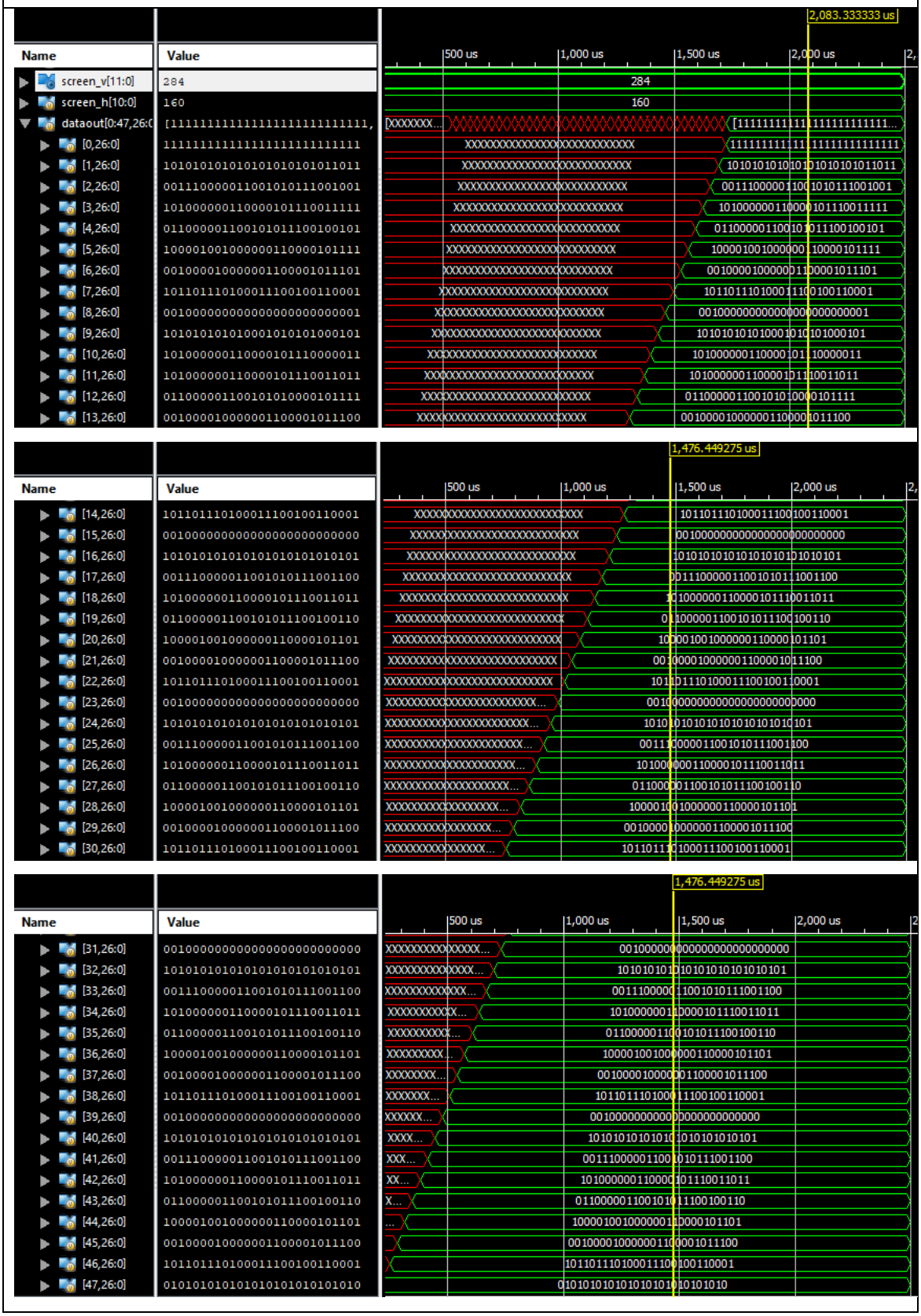


Figure 38. Maximization of cells' amount test.

Test results are shown on graph below. The reason for success rate drop is that when cell size on analysed image drops below 5x5 pixels, algorithm can miss cells' borders. The same reason will also appear in distance test.

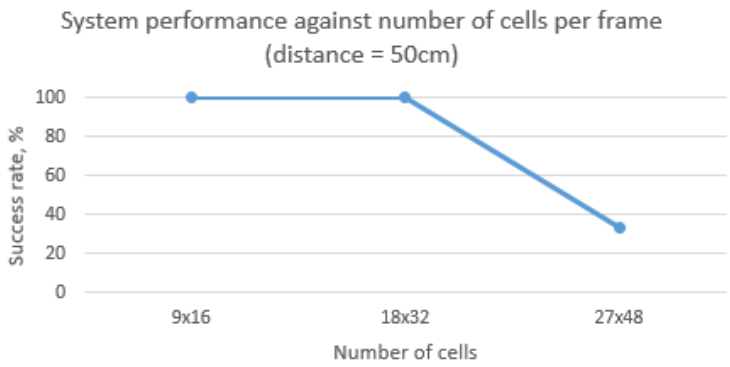


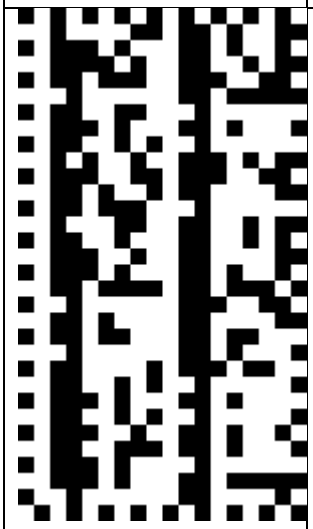
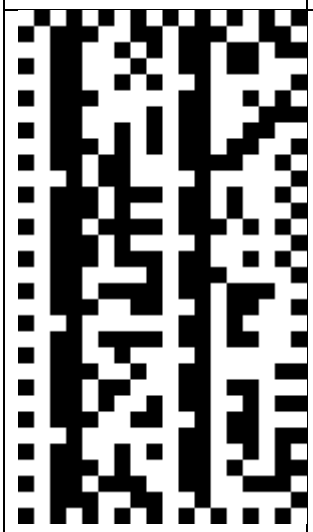
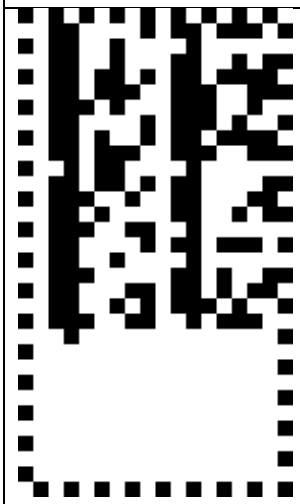
Figure 39. Success rate vs cells per frame.

4.3.5 Sequence test

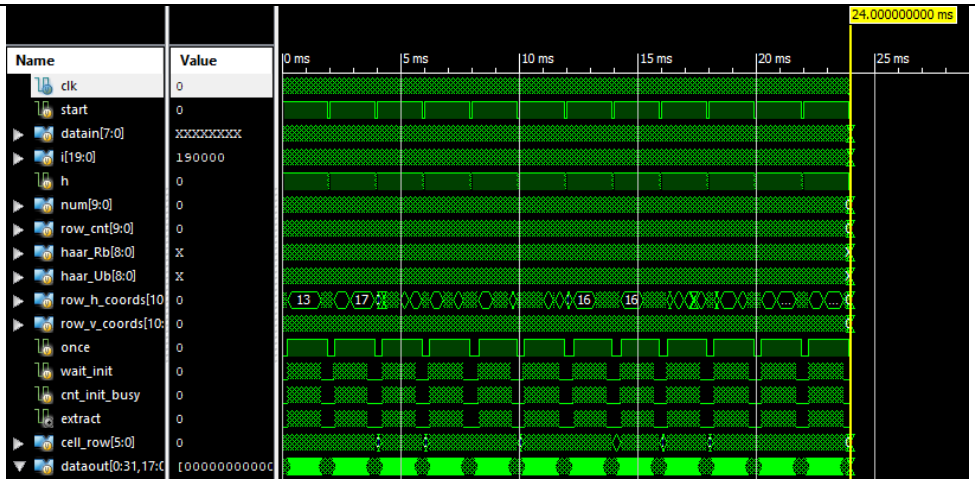
This test was done to check whether algorithm consistently refreshes all temporary parameters and registers between frames to be ready to analyse next frame.

Several images with encoded data (text of article 31 of Constitution of Russian Federation in ASCII) was shown on screen, while camera took 12 images of that screen (4 images per each data frame). 12 images were converted into hexadecimal format and sent to FPGA with 2ms delay between each of them (1900us for transmittion, 100us wait, transmit next image…). Whole process was finished in 24ms.

FPGA algorithm recognized all data correctly and stored data from new image only if it was different from previous one. Output data was manually put in ASCII decoder to check whether it matches with transmitted data (it did).

Information shown on screen	Image of the screen captured by camera			
				
				
				

FPGA algorithm performance:



[16:1] output (after deleting borders):			Output of binary-to-text converter:
01000011 01101001	01110011 01110011	01100001 01101110	Citizens of the Russian Federation shall have the right to a
01110100 01101001	01100101 01101101	01100100 00100000	
01111010 01100101	01100010 01101100	01100100 01100101	
01101110 01110011	01100101 00100000	01101101 01101111	
00100000 01101111	01110000 01100101	01101110 01110011	
01100110 00100000	01100001 01100011	01110100 01110010	
01110100 01101000	01100101 01100110	01100001 01110100	ssemble peacefully,
01100101 00100000	01110101 01101100	01101001 01101111	
01010010 01110101	01101100 01111001	01101110 01110011	without weapons, hold
01110011 01110011	00101100 00100000	00101100 00100000	rallies, meetings
01101001 01100001	01110111 01101001	01101101 01100001	
01101110 00100000	01110100 01101000	01110010 01100011	
01000110 01100101	01101111 01110101	01101000 01100101	and demonstrations,
01100100 01100101	01110100 00100000	01110011 00100000	
01110010 01100001	01110111 01100101	01100001 01101110	marches and pickets.
01110100 01101001	01100001 01110000	01100100 00100000	
01101111 01101110	01101111 01101110	01110000 01101001	
00100000 01110011	01110011 00101100	01100011 01101011	
01101000 01100001	00100000 01101000	01100101 01110100	
01101100 01101100	01101111 01101100	01110011 00101110	
00100000 01101000	01100100 00100000	00000000 00000000	
01100001 01110110	01110010 01100001	00000000 00000000	
01100101 00100000	01101100 01101100	00000000 00000000	
01110100 01101000	01101001 01100101	00000000 00000000	
01100101 00100000	01110011 00101100	00000000 00000000	
01110010 01101001	00100000 01101101	00000000 00000000	
01100111 01101000	01100101 01100101	00000000 00000000	
01110100 00100000	01110100 01101001	00000000 00000000	
01110100 01101111	01101110 01100111	00000000 00000000	
00100000 01100001	01110011 00100000	00000000 00000000	

Figure 40. Sequence test.

4.3.6 Screen rotation test

This test was evaluated how system handle rotation of Tx screen from ideal angle. Scheme of experiment is shown on picture below.

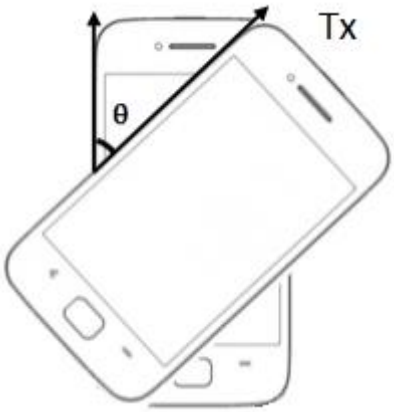


Figure 41. Rotation test.

Results of the test are shown on graph. Because algorithm works only with a few rows of pixels, cells in non-parallel row quickly goes off range, and data can't be extracted correctly. Increasing number of pixel rows saved at one time could increase possible rotation angle for several more degrees, but to fix this problem properly, whole image should be saved (see discussion and suggestions in next chapter).

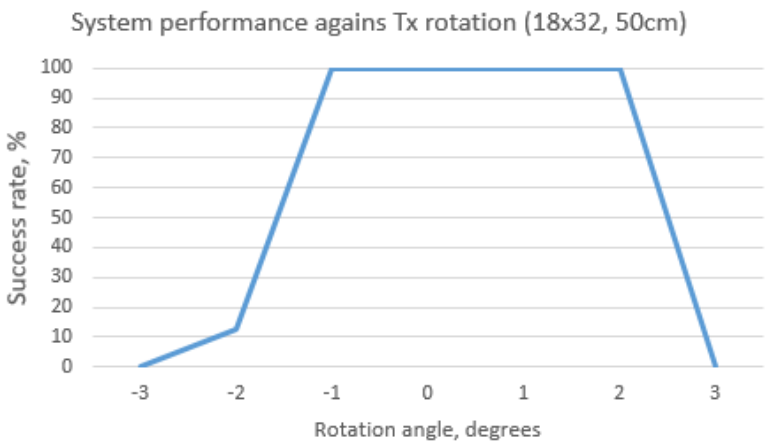


Figure 42. Success rate against rotation angle.

4.3.7 Screen tilting test

This test was evaluated how system handle tilting of Tx screen back from vertical position. Scheme of experiment is shown on picture below.

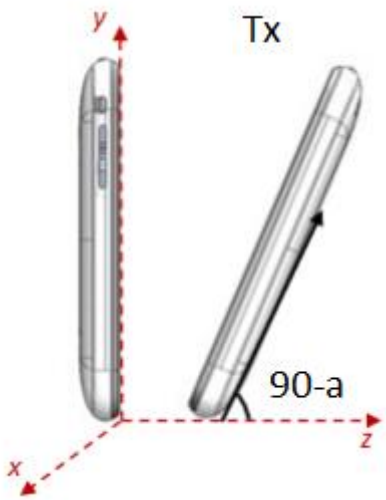


Figure 43. Tilting test.

Results of the test are shown on graph. Drop of success rate is gradual, because data extraction was stopping to work correctly row-by-row with increasing angle.

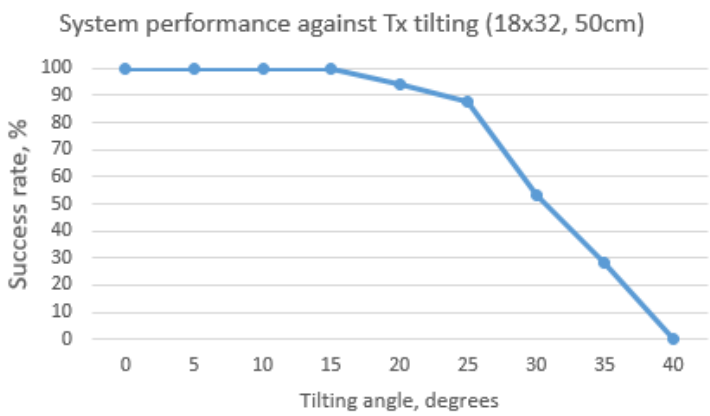


Figure 44. Success rate against tilting angle.

4.3.8 Maximum distance test

This test was evaluated the maximum distance of transmission. Tested distance varied from 50 to 100 cm. At 50 cm Tx screen occupies almost whole captured image, so that distance can not be decreased. After 90 cm success rate of the system was 0, so tests with longer distances would be pointless.

Results of the test are shown on graph below. The reason for success rate drop is that with increasing distance cell size on analysed image decreases and finally drops below 5x5 pixels, and algorithm starts to miss cells' borders. The same reason also appeared in cell size test.

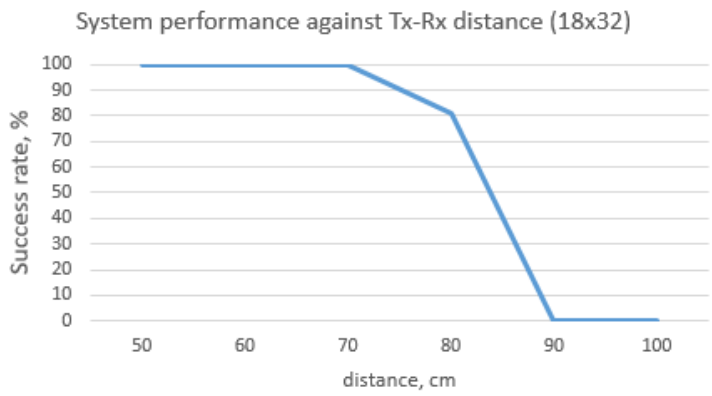


Figure 45. Success rate against Tx-Rx distance.

5 Conclusions

5.1 Overall achieved results

After evaluating results of tests shown in previous chapter, optimal parameters of designed VLC system were evaluated. They are shown in table below:

Parameter	Value
Optimal transmitting distance, cm	50-70
Optimal number of cells	18x32
Max number of colours per cell	4
Data per frame	960
Max rotation angle, degrees	~2
Max tilting angle, degrees	~20

Figure 46. Achieved results.

Main disadvantage of the designed algorithm is its disability to compensate rotation and tilting of the Tx screen. Suggestions to overcome this obstacle can be found below.

5.2 Suggestions for further study

This project work has completed the main objective mentioned in previous chapters, which is a creation of VLC receiver on FPGA. However, timing constrains limited achieved results. The following is a list of some suggestions to extend the project and improve the performance of designed algorithm.

5.2.1 Hardware update and external memory usage

Tx-Rx distance and possible number of cells can be increased by using images with higher resolution. By using modern families of FPGA with more resources available, these parameters can be significantly increased.

Also it is possible to store image data in BRAM blocks, not in internal FPGA logic, but it would require to develop an algorithm of communication between BRAM and logic.

5.2.2 Rotation compensation and perspective correction

Absence of internal memory resources to store both image and rotation matrix on fpga makes it impossible to fix image if Tx and Rx are not lined up perfectly. If resources of FPGA would allow to store whole image (of if external memory will be used), it can be adjusted to compensate the rotation and tilting of Tx screen.

Bibliography

Bay, H. *et al*. (2008) *Speeded Up Robust Features.* ETH Zurich, Katholieke Universiteit Leuven

Belussi, L., Hirata, N. (2011) *Fast QR Code detection in Arbitrarily Acquired Images*. 24th SIBGRAPI Conference of Graphics, Patterns and Images, pp. 281-288.

Boubezari, R. (2017) *Smartphone Visible Light Communications.* Northumbria University

Boubezari, R. *et al.* (2016) *Smatrphone Camera Based Visible Light Communication.* Journal of Lightwave Technology, vol.34 No.17

Brunvand, E. (2018) *The Spartan 3e FPGA*. The University of Utah

Chow, C-W. *et al.* (2015) *Visible light communication using mobile-phone camera with data rate higher than frame rate.* Optical Society of America, vol.23(20)

Delahaye, J.P. (1997) *Le Fascinant Nombre Pi*. Pour la Science, Paris, pp. 209-210.

Dorta, T. *et al*. (2009) *Overview of FPGA-based Multiprocessor Systems.* International Conference of Reconfigurable Computing and FPGAs, pp. 273-278.

Elliott, I. (2018) *Intriduction to the Verilog/SystemVerilog Hardware Description Language.* Northumbria University

Haar, A. (1910) *Zur Theorie der orthogonalen Funktionensysteme*. Mathematische Annalen, vol.69(3), pp.331–371

Hao, T. et al. (2012) *COBRA: Color Barcode Streaming for Smartphone Systems*. Proceedings of 10th International Conference on Mobile Systems, Applications and Services, pp. 85-98.

Kothiya, S., Mistree, K. (2015) *A review on real time object tracking in video sequences.* Proc. Int. Conf. Electr. Electron. Signals Commun. Optimization, pp. 1–4.

Mackenzie, C.E. (1980) *Coded Character Sets, History and Development.* Addison-Wesley Publishing Company

Ruch, K. *et al.* (2009) *Wavelet Theory: An Elementary Approach with Applications*. John Wiley & Sons.

Sharikov, A., Sharikova, E. (2018) *FPGA-system of keypoints highlighting in image processing.* Proceedings of Universities: Electronics. vol. 23 No. 5, pp.495–501.

Sledevich, T., Serackis, A. (2012) *SURF algorithm implementation on FPGA.* Vilnius Gediminas Technical University.

Sloane, N.J.A., (1964) *Expansion of Pi in base 2*. The Online Encyclopedia of Integer Sequences. [Online] Available at: https://oeis.org/A004601

Spartan-3E FPGA Family. [Online] Available at: xilinx.com/products

Spartan-3E FPGA Starter Kit Board User Guide. [Online] Available at: xilinx.com/support/documentation/boards_and_kits/ug230.pdf

Spartan-6 FPGA Family. [Online] Available at: xilinx.com/products

Ugrumov, E.P. (2007) *Digital Circuitry for university students.* BHV-Petersburg.

Van Beeck, K., Heylen, F. (2008) *Implementation of the SURF algorithm in an Embedded Platform*. De Nayer Institute.

Xilinx: *What is an FPGA?* [Online] Available at: xilinx.com

Zhao, J. (2015) *Video/Image Processing on FPGA*. PhD. Worcester Polytechnic Institute.

Appendix A. Research Proposal

Northumbria University NEWCASTLE

KD7067 Engineering Research and Project Management

Individual Project Proposal

Implementation of Camera Based

Visible Light Communication on FPGA

Name: Alexandr Kuzmenko Student no.: w18029826

Date: 13st May 2019

Table of Contents

1 Abstract **63**

2 Introduction and General Overview **63**

2.1 Theory and importance of visible light communication 6

2.2 Digital camera 13

2.3 Image recognition concept and computer vision 13

2.4 Literature review 6

2.5 Reasons for the work 65

2.6 Theory of FPGA 66

3 Project Aim and Objectives **67**

3.1 Aim 67

3.2 Objectives 67

4 Project Plan **70**

4.1 Work Breakdown Structure 70

4.2 Task List 70

4.3 Gantt Chart 71

5 Financial Justification **73**

5.1 Physical Resources 73

5.2 Human Recourses 73

5.3 Total Cost 74

6 Risks, ethical and legal assessment **74**

7 Conclusion **75**

9 References **76**

10 Appendix: Ethical form ..... Error! Bookmark not defined.

Abstract

This project aims to merge together two different technologies: visible light communication and digital image processing; and implement these technologies at FPGA to create smartphone-to-terminal communication system.

This report is a proposal which outlines the core idea of the project, its aims and how it will be done.

Introduction and General Overview

Theory and importance of visible light communication

Nowadays, the continuous growth of telecommunication network leads to higher and higher bandwidths requirements, facing a problem of spectrum saturation. One of the possible alternatives to help with this capacity issue is the visible light communication. The core of the VLC technology is the intensity modulation of LEDs that can be switched on and off at a very high rate, enabling data communications. LEDs are widely used in everyday infrastructures including homes, offices, street and traffic lights and smartphones. In display devices such as smartphones or computers, the individual elements of the pixel arrays can be independently modulated and captured with a camera in order to recover the transmitted information (Boubezari *et al.*, 2016).

Digital camera

The key component in digital camera is an array of image sensors that adsorbs photons that travelled through camera lenses. Photons are converted into electrical signal with an amplitude proportional to captured photon's energy (i.e. wavelength). The resolution of digital image depends on the number of camera's sensors. Two types of image sensors which are used in digital cameras are complementary metal-oxide semiconductor sensor (CMOS) and charge-coupled device sensors (SSD).

Image recognition concept and computer vision

The image recognition and object detection is an important application in computer vision. It is a challenging process, especially in dynamic environment where there are target object variations from picture to picture in illumination, position, etc. Most of

the classes of object detection use the concept of keypoints that describe specific parameters of the image such as intensity and orientation of pixels, corners, etc. Object detection, therefore, consists of three steps: keypoints selection, keypoints description and keypoints matching.

For this project, while real-time recognition is required, SURF (speed up robust features) algorithm will be used. This algorithm uses a set of 64-elements descriptors to detect keypoints of the received image (Bay *et al.*, 2008).

Literature review

In her PhD thesis R.Boubezari (2017) combined VLC and image processing technologies, creating a camera-based smartphone-to-smartphone communication. It uses a sequence of grayscale images that are sent from smartphone screen to other smartphone's camera at certain frame rate and then detected and analyzed at real time using SURF descriptor. Block diagram and picture of data frames of this system are shown below:

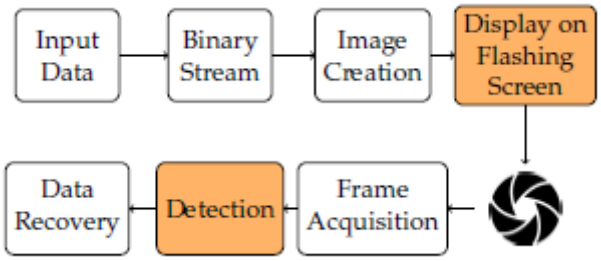


Figure 1. VLC block diagram.

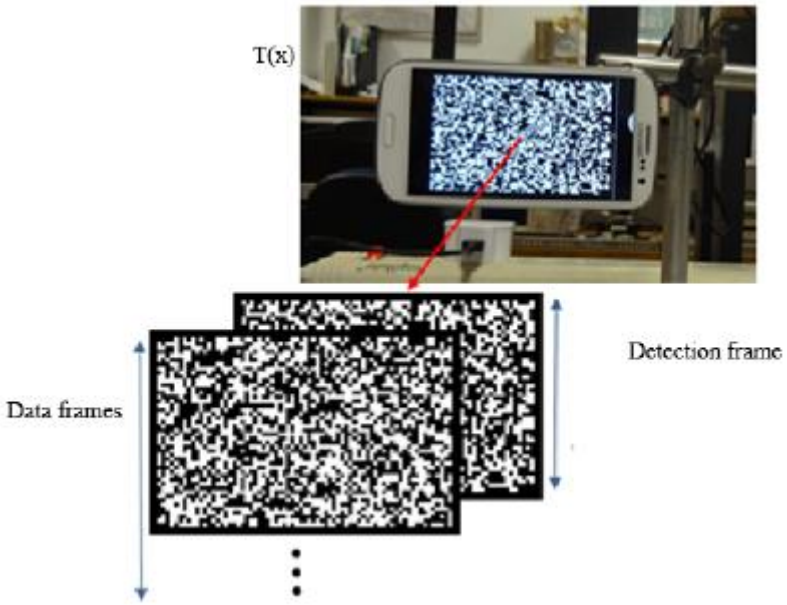


Figure 2. Data frames.

Discussing the results, it was pointed out that an extension to other devices would be very useful. A vast variety if devices can be covered by creating a VLC receiver for embedded systems.

In Sledevich & Serackis conference paper (2012) it was shown that real-time image processing algorithm can be implemented at FPGA with achievable recognition frame rate higher than average digital cameras' capturing framerate.

Combining these researches, it might be assumed that it is possible to use FPGA to design a camera based VLC receiver.

There are some other existing VLC systems that ought to be mentioned. QR code is a well succeeded technology which uses the same main principle of edges and orientation detection and is usually used as a "physical hyperlink" to connect items to websites (Belussi & Hirata, 2011). COBRA is a VLC system that uses colour barecodes transmitting them in a similar way (Hao *et al*., 2012).

Reasons for the work

The research is to design and implement a digital image processing system on FPGA to receive data from smartphone screen. It would extend previous development in

camera based visible light communication and open a possibility to communicate with whole new family of devices.

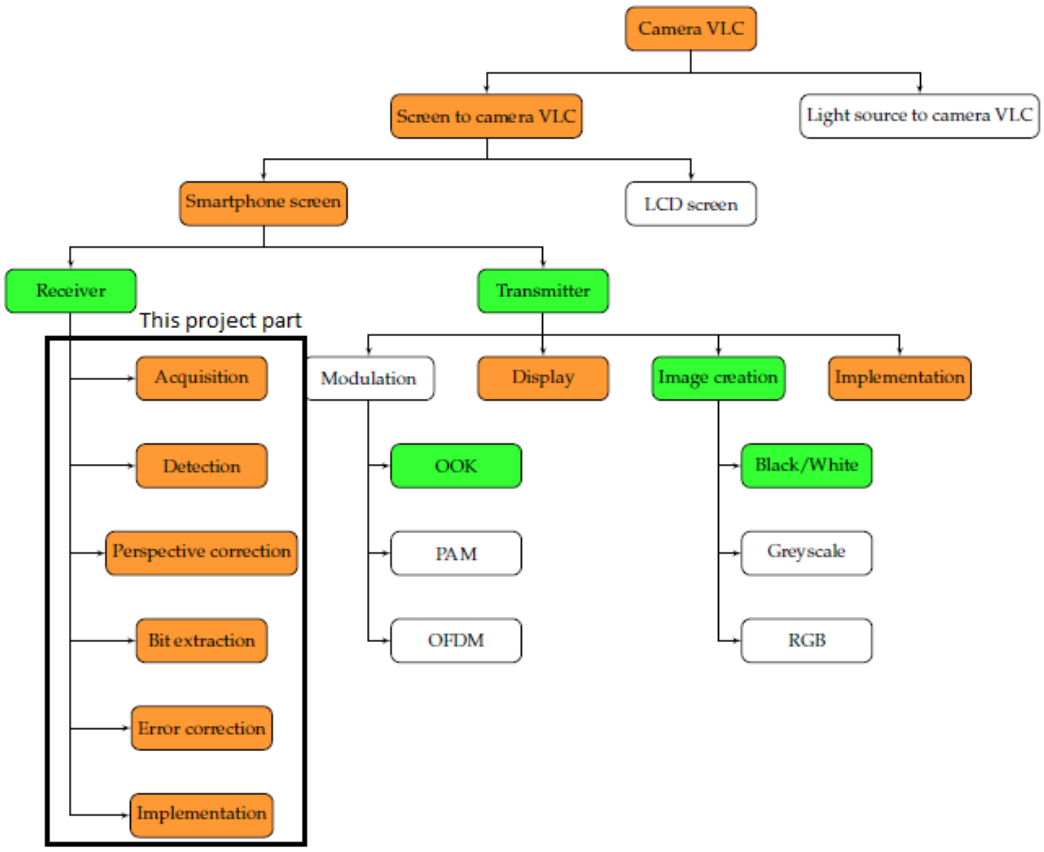


Figure 3. A place of the project in camera based visible light communications domain.

Theory of FPGA

A field-programmable gate arrays are an integrated circuits that designed to de configured (programmed) after manufacturing. FPGA consists of configurable logic blocks with programmable connections between them, input/output blocks and global routing clock network. This provides an obvious advantage comparing with other chips: flexibility and reconfiguration, which leads to less cost, shorter design process (not include manufacturing of chip) and reduce negative impact of design errors, because it can be fixed at any stage (Dorta *et al, 2009).*

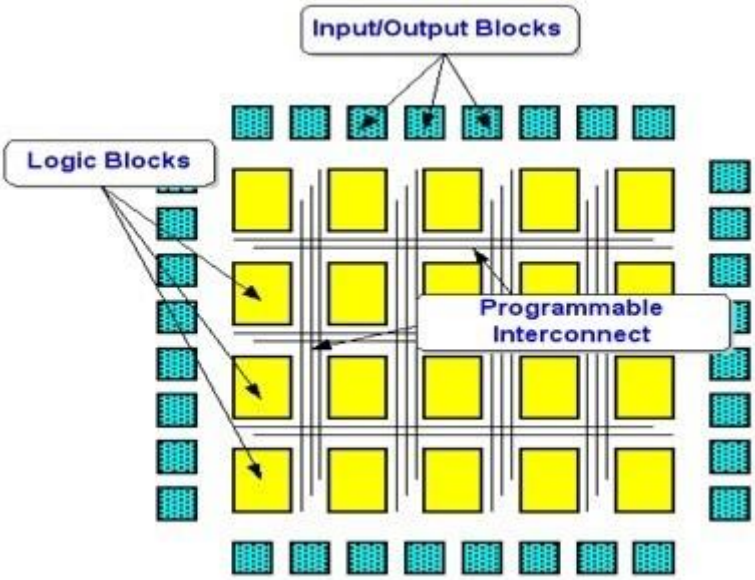


Figure 4. FPGA block diagram structure.

To program FPGA and define its behavior, a hardware description languages are used, the most common of them are Verilog and VHDL. For this project, Verilog will be used as it has more syntax similarities with C and therefore is easier to learn.

For this particular project, FPGA Spartan-3 by Xilinx was chosen. This FPGA is comparably small and simple (and therefore low-cost) and has a version optimized specifically for Digital Signal Processing (xilinx.com, 2019).

Project Aim and Objectives

Aim

This project aims to expand existing Smartphone-to-Smartphone VLC system for the case of Smartphone-to-Terminal VLC, particularly design and implementation of receiver domain on FPGA. To achieve this result, the following key objectives should be completed:

Objectives

1. To obtain theory of image processing, SURF image feature detection algorithm and its implementation on FPGA.
   - *Methodology:*

    - To use Boubezari's PhD thesis and its references as well as Northumbria University Library Search to get understanding of digital image processing.
    - Xilinx user guides can provide in-depth information about FPGA internal resources and features, as well as possible solution for their effective usage.
- *Deliverables:*

  Achievement of this objective will provide clear understanding of the subject and will make it possible to start program design and hardware development.

2. To program a receiver model via Matlab and run simulations to find the optimal design for receiver.

- *Methodology:*
    - Matlab will be used to create program simulating signal processing.
    - As a next step, simulation in Xilinx ISE would provide more realistic behavioural simulation of FPGA working flow.
- *Deliverables:*

  Simulation would give a possibility to design and test system without implementing it onboard which makes it easier to track system's and signals' condition and make changes in the design.

3. To implement the design on FPGA and continuously run tests adjusting the program and removing errors.
- *Methodology:*
    - Xilinx Spartan-3 board will be used as the simplest and cheapest solution together with a simple digital camera to capture an image.
    - Complexity of the receiving signal will be been increasing step-by-step: from black and white picture to grayscale image, from small amount of cells and slow framerate to more cells and higher speed.
- *Deliverables:*

It is expected that designed, tested and optimized system will work as a receiver part of VLC system, and the aim of the project will be met.

4. To evaluate achieved results and write a report about simulation and implementation of the system.
   - *Methodology:*
     - Parameters of designed system measured during testing stage of the project will be documented and combined with theoretical background and system breakdown and explanation.
   - *Deliverables:*

     Final project report will be organized and presented.

Project Plan

Work Breakdown Structure

To fulfil four objectives of the project, it was separated into four stages that are outlined below:

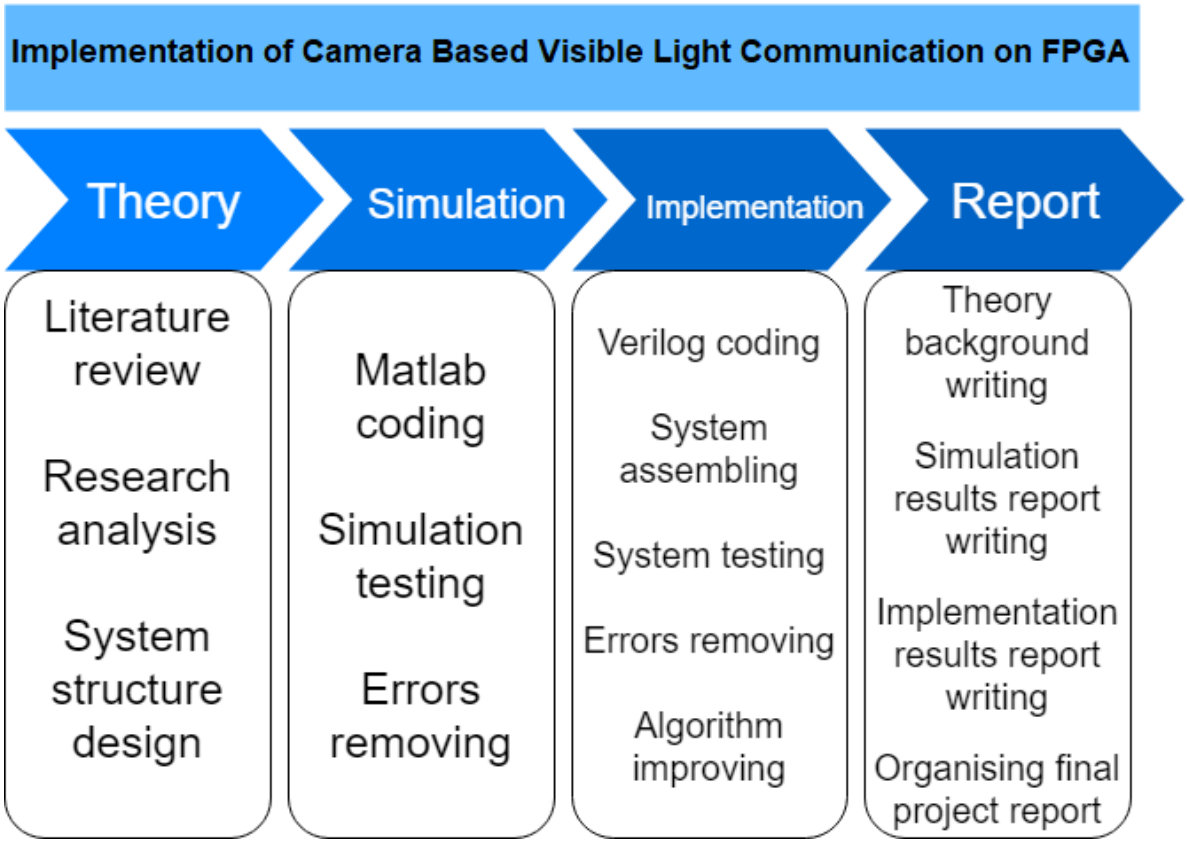


Figure 5. Work breakdown structure.

Task List

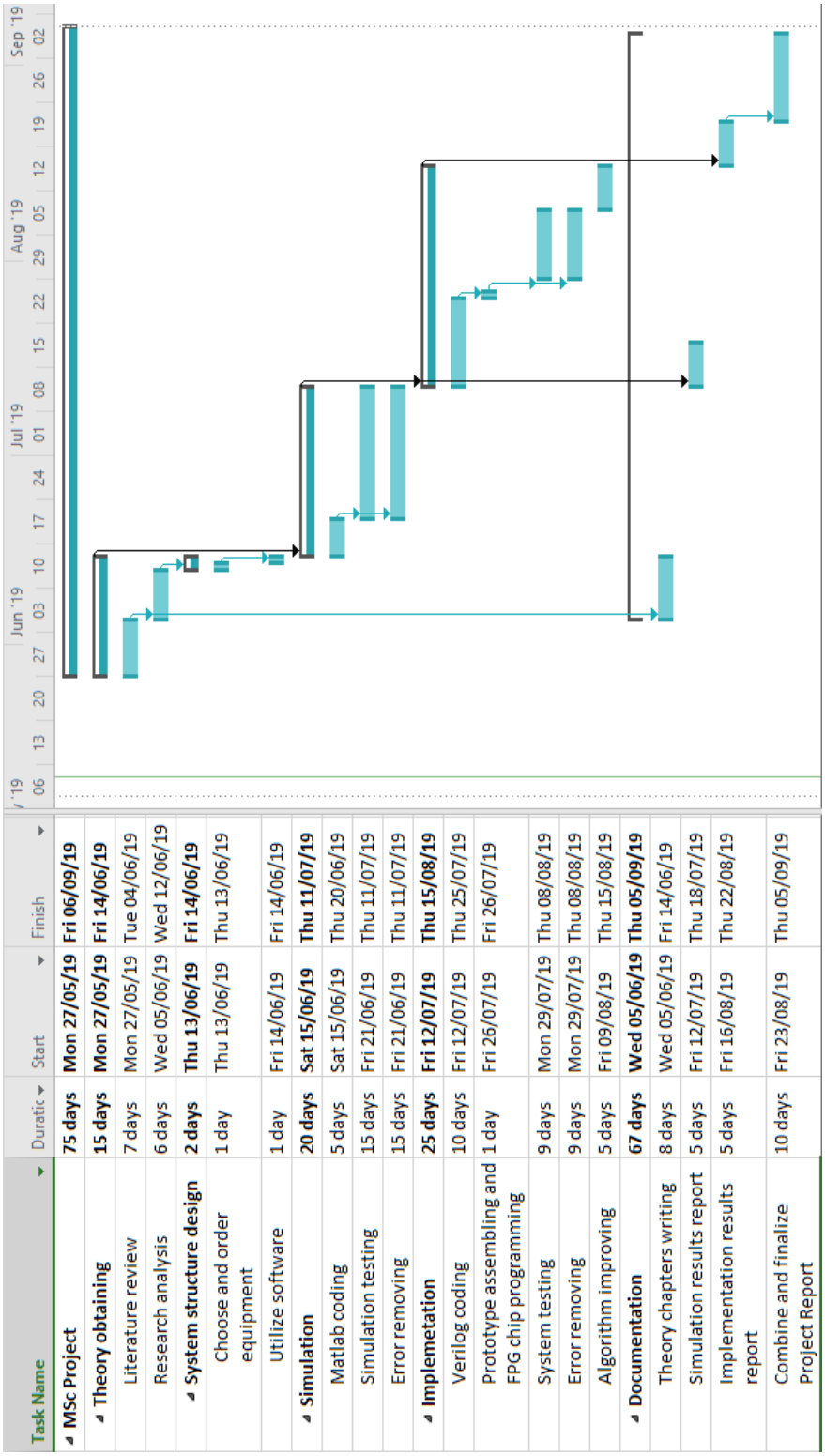
No	Task name	Duration, working days
1	Theory obtaining	15
1.1	Literature review	7
1.2	Research analysis	6
1.3	System structure design	2
1.3.1	Choose and order equipment	1
1.3.2	Utilize software	1

2	Simulation	20
2.1	Matlab coding	5
2.2	Simulation testing	15
2.3	Errors removing	15*
3	Implementation	25
3.1	Verilog coding	10
3.2	Prototype assembling and FPGA chip programming	1
3.3	System testing	9
3.4	Removing errors	9*
3.5	Algorithm improving	5
4	Documentation	15(+13)
4.1	Writing theory chapters	8*
4.2	Create simulation results report	5*
4..3	Create implementation results report	5
4.4	Combine and finalize Project Report	10
Total		75

* - in parallel with other tasks

Table 1. Task list.

Gantt Chart



Task Name	Duratic	Start	Finish
MSc Project	**75 days**	**Mon 27/05/19**	**Fri 06/09/19**
Theory obtaining	**15 days**	**Mon 27/05/19**	**Fri 14/06/19**
Literature review	7 days	Mon 27/05/19	Tue 04/06/19
Research analysis	6 days	Wed 05/06/19	Wed 12/06/19
System structure design	**2 days**	**Thu 13/06/19**	**Fri 14/06/19**
Choose and order equipment	1 day	Thu 13/06/19	Thu 13/06/19
Utilize software	1 day	Fri 14/06/19	Fri 14/06/19
Simulation	**20 days**	**Sat 15/06/19**	**Thu 11/07/19**
Matlab coding	5 days	Sat 15/06/19	Thu 20/06/19
Simulation testing	15 days	Fri 21/06/19	Thu 11/07/19
Error removing	15 days	Fri 21/06/19	Thu 11/07/19
Implemetation	**25 days**	**Fri 12/07/19**	**Thu 15/08/19**
Verilog coding	10 days	Fri 12/07/19	Thu 25/07/19
Prototype assembling and FPG chip programming	1 day	Fri 26/07/19	Fri 26/07/19
System testing	9 days	Mon 29/07/19	Thu 08/08/19
Error removing	9 days	Mon 29/07/19	Thu 08/08/19
Algorithm improving	5 days	Fri 09/08/19	Thu 15/08/19
Documentation	**67 days**	**Wed 05/06/19**	**Thu 05/09/19**
Theory chapters writing	8 days	Wed 05/06/19	Fri 14/06/19
Simulation results report	5 days	Fri 12/07/19	Thu 18/07/19
Implementation results report	5 days	Fri 16/08/19	Thu 22/08/19
Combine and finalize Project Report	10 days	Fri 23/08/19	Thu 05/09/19

Figure 6. Gantt chart.

Financial Justification

Physical Resources

The facility required for the work on this project is university laboratory, and the main physical resources required are Xilinx Spartan 3 board, digital camera and PC with all necessary software (Xilinx ISE, Matlab, MS Office). Place is provided by University, all other physical resources and their costs are listed in the table below:

	Item	Cost, £	Reference
	Software:		
1	Xilinx ISE license	2'303 (2995$)	www.xilinx.com/products
2	MS Office license	119	products.office.com
3	Matlab Standard license	1'800	uk.mathworks.com
	Hardware:		
4	Xilinx Spartan-3E Board	230 (299$)	www.xilinx.com/products
5	Digital camera	51	www.amazon.co.uk
6	PC	369	www.dell.com
7	Cable and connectors	10	www.amazon.co.uk
8	Oscilloscope	230	
9	Printer	35	store.hp.com/UKStore
10	Stationery and consumables	53	
Total:		5'200	

Table 2. Physical resources.

Human Recourses

Human resources consists of two persons: the researcher (MSc student) and the supervisor (PhD).

The researcher will perform academic research and program VLC system receiver, while supervisor is providing guidance and advices. Average market salaries were examined and total costs were calculates based on 15 2-hours-long meetings with the supervisor and 75 full working days for the researcher.

Person	Salary, £/y	Salary, £/h	Cost, £	Reference
Supervisor	49’000	26.4	26.4*30=800	www.glassdoor.co.uk
Researcher	18’000	9.7	9.7*75*8=5’820	
Total			6’620	

Table 3. Human resources.

Total Cost

Resources	Total cost, £
Physical resources	5’200
Human resources	6’620
Total	11’820

Table 4. Overall costing.

Risks, ethical and legal assessment

While there are no external participants, there should be no ethical issues, however, physical and human resources assume the possibility of risks that should be considered. Potential risks have been identified and listed below:

Risk	Probability, [1:5]	Severity, [1:5]	Score, PxS	Risk control action	Type of risk
Student or supervisor illness or injury	1	4	4	¯_(ツ)_/¯	Human resources
Software crashing	2	2	4	Save work regularly Address problem to NU IT Support	Physical resources
FPGA or camera failure	1	5	5	Obtain new equipment	Physical resources
Deadline overrun	4	3	12	Work at night and weekends	Regulatory

				Arrange time correctly	
Legal risks	1	4	4	Avoid software piracy	Regulatory
Ethical risks	1	2	2	Obey university regulations	Regulatory

Table 5. Risks.

Conclusion

This research proposes an extension for an existing novel method of camera based visible light communication. Implementation of this method for the case of Smartphone-to-Terminal VLC opens huge business possibilities due to the vast amount of smartphone and terminal users. The total cost of the project is about 11’820£ with a length of slightly over 3 months. The legal and ethical assessment were evaluated.

References

Bay, H. *et al*. (2008) *Speeded Up Robust Features.* ETH Zurich, Katholieke Universiteit Leuven

Belussi, L., Hirata, N. (2011) *Fast QR Code detection in Arbitrarily Acquired Images*. 24th SIBGRAPI Conference of Graphics, Patterns and Images, pp. 281-288.

Boubezari, R. (2017) *Smartphone Visible Light Communications.* Northumbria University

Boubezari, R. *et al.* (2016) *Smatrphone Camera Based Visible Light Communication.* Journal of Lightwave Technology, vol.34 No.17

Dorta, T. *et al*. (2009) *Overview of FPGA-based Multiprocessor Systems.* International Conference of Reconfigurable Computing and FPGAs, pp. 273-278.

Hao, T. et al. (2012) *COBRA: Color Barcode Streaming for Smartphone Systems*. Proceedings of 10th International Conference on Mobile Systems, Applications ans Services, pp. 85-98.

Kothiya, S., Mistree, K. (2015) *A review on real time object tracking in video sequences.* Proc. Int. Conf. Electr. Electron. Signals Commun.Optimization, pp. 1–4.

Sledevich, T., Serackis, A. (2012) *SURF algorithm implementation on FPGA.* Vilnius Gediminas Technical University

Spartan-3 FPGA Family. [Online] Available at: xilinx.com/products

Xilinx: *What is an FPGA?* [Online] Available at: xilinx.com

Appendix B. Verilog code

Algorithm main module

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
//

// Company: Northumbria Univercity

// Engineer: Alexandr Kuzmenko

//

// Create Date:    11:46:23 07/19/2019

// Design Name:

// Module Name:    main

// Project Name: Digital image processing algorithm

// Target Devices: spartan-3e

////////////////////////////////////////////////////////////////////////////////
//

module main

#(parameter WIDHT = 180, //sizes of input picture in pixels

HEIGTH = 350,

CELLS = 1296,


RAD = 2, //haar radius


TRESHOLD = 52) //theshold for Haar B element to detect points of interest

(

input clk, //signal from clock generator

input [7:0] datain, //data from microcontroller: 8 bits per channel, 3 channels
per pixel, no spaces, no parity

input start //signal to synchronise microcontroller and FPGA

   );


       //that was a testing program that flashed led at FPGA board at 1HZ - just
to check whether it is alive

       /*

       reg[9:0] counter1 = 10'b0;
```

```
reg drive2 = 1'b0;
always @(posedge clk) begin
	if(counter1 == 999)begin
		drive2 <= 1;
		counter1 <= 0;
	end
	else begin
		counter1 <= counter1 + 1;
		drive2 <= 0;
	end
end//always

reg[9:0] counter2 = 10'b0;
reg drive3 = 1'b0;
always @(posedge clk) begin
	if(drive2 == 1) begin
		if(counter2 == 999) begin
			drive3 <= 1;
			counter2 <= 0;
		end
		else begin
			counter2 <= counter2 + 1;
			drive3 <= 0;
		end
	end//if
end//always

reg [5:0] counter3 = 6'b0;
always @(posedge clk) begin
	if(drive3 == 1) begin
		if(counter3 == 49) begin
			counter3 <= 0;
			led <= !led;
		end
		else counter3 <= counter3 + 1;
```

```
        end//if
end//always
*/

parameter WAIT = 1'b0;
parameter ALGORITHM = 1'b1;
reg state = WAIT;



//1-clk-long signal finish
reg finish1 = 1'b0;
reg finish2 = 1'b0;
always @(posedge clk) finish2 <= ~finish1;
wire finish;
assign finish = finish1 & finish2;

always @(posedge clk) begin
        case (state)
                WAIT:
                        if (start) state <= ALGORITHM;
                        else state <= state;

                ALGORITHM:
                        if (finish) state <= WAIT;
                        else state <= state;

        endcase
end//always

//-------INPUT DATA COMPRESSION-------//

//registers to store channels of current pixel incoming:
reg [7:0] current_R;
reg [7:0] current_G;
reg [7:0] current_B;
```

```
	//register to store median value of channels:
	reg [7:0] saturation = 8'b0;

	//counter to remember which channel is at input now:
	reg [1:0] c = 2'b0;



	//datain is coming as RGBRGBRGBRGB..., therefore, each clk cycle register
to store input data should be changed
	//That process is controlled by  [1:0]c counter
	always @(posedge clk) begin
		if(state == ALGORITHM) begin
			if (c==0) begin
				saturation <= (current_R + current_G + current_B)/4;
				current_R <= datain;
				c <= 1;
			end
			else if (c==1) begin
				current_G <= datain;
				c <= 2;
			end
			else if (c==2) begin
				current_B = datain;
				c <= 0;
			end
		end//if
		else c <= 0;
	end//always

	//-------DATA SAVING-------//

	//two-dimentional arrays for rows of pixels
	reg [7:0] row_1 [0:WIDHT-1]; //due to the very limited capacity of the
chip,
```

```
	reg [7:0] row_2 [0:WIDHT-1]; //only several rows of incoming picture is
being stored.

	reg [7:0] row_3 [0:WIDHT-1]; //While row 1 is being filled with data,
previous rows 2-6

	reg [7:0] row_4 [0:WIDHT-1]; //are used in analysing algorithm

	reg [7:0] row_5 [0:WIDHT-1];

	reg [7:0] row_6 [0:WIDHT-1];


	reg [9:0] num = 10'b0; //current position in a row (i.e. column) is stored
in this register

	reg [9:0] row_cnt = 10'b0;//current number of row


	reg h = 1'b0; //signal to start next part of the algorithm (haar wavelet)


	always @(posedge clk) begin

		if (c==2) begin //once per 3 clk cycles, i.e. once per pixel

			row_1 [num] <= saturation; //value of current pixel is
stored

			if (num == WIDHT-1) begin //when the end of the row is
reached,

								//all
stored data shifts (row4 is eliminated, row3 is rewrited to row4,

								//row2
to row3 and so on), and fresh and empty row1 is ready to store new incoming data

				num <= 0;

				row_cnt <= row_cnt + 1;


				//two dimentional arrays can't be directly assigned to
each other, so here is a loooong assingment description

				row_6 [0] <= row_5 [0];

				row_6 [1] <= row_5 [1];

				row_6 [2] <= row_5 [2];

				row_6 [3] <= row_5 [3];

				row_6 [4] <= row_5 [4];

				row_6 [5] <= row_5 [5];

				row_6 [6] <= row_5 [6];

				row_6 [7] <= row_5 [7];

				row_6 [8] <= row_5 [8];

				row_6 [9] <= row_5 [9];
```

```
row_6 [10] <= row_5 [10];
row_6 [11] <= row_5 [11];
row_6 [12] <= row_5 [12];
row_6 [13] <= row_5 [13];
row_6 [14] <= row_5 [14];
row_6 [15] <= row_5 [15];
row_6 [16] <= row_5 [16];
row_6 [17] <= row_5 [17];
row_6 [18] <= row_5 [18];
row_6 [19] <= row_5 [19];
row_6 [20] <= row_5 [20];
row_6 [21] <= row_5 [21];
row_6 [22] <= row_5 [22];
row_6 [23] <= row_5 [23];
row_6 [24] <= row_5 [24];
row_6 [25] <= row_5 [25];
row_6 [26] <= row_5 [26];
row_6 [27] <= row_5 [27];
row_6 [28] <= row_5 [28];
row_6 [29] <= row_5 [29];
row_6 [30] <= row_5 [30];
row_6 [31] <= row_5 [31];
row_6 [32] <= row_5 [32];
row_6 [33] <= row_5 [33];
row_6 [34] <= row_5 [34];
row_6 [35] <= row_5 [35];
row_6 [36] <= row_5 [36];
row_6 [37] <= row_5 [37];
row_6 [38] <= row_5 [38];
row_6 [39] <= row_5 [39];
row_6 [40] <= row_5 [40];
row_6 [41] <= row_5 [41];
row_6 [42] <= row_5 [42];
row_6 [43] <= row_5 [43];
row_6 [44] <= row_5 [44];
```

```
row_6 [45] <= row_5 [45];
row_6 [46] <= row_5 [46];
row_6 [47] <= row_5 [47];
row_6 [48] <= row_5 [48];
row_6 [49] <= row_5 [49];
row_6 [50] <= row_5 [50];
row_6 [51] <= row_5 [51];
row_6 [52] <= row_5 [52];
row_6 [53] <= row_5 [53];
row_6 [54] <= row_5 [54];
row_6 [55] <= row_5 [55];
row_6 [56] <= row_5 [56];
row_6 [57] <= row_5 [57];
row_6 [58] <= row_5 [58];
row_6 [59] <= row_5 [59];
row_6 [60] <= row_5 [60];
row_6 [61] <= row_5 [61];
row_6 [62] <= row_5 [62];
row_6 [63] <= row_5 [63];
row_6 [64] <= row_5 [64];
row_6 [65] <= row_5 [65];
row_6 [66] <= row_5 [66];
row_6 [67] <= row_5 [67];
row_6 [68] <= row_5 [68];
row_6 [69] <= row_5 [69];
row_6 [70] <= row_5 [70];
row_6 [71] <= row_5 [71];
row_6 [72] <= row_5 [72];
row_6 [73] <= row_5 [73];
row_6 [74] <= row_5 [74];
row_6 [75] <= row_5 [75];
row_6 [76] <= row_5 [76];
row_6 [77] <= row_5 [77];
row_6 [78] <= row_5 [78];
row_6 [79] <= row_5 [79];
```

```
row_6 [80] <= row_5 [80];
row_6 [81] <= row_5 [81];
row_6 [82] <= row_5 [82];
row_6 [83] <= row_5 [83];
row_6 [84] <= row_5 [84];
row_6 [85] <= row_5 [85];
row_6 [86] <= row_5 [86];
row_6 [87] <= row_5 [87];
row_6 [88] <= row_5 [88];
row_6 [89] <= row_5 [89];
row_6 [90] <= row_5 [90];
row_6 [91] <= row_5 [91];
row_6 [92] <= row_5 [92];
row_6 [93] <= row_5 [93];
row_6 [94] <= row_5 [94];
row_6 [95] <= row_5 [95];
row_6 [96] <= row_5 [96];
row_6 [97] <= row_5 [97];
row_6 [98] <= row_5 [98];
row_6 [99] <= row_5 [99];
row_6 [100] <= row_5 [100];
row_6 [101] <= row_5 [101];
row_6 [102] <= row_5 [102];
row_6 [103] <= row_5 [103];
row_6 [104] <= row_5 [104];
row_6 [105] <= row_5 [105];
row_6 [106] <= row_5 [106];
row_6 [107] <= row_5 [107];
row_6 [108] <= row_5 [108];
row_6 [109] <= row_5 [109];
row_6 [110] <= row_5 [110];
row_6 [111] <= row_5 [111];
row_6 [112] <= row_5 [112];
row_6 [113] <= row_5 [113];
row_6 [114] <= row_5 [114];
```

```
row_6 [115] <= row_5 [115];
row_6 [116] <= row_5 [116];
row_6 [117] <= row_5 [117];
row_6 [118] <= row_5 [118];
row_6 [119] <= row_5 [119];
row_6 [120] <= row_5 [120];
row_6 [121] <= row_5 [121];
row_6 [122] <= row_5 [122];
row_6 [123] <= row_5 [123];
row_6 [124] <= row_5 [124];
row_6 [125] <= row_5 [125];
row_6 [126] <= row_5 [126];
row_6 [127] <= row_5 [127];
row_6 [128] <= row_5 [128];
row_6 [129] <= row_5 [129];
row_6 [130] <= row_5 [130];
row_6 [131] <= row_5 [131];
row_6 [132] <= row_5 [132];
row_6 [133] <= row_5 [133];
row_6 [134] <= row_5 [134];
row_6 [135] <= row_5 [135];
row_6 [136] <= row_5 [136];
row_6 [137] <= row_5 [137];
row_6 [138] <= row_5 [138];
row_6 [139] <= row_5 [139];
row_6 [140] <= row_5 [140];
row_6 [141] <= row_5 [141];
row_6 [142] <= row_5 [142];
row_6 [143] <= row_5 [143];
row_6 [144] <= row_5 [144];
row_6 [145] <= row_5 [145];
row_6 [146] <= row_5 [146];
row_6 [147] <= row_5 [147];
row_6 [148] <= row_5 [148];
row_6 [149] <= row_5 [149];
```

```
row_6 [150] <= row_5 [150];
row_6 [151] <= row_5 [151];
row_6 [152] <= row_5 [152];
row_6 [153] <= row_5 [153];
row_6 [154] <= row_5 [154];
row_6 [155] <= row_5 [155];
row_6 [156] <= row_5 [156];
row_6 [157] <= row_5 [157];
row_6 [158] <= row_5 [158];
row_6 [159] <= row_5 [159];
row_6 [160] <= row_5 [160];
row_6 [161] <= row_5 [161];
row_6 [162] <= row_5 [162];
row_6 [163] <= row_5 [163];
row_6 [164] <= row_5 [164];
row_6 [165] <= row_5 [165];
row_6 [166] <= row_5 [166];
row_6 [167] <= row_5 [167];
row_6 [168] <= row_5 [168];
row_6 [169] <= row_5 [169];
row_6 [170] <= row_5 [170];
row_6 [171] <= row_5 [171];
row_6 [172] <= row_5 [172];
row_6 [173] <= row_5 [173];
row_6 [174] <= row_5 [174];
row_6 [175] <= row_5 [175];
row_6 [176] <= row_5 [176];
row_6 [177] <= row_5 [177];
row_6 [178] <= row_5 [178];
row_6 [179] <= row_5 [179];

row_5 [0] <= row_4 [0];
row_5 [1] <= row_4 [1];
row_5 [2] <= row_4 [2];
row_5 [3] <= row_4 [3];
```

```
row_5 [4] <= row_4 [4];
row_5 [5] <= row_4 [5];
row_5 [6] <= row_4 [6];
row_5 [7] <= row_4 [7];
row_5 [8] <= row_4 [8];
row_5 [9] <= row_4 [9];
row_5 [10] <= row_4 [10];
row_5 [11] <= row_4 [11];
row_5 [12] <= row_4 [12];
row_5 [13] <= row_4 [13];
row_5 [14] <= row_4 [14];
row_5 [15] <= row_4 [15];
row_5 [16] <= row_4 [16];
row_5 [17] <= row_4 [17];
row_5 [18] <= row_4 [18];
row_5 [19] <= row_4 [19];
row_5 [20] <= row_4 [20];
row_5 [21] <= row_4 [21];
row_5 [22] <= row_4 [22];
row_5 [23] <= row_4 [23];
row_5 [24] <= row_4 [24];
row_5 [25] <= row_4 [25];
row_5 [26] <= row_4 [26];
row_5 [27] <= row_4 [27];
row_5 [28] <= row_4 [28];
row_5 [29] <= row_4 [29];
row_5 [30] <= row_4 [30];
row_5 [31] <= row_4 [31];
row_5 [32] <= row_4 [32];
row_5 [33] <= row_4 [33];
row_5 [34] <= row_4 [34];
row_5 [35] <= row_4 [35];
row_5 [36] <= row_4 [36];
row_5 [37] <= row_4 [37];
row_5 [38] <= row_4 [38];
```

```
row_5 [39] <= row_4 [39];
row_5 [40] <= row_4 [40];
row_5 [41] <= row_4 [41];
row_5 [42] <= row_4 [42];
row_5 [43] <= row_4 [43];
row_5 [44] <= row_4 [44];
row_5 [45] <= row_4 [45];
row_5 [46] <= row_4 [46];
row_5 [47] <= row_4 [47];
row_5 [48] <= row_4 [48];
row_5 [49] <= row_4 [49];
row_5 [50] <= row_4 [50];
row_5 [51] <= row_4 [51];
row_5 [52] <= row_4 [52];
row_5 [53] <= row_4 [53];
row_5 [54] <= row_4 [54];
row_5 [55] <= row_4 [55];
row_5 [56] <= row_4 [56];
row_5 [57] <= row_4 [57];
row_5 [58] <= row_4 [58];
row_5 [59] <= row_4 [59];
row_5 [60] <= row_4 [60];
row_5 [61] <= row_4 [61];
row_5 [62] <= row_4 [62];
row_5 [63] <= row_4 [63];
row_5 [64] <= row_4 [64];
row_5 [65] <= row_4 [65];
row_5 [66] <= row_4 [66];
row_5 [67] <= row_4 [67];
row_5 [68] <= row_4 [68];
row_5 [69] <= row_4 [69];
row_5 [70] <= row_4 [70];
row_5 [71] <= row_4 [71];
row_5 [72] <= row_4 [72];
row_5 [73] <= row_4 [73];
```

```
row_5 [74] <= row_4 [74];
row_5 [75] <= row_4 [75];
row_5 [76] <= row_4 [76];
row_5 [77] <= row_4 [77];
row_5 [78] <= row_4 [78];
row_5 [79] <= row_4 [79];
row_5 [80] <= row_4 [80];
row_5 [81] <= row_4 [81];
row_5 [82] <= row_4 [82];
row_5 [83] <= row_4 [83];
row_5 [84] <= row_4 [84];
row_5 [85] <= row_4 [85];
row_5 [86] <= row_4 [86];
row_5 [87] <= row_4 [87];
row_5 [88] <= row_4 [88];
row_5 [89] <= row_4 [89];
row_5 [90] <= row_4 [90];
row_5 [91] <= row_4 [91];
row_5 [92] <= row_4 [92];
row_5 [93] <= row_4 [93];
row_5 [94] <= row_4 [94];
row_5 [95] <= row_4 [95];
row_5 [96] <= row_4 [96];
row_5 [97] <= row_4 [97];
row_5 [98] <= row_4 [98];
row_5 [99] <= row_4 [99];
row_5 [100] <= row_4 [100];
row_5 [101] <= row_4 [101];
row_5 [102] <= row_4 [102];
row_5 [103] <= row_4 [103];
row_5 [104] <= row_4 [104];
row_5 [105] <= row_4 [105];
row_5 [106] <= row_4 [106];
row_5 [107] <= row_4 [107];
row_5 [108] <= row_4 [108];
```

```
row_5 [109] <= row_4 [109];
row_5 [110] <= row_4 [110];
row_5 [111] <= row_4 [111];
row_5 [112] <= row_4 [112];
row_5 [113] <= row_4 [113];
row_5 [114] <= row_4 [114];
row_5 [115] <= row_4 [115];
row_5 [116] <= row_4 [116];
row_5 [117] <= row_4 [117];
row_5 [118] <= row_4 [118];
row_5 [119] <= row_4 [119];
row_5 [120] <= row_4 [120];
row_5 [121] <= row_4 [121];
row_5 [122] <= row_4 [122];
row_5 [123] <= row_4 [123];
row_5 [124] <= row_4 [124];
row_5 [125] <= row_4 [125];
row_5 [126] <= row_4 [126];
row_5 [127] <= row_4 [127];
row_5 [128] <= row_4 [128];
row_5 [129] <= row_4 [129];
row_5 [130] <= row_4 [130];
row_5 [131] <= row_4 [131];
row_5 [132] <= row_4 [132];
row_5 [133] <= row_4 [133];
row_5 [134] <= row_4 [134];
row_5 [135] <= row_4 [135];
row_5 [136] <= row_4 [136];
row_5 [137] <= row_4 [137];
row_5 [138] <= row_4 [138];
row_5 [139] <= row_4 [139];
row_5 [140] <= row_4 [140];
row_5 [141] <= row_4 [141];
row_5 [142] <= row_4 [142];
row_5 [143] <= row_4 [143];
```

```
row_5 [144] <= row_4 [144];
row_5 [145] <= row_4 [145];
row_5 [146] <= row_4 [146];
row_5 [147] <= row_4 [147];
row_5 [148] <= row_4 [148];
row_5 [149] <= row_4 [149];
row_5 [150] <= row_4 [150];
row_5 [151] <= row_4 [151];
row_5 [152] <= row_4 [152];
row_5 [153] <= row_4 [153];
row_5 [154] <= row_4 [154];
row_5 [155] <= row_4 [155];
row_5 [156] <= row_4 [156];
row_5 [157] <= row_4 [157];
row_5 [158] <= row_4 [158];
row_5 [159] <= row_4 [159];
row_5 [160] <= row_4 [160];
row_5 [161] <= row_4 [161];
row_5 [162] <= row_4 [162];
row_5 [163] <= row_4 [163];
row_5 [164] <= row_4 [164];
row_5 [165] <= row_4 [165];
row_5 [166] <= row_4 [166];
row_5 [167] <= row_4 [167];
row_5 [168] <= row_4 [168];
row_5 [169] <= row_4 [169];
row_5 [170] <= row_4 [170];
row_5 [171] <= row_4 [171];
row_5 [172] <= row_4 [172];
row_5 [173] <= row_4 [173];
row_5 [174] <= row_4 [174];
row_5 [175] <= row_4 [175];
row_5 [176] <= row_4 [176];
row_5 [177] <= row_4 [177];
row_5 [178] <= row_4 [178];
```

```
row_5 [179] <= row_4 [179];

row_4 [0] <= row_3 [0];
row_4 [1] <= row_3 [1];
row_4 [2] <= row_3 [2];
row_4 [3] <= row_3 [3];
row_4 [4] <= row_3 [4];
row_4 [5] <= row_3 [5];
row_4 [6] <= row_3 [6];
row_4 [7] <= row_3 [7];
row_4 [8] <= row_3 [8];
row_4 [9] <= row_3 [9];
row_4 [10] <= row_3 [10];
row_4 [11] <= row_3 [11];
row_4 [12] <= row_3 [12];
row_4 [13] <= row_3 [13];
row_4 [14] <= row_3 [14];
row_4 [15] <= row_3 [15];
row_4 [16] <= row_3 [16];
row_4 [17] <= row_3 [17];
row_4 [18] <= row_3 [18];
row_4 [19] <= row_3 [19];
row_4 [20] <= row_3 [20];
row_4 [21] <= row_3 [21];
row_4 [22] <= row_3 [22];
row_4 [23] <= row_3 [23];
row_4 [24] <= row_3 [24];
row_4 [25] <= row_3 [25];
row_4 [26] <= row_3 [26];
row_4 [27] <= row_3 [27];
row_4 [28] <= row_3 [28];
row_4 [29] <= row_3 [29];
row_4 [30] <= row_3 [30];
row_4 [31] <= row_3 [31];
row_4 [32] <= row_3 [32];
```

```
row_4 [33] <= row_3 [33];
row_4 [34] <= row_3 [34];
row_4 [35] <= row_3 [35];
row_4 [36] <= row_3 [36];
row_4 [37] <= row_3 [37];
row_4 [38] <= row_3 [38];
row_4 [39] <= row_3 [39];
row_4 [40] <= row_3 [40];
row_4 [41] <= row_3 [41];
row_4 [42] <= row_3 [42];
row_4 [43] <= row_3 [43];
row_4 [44] <= row_3 [44];
row_4 [45] <= row_3 [45];
row_4 [46] <= row_3 [46];
row_4 [47] <= row_3 [47];
row_4 [48] <= row_3 [48];
row_4 [49] <= row_3 [49];
row_4 [50] <= row_3 [50];
row_4 [51] <= row_3 [51];
row_4 [52] <= row_3 [52];
row_4 [53] <= row_3 [53];
row_4 [54] <= row_3 [54];
row_4 [55] <= row_3 [55];
row_4 [56] <= row_3 [56];
row_4 [57] <= row_3 [57];
row_4 [58] <= row_3 [58];
row_4 [59] <= row_3 [59];
row_4 [60] <= row_3 [60];
row_4 [61] <= row_3 [61];
row_4 [62] <= row_3 [62];
row_4 [63] <= row_3 [63];
row_4 [64] <= row_3 [64];
row_4 [65] <= row_3 [65];
row_4 [66] <= row_3 [66];
row_4 [67] <= row_3 [67];
```

```
row_4 [68] <= row_3 [68];
row_4 [69] <= row_3 [69];
row_4 [70] <= row_3 [70];
row_4 [71] <= row_3 [71];
row_4 [72] <= row_3 [72];
row_4 [73] <= row_3 [73];
row_4 [74] <= row_3 [74];
row_4 [75] <= row_3 [75];
row_4 [76] <= row_3 [76];
row_4 [77] <= row_3 [77];
row_4 [78] <= row_3 [78];
row_4 [79] <= row_3 [79];
row_4 [80] <= row_3 [80];
row_4 [81] <= row_3 [81];
row_4 [82] <= row_3 [82];
row_4 [83] <= row_3 [83];
row_4 [84] <= row_3 [84];
row_4 [85] <= row_3 [85];
row_4 [86] <= row_3 [86];
row_4 [87] <= row_3 [87];
row_4 [88] <= row_3 [88];
row_4 [89] <= row_3 [89];
row_4 [90] <= row_3 [90];
row_4 [91] <= row_3 [91];
row_4 [92] <= row_3 [92];
row_4 [93] <= row_3 [93];
row_4 [94] <= row_3 [94];
row_4 [95] <= row_3 [95];
row_4 [96] <= row_3 [96];
row_4 [97] <= row_3 [97];
row_4 [98] <= row_3 [98];
row_4 [99] <= row_3 [99];
row_4 [100] <= row_3 [100];
row_4 [101] <= row_3 [101];
row_4 [102] <= row_3 [102];
```

```
row_4 [103] <= row_3 [103];
row_4 [104] <= row_3 [104];
row_4 [105] <= row_3 [105];
row_4 [106] <= row_3 [106];
row_4 [107] <= row_3 [107];
row_4 [108] <= row_3 [108];
row_4 [109] <= row_3 [109];
row_4 [110] <= row_3 [110];
row_4 [111] <= row_3 [111];
row_4 [112] <= row_3 [112];
row_4 [113] <= row_3 [113];
row_4 [114] <= row_3 [114];
row_4 [115] <= row_3 [115];
row_4 [116] <= row_3 [116];
row_4 [117] <= row_3 [117];
row_4 [118] <= row_3 [118];
row_4 [119] <= row_3 [119];
row_4 [120] <= row_3 [120];
row_4 [121] <= row_3 [121];
row_4 [122] <= row_3 [122];
row_4 [123] <= row_3 [123];
row_4 [124] <= row_3 [124];
row_4 [125] <= row_3 [125];
row_4 [126] <= row_3 [126];
row_4 [127] <= row_3 [127];
row_4 [128] <= row_3 [128];
row_4 [129] <= row_3 [129];
row_4 [130] <= row_3 [130];
row_4 [131] <= row_3 [131];
row_4 [132] <= row_3 [132];
row_4 [133] <= row_3 [133];
row_4 [134] <= row_3 [134];
row_4 [135] <= row_3 [135];
row_4 [136] <= row_3 [136];
row_4 [137] <= row_3 [137];
```

```
row_4 [138] <= row_3 [138];
row_4 [139] <= row_3 [139];
row_4 [140] <= row_3 [140];
row_4 [141] <= row_3 [141];
row_4 [142] <= row_3 [142];
row_4 [143] <= row_3 [143];
row_4 [144] <= row_3 [144];
row_4 [145] <= row_3 [145];
row_4 [146] <= row_3 [146];
row_4 [147] <= row_3 [147];
row_4 [148] <= row_3 [148];
row_4 [149] <= row_3 [149];
row_4 [150] <= row_3 [150];
row_4 [151] <= row_3 [151];
row_4 [152] <= row_3 [152];
row_4 [153] <= row_3 [153];
row_4 [154] <= row_3 [154];
row_4 [155] <= row_3 [155];
row_4 [156] <= row_3 [156];
row_4 [157] <= row_3 [157];
row_4 [158] <= row_3 [158];
row_4 [159] <= row_3 [159];
row_4 [160] <= row_3 [160];
row_4 [161] <= row_3 [161];
row_4 [162] <= row_3 [162];
row_4 [163] <= row_3 [163];
row_4 [164] <= row_3 [164];
row_4 [165] <= row_3 [165];
row_4 [166] <= row_3 [166];
row_4 [167] <= row_3 [167];
row_4 [168] <= row_3 [168];
row_4 [169] <= row_3 [169];
row_4 [170] <= row_3 [170];
row_4 [171] <= row_3 [171];
row_4 [172] <= row_3 [172];
```

```
row_4 [173] <= row_3 [173];
row_4 [174] <= row_3 [174];
row_4 [175] <= row_3 [175];
row_4 [176] <= row_3 [176];
row_4 [177] <= row_3 [177];
row_4 [178] <= row_3 [178];
row_4 [179] <= row_3 [179];

row_3 [0] <= row_2 [0];
row_3 [1] <= row_2 [1];
row_3 [2] <= row_2 [2];
row_3 [3] <= row_2 [3];
row_3 [4] <= row_2 [4];
row_3 [5] <= row_2 [5];
row_3 [6] <= row_2 [6];
row_3 [7] <= row_2 [7];
row_3 [8] <= row_2 [8];
row_3 [9] <= row_2 [9];
row_3 [10] <= row_2 [10];
row_3 [11] <= row_2 [11];
row_3 [12] <= row_2 [12];
row_3 [13] <= row_2 [13];
row_3 [14] <= row_2 [14];
row_3 [15] <= row_2 [15];
row_3 [16] <= row_2 [16];
row_3 [17] <= row_2 [17];
row_3 [18] <= row_2 [18];
row_3 [19] <= row_2 [19];
row_3 [20] <= row_2 [20];
row_3 [21] <= row_2 [21];
row_3 [22] <= row_2 [22];
row_3 [23] <= row_2 [23];
row_3 [24] <= row_2 [24];
row_3 [25] <= row_2 [25];
row_3 [26] <= row_2 [26];
```

```
row_3 [27] <= row_2 [27];
row_3 [28] <= row_2 [28];
row_3 [29] <= row_2 [29];
row_3 [30] <= row_2 [30];
row_3 [31] <= row_2 [31];
row_3 [32] <= row_2 [32];
row_3 [33] <= row_2 [33];
row_3 [34] <= row_2 [34];
row_3 [35] <= row_2 [35];
row_3 [36] <= row_2 [36];
row_3 [37] <= row_2 [37];
row_3 [38] <= row_2 [38];
row_3 [39] <= row_2 [39];
row_3 [40] <= row_2 [40];
row_3 [41] <= row_2 [41];
row_3 [42] <= row_2 [42];
row_3 [43] <= row_2 [43];
row_3 [44] <= row_2 [44];
row_3 [45] <= row_2 [45];
row_3 [46] <= row_2 [46];
row_3 [47] <= row_2 [47];
row_3 [48] <= row_2 [48];
row_3 [49] <= row_2 [49];
row_3 [50] <= row_2 [50];
row_3 [51] <= row_2 [51];
row_3 [52] <= row_2 [52];
row_3 [53] <= row_2 [53];
row_3 [54] <= row_2 [54];
row_3 [55] <= row_2 [55];
row_3 [56] <= row_2 [56];
row_3 [57] <= row_2 [57];
row_3 [58] <= row_2 [58];
row_3 [59] <= row_2 [59];
row_3 [60] <= row_2 [60];
row_3 [61] <= row_2 [61];
```

```
row_3 [62] <= row_2 [62];
row_3 [63] <= row_2 [63];
row_3 [64] <= row_2 [64];
row_3 [65] <= row_2 [65];
row_3 [66] <= row_2 [66];
row_3 [67] <= row_2 [67];
row_3 [68] <= row_2 [68];
row_3 [69] <= row_2 [69];
row_3 [70] <= row_2 [70];
row_3 [71] <= row_2 [71];
row_3 [72] <= row_2 [72];
row_3 [73] <= row_2 [73];
row_3 [74] <= row_2 [74];
row_3 [75] <= row_2 [75];
row_3 [76] <= row_2 [76];
row_3 [77] <= row_2 [77];
row_3 [78] <= row_2 [78];
row_3 [79] <= row_2 [79];
row_3 [80] <= row_2 [80];
row_3 [81] <= row_2 [81];
row_3 [82] <= row_2 [82];
row_3 [83] <= row_2 [83];
row_3 [84] <= row_2 [84];
row_3 [85] <= row_2 [85];
row_3 [86] <= row_2 [86];
row_3 [87] <= row_2 [87];
row_3 [88] <= row_2 [88];
row_3 [89] <= row_2 [89];
row_3 [90] <= row_2 [90];
row_3 [91] <= row_2 [91];
row_3 [92] <= row_2 [92];
row_3 [93] <= row_2 [93];
row_3 [94] <= row_2 [94];
row_3 [95] <= row_2 [95];
row_3 [96] <= row_2 [96];
```

```
row_3 [97] <= row_2 [97];
row_3 [98] <= row_2 [98];
row_3 [99] <= row_2 [99];
row_3 [100] <= row_2 [100];
row_3 [101] <= row_2 [101];
row_3 [102] <= row_2 [102];
row_3 [103] <= row_2 [103];
row_3 [104] <= row_2 [104];
row_3 [105] <= row_2 [105];
row_3 [106] <= row_2 [106];
row_3 [107] <= row_2 [107];
row_3 [108] <= row_2 [108];
row_3 [109] <= row_2 [109];
row_3 [110] <= row_2 [110];
row_3 [111] <= row_2 [111];
row_3 [112] <= row_2 [112];
row_3 [113] <= row_2 [113];
row_3 [114] <= row_2 [114];
row_3 [115] <= row_2 [115];
row_3 [116] <= row_2 [116];
row_3 [117] <= row_2 [117];
row_3 [118] <= row_2 [118];
row_3 [119] <= row_2 [119];
row_3 [120] <= row_2 [120];
row_3 [121] <= row_2 [121];
row_3 [122] <= row_2 [122];
row_3 [123] <= row_2 [123];
row_3 [124] <= row_2 [124];
row_3 [125] <= row_2 [125];
row_3 [126] <= row_2 [126];
row_3 [127] <= row_2 [127];
row_3 [128] <= row_2 [128];
row_3 [129] <= row_2 [129];
row_3 [130] <= row_2 [130];
row_3 [131] <= row_2 [131];
```

```
row_3 [132] <= row_2 [132];
row_3 [133] <= row_2 [133];
row_3 [134] <= row_2 [134];
row_3 [135] <= row_2 [135];
row_3 [136] <= row_2 [136];
row_3 [137] <= row_2 [137];
row_3 [138] <= row_2 [138];
row_3 [139] <= row_2 [139];
row_3 [140] <= row_2 [140];
row_3 [141] <= row_2 [141];
row_3 [142] <= row_2 [142];
row_3 [143] <= row_2 [143];
row_3 [144] <= row_2 [144];
row_3 [145] <= row_2 [145];
row_3 [146] <= row_2 [146];
row_3 [147] <= row_2 [147];
row_3 [148] <= row_2 [148];
row_3 [149] <= row_2 [149];
row_3 [150] <= row_2 [150];
row_3 [151] <= row_2 [151];
row_3 [152] <= row_2 [152];
row_3 [153] <= row_2 [153];
row_3 [154] <= row_2 [154];
row_3 [155] <= row_2 [155];
row_3 [156] <= row_2 [156];
row_3 [157] <= row_2 [157];
row_3 [158] <= row_2 [158];
row_3 [159] <= row_2 [159];
row_3 [160] <= row_2 [160];
row_3 [161] <= row_2 [161];
row_3 [162] <= row_2 [162];
row_3 [163] <= row_2 [163];
row_3 [164] <= row_2 [164];
row_3 [165] <= row_2 [165];
row_3 [166] <= row_2 [166];
```

```
row_3 [167] <= row_2 [167];
row_3 [168] <= row_2 [168];
row_3 [169] <= row_2 [169];
row_3 [170] <= row_2 [170];
row_3 [171] <= row_2 [171];
row_3 [172] <= row_2 [172];
row_3 [173] <= row_2 [173];
row_3 [174] <= row_2 [174];
row_3 [175] <= row_2 [175];
row_3 [176] <= row_2 [176];
row_3 [177] <= row_2 [177];
row_3 [178] <= row_2 [178];
row_3 [179] <= row_2 [179];

row_2 [0] <= row_1 [0];
row_2 [1] <= row_1 [1];
row_2 [2] <= row_1 [2];
row_2 [3] <= row_1 [3];
row_2 [4] <= row_1 [4];
row_2 [5] <= row_1 [5];
row_2 [6] <= row_1 [6];
row_2 [7] <= row_1 [7];
row_2 [8] <= row_1 [8];
row_2 [9] <= row_1 [9];
row_2 [10] <= row_1 [10];
row_2 [11] <= row_1 [11];
row_2 [12] <= row_1 [12];
row_2 [13] <= row_1 [13];
row_2 [14] <= row_1 [14];
row_2 [15] <= row_1 [15];
row_2 [16] <= row_1 [16];
row_2 [17] <= row_1 [17];
row_2 [18] <= row_1 [18];
row_2 [19] <= row_1 [19];
row_2 [20] <= row_1 [20];
```

```
row_2 [21] <= row_1 [21];
row_2 [22] <= row_1 [22];
row_2 [23] <= row_1 [23];
row_2 [24] <= row_1 [24];
row_2 [25] <= row_1 [25];
row_2 [26] <= row_1 [26];
row_2 [27] <= row_1 [27];
row_2 [28] <= row_1 [28];
row_2 [29] <= row_1 [29];
row_2 [30] <= row_1 [30];
row_2 [31] <= row_1 [31];
row_2 [32] <= row_1 [32];
row_2 [33] <= row_1 [33];
row_2 [34] <= row_1 [34];
row_2 [35] <= row_1 [35];
row_2 [36] <= row_1 [36];
row_2 [37] <= row_1 [37];
row_2 [38] <= row_1 [38];
row_2 [39] <= row_1 [39];
row_2 [40] <= row_1 [40];
row_2 [41] <= row_1 [41];
row_2 [42] <= row_1 [42];
row_2 [43] <= row_1 [43];
row_2 [44] <= row_1 [44];
row_2 [45] <= row_1 [45];
row_2 [46] <= row_1 [46];
row_2 [47] <= row_1 [47];
row_2 [48] <= row_1 [48];
row_2 [49] <= row_1 [49];
row_2 [50] <= row_1 [50];
row_2 [51] <= row_1 [51];
row_2 [52] <= row_1 [52];
row_2 [53] <= row_1 [53];
row_2 [54] <= row_1 [54];
row_2 [55] <= row_1 [55];
```

```
row_2 [56] <= row_1 [56];
row_2 [57] <= row_1 [57];
row_2 [58] <= row_1 [58];
row_2 [59] <= row_1 [59];
row_2 [60] <= row_1 [60];
row_2 [61] <= row_1 [61];
row_2 [62] <= row_1 [62];
row_2 [63] <= row_1 [63];
row_2 [64] <= row_1 [64];
row_2 [65] <= row_1 [65];
row_2 [66] <= row_1 [66];
row_2 [67] <= row_1 [67];
row_2 [68] <= row_1 [68];
row_2 [69] <= row_1 [69];
row_2 [70] <= row_1 [70];
row_2 [71] <= row_1 [71];
row_2 [72] <= row_1 [72];
row_2 [73] <= row_1 [73];
row_2 [74] <= row_1 [74];
row_2 [75] <= row_1 [75];
row_2 [76] <= row_1 [76];
row_2 [77] <= row_1 [77];
row_2 [78] <= row_1 [78];
row_2 [79] <= row_1 [79];
row_2 [80] <= row_1 [80];
row_2 [81] <= row_1 [81];
row_2 [82] <= row_1 [82];
row_2 [83] <= row_1 [83];
row_2 [84] <= row_1 [84];
row_2 [85] <= row_1 [85];
row_2 [86] <= row_1 [86];
row_2 [87] <= row_1 [87];
row_2 [88] <= row_1 [88];
row_2 [89] <= row_1 [89];
row_2 [90] <= row_1 [90];
```

```
row_2 [91] <= row_1 [91];
row_2 [92] <= row_1 [92];
row_2 [93] <= row_1 [93];
row_2 [94] <= row_1 [94];
row_2 [95] <= row_1 [95];
row_2 [96] <= row_1 [96];
row_2 [97] <= row_1 [97];
row_2 [98] <= row_1 [98];
row_2 [99] <= row_1 [99];
row_2 [100] <= row_1 [100];
row_2 [101] <= row_1 [101];
row_2 [102] <= row_1 [102];
row_2 [103] <= row_1 [103];
row_2 [104] <= row_1 [104];
row_2 [105] <= row_1 [105];
row_2 [106] <= row_1 [106];
row_2 [107] <= row_1 [107];
row_2 [108] <= row_1 [108];
row_2 [109] <= row_1 [109];
row_2 [110] <= row_1 [110];
row_2 [111] <= row_1 [111];
row_2 [112] <= row_1 [112];
row_2 [113] <= row_1 [113];
row_2 [114] <= row_1 [114];
row_2 [115] <= row_1 [115];
row_2 [116] <= row_1 [116];
row_2 [117] <= row_1 [117];
row_2 [118] <= row_1 [118];
row_2 [119] <= row_1 [119];
row_2 [120] <= row_1 [120];
row_2 [121] <= row_1 [121];
row_2 [122] <= row_1 [122];
row_2 [123] <= row_1 [123];
row_2 [124] <= row_1 [124];
row_2 [125] <= row_1 [125];
```

```
row_2 [126] <= row_1 [126];
row_2 [127] <= row_1 [127];
row_2 [128] <= row_1 [128];
row_2 [129] <= row_1 [129];
row_2 [130] <= row_1 [130];
row_2 [131] <= row_1 [131];
row_2 [132] <= row_1 [132];
row_2 [133] <= row_1 [133];
row_2 [134] <= row_1 [134];
row_2 [135] <= row_1 [135];
row_2 [136] <= row_1 [136];
row_2 [137] <= row_1 [137];
row_2 [138] <= row_1 [138];
row_2 [139] <= row_1 [139];
row_2 [140] <= row_1 [140];
row_2 [141] <= row_1 [141];
row_2 [142] <= row_1 [142];
row_2 [143] <= row_1 [143];
row_2 [144] <= row_1 [144];
row_2 [145] <= row_1 [145];
row_2 [146] <= row_1 [146];
row_2 [147] <= row_1 [147];
row_2 [148] <= row_1 [148];
row_2 [149] <= row_1 [149];
row_2 [150] <= row_1 [150];
row_2 [151] <= row_1 [151];
row_2 [152] <= row_1 [152];
row_2 [153] <= row_1 [153];
row_2 [154] <= row_1 [154];
row_2 [155] <= row_1 [155];
row_2 [156] <= row_1 [156];
row_2 [157] <= row_1 [157];
row_2 [158] <= row_1 [158];
row_2 [159] <= row_1 [159];
row_2 [160] <= row_1 [160];
```

```
                        row_2 [161] <= row_1 [161];
                        row_2 [162] <= row_1 [162];
                        row_2 [163] <= row_1 [163];
                        row_2 [164] <= row_1 [164];
                        row_2 [165] <= row_1 [165];
                        row_2 [166] <= row_1 [166];
                        row_2 [167] <= row_1 [167];
                        row_2 [168] <= row_1 [168];
                        row_2 [169] <= row_1 [169];
                        row_2 [170] <= row_1 [170];
                        row_2 [171] <= row_1 [171];
                        row_2 [172] <= row_1 [172];
                        row_2 [173] <= row_1 [173];
                        row_2 [174] <= row_1 [174];
                        row_2 [175] <= row_1 [175];
                        row_2 [176] <= row_1 [176];
                        row_2 [177] <= row_1 [177];
                        row_2 [178] <= row_1 [178];
                        row_2 [179] <= row_1 [179];

                        if (row_cnt == HEIGTH + 2) begin //whole image went
through analyzer
                              finish1 <= 1;
                              h <= 0;
                              num <= 0;
                              row_cnt <= 0;
                        end
                        else if (row_6[0] > 0) h <= 1; //when rows 2-4 are
filled with data, algorithm can start to analyse it
                  end
                  else begin
                        num <= num+1;
                        finish1 <= 0;
                  end
            end//if
      end//always
```

```
//-------DATA ANALYZING-------//

//just for waveform analysing purposes
//two signals to spot dark and light pixels using simple tresholds
wire light;
wire dark;
assign light = (saturation > 110) ? 1:0;
assign dark = (saturation <30) ? 1:0;

//Haar function for discrete signal:
//for each pair 'current pixel+its neighbour', two values are calculated:
//for neighbour to the rignt
reg [7:0] haar_Ra = 8'b0;
reg signed [8:0] haar_Rb = 9'b0;
//for neighbour to the left
reg [7:0] haar_La = 8'b0;
reg signed [8:0] haar_Lb = 9'b0;
//for neighbour up
reg [7:0] haar_Ua = 8'b0;
reg signed [8:0] haar_Ub = 9'b0;
//for neighbour down
reg [7:0] haar_Ba = 8'b0;
reg signed [8:0] haar_Bb = 9'b0;

always @(posedge clk) begin
       if (h) begin
              haar_Ra <= (row_4[num] + row_4[num+RAD])/2;
              haar_Rb <= (row_4[num] - row_4[num+RAD])/2;
              haar_La <= (row_4[num] + row_4[num-RAD])/2;
              haar_Lb <= (row_4[num] - row_4[num-RAD])/2;
              haar_Ua <= (row_4[num] + row_2[num])/2;
              haar_Ub <= (row_4[num] - row_2[num])/2;
              haar_Ba <= (row_4[num] + row_6[num])/2;
              haar_Bb <= (row_4[num] - row_6[num])/2;
```

```
            end//if h

        end//always


        //-------ROW BORDERS SEARCHING-------//


        reg cnt_init = 1'b0;//signal to start count number and sizes of cells when
border of screen is spotted


        //weird connection to swithch berween screen and cells borders searchings

        reg cnt_init_busy = 1'b0;

        //dont even ask

        //but if you do, cnt_init starts cell searching and stops screen borders
searching

        //it goes high when screen border found (and  therefore signal cnt_init_1
generated) AND

        //AND cnt_init_busy is not high (_busy indicates that cell searching is
going on)


        //as a result, screen and cells searchings should go one after another


        reg wait_init = 1'b0;


        //coordinates of the found row's edge will be stored in these registers:

        reg [10:0] row_h_coords = 11'b0;

        reg [10:0] row_v_coords = 11'b0;


        reg [10:0] screen_h = 11'b0; //to store horisontal size of the screen (in
pixels)

        wire [11:0] screen_v; //to store vertical size of the screen (in pixels)


        always @(posedge clk) begin

            if (h) begin

                if (c == 2) begin

                    if (num < WIDHT/8 && (haar_Ub < -100 || haar_Ub > 100)
&& cnt_init_busy == 0 && wait_init == 0) begin

                        //if the value of Haar B functions for upper
neighbour is larger

                        //than treshold, we spotted a start of new row
of cells
```

```
                                wait_init <= 1;

                                row_h_coords <= num;

                                row_v_coords <= row_cnt - 3;

                        end

                        //wait for the next row for certainty

                        else if (wait_init == 1 && num == row_h_coords &&
row_cnt == row_v_coords + 6) begin

                                cnt_init <= 1;

                                wait_init <= 0;

                        end

                        else cnt_init <= 0;

                end//if

            end//ifh

            else if (state == WAIT) begin

                wait_init <= 0;

                cnt_init <= 0;

                row_h_coords <= 0;

                row_v_coords <= 0;

            end

        end//always


        //-------CELLS BORDERS SEARCHING-------//


        reg extract_init = 1'b0;


        reg [5:0] cell_number = 6'b0;//register to store calculeted number of
cells in an each row


        //for each cell in a row its size (in pixels) will be counted

        //(sizes are the same on a smartphone screen, but captured image of the
screen may have distortions)

        reg [3:0] cell_size_0 = 4'b0;

        reg [3:0] cell_size_1 = 4'b0;

        reg [3:0] cell_size_2 = 4'b0;

        reg [3:0] cell_size_3 = 4'b0;

        reg [3:0] cell_size_4 = 4'b0;
```

```
        reg [3:0] cell_size_5 = 4'b0;
        reg [3:0] cell_size_6 = 4'b0;
        reg [3:0] cell_size_7 = 4'b0;
        reg [3:0] cell_size_8 = 4'b0;

        reg [3:0] cell_size_9 = 4'b0;
        reg [3:0] cell_size_10 = 4'b0;
        reg [3:0] cell_size_11 = 4'b0;
        reg [3:0] cell_size_12 = 4'b0;
        reg [3:0] cell_size_13 = 4'b0;
        reg [3:0] cell_size_14 = 4'b0;
        reg [3:0] cell_size_15 = 4'b0;
        reg [3:0] cell_size_16 = 4'b0;
        reg [3:0] cell_size_17 = 4'b0;

        reg [3:0] cell_size_18 = 4'b0;
        reg [3:0] cell_size_19 = 4'b0;
        reg [3:0] cell_size_20 = 4'b0;
        reg [3:0] cell_size_21 = 4'b0;
        reg [3:0] cell_size_22 = 4'b0;
        reg [3:0] cell_size_23 = 4'b0;
        reg [3:0] cell_size_24 = 4'b0;
        reg [3:0] cell_size_25 = 4'b0;
        reg [3:0] cell_size_26 = 4'b0;

        reg once = 1'b0;

        always @(posedge clk) begin
               if ((cnt_init || cnt_init_busy) && !once) begin
                      if(c == 2) begin
                             screen_h <= screen_h + 1;//we can start counting
horisontal length of the screen
                             if (cell_number == 0) begin
                                    extract_init <= 0;
                                    cnt_init_busy <= 1;
```

```
                                cell_size_0 <= cell_size_0 + 1;
                                if ((haar_Rb < -TRESHOLD || haar_Rb > TRESHOLD)
&& cell_size_0 > 2) begin
                                        //border between cells found, switch to
count next cell's size
                                        cell_number <= 1;
                                end
                        end
                        else if (cell_number == 1) begin
                                cell_size_1 <= cell_size_1 + 1;
                                if ((haar_Rb < -TRESHOLD || haar_Rb > TRESHOLD)
&& cell_size_1 > 2) begin
                                        cell_number <= 2;
                                end
                        end
                        else if (cell_number == 2) begin
                                cell_size_2 <= cell_size_2 + 1;
                                if ((haar_Rb < -TRESHOLD || haar_Rb > TRESHOLD)
&& cell_size_2 > 2) begin
                                        cell_number <= 3;
                                end
                        end
                        else if (cell_number == 3) begin
                                cell_size_3 <= cell_size_3 + 1;
                                if ((haar_Rb < -TRESHOLD || haar_Rb > TRESHOLD)
&& cell_size_3 > 2) begin
                                        cell_number <= 4;
                                end
                        end
                        else if (cell_number == 4) begin
                                cell_size_4 <= cell_size_4 + 1;
                                if ((haar_Rb < -TRESHOLD || haar_Rb > TRESHOLD)
&& cell_size_4 > 2) begin
                                        cell_number <= 5;
                                end
                        end
                        else if (cell_number == 5) begin
                                cell_size_5 <= cell_size_5 + 1;
```

```
                                if ((haar_Rb < -TRESHOLD || haar_Rb > TRESHOLD)
&& cell_size_5 > 2) begin
                                        cell_number <= 6;
                                end
                        end
                        else if (cell_number == 6) begin
                                cell_size_6 <= cell_size_6 + 1;
                                if ((haar_Rb < -TRESHOLD || haar_Rb > TRESHOLD)
&& cell_size_6 > 2) begin
                                        cell_number <= 7;
                                end
                        end
                        else if (cell_number == 7) begin
                                cell_size_7 <= cell_size_7 + 1;
                                if ((haar_Rb < -TRESHOLD || haar_Rb > TRESHOLD)
&& cell_size_7 > 2) begin
                                        cell_number <= 8;
                                end
                        end
                        else if (cell_number == 8) begin
                                cell_size_8 <= cell_size_8 + 1;
                                if ((haar_Rb < -TRESHOLD || haar_Rb > TRESHOLD)
&& cell_size_8 > 2) begin
                                        cell_number <= 9;
                                end
                        end
                        else if (cell_number == 9) begin
                                cell_size_9 <= cell_size_9 + 1;
                                if ((haar_Rb < -TRESHOLD || haar_Rb > TRESHOLD)
&& cell_size_9 > 2) begin
                                        cell_number <= 10;
                                end
                        end
                        else if (cell_number == 10) begin
                                cell_size_10 <= cell_size_10 + 1;
                                if ((haar_Rb < -TRESHOLD || haar_Rb > TRESHOLD)
&& cell_size_10 > 2) begin
                                        cell_number <= 11;
```

```
                                end
                        end
                        else if (cell_number == 11) begin
                                cell_size_11 <= cell_size_11 + 1;
                                if ((haar_Rb < -TRESHOLD || haar_Rb > TRESHOLD)
&& cell_size_11 > 2) begin
                                        cell_number <= 12;
                                end
                        end
                        else if (cell_number == 12) begin
                                cell_size_12 <= cell_size_12 + 1;
                                if ((haar_Rb < -TRESHOLD || haar_Rb > TRESHOLD)
&& cell_size_12 > 2) begin
                                        cell_number <= 13;
                                end
                        end
                        else if (cell_number == 13) begin
                                cell_size_13 <= cell_size_13 + 1;
                                if ((haar_Rb < -TRESHOLD || haar_Rb > TRESHOLD)
&& cell_size_13 > 2) begin
                                        cell_number <= 14;
                                end
                        end
                        else if (cell_number == 14) begin
                                cell_size_14 <= cell_size_14 + 1;
                                if ((haar_Rb < -TRESHOLD || haar_Rb > TRESHOLD)
&& cell_size_14 > 2) begin
                                        cell_number <= 15;
                                end
                        end
                        else if (cell_number == 15) begin
                                cell_size_15 <= cell_size_15 + 1;
                                if ((haar_Rb < -TRESHOLD || haar_Rb > TRESHOLD)
&& cell_size_15 > 2) begin
                                        cell_number <= 16;
                                end
                        end
```

```
                        else if (cell_number == 16) begin
                                cell_size_16 <= cell_size_16 + 1;
                                if ((haar_Rb < -TRESHOLD || haar_Rb > TRESHOLD)
&& cell_size_16 > 2) begin
                                        cell_number <= 17;
                                end
                        end
                        //the following addition was made for the testing of
image with 27x42 cells
//                      else if (cell_number == 17) begin
//                              cell_size_17 <= cell_size_17 + 1;
//                              if ((haar_Rb < -TRESHOLD || haar_Rb > TRESHOLD)
&& cell_size_17 > 2) begin
//                                      cell_number <= 18;
//                              end
//                      end
//                      else if (cell_number == 18) begin
//                              cell_size_18 <= cell_size_18 + 1;
//                              if ((haar_Rb < -TRESHOLD || haar_Rb > TRESHOLD)
&& cell_size_18 > 2) begin
//                                      cell_number <= 19;
//                              end
//                      end
//                      else if (cell_number == 19) begin
//                              cell_size_19 <= cell_size_19 + 1;
//                              if ((haar_Rb < -TRESHOLD || haar_Rb > TRESHOLD)
&& cell_size_19 > 2) begin
//                                      cell_number <= 20;
//                              end
//                      end
//                      else if (cell_number == 20) begin
//                              cell_size_20 <= cell_size_20 + 1;
//                              if ((haar_Rb < -TRESHOLD || haar_Rb > TRESHOLD)
&& cell_size_20 > 2) begin
//                                      cell_number <= 21;
//                              end
//                      end
//                      else if (cell_number == 21) begin
```

```
//                                cell_size_21 <= cell_size_21 + 1;

//                                if ((haar_Rb < -TRESHOLD || haar_Rb > TRESHOLD)
&& cell_size_21 > 2) begin

//                                       cell_number <= 22;

//                                end

//                         end

//                         else if (cell_number == 22) begin

//                                cell_size_22 <= cell_size_22 + 1;

//                                if ((haar_Rb < -TRESHOLD || haar_Rb > TRESHOLD)
&& cell_size_22 > 2) begin

//                                       cell_number <= 23;

//                                end

//                         end

//                         else if (cell_number == 23) begin

//                                cell_size_23 <= cell_size_23 + 1;

//                                if ((haar_Rb < -TRESHOLD || haar_Rb > TRESHOLD)
&& cell_size_23 > 2) begin

//                                       cell_number <= 24;

//                                end

//                         end

//                         else if (cell_number == 24) begin

//                                cell_size_24 <= cell_size_24 + 1;

//                                if ((haar_Rb < -TRESHOLD || haar_Rb > TRESHOLD)
&& cell_size_24 > 2) begin

//                                       cell_number <= 25;

//                                end

//                         end

//                         else if (cell_number == 25) begin

//                                cell_size_25 <= cell_size_25 + 1;

//                                if ((haar_Rb < -TRESHOLD || haar_Rb > TRESHOLD)
&& cell_size_25 > 2) begin

//                                       cell_number <= 26;

//                                end

//                         end

                           else if (cell_number == 17) begin

                                  cell_size_17 <= cell_size_17 + 1;

                                  if ((haar_Rb < -TRESHOLD/2 || haar_Rb >
TRESHOLD/2) && cell_size_17 > 2) begin
```

```
                                        //if this cell is the last, stop
counting
                                        cell_number <= 0;
                                        cnt_init_busy <= 0;
                                        //and start extracting
                                        extract_init <= 1;
                                        once <= 1;
                                end
                                else if (num == WIDHT - 2) begin
                                        cell_size_17 <= cell_size_16;
                                        cell_number <= 0;
                                        cnt_init_busy <= 0;
                                        extract_init <= 1;
                                        once <= 1;
                                end
                        end//cell_17
                end//if c==2
        end//if init
        else if ((cnt_init || cnt_init_busy) && once) begin //if cell sizes
are already counted
                if (c == 2) begin
                        //just wait till the end of the row
                        if (num < WIDHT - 2) begin
                                cnt_init_busy <= 1;
                                extract_init <= 0;
                        end
                        else if (num == WIDHT - 2) begin
                                cnt_init_busy <= 0;
                                extract_init <= 1;
                        end
                end//if c==2
        end//if init & once
        else if (state == WAIT) begin
                screen_h <= 0;
                cell_size_0 <= 0;
                cell_size_1 <= 0;
```

```
                cell_size_2 <= 0;

                cell_size_3 <= 0;

                cell_size_4 <= 0;

                cell_size_5 <= 0;

                cell_size_6 <= 0;

                cell_size_7 <= 0;

                cell_size_8 <= 0;

                cell_size_9 <= 0;

                cell_size_10 <= 0;

                cell_size_11 <= 0;

                cell_size_12 <= 0;

                cell_size_13 <= 0;

                cell_size_14 <= 0;

                cell_size_15 <= 0;

                cell_size_16 <= 0;

                cell_size_17 <= 0;


                extract_init <= 0;

                cnt_init_busy <= 0;

                once <= 0;

            end

        end//always


        //-------DATA EXTRACTION-------//


        //data is being extracted from the centre of each cell by targeting to

        //[row_horizontal_coordinates + size_of_all_previous_cells +
1/2_size_of_target_cell] position in a

        //[row_vertical_coordinates + size_of_all_previous_rows_of_cells +
1/2_size_of_targer_row_of_cells] row


        //the following are two differenr extraction algorithms: for black&white
encoding and for grayscale encoding

        //uncomment the one you need right now and comment other

        //suggestion: create a parameter TYPE (of image) and write FSM that
chooses algorithm depending on this parameter.

        //But now I'm to lasy to do it
```

```
//for black&white:


wire [18:0] data; //each cell stores one bit of data

assign data [17] = (row_4[row_h_coords + (cell_size_0)/2] > 127) ? 0:1;

assign data [16] = (row_4[row_h_coords + cell_size_0 + (cell_size_1)/2] >
127) ? 0:1;

assign data [15] = (row_4[row_h_coords + cell_size_0 + cell_size_1 +
(cell_size_2)/2] > 127) ? 0:1;

assign data [14] = (row_4[row_h_coords + cell_size_0 + cell_size_1 +
cell_size_2 + (cell_size_3)/2] > 127) ? 0:1;

assign data [13] = (row_4[row_h_coords + cell_size_0 + cell_size_1 +
cell_size_2 + cell_size_3 + (cell_size_4)/2] > 127) ? 0:1;

assign data [12] = (row_4[row_h_coords + cell_size_0 + cell_size_1 +
cell_size_2 + cell_size_3 + cell_size_4 + (cell_size_5)/2] > 127) ? 0:1;

assign data [11] = (row_4[row_h_coords + cell_size_0 + cell_size_1 +
cell_size_2 + cell_size_3 + cell_size_4 + cell_size_5 + (cell_size_6)/2] > 127)
? 0:1;

assign data [10] = (row_4[row_h_coords + cell_size_0 + cell_size_1 +
cell_size_2 + cell_size_3 + cell_size_4 + cell_size_5 + cell_size_6 +
(cell_size_7)/2] > 127) ? 0:1;

assign data [9] = (row_4[row_h_coords + cell_size_0 + cell_size_1 +
cell_size_2 + cell_size_3 + cell_size_4 + cell_size_5 + cell_size_6 +
cell_size_7 + (cell_size_8)/2] > 127) ? 0:1;

assign data [8] = (row_4[row_h_coords + cell_size_0 + cell_size_1 +
cell_size_2 + cell_size_3 + cell_size_4 + cell_size_5 + cell_size_6 +
cell_size_7 + cell_size_8 + (cell_size_9)/2] > 127) ? 0:1;

assign data [7] = (row_4[row_h_coords + cell_size_0 + cell_size_1 +
cell_size_2 + cell_size_3 + cell_size_4 + cell_size_5 + cell_size_6 +
cell_size_7 + cell_size_8 + cell_size_9 + (cell_size_10)/2] > 127) ? 0:1;

assign data [6] = (row_4[row_h_coords + cell_size_0 + cell_size_1 +
cell_size_2 + cell_size_3 + cell_size_4 + cell_size_5 + cell_size_6 +
cell_size_7 + cell_size_8 + cell_size_9 + cell_size_10 + (cell_size_11)/2] >
127) ? 0:1;

assign data [5] = (row_4[row_h_coords + cell_size_0 + cell_size_1 +
cell_size_2 + cell_size_3 + cell_size_4 + cell_size_5 + cell_size_6 +
cell_size_7 + cell_size_8 + cell_size_9 + cell_size_10 + cell_size_11 +
(cell_size_12)/2] > 127) ? 0:1;

assign data [4] = (row_4[row_h_coords + cell_size_0 + cell_size_1 +
cell_size_2 + cell_size_3 + cell_size_4 + cell_size_5 + cell_size_6 +
cell_size_7 + cell_size_8 + cell_size_9 + cell_size_10 + cell_size_11 +
cell_size_12 + (cell_size_13)/2] > 127) ? 0:1;

assign data [3] = (row_4[row_h_coords + cell_size_0 + cell_size_1 +
cell_size_2 + cell_size_3 + cell_size_4 + cell_size_5 + cell_size_6 +
cell_size_7 + cell_size_8 + cell_size_9 + cell_size_10 + cell_size_11 +
cell_size_12 + cell_size_13 + (cell_size_14)/2] > 127) ? 0:1;

assign data [2] = (row_4[row_h_coords + cell_size_0 + cell_size_1 +
cell_size_2 + cell_size_3 + cell_size_4 + cell_size_5 + cell_size_6 +
```

```
cell_size_7 + cell_size_8 + cell_size_9 + cell_size_10 + cell_size_11 +
cell_size_12 + cell_size_13 + cell_size_14 + (cell_size_15)/2] > 127) ? 0:1;

       assign data [1] = (row_4[row_h_coords + cell_size_0 + cell_size_1 +
cell_size_2 + cell_size_3 + cell_size_4 + cell_size_5 + cell_size_6 +
cell_size_7 + cell_size_8 + cell_size_9 + cell_size_10 + cell_size_11 +
cell_size_12 + cell_size_13 + cell_size_14 + cell_size_15 + (cell_size_16)/2] >
127) ? 0:1;

       assign data [0] = (row_4[row_h_coords + cell_size_0 + cell_size_1 +
cell_size_2 + cell_size_3 + cell_size_4 + cell_size_5 + cell_size_6 +
cell_size_7 + cell_size_8 + cell_size_9 + cell_size_10 + cell_size_11 +
cell_size_12 + cell_size_13 + cell_size_14 + cell_size_15 + cell_size_16 +
(cell_size_17)/2] > 127) ? 0:1;

//     assign data [8] = (row_4[row_h_coords + cell_size_0 + cell_size_1 +
cell_size_2 + cell_size_3 + cell_size_4 + cell_size_5 + cell_size_6 +
cell_size_7 + cell_size_8 + cell_size_9 + cell_size_10 + cell_size_11 +
cell_size_12 + cell_size_13 + cell_size_14 + cell_size_15 + cell_size_16 +
cell_size_17 + (cell_size_18)/2] > 127) ? 0:1;

//     assign data [7] = (row_4[row_h_coords + cell_size_0 + cell_size_1 +
cell_size_2 + cell_size_3 + cell_size_4 + cell_size_5 + cell_size_6 +
cell_size_7 + cell_size_8 + cell_size_9 + cell_size_10 + cell_size_11 +
cell_size_12 + cell_size_13 + cell_size_14 + cell_size_15 + cell_size_16 +
cell_size_17 + cell_size_18 + (cell_size_19)/2] > 127) ? 0:1;

//     assign data [6] = (row_4[row_h_coords + cell_size_0 + cell_size_1 +
cell_size_2 + cell_size_3 + cell_size_4 + cell_size_5 + cell_size_6 +
cell_size_7 + cell_size_8 + cell_size_9 + cell_size_10 + cell_size_11 +
cell_size_12 + cell_size_13 + cell_size_14 + cell_size_15 + cell_size_16 +
cell_size_17 + cell_size_18 + cell_size_19 + (cell_size_20)/2] > 127) ? 0:1;

//     assign data [5] = (row_4[row_h_coords + cell_size_0 + cell_size_1 +
cell_size_2 + cell_size_3 + cell_size_4 + cell_size_5 + cell_size_6 +
cell_size_7 + cell_size_8 + cell_size_9 + cell_size_10 + cell_size_11 +
cell_size_12 + cell_size_13 + cell_size_14 + cell_size_15 + cell_size_16 +
cell_size_17 + cell_size_18 + cell_size_19 + cell_size_20 + (cell_size_21)/2] >
127) ? 0:1;

//     assign data [4] = (row_4[row_h_coords + cell_size_0 + cell_size_1 +
cell_size_2 + cell_size_3 + cell_size_4 + cell_size_5 + cell_size_6 +
cell_size_7 + cell_size_8 + cell_size_9 + cell_size_10 + cell_size_11 +
cell_size_12 + cell_size_13 + cell_size_14 + cell_size_15 + cell_size_16 +
cell_size_17 + cell_size_18 + cell_size_19 + cell_size_20 + cell_size_21 +
(cell_size_22)/2] > 127) ? 0:1;

//     assign data [3] = (row_4[row_h_coords + cell_size_0 + cell_size_1 +
cell_size_2 + cell_size_3 + cell_size_4 + cell_size_5 + cell_size_6 +
cell_size_7 + cell_size_8 + cell_size_9 + cell_size_10 + cell_size_11 +
cell_size_12 + cell_size_13 + cell_size_14 + cell_size_15 + cell_size_16 +
cell_size_17 + cell_size_18 + cell_size_19 + cell_size_20 + cell_size_21 +
cell_size_22 + (cell_size_23)/2] > 127) ? 0:1;

//     assign data [2] = (row_4[row_h_coords + cell_size_0 + cell_size_1 +
cell_size_2 + cell_size_3 + cell_size_4 + cell_size_5 + cell_size_6 +
cell_size_7 + cell_size_8 + cell_size_9 + cell_size_10 + cell_size_11 +
cell_size_12 + cell_size_13 + cell_size_14 + cell_size_15 + cell_size_16 +
cell_size_17 + cell_size_18 + cell_size_19 + cell_size_20 + cell_size_21 +
cell_size_22 + cell_size_23 + (cell_size_24)/2] > 127) ? 0:1;

//     assign data [1] = (row_4[row_h_coords + cell_size_0 + cell_size_1 +
cell_size_2 + cell_size_3 + cell_size_4 + cell_size_5 + cell_size_6 +
cell_size_7 + cell_size_8 + cell_size_9 + cell_size_10 + cell_size_11 +
cell_size_12 + cell_size_13 + cell_size_14 + cell_size_15 + cell_size_16 +
cell_size_17 + cell_size_18 + cell_size_19 + cell_size_20 + cell_size_21 +
cell_size_22 + cell_size_23 + cell_size_24 + (cell_size_25)/2] > 127) ? 0:1;
```

```
//     assign data [0] = (row_4[row_h_coords + cell_size_0 + cell_size_1 +
cell_size_2 + cell_size_3 + cell_size_4 + cell_size_5 + cell_size_6 +
cell_size_7 + cell_size_8 + cell_size_9 + cell_size_10 + cell_size_11 +
cell_size_12 + cell_size_13 + cell_size_14 + cell_size_15 + cell_size_16 +
cell_size_17 + cell_size_18 + cell_size_19 + cell_size_20 + cell_size_21 +
cell_size_22 + cell_size_23 + cell_size_24 + cell_size_25 + (cell_size_26)/2] >
127) ? 0:1;



       //for grayscale;

//     reg [17:0] data; //each cell stores two bits of data

//

//     always @(posedge clk) begin

//           data [17:16] <= ~(row_4[row_h_coords + (cell_size_0)/2]+1)/70;

//           data [15:14] <= ~(row_4[corner_h_coords + cell_size_0 +
(cell_size_1)/2]+1)/70;

//           data [13:12] <= ~(row_4[corner_h_coords + cell_size_0 + cell_size_1
+ (cell_size_2)/2]+1)/70;

//           data [11:10] <= ~(row_4[corner_h_coords + cell_size_0 + cell_size_1
+ cell_size_2 + (cell_size_3)/2]+1)/70;

//           data [9:8] <= ~(row_4[corner_h_coords + cell_size_0 + cell_size_1 +
cell_size_2 + cell_size_3 + (cell_size_4)/2]+1)/70;

//           data [7:6] <= ~(row_4[corner_h_coords + cell_size_0 + cell_size_1 +
cell_size_2 + cell_size_3 + cell_size_4 + (cell_size_5)/2]+1)/70;

//           data [5:4] <= ~(row_4[corner_h_coords + cell_size_0 + cell_size_1 +
cell_size_2 + cell_size_3 + cell_size_4 + cell_size_5 + (cell_size_6)/2]+1)/70;

//           data [3:2] <= ~(row_4[corner_h_coords + cell_size_0 + cell_size_1 +
cell_size_2 + cell_size_3 + cell_size_4 + cell_size_5 + cell_size_6 +
(cell_size_7)/2]+1)/70;

//           data [1:0] <= ~(row_4[corner_h_coords + cell_size_0 + cell_size_1 +
cell_size_2 + cell_size_3 + cell_size_4 + cell_size_5 + cell_size_6 +
cell_size_7 + (cell_size_8)/2]+1)/70;

//     end

//

       reg [17:0] dataout [0:31]; //two-dimentional array of data for two-
dimentional array of cells on a screen

       //the folloving signal exists and belongs to this part of algorithm

       //it was commented out, because it's already declarated in Inputs/Outputs
section

       //commennts were left here to describe what it is

       //reg finish = 1'b0;//signal that shows that all data was extracted, can
be used for synchronisation with next stage

       reg [5:0] cell_row = 6'b0;



       reg extract2;
```

```
always @(posedge clk) extract2 <= ~extract_init;
wire extract;
assign extract = extract_init & extract2;

always @(posedge clk) begin
	if (extract) begin //if previous stage stopped measuring borders
		dataout [31 - cell_row] <= data;
		cell_row <= cell_row + 1;
	end //if extract_init

	else if (state == WAIT) begin
		dataout [31] <= 0;
		dataout [30] <= 0;
		dataout [29] <= 0;
		dataout [28] <= 0;
		dataout [27] <= 0;
		dataout [26] <= 0;
		dataout [25] <= 0;
		dataout [24] <= 0;
		dataout [23] <= 0;
		dataout [22] <= 0;
		dataout [21] <= 0;
		dataout [20] <= 0;
		dataout [19] <= 0;
		dataout [18] <= 0;
		dataout [17] <= 0;
		dataout [16] <= 0;
		dataout [15] <= 0;
		dataout [14] <= 0;
		dataout [13] <= 0;
		dataout [12] <= 0;
		dataout [11] <= 0;
		dataout [10] <= 0;
		dataout [9] <= 0;
		dataout [8] <= 0;
```

```
                    dataout [7] <= 0;
                    dataout [6] <= 0;
                    dataout [5] <= 0;
                    dataout [4] <= 0;
                    dataout [3] <= 0;
                    dataout [2] <= 0;
                    dataout [1] <= 0;
                    dataout [0] <= 0;

                    cell_row <= 0;
              end
       end//always

endmodule
```

Testbench file for simulation

```
`timescale 1ns / 1ps

module testbench
#(parameter WIDHT = 180,
HEIGTH = 350,
TRESHOLD = 90,
//put your input files below
INFILE1 = "./real_1.hex",
INFILE2 = "./real_2.hex",
INFILE3 = "./real_3.hex",
INFILE4 = "./real_4.hex",
INFILE5 = "./real_5.hex",
INFILE6 = "./real_6.hex",
INFILE7 = "./real_7.hex",
INFILE8 = "./real_8.hex",
INFILE9 = "./real_9.hex",
INFILE10 = "./real_10.hex",
INFILE11 = "./real_11.hex",
INFILE12 = "./real_12.hex"
```

```
)
();
reg clk;
reg start;
reg [7:0] total [0:189000];//width*heigth*3
reg [7:0] datain;
reg [19:0] i;

initial begin
	clk = 1'b0;
	forever begin
		#5
		clk = ~clk;
	end
end

initial begin
	start = 1'b0;
	i = 0;
	#1;
   $readmemh(INFILE1,total,0,188999); // read file from INFILE to total
	start = 1'b1;
	while (i < 190000) begin
		datain = total[i];
		i = i+1;
		#10;
	end
	start = 1'b0;
	#100000;
	$readmemh(INFILE2,total,0,188999); // read file from INFILE to total
	i = 0;
	start = 1'b1;
	while (i < 190000) begin
		datain = total[i];
		i = i+1;
```

```
        #10;
end
start = 1'b0;
#100000;
$readmemh(INFILE3,total,0,188999); // read file from INFILE to total
i = 0;
start = 1'b1;
while (i < 190000) begin
        datain = total[i];
        i = i+1;
        #10;
end
start = 1'b0;
#100000;
$readmemh(INFILE4,total,0,188999); // read file from INFILE to total
i = 0;
start = 1'b1;
while (i < 190000) begin
        datain = total[i];
        i = i+1;
        #10;
end
start = 1'b0;
#100000;
$readmemh(INFILE5,total,0,188999); // read file from INFILE to total
i = 0;
start = 1'b1;
while (i < 190000) begin
        datain = total[i];
        i = i+1;
        #10;
end
start = 1'b0;
#100000;
$readmemh(INFILE6,total,0,188999); // read file from INFILE to total
```

```
i = 0;
start = 1'b1;
while (i < 190000) begin
	datain = total[i];
	i = i+1;
	#10;
end
start = 1'b0;
#100000;
$readmemh(INFILE7,total,0,188999); // read file from INFILE to total
i = 0;
start = 1'b1;
while (i < 190000) begin
	datain = total[i];
	i = i+1;
	#10;
end
start = 1'b0;
#100000;
$readmemh(INFILE8,total,0,188999); // read file from INFILE to total
i = 0;
start = 1'b1;
while (i < 190000) begin
	datain = total[i];
	i = i+1;
	#10;
end
start = 1'b0;
#100000;
$readmemh(INFILE9,total,0,188999); // read file from INFILE to total
i = 0;
start = 1'b1;
while (i < 190000) begin
	datain = total[i];
	i = i+1;
```

```
            #10;
        end
        start = 1'b0;
        #100000;
        $readmemh(INFILE10,total,0,188999); // read file from INFILE to total
        i = 0;
        start = 1'b1;
        while (i < 190000) begin
            datain = total[i];
            i = i+1;
            #10;
        end
        start = 1'b0;
        #100000;
        $readmemh(INFILE11,total,0,188999); // read file from INFILE to total
        i = 0;
        start = 1'b1;
        while (i < 190000) begin
            datain = total[i];
            i = i+1;
            #10;
        end
        start = 1'b0;
        #100000;
        $readmemh(INFILE12,total,0,188999); // read file from INFILE to total
        i = 0;
        start = 1'b1;
        while (i < 190000) begin
            datain = total[i];
            i = i+1;
            #10;
        end
        start = 1'b0;
        #100000;
end
```

```
main main(
        .clk(clk),
        .start(start),
        .datain(datain)
);

endmodule
```

User Constrain File for implementation

```
#####################################################
### SPARTAN-3E STARTER KIT BOARD CONSTRAINTS FILE
#####################################################
## ==== Analog-to-Digital Converter (ADC) ====
## some connections shared with SPI Flash, DAC, ADC, and AMP
#NET "AD_CONV" LOC = "P11" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 6 ;
## ==== Programmable Gain Amplifier (AMP) ====
## some connections shared with SPI Flash, DAC, ADC, and AMP
#NET "AMP_CS" LOC = "N7" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 6 ;
#NET "AMP_DOUT" LOC = "E18" | IOSTANDARD = LVCMOS33 ;
#NET "AMP_SHDN" LOC = "P7" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 6 ;
## ==== Pushbuttons (BTN) ====
#NET "BTN_EAST" LOC = "H13" | IOSTANDARD = LVTTL | PULLDOWN ;
#NET "BTN_NORTH" LOC = "V4" | IOSTANDARD = LVTTL | PULLDOWN ;
#NET "BTN_SOUTH" LOC = "K17" | IOSTANDARD = LVTTL | PULLDOWN ;
#NET "BTN_WEST" LOC = "D18" | IOSTANDARD = LVTTL | PULLDOWN ;
# ==== Clock inputs (CLK) ====
NET "clk" LOC = "C9" | IOSTANDARD = LVCMOS33;
# Define clock period for 50 MHz oscillator (40%/60% duty-cycle)
NET "clk" PERIOD = 20.0ns HIGH 50%;
#NET "CLK_AUX" LOC = "B8" | IOSTANDARD = LVCMOS33 ;
#NET "CLK_SMA" LOC = "A10" | IOSTANDARD = LVCMOS33 ;
## ==== Digital-to-Analog Converter (DAC) ====
## some connections shared with SPI Flash, DAC, ADC, and AMP
#NET "DAC_CLR" LOC = "P8" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
#NET "DAC_CS" LOC = "N8" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
```

```
## ==== 1-Wire Secure EEPROM (DS)
#NET "DS_WIRE" LOC = "U4" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
## ==== Ethernet PHY (E) ====
#NET "E_COL" LOC = "U6" | IOSTANDARD = LVCMOS33 ;
#NET "E_CRS" LOC = "U13" | IOSTANDARD = LVCMOS33 ;
#NET "E_MDC" LOC = "P9" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
#
#NET "E_MDIO" LOC = "U5" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
#NET "E_RX_CLK" LOC = "V3" | IOSTANDARD = LVCMOS33 ;
#NET "E_RX_DV" LOC = "V2" | IOSTANDARD = LVCMOS33 ;
#NET "E_RXD<0>" LOC = "V8" | IOSTANDARD = LVCMOS33 ;
#NET "E_RXD<1>" LOC = "T11" | IOSTANDARD = LVCMOS33 ;
#NET "E_RXD<2>" LOC = "U11" | IOSTANDARD = LVCMOS33 ;
#NET "E_RXD<3>" LOC = "V14" | IOSTANDARD = LVCMOS33 ;
#NET "E_RXD<4>" LOC = "U14" | IOSTANDARD = LVCMOS33 ;
#NET "E_TX_CLK" LOC = "T7" | IOSTANDARD = LVCMOS33 ;
#NET "E_TX_EN" LOC = "P15" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
#NET "E_TXD<0>" LOC = "R11" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
#NET "E_TXD<1>" LOC = "T15" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
#NET "E_TXD<2>" LOC = "R5" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
#NET "E_TXD<3>" LOC = "T5" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
#NET "E_TXD<4>" LOC = "R6" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
## ==== FPGA Configuration Mode, INIT_B Pins (FPGA) ====
#NET "FPGA_M0" LOC = "M10" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
#NET "FPGA_M1" LOC = "V11" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
#NET "FPGA_M2" LOC = "T10" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
#NET "FPGA_INIT_B" LOC = "T3" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 4
;
#NET "FPGA_RDWR_B" LOC = "U10" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 4
;
#NET "FPGA_HSWAP" LOC = "B3" | IOSTANDARD = LVCMOS33 ;
## ==== FX2 Connector (FX2) ====
#NET "FX2_CLKIN" LOC = "E10" | IOSTANDARD = LVCMOS33 ;
#NET "FX2_CLKIO" LOC = "D9" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8 ;
#NET "FX2_CLKOUT" LOC = "D10" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8
;
```

```
## These four connections are shared with the J1 6-pin accessory header
#NET "FX2_IO<1>" LOC = "B4" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8 ;
#NET "FX2_IO<2>" LOC = "A4" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8 ;
#NET "FX2_IO<3>" LOC = "D5" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8 ;
#NET "FX2_IO<4>" LOC = "C5" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8 ;
## These four connections are shared with the J2 6-pin accessory header
#NET "FX2_IO<5>" LOC = "A6" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8 ;
#NET "FX2_IO<6>" LOC = "B6" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8 ;
#NET "FX2_IO<7>" LOC = "E7" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8 ;
#NET "FX2_IO<8>" LOC = "F7" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8 ;
## These four connections are shared with the J4 6-pin accessory header
#NET "FX2_IO<9>" LOC = "D7" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8 ;
#NET "FX2_IO<10>" LOC = "C7" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8 ;
#NET "FX2_IO<11>" LOC = "F8" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8 ;
#NET "FX2_IO<12>" LOC = "E8" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8 ;
## The discrete LEDs are shared with the following 8 FX2 connections
##NET "FX2_IO<13>" LOC = "F9" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8
;
##NET "FX2_IO<14>" LOC = "E9" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8
;
##NET "FX2_IO<15>" LOC = "D11" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8
;
##NET "FX2_IO<16>" LOC = "C11" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8
;
##NET "FX2_IO<17>" LOC = "F11" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8
;
##NET "FX2_IO<18>" LOC = "E11" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8
;
##NET "FX2_IO<19>" LOC = "E12" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8
;
##NET "FX2_IO<20>" LOC = "F12" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8
;
#NET "FX2_IO<21>" LOC = "A13" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8
;
#NET "FX2_IO<22>" LOC = "B13" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8
;
#NET "FX2_IO<23>" LOC = "A14" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8
;
#NET "FX2_IO<24>" LOC = "B14" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8
;
```

```
#NET "FX2_IO<25>" LOC = "C14" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8
;

#NET "FX2_IO<26>" LOC = "D14" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8
;

#

#NET "FX2_IO<27>" LOC = "A16" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8
;

#NET "FX2_IO<28>" LOC = "B16" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8
;

#NET "FX2_IO<29>" LOC = "E13" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8
;

#NET "FX2_IO<30>" LOC = "C4" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8 ;

#NET "FX2_IO<31>" LOC = "B11" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8
;

#NET "FX2_IO<32>" LOC = "A11" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8
;

#NET "FX2_IO<33>" LOC = "A8" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8 ;

#NET "FX2_IO<34>" LOC = "G9" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8 ;

#NET "FX2_IP<35>" LOC = "D12" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8
;

#NET "FX2_IP<36>" LOC = "C12" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8
;

#NET "FX2_IP<37>" LOC = "A15" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8
;

#NET "FX2_IP<38>" LOC = "B15" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8
;

#NET "FX2_IO<39>" LOC = "C3" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8 ;

#NET "FX2_IP<40>" LOC = "C15" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8
;

## ==== 6-pin header J1 ====

## These are shared connections with the FX2 connector

NET "datain<0>" LOC = "B4" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;

NET "datain<1>" LOC = "A4" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;

NET "datain<2>" LOC = "D5" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;

NET "datain<3>" LOC = "C5" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;

## ==== 6-pin header J2 ====

## These are shared connections with the FX2 connector

NET "datain<4>" LOC = "A6" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;

NET "datain<5>" LOC = "B6" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;

NET "datain<6>" LOC = "E7" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;
```

```
NET "datain<7>" LOC = "F7" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;
## ==== 6-pin header J4 ====
## These are shared connections with the FX2 connector
NET "start" LOC = "D7" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;
##NET "J4<1>" LOC = "C7" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;
##NET "J4<2>" LOC = "F8" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;
##NET "J4<3>" LOC = "E8" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;
## ==== Character LCD (LCD) ====
#NET "LCD_E" LOC = "M18" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "LCD_RS" LOC = "L18" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "LCD_RW" LOC = "L17" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
## LCD data connections are shared with StrataFlash connections SF_D<11:8>
##NET "SF_D<8>" LOC = "R15" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
##NET "SF_D<9>" LOC = "R16" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
##NET "SF_D<10>" LOC = "P17" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
##NET "SF_D<11>" LOC = "M15" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
# ==== Discrete LEDs (LED) ====
# These are shared connections with the FX2 connector
#NET "led" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
#NET "LED<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
#NET "LED<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
#NET "LED<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
#NET "LED<4>" LOC = "C11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
#NET "LED<5>" LOC = "D11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
#
#NET "LED<6>" LOC = "E9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
#NET "LED<7>" LOC = "F9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
## ==== PS/2 Mouse/Keyboard Port (PS2) ====
#NET "PS2_CLK" LOC = "G14" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = SLOW ;
#NET "PS2_DATA" LOC = "G13" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = SLOW ;
## ==== Rotary Pushbutton Switch (ROT) ====
#NET "ROT_A" LOC = "K18" | IOSTANDARD = LVTTL | PULLUP ;
#NET "ROT_B" LOC = "G18" | IOSTANDARD = LVTTL | PULLUP ;
#NET "ROT_CENTER" LOC = "V16" | IOSTANDARD = LVTTL | PULLDOWN ;
## ==== RS-232 Serial Ports (RS232) ====
```

```
#NET "RS232_DCE_RXD" LOC = "R7" | IOSTANDARD = LVTTL ;

#NET "RS232_DCE_TXD" LOC = "M14" | IOSTANDARD = LVTTL | DRIVE = 8 | SLEW = SLOW
;

#NET "RS232_DTE_RXD" LOC = "U8" | IOSTANDARD = LVTTL ;

#NET "RS232_DTE_TXD" LOC = "M13" | IOSTANDARD = LVTTL | DRIVE = 8 | SLEW = SLOW
;

## ==== DDR SDRAM (SD) ==== (I/O Bank 3, VCCO=2.5V)

#NET "SD_A<0>" LOC = "T1" | IOSTANDARD = SSTL2_I ;

#NET "SD_A<1>" LOC = "R3" | IOSTANDARD = SSTL2_I ;

#NET "SD_A<2>" LOC = "R2" | IOSTANDARD = SSTL2_I ;

#NET "SD_A<3>" LOC = "P1" | IOSTANDARD = SSTL2_I ;

#NET "SD_A<4>" LOC = "F4" | IOSTANDARD = SSTL2_I ;

#NET "SD_A<5>" LOC = "H4" | IOSTANDARD = SSTL2_I ;

#NET "SD_A<6>" LOC = "H3" | IOSTANDARD = SSTL2_I ;

#NET "SD_A<7>" LOC = "H1" | IOSTANDARD = SSTL2_I ;

#NET "SD_A<8>" LOC = "H2" | IOSTANDARD = SSTL2_I ;

#NET "SD_A<9>" LOC = "N4" | IOSTANDARD = SSTL2_I ;

#NET "SD_A<10>" LOC = "T2" | IOSTANDARD = SSTL2_I ;

#NET "SD_A<11>" LOC = "N5" | IOSTANDARD = SSTL2_I ;

#NET "SD_A<12>" LOC = "P2" | IOSTANDARD = SSTL2_I ;

#NET "SD_BA<0>" LOC = "K5" | IOSTANDARD = SSTL2_I ;

#NET "SD_BA<1>" LOC = "K6" | IOSTANDARD = SSTL2_I ;

#NET "SD_CAS" LOC = "C2" | IOSTANDARD = SSTL2_I ;

#NET "SD_CK_N" LOC = "J4" | IOSTANDARD = SSTL2_I ;

#NET "SD_CK_P" LOC = "J5" | IOSTANDARD = SSTL2_I ;

#NET "SD_CKE" LOC = "K3" | IOSTANDARD = SSTL2_I ;

#NET "SD_CS" LOC = "K4" | IOSTANDARD = SSTL2_I ;

#NET "SD_DQ<0>" LOC = "L2" | IOSTANDARD = SSTL2_I ;

#NET "SD_DQ<1>" LOC = "L1" | IOSTANDARD = SSTL2_I ;

#NET "SD_DQ<2>" LOC = "L3" | IOSTANDARD = SSTL2_I ;

#NET "SD_DQ<3>" LOC = "L4" | IOSTANDARD = SSTL2_I ;

#NET "SD_DQ<4>" LOC = "M3" | IOSTANDARD = SSTL2_I ;

#NET "SD_DQ<5>" LOC = "M4" | IOSTANDARD = SSTL2_I ;

#NET "SD_DQ<6>" LOC = "M5" | IOSTANDARD = SSTL2_I ;

#NET "SD_DQ<7>" LOC = "M6" | IOSTANDARD = SSTL2_I ;

#NET "SD_DQ<8>" LOC = "E2" | IOSTANDARD = SSTL2_I ;
```

```
#NET "SD_DQ<9>" LOC = "E1" | IOSTANDARD = SSTL2_I ;
#NET "SD_DQ<10>" LOC = "F1" | IOSTANDARD = SSTL2_I ;
#NET "SD_DQ<11>" LOC = "F2" | IOSTANDARD = SSTL2_I ;
#NET "SD_DQ<12>" LOC = "G6" | IOSTANDARD = SSTL2_I ;
#NET "SD_DQ<13>" LOC = "G5" | IOSTANDARD = SSTL2_I ;
#NET "SD_DQ<14>" LOC = "H6" | IOSTANDARD = SSTL2_I ;
#NET "SD_DQ<15>" LOC = "H5" | IOSTANDARD = SSTL2_I ;
#
#NET "SD_LDM" LOC = "J2" | IOSTANDARD = SSTL2_I ;
#NET "SD_LDQS" LOC = "L6" | IOSTANDARD = SSTL2_I ;
#NET "SD_RAS" LOC = "C1" | IOSTANDARD = SSTL2_I ;
#NET "SD_UDM" LOC = "J1" | IOSTANDARD = SSTL2_I ;
#NET "SD_UDQS" LOC = "G3" | IOSTANDARD = SSTL2_I ;
#NET "SD_WE" LOC = "D1" | IOSTANDARD = SSTL2_I ;
## Path to allow connection to top DCM connection
#NET "SD_CK_FB" LOC = "B9" | IOSTANDARD = LVCMOS33 ;
## Prohibit VREF pins
#CONFIG PROHIBIT = D2;
#CONFIG PROHIBIT = G4;
#CONFIG PROHIBIT = J6;
#CONFIG PROHIBIT = L5;
#CONFIG PROHIBIT = R4;
## ==== Intel StrataFlash Parallel NOR Flash (SF) ====
#NET "SF_A<0>" LOC = "H17" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_A<1>" LOC = "J13" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_A<2>" LOC = "J12" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_A<3>" LOC = "J14" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_A<4>" LOC = "J15" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_A<5>" LOC = "J16" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_A<6>" LOC = "J17" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_A<7>" LOC = "K14" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_A<8>" LOC = "K15" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_A<9>" LOC = "K12" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_A<10>" LOC = "K13" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_A<11>" LOC = "L15" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
```

```
#NET "SF_A<12>" LOC = "L16" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_A<13>" LOC = "T18" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_A<14>" LOC = "R18" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_A<15>" LOC = "T17" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_A<16>" LOC = "U18" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_A<17>" LOC = "T16" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_A<18>" LOC = "U15" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_A<19>" LOC = "V15" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_A<20>" LOC = "T12" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_A<21>" LOC = "V13" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_A<22>" LOC = "V12" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_A<23>" LOC = "N11" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_A<24>" LOC = "A11" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_BYTE" LOC = "C17" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_CE0" LOC = "D16" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_D<1>" LOC = "P10" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_D<2>" LOC = "R10" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_D<3>" LOC = "V9" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_D<4>" LOC = "U9" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_D<5>" LOC = "R9" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_D<6>" LOC = "M9" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_D<7>" LOC = "N9" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_D<8>" LOC = "R15" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_D<9>" LOC = "R16" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_D<10>" LOC = "P17" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_D<11>" LOC = "M15" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_D<12>" LOC = "M16" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_D<13>" LOC = "P6" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_D<14>" LOC = "R8" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_D<15>" LOC = "T8" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#
#NET "SF_OE" LOC = "C18" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
#NET "SF_STS" LOC = "B18" | IOSTANDARD = LVCMOS33 ;
#NET "SF_WE" LOC = "D17" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
## ==== STMicro SPI serial Flash (SPI) ====
```

```
## some connections shared with SPI Flash, DAC, ADC, and AMP

#NET "SPI_MISO" LOC = "N10" | IOSTANDARD = LVCMOS33 ;

#NET "SPI_MOSI" LOC = "T4" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 6 ;

#NET "SPI_SCK" LOC = "U16" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 6 ;

#NET "SPI_SS_B" LOC = "U3" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 6 ;

#NET "SPI_ALT_CS_JP11" LOC = "R12" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE
= 6 ;

## ==== Slide Switches (SW) ====

#NET "SW<0>" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP ;

#NET "SW<1>" LOC = "L14" | IOSTANDARD = LVTTL | PULLUP ;

#NET "SW<2>" LOC = "H18" | IOSTANDARD = LVTTL | PULLUP ;

#NET "SW<3>" LOC = "N17" | IOSTANDARD = LVTTL | PULLUP ;

## ==== VGA Port (VGA) ====

#NET "VGA_BLUE" LOC = "G15" | IOSTANDARD = LVTTL | DRIVE = 8 | SLEW = FAST ;

#NET "VGA_GREEN" LOC = "H15" | IOSTANDARD = LVTTL | DRIVE = 8 | SLEW = FAST ;

#NET "VGA_HSYNC" LOC = "F15" | IOSTANDARD = LVTTL | DRIVE = 8 | SLEW = FAST ;

#NET "VGA_RED" LOC = "H14" | IOSTANDARD = LVTTL | DRIVE = 8 | SLEW = FAST ;

#NET "VGA_VSYNC" LOC = "F14" | IOSTANDARD = LVTTL | DRIVE = 8 | SLEW = FAST ;

## ==== Xilinx CPLD (XC) ====

#NET "XC_CMD<0>" LOC = "P18" | IOSTANDARD = LVTTL | DRIVE = 4 | SLEW = SLOW ;

#NET "XC_CMD<1>" LOC = "N18" | IOSTANDARD = LVTTL | DRIVE = 4 | SLEW = SLOW ;

#NET "XC_CPLD_EN" LOC = "B10" | IOSTANDARD = LVTTL ;

#NET "XC_D<0>" LOC = "G16" | IOSTANDARD = LVTTL | DRIVE = 4 | SLEW = SLOW ;

#NET "XC_D<1>" LOC = "F18" | IOSTANDARD = LVTTL | DRIVE = 4 | SLEW = SLOW ;

#NET "XC_D<2>" LOC = "F17" | IOSTANDARD = LVTTL | DRIVE = 4 | SLEW = SLOW ;

#NET "XC_TRIG" LOC = "R17" | IOSTANDARD = LVCMOS33 ;

#NET "XC_GCK0" LOC = "H16" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;

#NET "GCLK10" LOC = "C9" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
```

Matlab code for bitmap-to-hex converter

```
pic = imread('U:\Desktop\test\real_n3.bmp');


k = 1;

for i = 350:-1:1

    for j = 1:180
```

```
            a(k)=pic(i,j,1);
            a(k+1)=pic(i,j,2);
            a(k+2)=pic(i,j,3);
            k=k+3;
    end
end

fid = fopen('U:\Desktop\test\real_n3.hex', 'wt');
fprintf(fid, '%x\n', a);
disp('done');
fclose(fid);
```