Contextualised Information

Group 7
Electronics and Information Engineering
5/18/18

Word count: 7448

Tharusha Amarasinghe (CID: 01337937)

Ramon Jutaviriya (CID: 01412091)

Hardik Aggarwal (CID: 01357191)

Contents

1 Introduction:

This project examines the display of information and explores how contextualisation can lead to a more interesting outcome. By contextualising information, specific information is provided to a user, rather than simply providing a general overview. The design not only factors in an extra way to authenticate identity but could also shield information from users based on their level of clearance. An example of this could be the personal details of a person, whereas age may be important in medical diagnosis, but isn't required when looking at a job applicant's details. This project focuses around the concept of being able to give and/or restrict information provided to a user depending on who they are.

Hence, a system was created in which the user is first identified, either via facial recognition or using a login/password, then a special cipher is used to determine the type of information to be displayed. Once this was done the system would determine which output image to use and display it on the screen as a corner overlay. The system was designed to run and process a video feed on the PYNQ-Z1 board, from which it received an input video stream and outputs to a display, with both interfaces being done via HDMI. The system uses live image processing to convert the image into different forms to aid with detection and will alter the resultant output image to display information.

2 Progress since management report

Over April, the group did extensive research, exploring various algorithms to implement the design and getting it ready before the start of the term, further analysis of the test bench and the demo code was also done. The various ways researched can be found under design process.

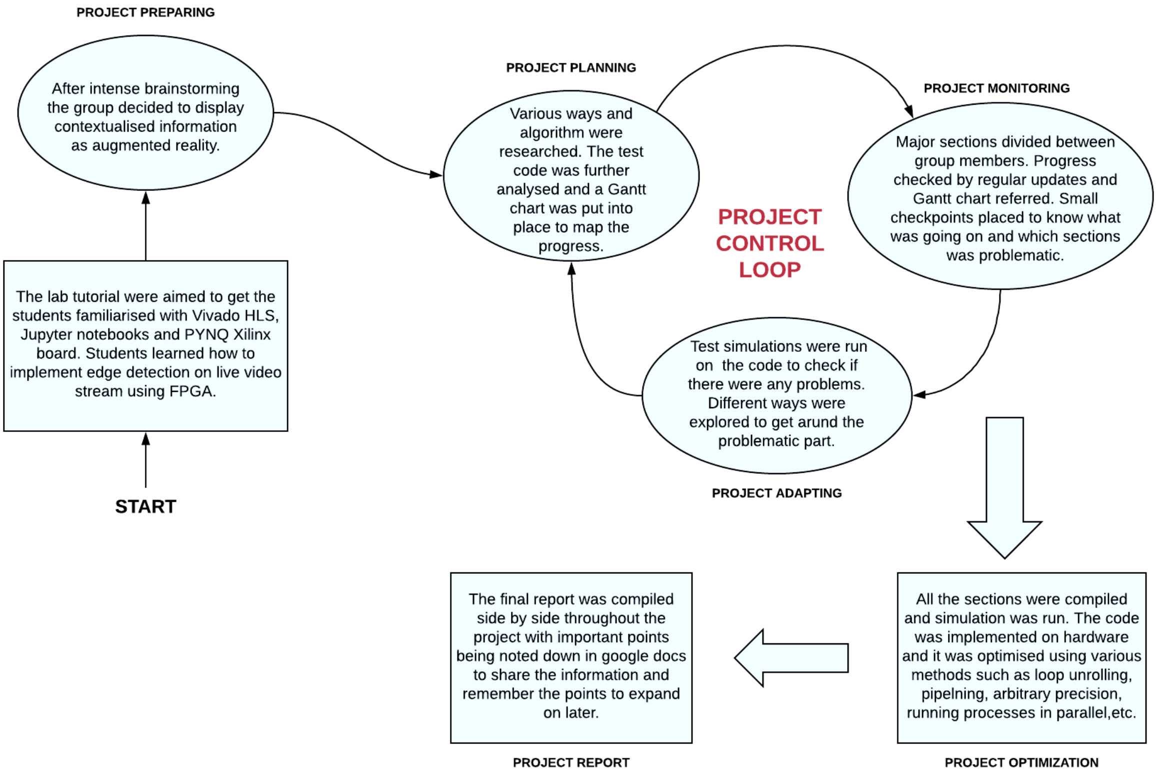


Figure 1

2.1 Team communication

In the management report, the work was split into three independent phases. Following the Gantt chart and dividing the coding between members not only increased the efficiency but also gave everyone autonomy over their chosen sections. However, towards the end of the April, the group decided to have a few alterations in the approach such as more communication between the group members with constant discussion about the problems being faced in each section and factoring in everyone's input to arrive at a solution.

Hence, the group met and worked together on the project from the start of term. This resulted in increased communication allowing the group members to work more effectively together. The roles were re-allocated based on the basic skill set of the person such as knowledge of python or report writing capability to maximise output and achieve high standard of performance. This was reflected in the revision of the Gantt chart from the management report (figure 2), which included unforeseen delays and restricting of future tasks and timeframes as seen in figure 3, (enlarged version found in appendix A).

2.2 Gantt chart revision

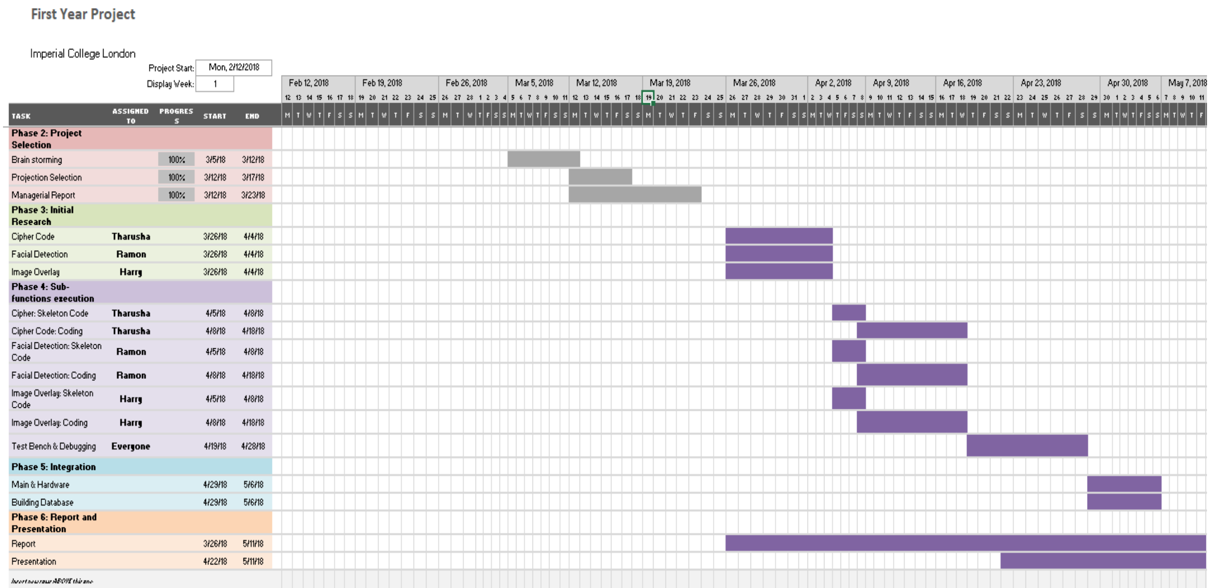


Figure 2: Management Report Gantt Chart

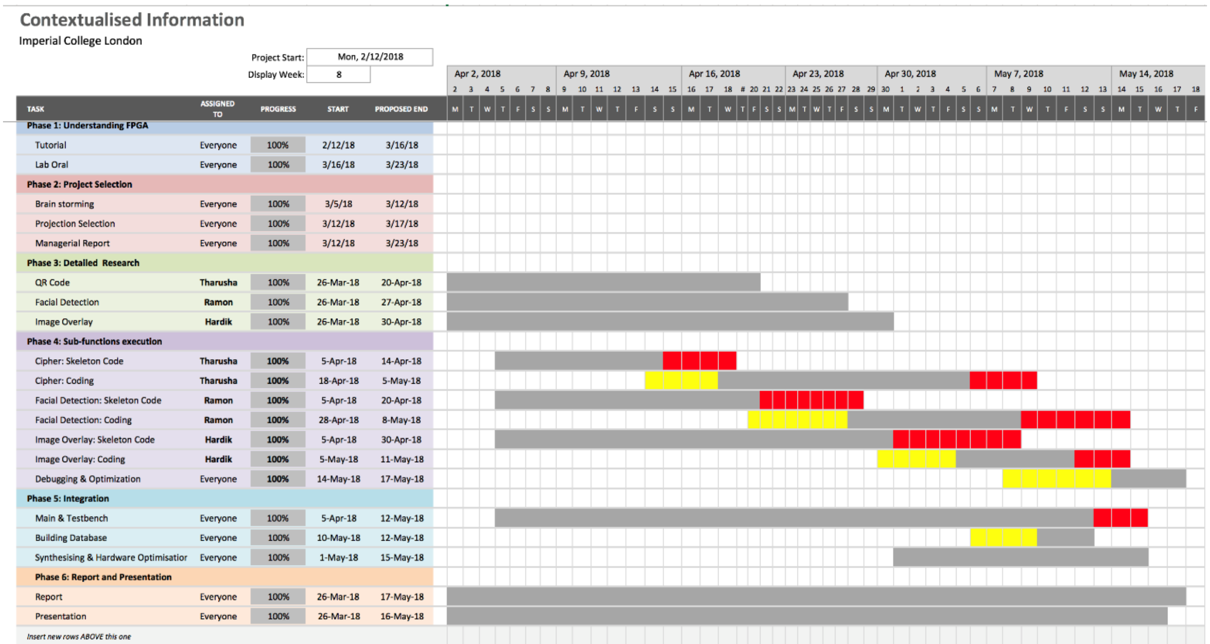


Figure 3: Updated Gantt chart

The updated Gantt chart was created at the beginning of phase three, hence the updated chart omits the first phases, as they are identical. The grey area indicates the allocated time in which each task was supposed to be performed. The red bars indicate an extension to the time taken for the task, as a result some subsequent tasks were delayed from starting, as indicated in yellow. In this revision, each member would still work on a section of the project but working in a group allowed the members to ask for advice and help from each other and allowed all members to take key decisions together.

2.3 Division of functions

The whole project was divided into three main phases. This helped the team later in crosschecking how each form of code performed independently of each other, and to determine the bottlenecks in the system. Having set many smaller objectives instead of few large objectives in each section gave the team regular checks on how much they have covered and what the timeline looks like. It also gave reference points to map progress according the Gantt chart.

3 Updated High Level Description:

The flowcharts in figure 4 and 5 illustrates the main changes in the system flow between the management report and the final report:

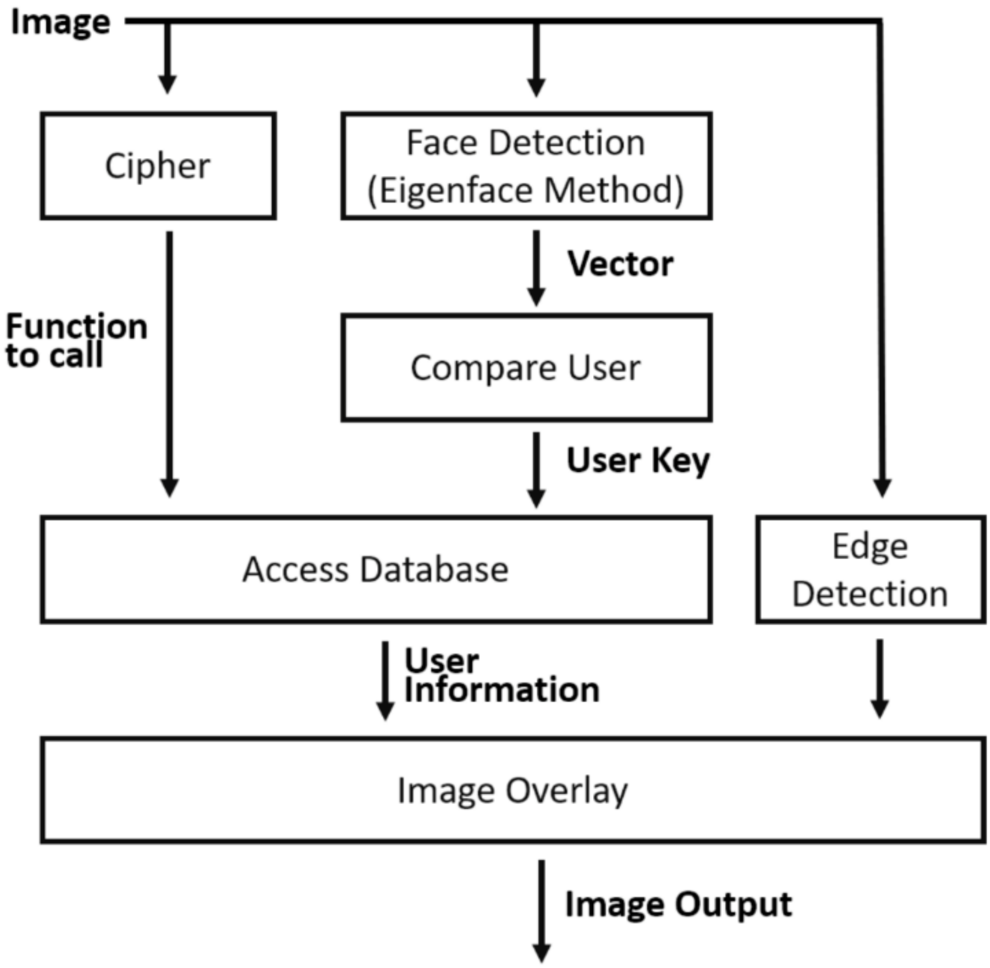


Figure 4: Flowchart of system in management report

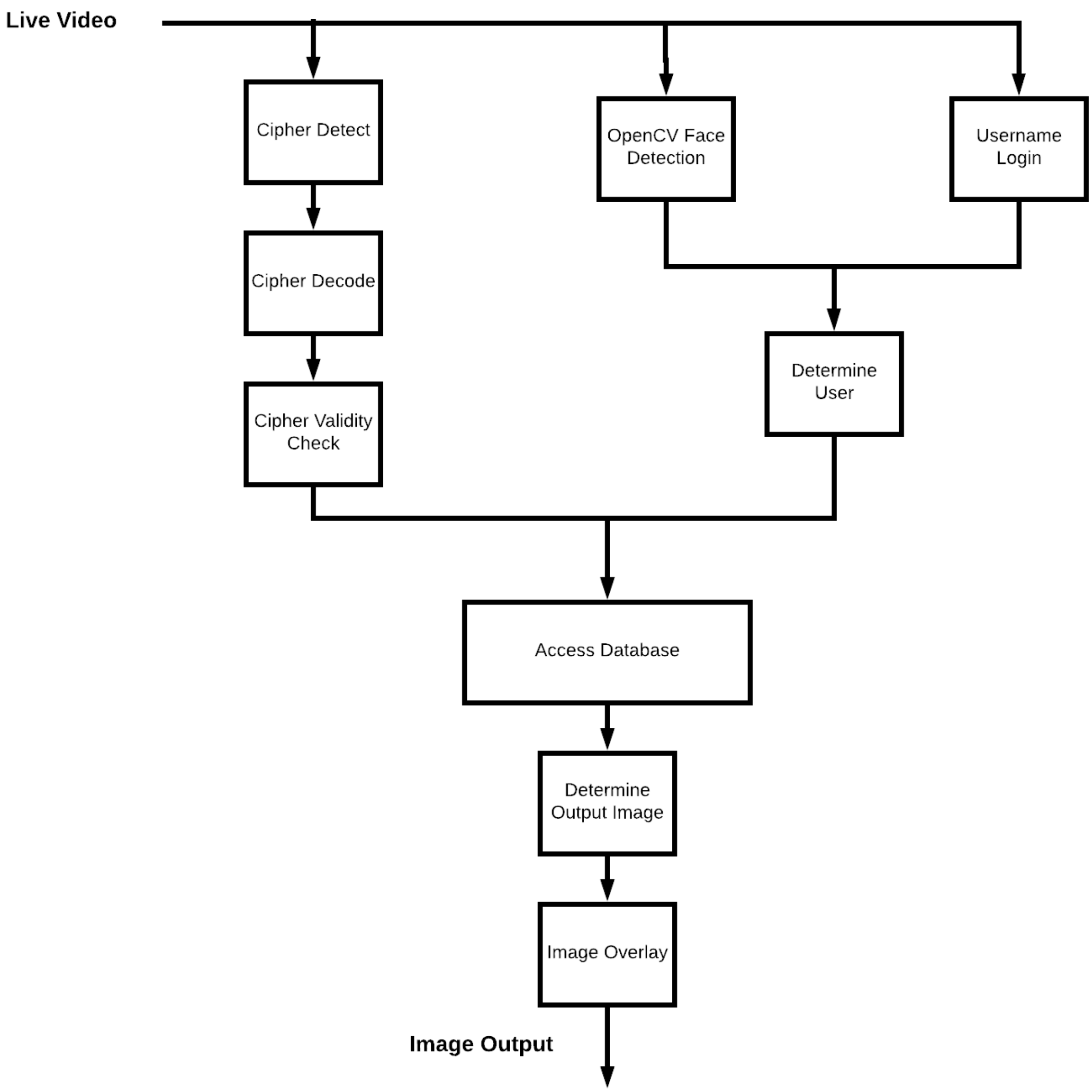


Figure 5: Updated flowchart

As described in the management report, the function of the design is to first determine the user, then scan the cipher. From this an image is outputted, which is specific to both the user and the cipher, and hence the output information is contextualised depending on the user.

3.1 Updated functional block diagram of the system

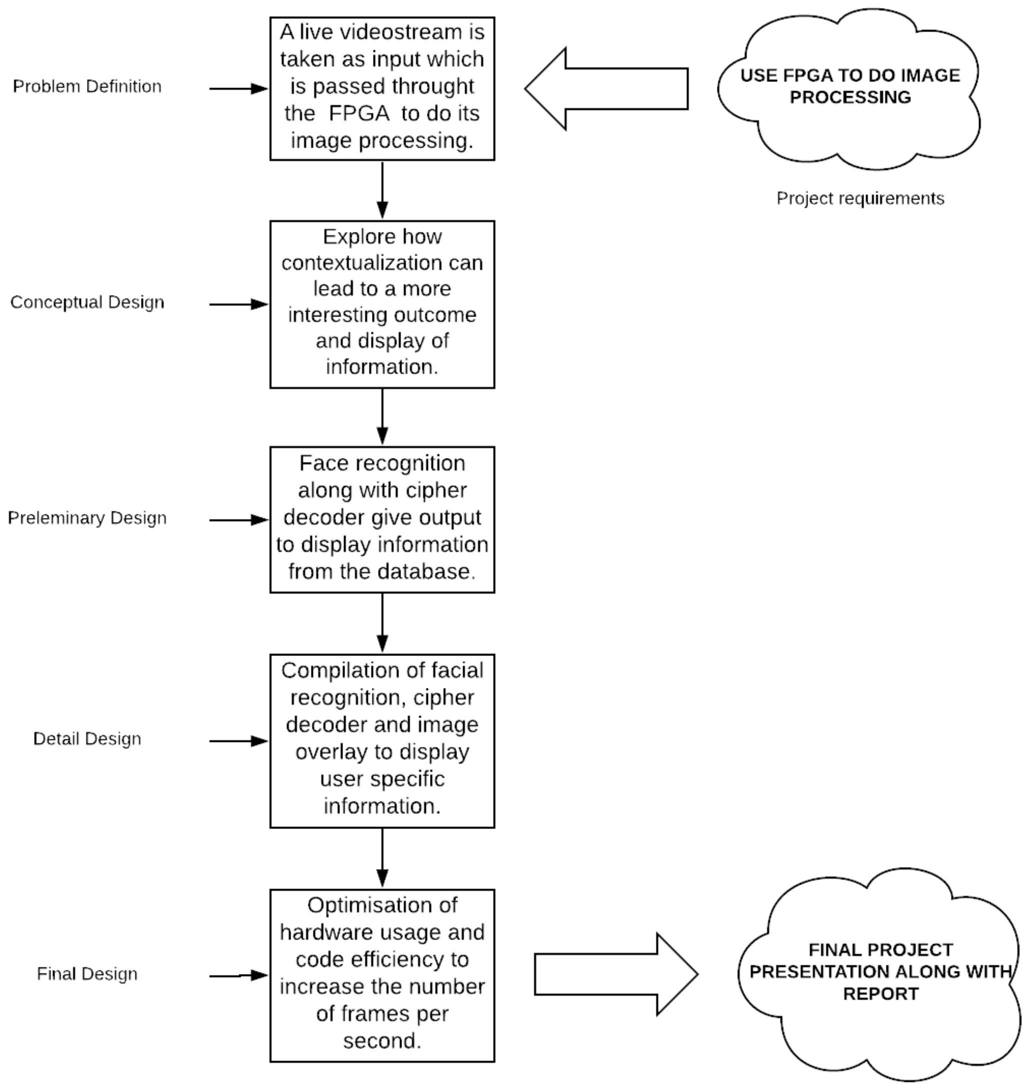


Figure 6

3.2 Detecting and determining the cipher

The first part of the function is segmented into three further parts: locating, translating, and outputting the result of the cipher. Determining the location of the cipher was initially planned by filtering the entire image using edge detection and searching for the main characteristics of the code. Using edge detection, the cipher would be detected successfully, but detecting the orientation, and reading the value of the cipher proved to be difficult. However, the cipher features an orientation marker, but it is intended to indicate to the user which way is the correct orientation of the code. The intended input source, a camera, will produce images where the cipher will most often be slanted or distorted due to the camera angle. Therefore, the edges present wouldn't be straight, making it harder to both detect and read the cipher.

Instead, the cipher location is fixed (and indicated) on the screen. It is read horizontally, where the 'read' segments are tall rectangles (much like a barcode), which means the mid-section won't be affected much by a slight slant. The cipher features a characteristic black ring, which is used to validate that it is present in the search area and to prevent false positives. The data stored in segments within the cipher is read via binary image conversion, in which the output can then be generated and used in the third phase as before.

3.3 Determining User

The second part of the project aims to obtain differentiate users to personalise the information. This is done in two ways: login and face recognition. In contrast to the management report, face recognition is done via detection of facial features using OpenCV library instead of using eigenvectors, as it requires less machine-learning to be able to detect the person. Hence, it become easier for new users to sign up to the system. The eyes and nose are detected using the Haar-cascade algorithm from OpenCV library. After determining their position of the eyes and nose, the angle between these are found. It is then used to compared to the pre-existing data on the database to identify the user.

Also, as the HDMI input is connected to mobile phone, the cipher is detected using back camera and face is detection using the front camera. Thus, the face is detected from the entire of the screen instead of just above the cipher, requiring the face to be detected prior the recognition.

In addition to face-detection, a login system is also being implemented as it allows users to log into the system in situation where face detection might not be possible or should not be used, for example, night-time, or high security system.

3.4 Displaying information

In the last part of the code, the original plan was to use the outputs from cipher decoder and facial mapping to control the image being displayed on the plane of the cipher via augmented reality. The images themselves are stored as files on the board's SD card (alongside the python notebooks used to run the board) and specific images are extracted based on who the user is. Depending on whether the processing system has stated that an image should be shown (decided by a parameter value), the system will include the image in the output frame alongside the processed video frame. However, although the plan was to display the image over the plane of the cipher, it proved to be far more complex and computationally expensive to implement augmented reality alongside the other two functions. Hence the image is instead output as an overlay in the top corner of the output frame and will only remain there until a button is pressed on the board itself.

3.5 Outcome of the system

The outcome of the system is that when someone scans a cipher, then the system will output information on the screen based on the person scanning the code. This information will appear in the top left of the screen, which will remain until the user specifies that they want to change the image (via a hardware button). The system will also provide the option to switch the user, either via face detection or by entering a username and password, as seen in figure 5.

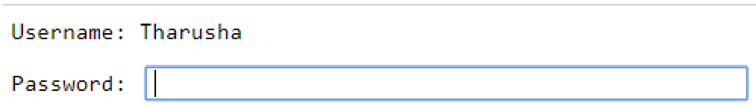


Figure 7: Username and password fields

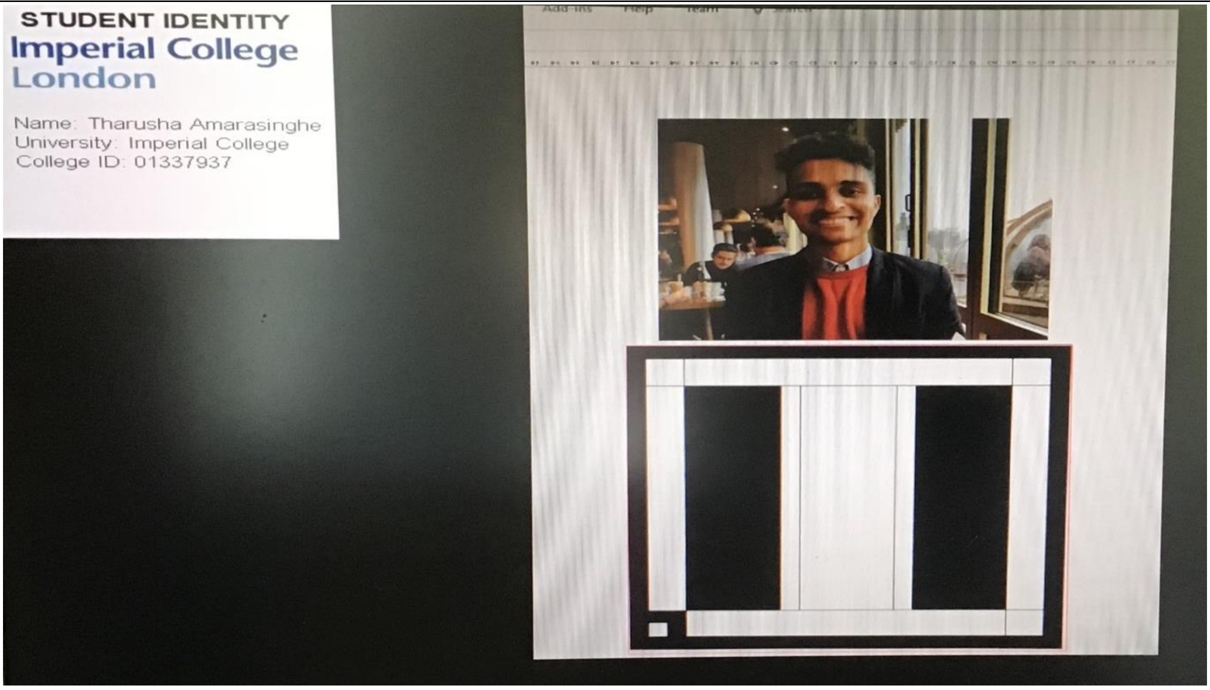


Figure 8: Student ID of first user

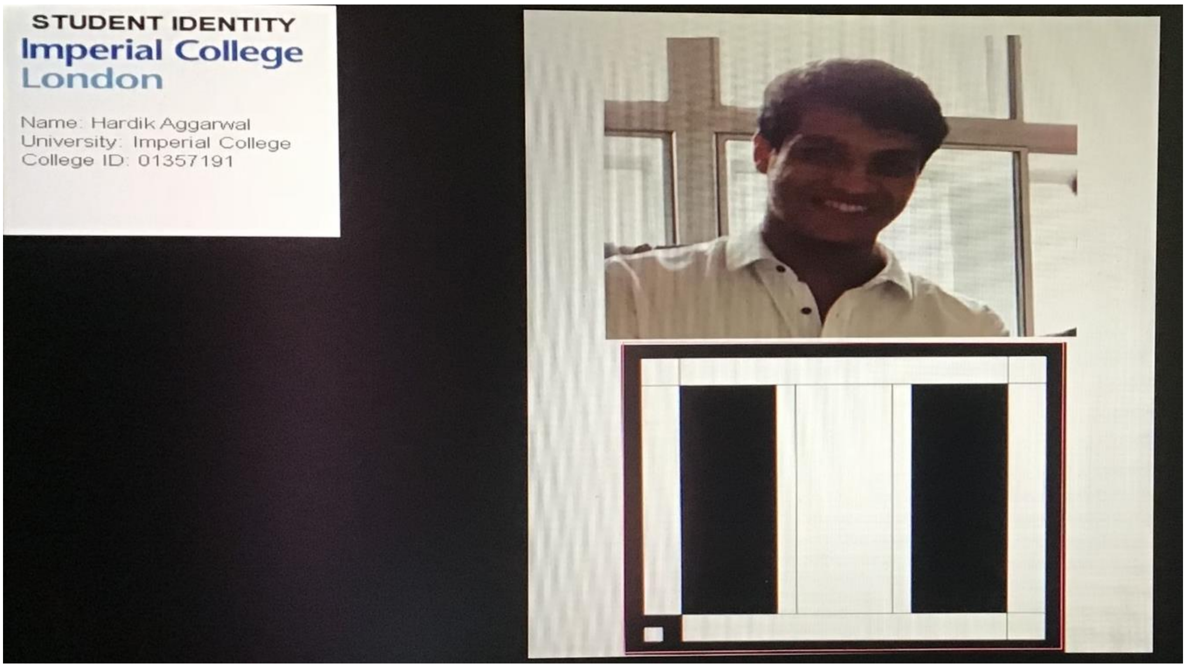


Figure 9: Student ID of second user

Figures 8 and 9 shows how the output image in the corner will change depending on the user. For illustration purposes, the image of the user is also displayed above the cipher (the face scanner will wherever the face is as long as it is large enough). Although the cipher is the same, as the user is different the information shown is also changed.

4 Design Process

4.1 Hardware

For our project the system is made to run on the PYNQ-Z1 board, which unites a processing system (PS) with programmable logic (PL). This allows the PS to work functionally to complete tasks, treating the PL like a software library of functions that is imported into the PS. Since the PL is custom-made to execute its functions as fast as possible, it will be able to work a lot faster than the general purpose PS. This makes image processing on the PL far more efficient, since each frame involves many pixels which are individually processed (for example, a 1080p image has 2,073,600 pixels in each frame).

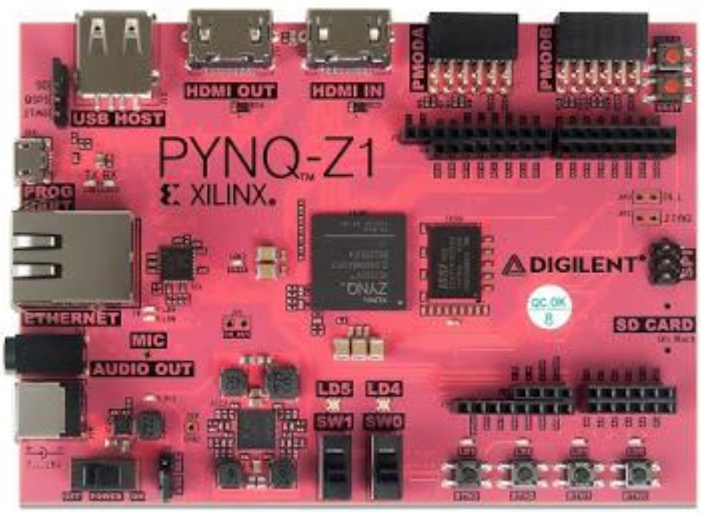


Figure 10

The hardware in the PYNQ-Z1 board[6], that we used in our system is as follows:

- 650 MHz dual-core ARM processor
  Core part of processing system, and is used to execute the system
- 512MB DDR3
- HDMI input port
  Interfaces with the source (with a pre-processed video stream)
- HDMI output port
  Interfaces with a display to show the processed video
- Gigabit Ethernet port
  Communicates between the firmware running on the board and the host computer
- USB 2.0
  Powers the board
- MicroSD card slot
  Holds an SD-card which contains the firmware for the board
- Physical buttons (labelled 0 to 3)
- FPGA Fabric:
  - 13,300 logic slices, with 8 flip-flops and 4 look-up tables (LUTs) each
  - 630 KB of fast block RAM
  - 220 DSP slices

  The fabric is the programmable logic, used to implement the specific image processing functions used by the system

In addition to the board, we also used the following:

- Laptop to interface with board firmware, provide power and act as the HDMI input source
- Smartphone camera as video input via the laptop

4.2 Product Design Specification

Product Design Specification (PDS) is employed to define the design criteria for this project. It considers several factors that could influence the design of the project and this document is considered every now and then for the purposes of achieving the most optimal solution needed for each subsystem. We considered the element listed below when designing our project:

PDS ELEMENT	PROBLEMS	CONSTRAINTS
PERFORMANCE	What will the project be able to do in the end? How much will be functional? What all can be done to optimise the performance?	Synthesis constraint- how synthesis of HDL code to RTL occurs. I/0 constraint- assign signal to specific I/O and specify user configurable I/O.
ENVIRONMENT	Which language is the code written in and in which environment is it synthesised?	Web server hosts Jupyter Notebook Design environment. The source code is in C.
SIZE	Although the FPGA is small in size, but it needs to be connected to a desktop and camera for display and input. Hence, setup is large and difficult to move around.	Most of the work needs to be done on lab computers as desktop is required. Electronics lab closes at 5pm.
STANDARDS AND SPECIFICATIONS	What is the computational power of the FPGA? How much memory is available for usage?	220 DSP slices, 630 KB of fast block RAM, 650 MHz dual-core Cortex-A9 processor. 512 MB DDR3/FLASH.
QUALITY AND RELIABILITY	Do all the functions work all the time? Are there any issues with the PYNQ board?	Logical complexity needs to be in check, power consumption should not be exceedingly high
COMPATIBILITY	What kind of software and hardware compatibility is needed to run the FPGA?	Jupyter Notebook design environment, iPython kernel and packages, Linux, Base Hardware library and API.
HARDWARE	Apart from video what other inputs can be used and how? What circuitry is available on the board?	Zynq XC7Z020, Ethernet, HDMI in/out, MIC in, Audio out, Arduino Interface,2 Pmods, 16 pin GPIO header, user LEDs, pushbuttons and switches.
TIME	How much time can be devoted to the project in the last term? What all work should be done over April to ease the load?	Project deadline a week before final exams commence.

Figure 11: PDS

4.3 Brainstorming of ideas

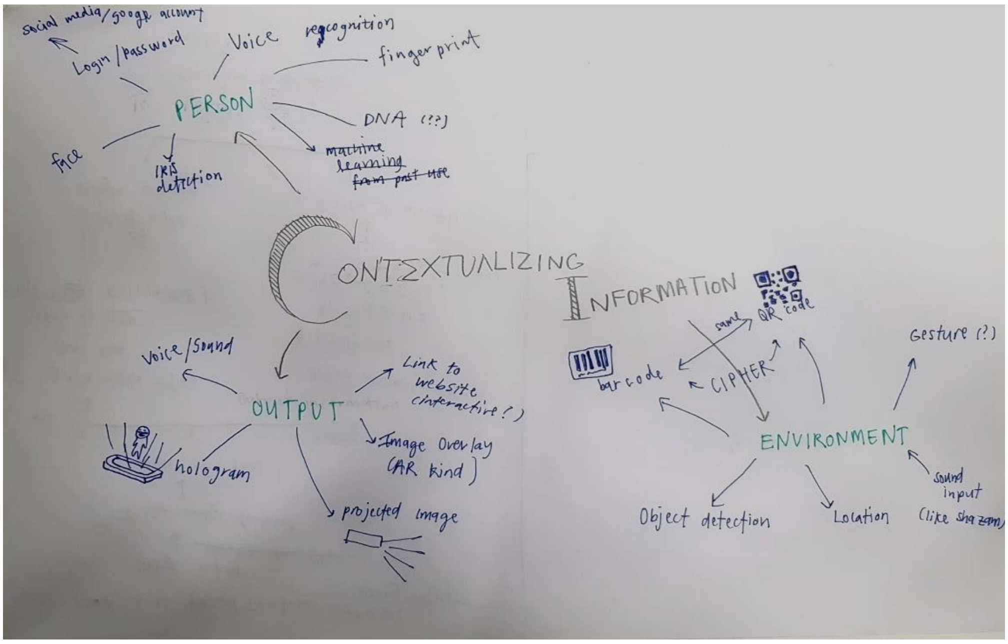


Figure 12: Brainstorm

The three main functions that the system will perform were selected and are as follows:

- Able to detect an object in an environment,
- Able to distinguish between the users that are in that environment,
- Displaying the data back to the user.

Upon brainstorming some ideas for each function (figure 12), the advantages and disadvantages of each design were evaluated, and summarised in the tables below:

Methods to distinguish between people			
Solution	Description	Advantages	Disadvantages
Login Account	• User signs into system with a username and password. • System assumes user remains logged in until they logout	• Very secure if password is private • Can reliably identify individual	• Easy to fake, if the password is known to another person • Can be time consuming • User may forget to logout
Fingerprint	• A fingerprint scanner is used to identify the users	• Secure Quick to access • Could be used to unlock specific information	• Can sometimes cause delays if finger is dirty/unrecognisable • Requires I/O interfacing with external hardware
Face recognition	• Determine and compares peoples' faces to an already present database,	• Convenient Relatively secure	• Complex to implement • May fail for specific cases such as user

	needs to be checked periodically		wearing glasses, or poor lighting
Voice Detection	• Login by detecting the user's voice when they say a phrase	• Secure • Convenient for user • Quick to access	• Complicated to implement for a lot of people • Unreliable in a noisy environment

Figure 13

Methods to determine environment			
Solution	Description	Advantages	Disadvantages
Object Detection	• User scans the object to obtain information	• Simpler to implement • Will work for many variations of a shape • Detection isn't affected by colour	• May not recognise if object is partially blocked • Edge detection won't work in bad lighting scenarios (e.g. outdoors)
QR Code/Cipher	• Encode information using a cipher, such as a QR code • Create a scanner to read and output the result	• Can be very small, but holds a lot of information • Easy for user to scan • QR decoders available	• Vulnerable to poor lighting conditions • Hard to detect in an image • Depends on database which uses memory
Location	• Use GPS to obtain the location of the system • Provide location-based data	• Won't require any additional effort	• Requires external internet access to obtain GPS data • Restricted to that area, other parts of information can't be accessed • GPS is unable to provide very specific location
Gesture Detection	• Detects specific gestures made by the user to display different information.	• Quick to use • Convenient • Interactive • Easier to remember than password	• Easy to confuse similar gestures • May not recognise small variations/orientations of gestures • A very large database is required to compare the gestures.

Figure 14

Methods of outputting data			
Solution	Description	Advantages	Disadvantages
Overlay	• Information is displayed at the	• Real-time information changing	• Can be intrusive, and may cover object of interest

	side/a corner of the image	• Keeps environment visible • Possibility to show or hide	
AR	• Displays the information, by augmenting it whilst tracking an object	• Real-time information • Environment is always visible • Less intrusive • Very immersive	• Complex to implement - more likely to work unexpectedly • Difficult to see if display area is restricted
Sound	• Information relayed verbally back to the user	• Interactive • Immersive – allows user to observe environment more • Subtitles could be displayed alongside	• May be irritating if unwanted • Hard to hear in noisy surroundings • May miss information if there is no subtitles
Website	• System outputs to a website, either via a link or in the video stream	• Will provide a lot more information • Quick to implement	• Requires internet access/interface • Can't change the output dynamically as input is no longer visible

Figure 15

5 Proposed Solutions

In this section the main methods that were considered for implementation in the project are explained. Each phase had several possible ways to execute, and all the possible ways were evaluated in terms of how they would work as well as any limitations that may be present. The chosen methods are also explained in this section, with the reasoning on why they were chosen being explained in the next section. The code for the implementation of the chosen solutions can be found in the appendix.

5.1 Determining the user

5.1.1 Chosen solution: Login and password

The user of the system indicates their identity by first signing into the system, via a username and password. The system would then assume that the user doesn't change for the remainder of the session, until that user logs out, and all information displayed in the session would be specific to that user.

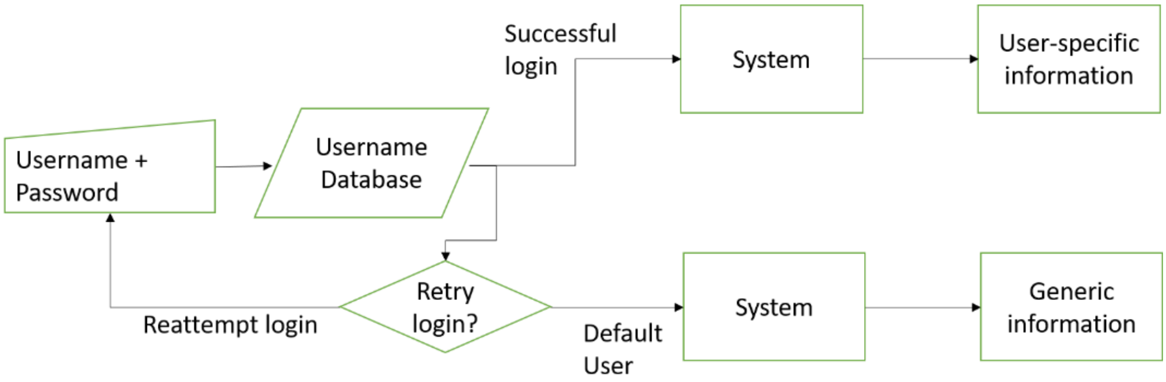


Figure 16: Login process and expected outcomes

As outlined in figure 16, the user is required to enter a username and corresponding password, this is done via the python interface for the processing system as it requires direct user input and wouldn't benefit from running on the FPGA fabric. Upon a successful login, the username is sent to the rest of the system, which results in the information display being specific to the user. However, if the username/password combination is incorrect, the user will have the option of either retrying the login or sending a default case to the system and logging in as a guest.

Figure 17

The processing system also utilises the hardware buttons found on the board. If the user needs to be changed during the video processing, a button can be pressed on the board, which causes the output to display an image as in figure 17. The PS will then provide a username and password to the user via the python interface (see appendix D), which in turn will generate new images specific to that user to display in phase 3.

5.1.2 Face Recognition

Face Recognition utilises facial features as a mean to differentiate users from each other. Features compared include: length between eyes, width-to-height ratio of the person's face, and eigenface values. These are stored in the system and used to verify the user.

Additionally, facial features can be used to distinguish between users to separate different age groups or gender. It can be done by machine-learning to generate statistic data to differentiate the group. This feature can be implemented as a supplement to generic information in case of invalid login, or as a stand-alone system. Some methods of facial recognition considered are as followed:

Facial Features Detection Using Colour [5]

First after the face is found using edge detection, the pixels in the region are convert into YCbCr format as despite different skin colour, chrominace and luminance of people falls in similar range. Using the formulae in Figure 18, YCrCb can be obtained.

$$\begin{cases} Y = 16 + \frac{65.738R}{256} + \frac{129.057G}{256} + \frac{25.064B}{256} & (1) \\ Cb = 128 - \frac{37.945R}{256} - \frac{74.494G}{256} + \frac{112.439B}{256} & (2) \\ Cr = 128 + \frac{112.439R}{256} - \frac{94.154G}{256} - \frac{18.285B}{256} & (3) \end{cases}$$

Figure 18: Source – Wikipedia

After this is done, eyes and mouth can be detected by checking colour space. (Refer to Figure 19).The recognition is done by check length of each eye, distance between eyes and length of mouth. These can be obtained using the pixel location. Once the face matched the criteria used to determine the user, this user is outputted from the function to be used in the third phase.

Facial feature detection using colour

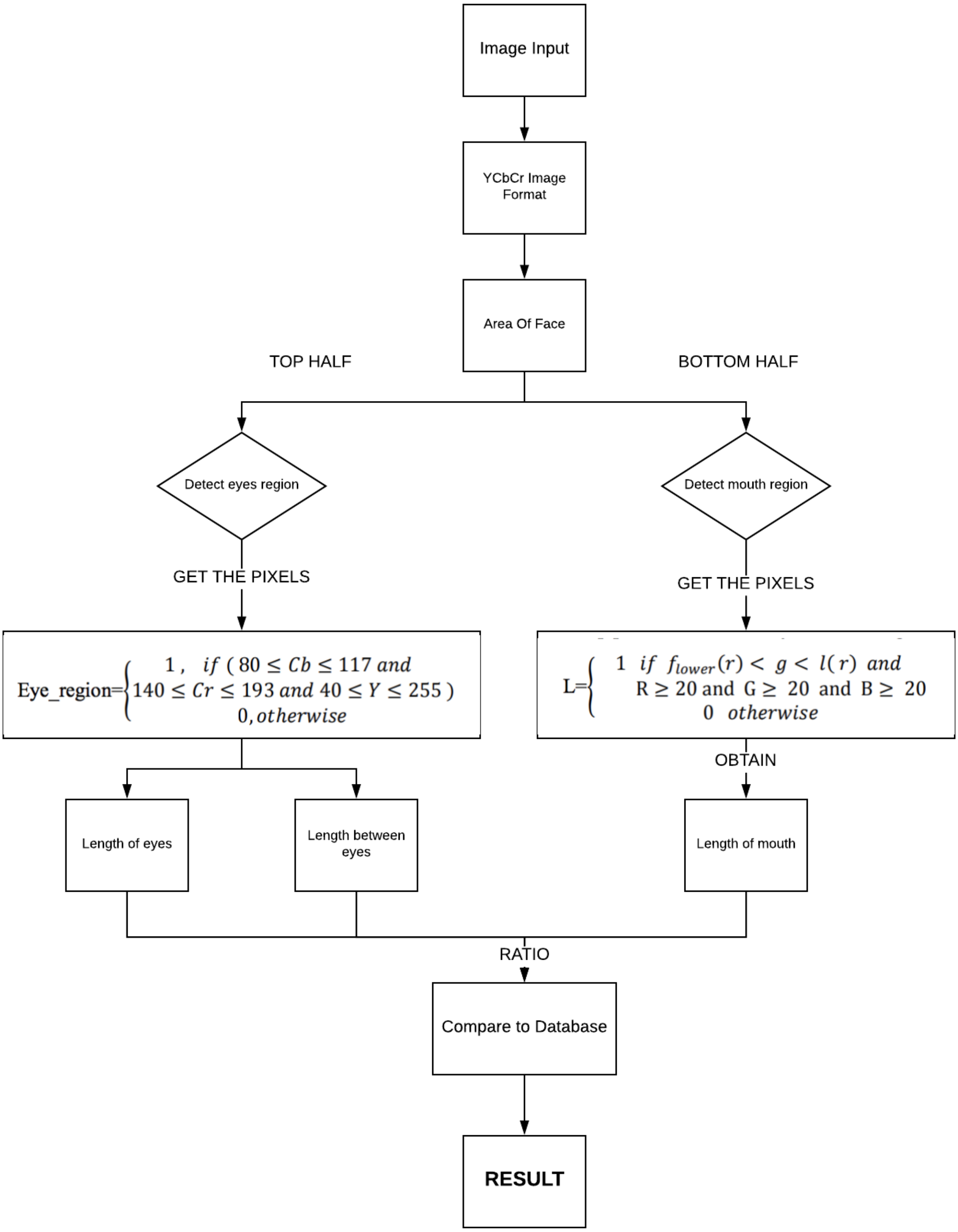


Figure 19

OpenCV – Local Binary Patterns Histogram [3]

Local Binary Patterns Histogram is used to differentiate between users. This works with multiple criteria such as gender-wise detection, age-wise detection in addition of facial recognition. This algorithm works by comparing the pixel to its neighbouring pixel. If the pixel is darker than the centre pixel, it is marked as 1, and 0 otherwise. After which the 8 binary digits can be obtained by concatenate the number from each pixel clockwise. After which the 8 binary digits can be obtained by concatenating the number from each pixel clockwise. Now, a unique pattern of the area is found. This is repeated through out the image. The total occurances of each number is then used to generate a histogram containing features of each person. Using machine learning techniques, histogram can be generalised to specific cases or general case; from person-specific to age-wise recognition, for instances.

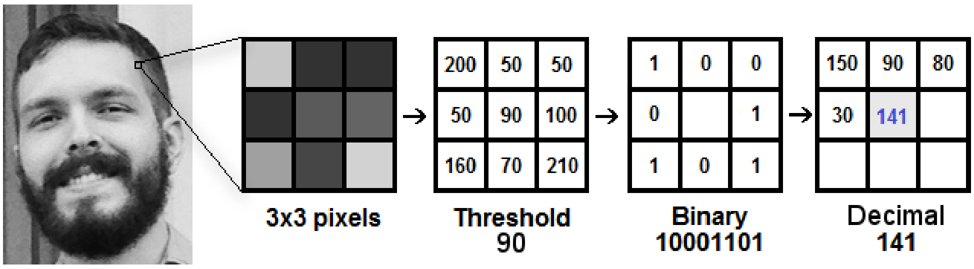


Figure 20: Source: towardsdatascience.com

After the generalised histogram is generated, this can be stored on the PYNQ SD card itself, hence no more training need to be done. Once the user face is scanned, face is detected using Haar-cascade (as explained in the next section.) LBPH is then applied to the face-region and compared to histogram. If the result differs below the threshold with a person's histogram, it can be concluded that the face belongs to that person.

Chosen Solution: OpenCV – Haar Cascade[8]

This algorithm involves using Haar-Cascade to detect facial parts where unique facial characteristic of the user can be obtained. Haar-cascade is a series of Haar-like features test where the image can to satisfied in order to be consider as a face. Haar-like features is an algorithm used to detect specific shape such as edge or line. This works by calculating the average value of pixel (in grayscale) under the white part and black part of kernel. If the difference is above a certain threshold, Haar-like features is considered detected. This kernel is passed through the whole image.

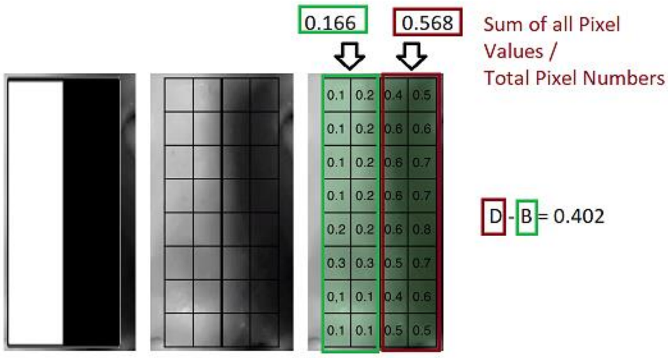


Figure 21 – Source: http://vitalimueller.com/

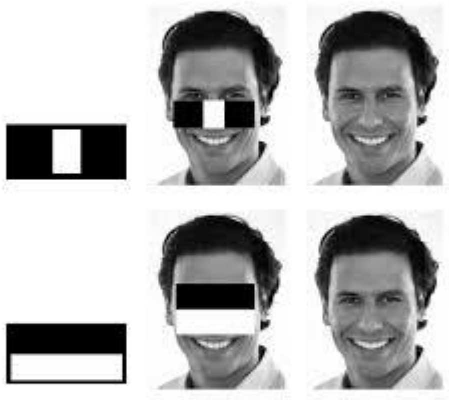


Figure 22

In order to obtain Haar-cascade, AdaBoost is used to train the cascade based on positive and negative training set. Haar-cascade consist of series of haar-like features that must be all detected in a region in order to be considered that feature.

In this algorithm, Haar-cascade is used to detect face, eyes and nose. The face is detect in order to obtain an smaller search area where eyes and nose should be located. This can be done using OpenCV library on PYNQ board. The length between the eyes and the nose can be obtained. As a user's unique characteristic, the angle of eye-nose-eye is calculated using the data obtained earlier. It is chosen as the angle remains the same despite the distance away from the camera.

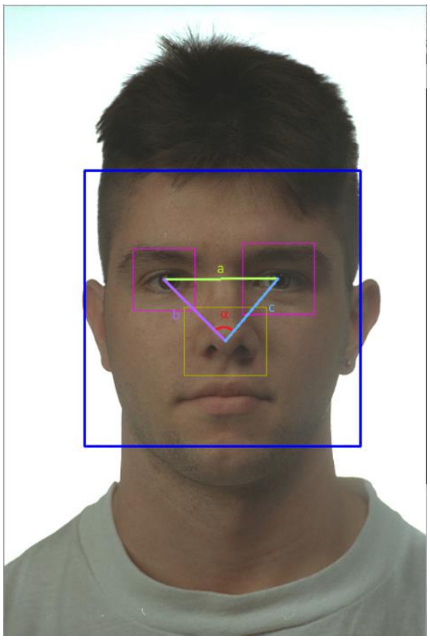


Figure 23

5.2 Determining the environment

5.2.1 1D: Pattern-detection

The 1D Barcode represents information in terms of width and spacing of parallel line. This algorithm utilises first few bits of the code to form a certain pattern to aid with localisation. This algorithm allows skewed and scaled barcode to be read as long as one straight line cuts through the entirety of the barcode (Figure 24.2) From figure 24.1, bit 0 – 3 of the barcode serve as a pattern allowing the barcode to be detected and the width of each bar to be obtained. Bit 4 – 11 is information bit, and bit 12 is odd parity bit used to ensure that information read is accurate.

The algorithm works when the user line up the bar code with the line display on the screen, which is where the data will be read. When the first black pixel is found, the width of the black pixel till it becomes white again is found. The same thing is done for the following white and black regions. Then the width of the three are compared. If they are the same, the barcode is considered detected. Once the pattern is found the value of each bit can be decoded by checking value of pixels at the middle of each bar by incrementing each time by the width obtained from pattern-detection.

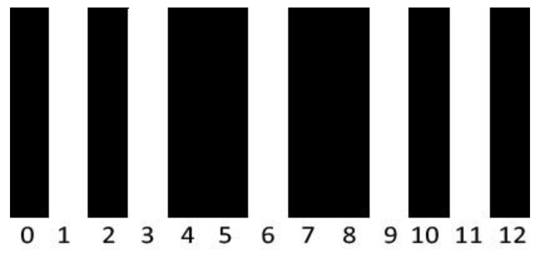


Figure 24 .1

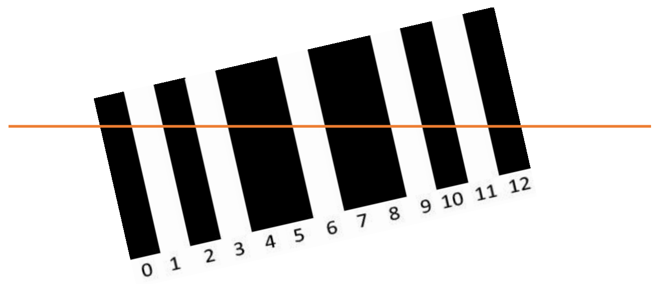


Figure 24.2

5.2.2 Region-Based Gradient Analysis

This algorithm works on the fact that barcode are parallel lines, hence have a very high gradient change horizontally but very low vertically or vice versa depending on the orientation of the barcode. This can be used to find the width and height of the whole barcode. To decode, the gradient can be analysed. The increase in gradient indicate change from black bit to white bit and vice versa. The magnitude of gradient indicates the number of same coloured bit besides each other; the gradient is lower for higher number of bits being besides each other. Another is by using scaled array. Using width and height and width, the scaling factor of the barcode and be calculated. The scaling face can be used to scale the decoding array to match the size and compare the value to each element of the decoding array.

5.2.3 Chosen solution: Segment Searching

To perform a segment search, the process is divided into 3 parts. Converting the video stream to Binary, marking the area by a box and using bars as bits.

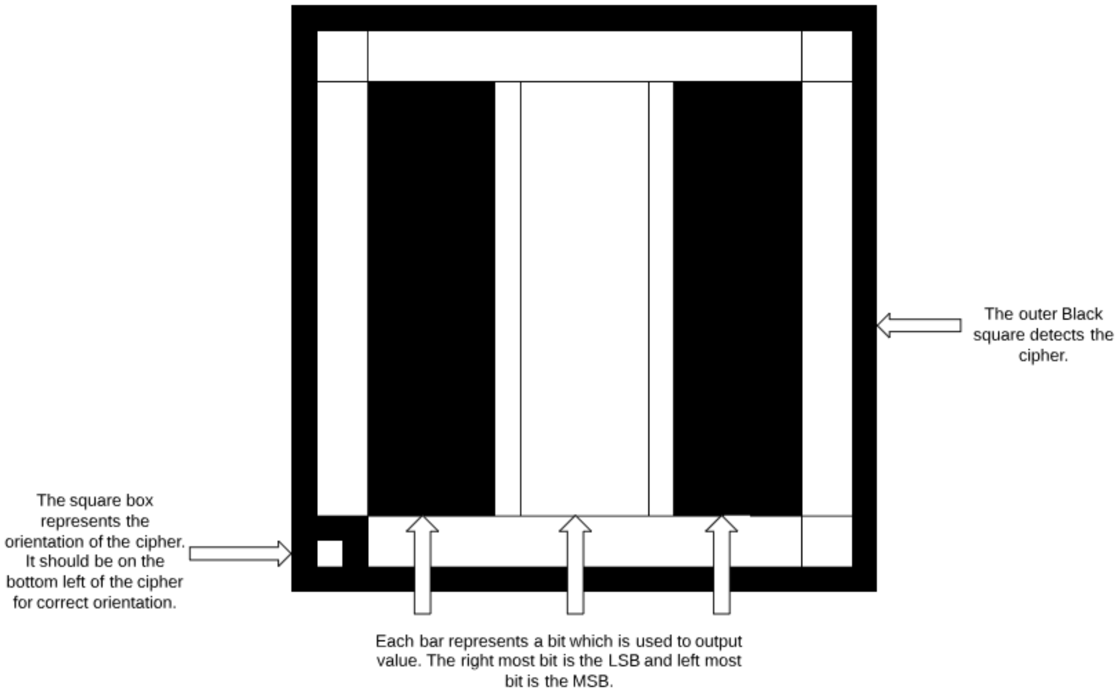


Figure 25

For the first part the input video stream is converted to Binary stream. To do this a greyscale filter is used which gives the intensity of each pixel, which in turn is used to create a binary image [2]. This is done by first deciding on a threshold value, which in this case is just half the intensity of a white pixel. Each pixel intensity is compared to this threshold value so that if pixel intensity is greater than threshold, the pixel is coloured white, and if less it becomes black.

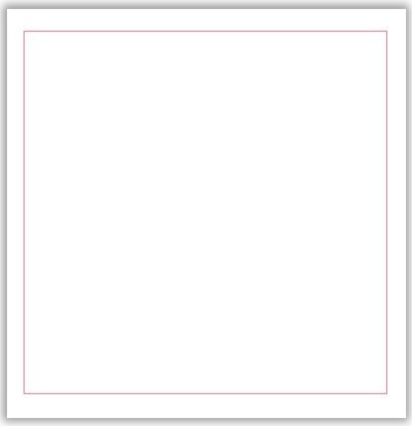


Figure 26: Search area indicated by pink box

This method doesn't involve detection, but instead searches a restricted area of the image, and checks if the cipher is valid. The search area for the cipher is already indicated using a box on the screen, as in figure 26. In this area, the code finds a black square ring with width approximately equal to 15 pixels. On detection of such a box it recognizes the presence of cipher, and validates the output is correct. The orientation of the cipher assumed to be correct, so the cipher features an orientation marker for the user to known when the cipher is orientated in the correct way.

The bars inside the cipher are interpreted as bits with colour black being 1 and white being 0. If 80% of the area of bar is black it is read as a black bar otherwise it is a white bar. There are 3 bars present representing 3 bits with MSB (most significant bit) on the left and the LSB (least significant bit) on the right. Hence, these 3 bits can represent numbers from 0 to 7. As soon as any black bar is detected the bit value is raised to 1, which is then multiplied by the bit weightage of that bit. All these values are added to obtain an integer value. Finally, if the black ring is present to prove that the cipher is valid, the function will output the integer value, but if not, it will instead output a default 'empty' value (set as 10 in the HLS code).

5.3 Displaying Information

5.3.1 AR: Affine Transformation

Image is displayed in Artificial Reality by overlaying over the cipher. When the cipher moves, or rotates, the image also follows accordingly. The movement of cipher can be detected using edge detection on the edge of the cipher. The co-ordinate to indicates where the image should be drawn. The scaling and rotation of the cipher can be done via Affine Transformation.

Affine transformation is used to correct geometrical distortion caused due to parallax in camera angles. This method is quite useful for digital image processing as by linear transformations it assigns new values to pixels such that they are shaded and any colour discontinuities in pixels due to the moved pixels are eliminated. The linear transformations can be understood with the help of some maths. Suppose, X and Y are affine space then the transform f: X->Y is given by x=Ax+b, where b is a vector in Y. If A is a matrix this can be represented as an augmented matrix by:

$$\begin{bmatrix} \vec{y} \\ 1 \end{bmatrix} = \left[\begin{array}{ccc|c} & A & & \vec{b} \\ 0 & \dots & 0 & 1 \end{array} \right] \begin{bmatrix} \vec{x} \\ 1 \end{bmatrix}$$

Figure 27: Source – Wikipedia [2]

This matrix is called an Affine transformation matrix. Using affine transformations, the image can be rotated, scaled and mirrored.

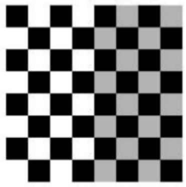
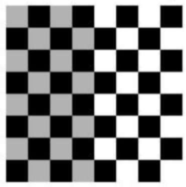
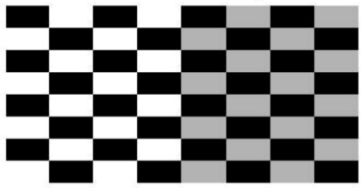
Transformation name	Affine matrix	Example
Identity	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	Identity (Original) 
Reflection	$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	Reflected Horizontaly 
Scale	$\begin{bmatrix} c_x = 2 & 0 & 0 \\ 0 & c_y = 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	Scaled 2x Horizontaly 

Figure 28: Source - Wikipedia [2]

while loading an image from the stored database an Affine transformation matrix could have been used to rotate, scale or mirror the images.

5.3.2 Hardcoding the Array:

One option of display text/images on the output frame is to hardcode a 2D-array, whereby an individual pixel has the value 1 whenever a black pixel is present, 0 for white pixels. In this method rather than passing an input image as an array to the programmable logic, arrays containing images are coded into the programmable logic. This would also allow for combinations of arrays to form longer images. For example, the letters of the alphabet could be already stored in the data and combined to form different words. In figure 29 an array of the size of the number of pixels is created. Taking any black pixels (ignoring the gridlines), these would be stored as 1 in the array, whilst the remaining white pixels are stored as 0, as illustrated in figure 30. When the image, and hence the array is called via a passed parameter, the output image is generated: when 0, the output pixel is white and when 1, the output pixel is black. Implementing this method to work is relatively easy and would allow for very quick image processing as the array could be stored on ram. However, an array would have to be assigned for each image, which can easily use a huge amount of hardware to store, which is also shared with the other functions and hence would limit the number of images greatly. Additionally, this method only allows for the output of binary images (black or white), as using colour values would require much larger array elements and would use even more hardware.

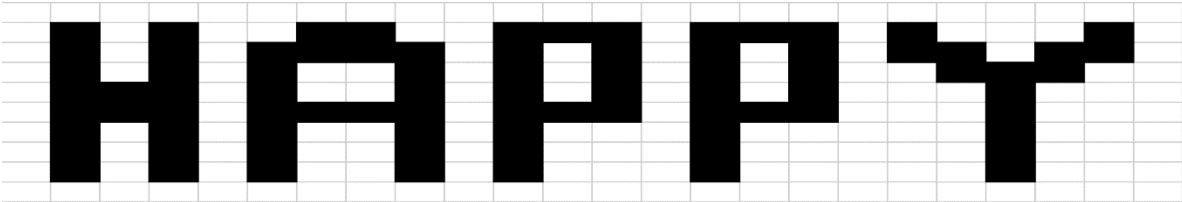


Figure 29

Suppose, this is the word to be displayed. Then the following array would be made

	#1		#1			#1	#1			#1	#1	#1		#1	#1	#1		#1				#1	
	#1		#1		#1	#1	#1	#1		#1		#1		#1		#1		#1	#1		#1	#1	
	#1		#1		#1			#1		#1		#1		#1		#1			#1	#1	#1		
	#1	#1	#1		#1			#1		#1		#1		#1		#1				#1			
	#1		#1		#1	#1	#1	#1		#1	#1	#1		#1	#1	#1				#1			
	#1		#1		#1			#1		#1				#1						#1			
	#1		#1		#1			#1		#1				#1						#1			

Figure 30

5.3.3 Chosen Solution: Image Array Conversion

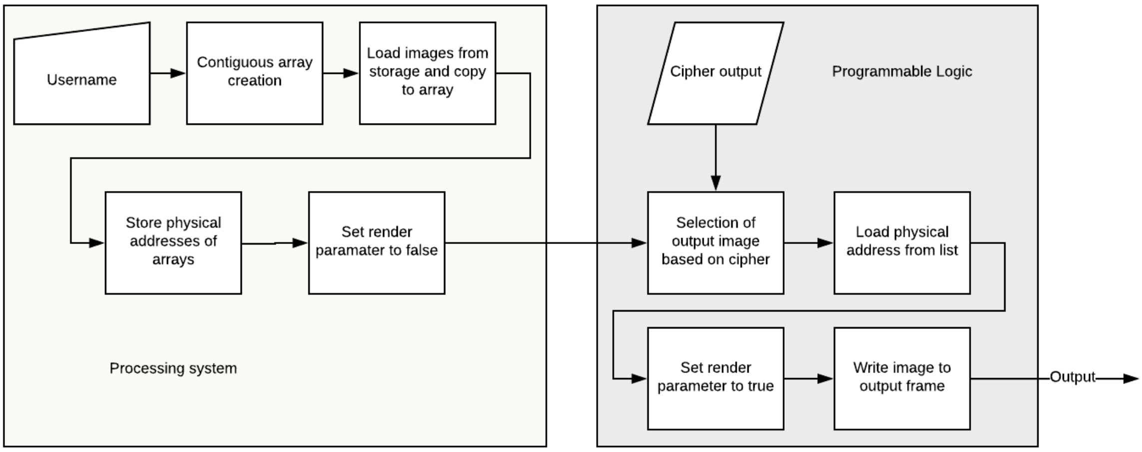


Figure 31

The image array conversion involves processing on both the processing system (PS) and programmable logic (PL). The processing system is run using python (via Jupyter), and hence the images are all stored in the same directory as the notebook used to run the board (in this case the SD card). The python code used to process the image in the PS is found in appendix C, and is run before the code that processes the video. Once a user has been specified as in the second phase, the PS will create empty multi-dimensional contiguous arrays[7] to store the images. This is required as the arrays need to be stored in a contiguous portion of physical memory on the board, so that it can be used by the PL in the same manner as the HDMI video frames.

The images corresponding to the user stored on the SD card are loaded and converted into standard multi-dimensional arrays, in which the pixels are individually stored. In this case the image names are prefixed by the username or characteristic of the person. However, the contiguous array is part of the pynq board python module, and hence the images cannot be directly imported into the arrays. Hence each element is instead copied element by element from the standard arrays to the pynq contiguous arrays, via nested for loops. This results in the image being stored in a contiguous array, which has a physical starting address and is therefore useable as a pointer for the programmable logic. The physical addresses of the arrays of each image are stored in a list for easy access during the video runtime.

Once the PS has created the image arrays, the physical address of the array is passed to the PL, which in turn will load the image and store it in a hardware array for quick access (utilising the RAM on the PL). When during the processing loops, the pixel location is that of the printing area, the pixels stored in the hardware array are loaded into each pixel. This will in turn overlay the input image pixels instead of the input video pixels onto the output video pixels for that area, as illustrated in figure 32. Since the images are already stored as memory, they can be processed via a pipelined loop, as seen in appendix B. The image being displayed is changed by simply changing the physical address input to the PL (from the list generated in the PS). Additionally, the PL also includes an option to not display the image at all (the *render_image* variable in the code), which will only display the output image if the variable is passed as 1 and will instead just output the input video pixels instead, as in figure 33.

Figure 32: Part of output image, with display parameter passed as 1

Figure 33: Part of output image, with display parameter passed as 0

The PS utilises the hardware buttons to allows the user to be changed, as mentioned earlier in section 5.1.1. Once a new user has been specified, the output images are reselected as before (using the name as a prefix), and the contents of the contiguous arrays are replaced with the new images. Therefore, the outputs on the screen will be relevant to the user. As there will be a slight delay whilst the new images are processed, the loading image as in figure 27 is displayed instead.

6 Final design

6.1 Method selection

Once the proposed solutions were explored, each was evaluated in terms of specific criteria, and given a score as seen in figures 34-36. Each criterion for method selection was given a specific weightage based on its importance for the final project for e.g. implementation time is a more important criteria as compared to user ergonomics or algorithm efficiency. Though both are also important for the project but even a code with lower efficiency can be run if it gives better system performance as compared to an efficient algorithm with bad system performance. These scores were then totalled together to give an overall score for each method. The highest scorer was then deemed the most suitable method to use to implement the system:

FACE RECOGNITION	POSSIBLE WAYS	FACIAL FEATURES DETECTION USING COLOURS		OPEN CV - LOCAL BINARY PATTERN HISTOGRAM		OPEN CV - HAAR CASCADE	
CRITERIA	WEIGHTAGE (W)	SCORE (S)	W x S	SCORE (S)	W x S	SCORE (S)	W x S
SYSTEM PERFORMANCE High throughput, low utilisation of resources and increased frames per second.	8	4	32	6	48	6	48
COMPLEXITY Difficulty of implementation and understanding. (Higher means less complexity)	8	3	24	6	48	5	40
IMPLEMENTATION TIME Time required to have a bug free workable code. (Higher means less implementation time)	10	3	30	5	50	4	40
USER ERGONOMICS How easy it is to use the feature.	6	3	18	7	42	5	30
HARDWARE UTILISATION Amount of hardware used by the method. (Higher means less hardware usage)	8	1	8	7	56	7	56
ALGORITHM EFFICIENCY Measure of average execution time necessary for the code.	6	2	12	8	48	7	42
TOTAL		124		292		256	

Figure 34

Based on Figure 34, OpenCV - LBPH is attempted due to its accuracy and high algorithm efficiency. However, OpenCV – Haar-cascade is implemented due to the difficulties faced by the group explained in section 7.1. Haar-cascade is chosen over facial features detection using colour as it has lower hardware utilisation.

DISPLAYING INFORMATION	POSSIBLE WAYS	AFFINE TRANSFORMATION		HARDCODING THE ARRAY		IMAGE ARRAY CONVERSION	
CRITERIA	**WEIGHTAGE (W)**	SCORE (S)	W x S	SCORE (S)	W x S	SCORE (S)	W x S
SYSTEM PERFORMANCE High throughput, low utilisation of resources and increased frames per second.	8	5	40	2	16	7	56
COMPLEXITY Difficulty of implementation and understanding. (Higher means less complexity)	8	1	8	7	56	5	40
IMPLEMENTATION TIME Time required to have a bug free workable code. (Higher means less implementation time)	10	2	60	5	50	5	50
USER ERGONOMICS How easy it is to use the feature.	6	7	42	3	18	7	42
HARDWARE UTILISATION Amount of hardware used by the method. (Higher means less hardware usage)	8	4	32	7	56	7	56
ALGORITHM EFFICIENCY Measure of average execution time necessary for the code.	6	4	24	1	6	8	48
TOTAL		206		202		292	

Figure 35

From figure 35, Image array conversion is chosen as it can display detailed images and utilises less hardware than hardcoding the array. Also, it is easier to implement in comparison to affine transformation while provide the same amount of information to the users.

READING THE CIPHER	POSSIBLE WAYS	1D PATTERN DETECTION		REGION-BASED GRADIENT ANALYSIS		SEGMENT SEARCHING	
CRITERIA	**WEIGHTAGE (W)**	SCORE (S)	W x S	SCORE (S)	W x S	SCORE (S)	W x S
SYSTEM PERFORMANCE High throughput, low utilisation of resources and increased frames per second.	8	4	32	6	48	8	64
COMPLEXITY Difficulty of implementation and understanding. (Higher means less complexity)	8	4	32	4	32	6	48
IMPLEMENTATION TIME Time required to have a bug free workable code. (Higher means less implementation time)	10	5	50	5	50	5	50
USER ERGONOMICS How easy it is to use the feature.	6	5	30	6	36	3	18
HARDWARE UTILISATION Amount of hardware used by the method. (Higher means less hardware usage)	8	2	16	4	32	6	48
ALGORITHM EFFICIENCY Measure of average execution time necessary for the code.	6	4	24	5	30	8	48
TOTAL		184		228		**264**	

Figure 36

Despite 1D Pattern detection and region-based gradient analysis' ability to detect scaled and slanted cipher, segment search was implemented due to its low hardware utilisation and high algorithm efficiency hence allowing the FPS to reach the acceptable level (14.9 FPS for segment searching and 9.4 FPS for 1D Pattern detection)

6.2 Optimizations

When using programmable logic, it is important to optimise the synthesis of the code, so that it operates as fast as possible, whilst maintaining efficient hardware utilization. Optimization is achieved by writing the code efficiently and making use of the directives in Vivado HLS. One way of optimizing the efficiency of the C code written was to reduce the number of calculated that the code performed. This was the case as some fixed calculations were being performed in loops, which was a waste of resources and instead it was more efficient to perform the calculations before the loops, which greatly reduced the number of flip-flops and lookup tables needed. Some of the methods of optimization in Vivado HLS are outlined as follows:

6.2.1 Arbitrary precision types

As described in the tutorial section of the management report. Arbitrary precision is used to reduce the number of bits used by a variable, rather than using standard C types. This has the effect of reducing the amount of hardware needed to operate on the variables, as some hardware such as multipliers can only work with a maximum number of pixels at a time. Therefore, using arbitrary precision will reduce the amount of hardware needed. Figure 37 shows the hardware utilization by the PL function, when the variables are standard types (mostly unsigned int).

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	4067
FIFO	-	-	-	-
Instance	2	25	3567	6667
Memory	-	-	-	-
Multiplexer	-	-	-	695
Register	-	-	2406	-
Total	2	25	5973	11429
Available	280	220	106400	53200
Utilization (%)	~0	11	5	21

Figure 37: No optimization

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	3	0	2662
FIFO	-	-	-	-
Instance	2	17	2396	4527
Memory	-	-	-	-
Multiplexer	-	-	-	400
Register	-	-	1554	-
Total	2	20	3950	7589
Available	280	220	106400	53200
Utilization (%)	~0	9	3	14

Figure 38: Using arbitrary precision

When implementing arbitrary precision, the maximum number stored in the variable found, and the number of bits needed to store it was calculated using $number\ of\ require\ bits = \log_2(\max number\ in\ variable)$. Hence the variable types were made arbitrary with minimal bits for each variable, the types can be in the lines 14-18 of appendix B. In addition, parts of the C code were rewritten to further improve efficiency. The resulting utilization is shown in figure 38: there is a reduction in the number of flip-flops and look-up tables used to implement the design, as well as DSPs (as less calculations were performed). This will result in faster processing and more efficient use of the hardware.

6.2.2 Loop Pipelining

Loop pipelining is a method of parallelising the workload, using the fact that C is a sequential logic. In loops, the next iteration is sequentially run after the current iteration finishes. However, using HLS pipelining, the next iteration's actions can be run in parallel with the current loops actions with a delay of at least 1 clock cycle to prevent overlap. Code outside the loops can also be pipelined. This reduces the number of clock cycles needed to operate the overall code, allowing the PL to operate at a faster rate, as shown in the reduction in clock cycles from 43 to 35 as show between figure 39 and 40.

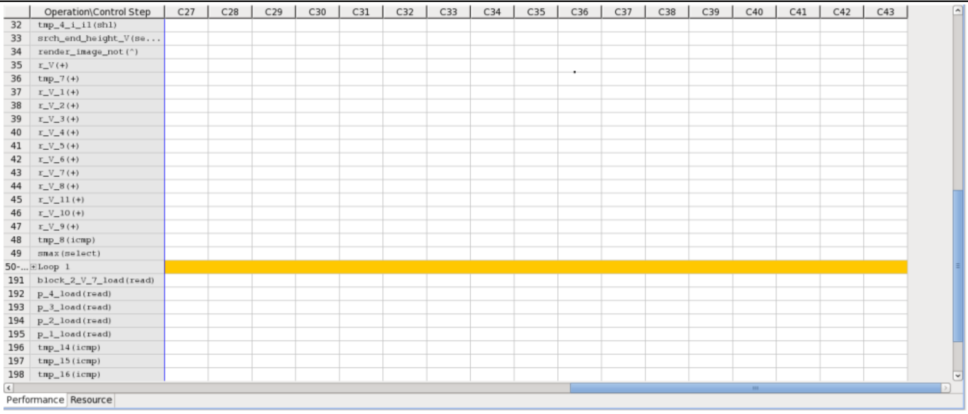


Figure 39: No pipelining

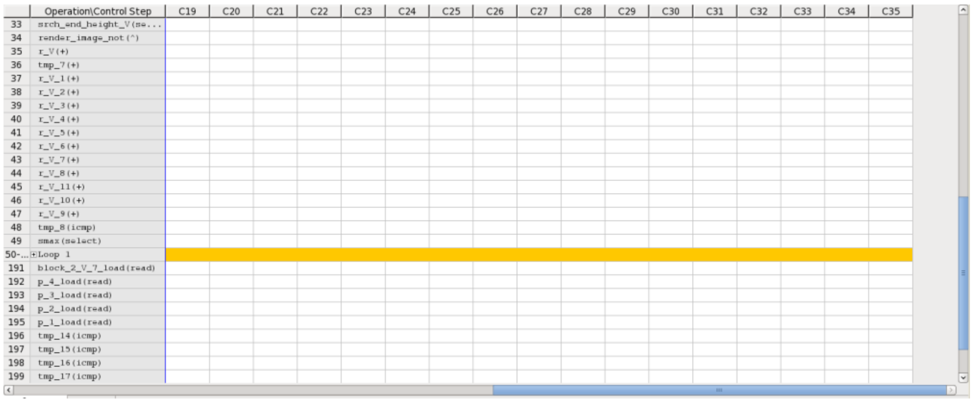


Figure 40: Pipelined

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	3	0	2679
FIFO	-	-	-	-
Instance	2	17	2396	4527
Memory	-	-	-	-
Multiplexer	-	-	-	383
Register	-	-	1604	70
Total	2	20	4000	7659
Available	280	220	106400	53200
Utilization (%)	~0	9	3	14

Figure 41: Pipeline utilization

Pipelining evidently reduces the number of clocks cycles needed to run the code, which translates to the FPGA running and interfacing faster with the processing system. This translates to a greater frame-rate in the output video.

6.2.3 Array Loading

The HDMI inputs and outputs, as well as the input image are all stored in contiguous memory on the processing system ram. However, arrays that are programmed in HLS can be used to store data on the local ram of the FPGA. This allows for much faster access rather than accessing the memory on the processing system. Although the HDMI frames are too big to store in arrays on the board, the output image is small enough (250 x 250) to store its individual pixels in an array. Hence, before processing the video, a pipelined loop was created to store the pixels from the contiguous memory into the array (see lines 56 – 59 of appendix B).

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	3	0	2712
FIFO	-	-	-	-
Instance	2	17	2396	4527
Memory	96	-	0	0
Multiplexer	-	-	-	366
Register	-	-	1382	46
Total	98	20	3778	7651
Available	280	220	106400	53200
Utilization (%)	35	9	3	14

Figure 42

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	9.13	1.25

Figure 43

This required more hardware, as shown in figure 42, especially in the amount of RAM used, and slightly increase the time to complete each clock (figure 43) from 8.75 nanoseconds to 9.13. However, this resulted in an increase in the framerate from 9.7 to 14.6 was observed when running the system. This is because by loading the image into a pipelined loop, the pixels were requested from the PS RAM in bursts, allowing data transfer to occur a lot faster, rather than requesting a single pixel every iteration of the video loop, and hence despite the apparent decrease in performance from the synthesis results, the actual performance of the board was much better.

6.3 Testing

The final design involves the combination of the three phases. Hence during the implementation of the phases, it was important to test each one, so any debugging once the code was combined was simpler.

Cipher decoder

```
   Compiling ../../../src/template_TB.cpp in release mode
   Generating csim.exe
Image type: 16, no. of channels: 3
VIDEO SIZE  [1920 x 1080]
Image type: 16, no. of channels: 3
block2: 1
block1: 0
block0: 1
Valid cipher detected
cipher_output: 5
INFO: [SIM 1] CSim done with 0 errors.
```

Figure 44: Cipher testbench output

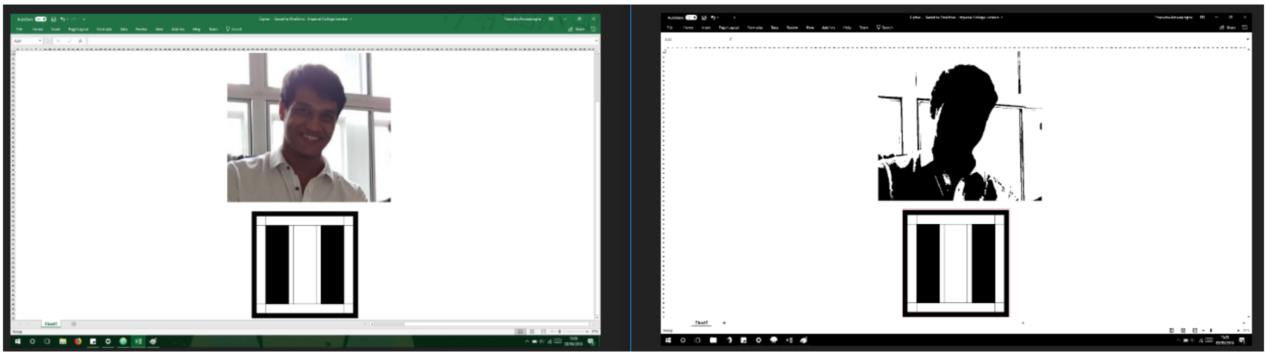


Figure 45: Image processing done by cipher reader

The first part of the decoder was to convert the input to a binary image. Hence the binary stream was given as an output to check for validity, as shown in figure 45. After testing with different values of threshold intensity, it was decided to choose the threshold intensity as half the intensity of white pixel for a better contrast. After this the code checked for the black ring in the restricted region, which marked the presence of a cipher, which was confirmed by the testbench in line 9 of figure 44. When searching the section of the cipher for the bits, their values were given as an output to check if the right bits were being detected in the correct order, as shown in lines 6-8 of figure 44.

7 Outcomes and Reflections

7.1 Challenges Faced by the Group:

1) CIPHER DECODER: The decoder function worked based on assumption that cipher was present in a predefined area of the input video stream. This is particularly difficult for the user as it is difficult for user to hold in place. To solve this problem the group had a few ideas such as making a stable base for the cipher or hanging it from the neck of the user, but none seemed successful as even small movements would take the cipher out of the grid and the camera also recognizes the face so had to be moved around. Finally, after discussion the group concluded that it was better to rewrite code using edge detection which would find the cipher anywhere in a larger area and then decode it. However, due to time constraint this task could not be accomplished, and the group had to use the original design.

2) FACE DETECTION: Face recognition using LBPH was attempted, which successfully work on software simulation, however fails on hardware. The reason is that OpenCV libraries installed on PYNQ was base-version which does not have LBPH function. An attempt has been made to install the library, however, this required installation using cmake (to convert c++ code to python library) which takes a long-time and required the board to be connect to internet constantly. This is practically impossible for the group due to unstable WIFI connection. Hence, Haar-cascade is used by the group instead.

3) IMAGE OVERLAY: In this part a predefined image present in the database had to be loaded onto the screen and displayed based on the outputs of other functions. The first solution that came to mind was to store the images in the chip on the FPGA as part of hardware and then interleave it with the displayed video stream. However, when the group tried to implement it a memory access error showed. Due to limited memory of the chip it was not possible to store them on the chip. After quite a lot of brainstorming, the group decided to access it by CPU (exposing it to python). The array required to be used by FPGA was a special type belonging to PYNQ library. There was difficulty converting the array as it had to be contiguous in memory and standard python array is not contiguous in memory.

7.2 Future Improvements

Many solutions and features are not implemented due to the lack of time, resources and technical skills. Many improvements can be made to the project. Some of which include:

7.2.1 QR Code

As a possible solution, the QR is a more complex way, but heavily standardised way of storing information. A natural way of improving the system would be to scan a QR code rather than a cipher, allowing for far more possibilities of displaying data.

7.2.2 More Accurate Facial Recognition

Currently, face detection only works only with small set of user groups as the criteria is not very specific to users. More facial characteristic can be added to achieve high accuracy such as the eye-brow shape, lip length or eye length. Also, if the OpenCV-contrib could be installed in PYNQ, this will allow LBPH algorithm to be implemented thus allowing the system to differentiate the users outside the database into smaller group (by gender or age) and displayed information relevant to the group.

8 References

1. "Affine transformation," *Wikipedia*, 29-Apr-2018. [Online]. Available: https://en.wikipedia.org/wiki/Affine_transformation. [Accessed: 02-May-2018].
2. "Binary image," *Wikipedia*, 10-Apr-2018. [Online]. Available: https://en.wikipedia.org/wiki/Binary_image. [Accessed: 15-Apr-2018].
3. "Face Recognition with OpenCV," *OpenCV: Image Thresholding*. [Online]. Available: https://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html#id40. [Accessed: 18-May-2018].
4. K. Salton, "Face Recognition: Understanding LBPH Algorithm – Towards Data Science," *Towards Data Science*, 10-Nov-2017. [Online]. Available: https://towardsdatascience.com/face-recognition-how-lbph-works-90ec258c3d6b. [Accessed: 18-May-2018].
5. M. M. Hasan and M. F. Hossain, "Facial features detection in color images based on skin color segmentation," 2014 International Conference on Informatics, Electronics & Vision (ICIEV), Dhaka, 2014, pp. 1-5.
6. "PYNQ-Z1 Reference Manual," *PYNQ-Z1 Reference Manual [Reference.Digilentinc]*. [Online]. Available: https://reference.digilentinc.com/reference/programmable-logic/pynq-z1/reference-manual. [Accessed: 02-May-2018].
7. "pynq.xlnk Module," *PYNQ Introduction - Python productivity for Zynq (Pynq) v1.0*. [Online]. Available: http://pynq.readthedocs.io/en/v2.1/pynq_package/pynq.xlnk.html. [Accessed: 10-May-2018].
8. P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*.
9. Vmueller, "Face Detection Project – Python3 – OpenCV," *Vitali Mueller – Software Engineer - Personal Research Lab on Artificial Intelligence*. [Online]. Available: http://vitalimueller.com/2018/03/08/face-detection-project/. [Accessed: 18-May-2018].

Appendix A: Revised Gantt Chart

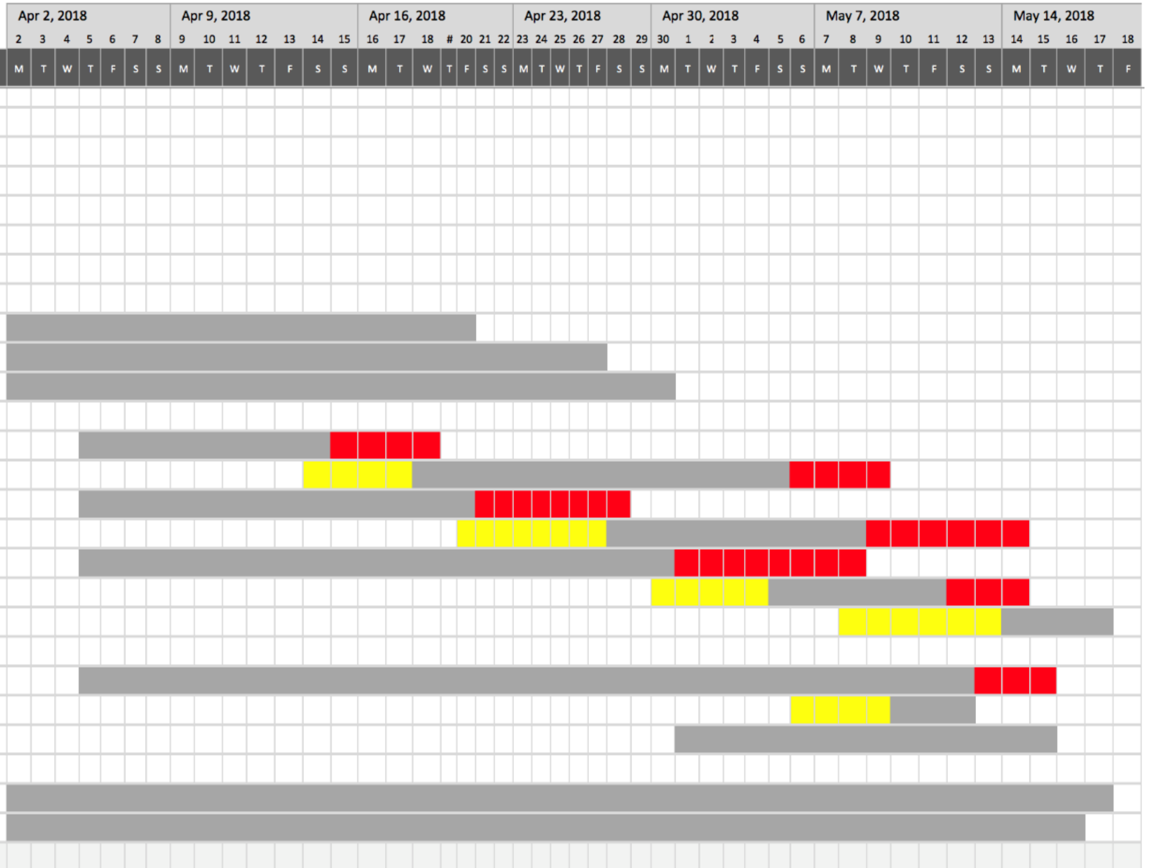
Contextualised Information

Imperial College London

Project Start: Mon, 2/12/2018

Display Week: 8

TASK	ASSIGNED TO	PROGRESS	START	PROPOSED END
Phase 1: Understanding FPGA				
Tutorial	Everyone	100%	2/12/18	3/16/18
Lab Oral	Everyone	100%	3/16/18	3/23/18
Phase 2: Project Selection				
Brain storming	Everyone	100%	3/5/18	3/12/18
Projection Selection	Everyone	100%	3/12/18	3/17/18
Managerial Report	Everyone	100%	3/12/18	3/23/18
Phase 3: Detailed Research				
QR Code	**Tharusha**	100%	26-Mar-18	20-Apr-18
Facial Detection	**Ramon**	100%	26-Mar-18	27-Apr-18
Image Overlay	**Hardik**	100%	26-Mar-18	30-Apr-18
Phase 4: Sub-functions execution				
Cipher: Skeleton Code	**Tharusha**	100%	5-Apr-18	14-Apr-18
Cipher: Coding	**Tharusha**	100%	18-Apr-18	5-May-18
Facial Detection: Skeleton Code	**Ramon**	100%	5-Apr-18	20-Apr-18
Facial Detection: Coding	**Ramon**	100%	28-Apr-18	8-May-18
Image Overlay: Skeleton Code	**Hardik**	100%	5-Apr-18	30-Apr-18
Image Overlay: Coding	**Hardik**	100%	5-May-18	11-May-18
Debugging & Optimization	Everyone	100%	14-May-18	17-May-18
Phase 5: Integration				
Main & Testbench	Everyone	100%	5-Apr-18	12-May-18
Building Database	Everyone	100%	10-May-18	12-May-18
Synthesising & Hardware Optimisatior	Everyone	100%	1-May-18	15-May-18
Phase 6: Report and Presentation				
Report	Everyone	100%	26-Mar-18	17-May-18
Presentation	Everyone	100%	26-Mar-18	16-May-18
Insert new rows ABOVE this one				



Appendix B: C code for programmable logic

```
#include <ap_fixed.h>
#include <ap_int.h>
#include <stdint.h>
#include <assert.h>

typedef ap_uint<8> pixel_type;
typedef ap_int<8> pixel_type_s;
typedef ap_uint<96> u96b;
typedef ap_uint<32> word_32;
typedef ap_ufixed<8,0, AP_RND, AP_SAT> comp_type;
typedef ap_fixed<10,2, AP_RND, AP_SAT> coeff_type;

//arbitrary precision
typedef ap_uint<11> screen;
typedef ap_uint<14> thresh_t;
typedef ap_uint<3> sys_output;
typedef ap_uint<24> pixel_val;
typedef ap_uint<17> block_t;

struct pixel_data {
	pixel_type blue;
	pixel_type green;
	pixel_type red;
};

unsigned int process_pixels (pixel_val current, bool greyscale);
int gray_filter(volatile uint32_t* in_data, volatile uint32_t* out_data, volatile uint32_t* in_image, int w, int h, bool render_image) {
#pragma HLS INTERFACE s_axilite port=render_image
#pragma HLS INTERFACE s_axilite port=return
#pragma HLS INTERFACE s_axilite port=w
```

```
#pragma HLS INTERFACE s_axilite port=h

#pragma HLS INTERFACE m_axi depth=2073600 port=in_data offset=slave // This will NOT work for resolutions higher than 1080p
#pragma HLS INTERFACE m_axi depth=2073600 port=out_data offset=slave
#pragma HLS INTERFACE m_axi depth=62500 port=in_image offset=slave


    screen srch_st_width = w/2 - 175;
    screen srch_end_width = w/2 + 175;
    screen srch_st_height = h*0.75 - 175;
    screen srch_end_height = h*0.75 + 175;
    thresh_t h_lines = 0;
    thresh_t v_lines = 0;
    block_t block_0 = 0;
    block_t block_1 = 0;
    block_t block_2 = 0;
    pixel_val output;
    pixel_val white_p = 16777215;
    pixel_val bin_p;
    pixel_val pink_p = 254 | (127 << 8) | (155 << 16);
    thresh_t threshold = 15300;

    //store image in array
    int pic_count = 0;
    pixel_val image_store[62500];
    for (int m = 0;m < 62500;m++) {
        #pragma HLS PIPELINE II=1
        image_store[m] = process_pixels(*in_image++,0);
    }

```

```
for (screen i = 0; i < h; ++i) {
	for (screen j = 0; j < w; ++j) {
		//process each individual pixel
		#pragma HLS PIPELINE II=1
		#pragma HLS LOOP_FLATTEN off

		pixel_val current = *in_data++;
		output = process_pixels(current, 1);

		//convert to binary image
		if (output < white_p/2) {
			bin_p = 0;
		}
		else {
			bin_p = white_p;
		}

		//write to image:
		//output of image from array
		if ((j <250) && (i < 250) && (render_image)) {
			output = image_store[pic_count++];
		}
		//print search square
		else if (((i == srch_st_height) || (i == srch_end_height)) && ((j>=srch_st_width) && (j<=srch_end_width))) {
			output = pink_p;
		}
		else if (((j == srch_st_width) || (j == srch_end_width)) && (i >= srch_st_height) && (i<= srch_end_height)) {
			output = pink_p;
		}
```

```
        else {
                output = current;
        }

        *out_data++ = output;


        //search the square
        if ((i>=srch_st_height) && (i<=srch_end_height) && (j>=srch_st_width)
        && (j<=srch_end_width)) {
                //check horizontnal lines of black validity square
                if ((i<=srch_st_height + 15) || (i>=srch_end_height - 15)) {
                        if (bin_p == 0)
                                h_lines++;
                }

                //checks vertical lines of black validity square
                if ((j<=srch_st_width + 15) || (j>=srch_end_width - 15)) {
                        if (bin_p == 0)
                                v_lines++;
                }


                //reading the cipher
                if ((i>=srch_st_height + 45) && (i<=srch_end_height - 45) &&
                (j>=srch_st_width + 45) && (j<=srch_end_width - 45)) {
                        //block #2
                        if ((j<=srch_st_width + 120) && (bin_p == 0)) {
                                block_2++;
                        }
```

```
                    //block #1
                    if ((j>=srch_st_width + 135) && (j<=srch_st_width + 210) && (bin_p == 0)) {
                        block_1++;
                    }
                    //block #0
                    if ((j>=srch_st_width + 225) && (j<=srch_st_width + 300) && (bin_p == 0)) {
                        block_0++;
                    }
                }

            }
        }
    }

    if (block_2 >= threshold) {
        block_2 = 1;
    }
    else {
        block_2 = 0;
    }

    if (block_1 >= threshold) {
        block_1 = 1;
    }
    else {
        block_1 = 0;
    }

    if (block_0 >= threshold) {
        block_0 = 1;
    }
```

```
	else {
		block_0 = 0;
	}

	//outputs if cipher has valid black ring
	if ((v_lines >= 8400) && (h_lines >= 8400)) {
		sys_output out_index = (block_2 * 4) + (block_1 * 2) + block_0;
		return out_index;
	}
	else {
		return 10;
	}
}

unsigned int process_pixels (pixel_val current, bool greyscale) {
	unsigned char in_r = current & 0xFF;
	unsigned char in_b = (current >> 8) & 0xFF;
	unsigned char in_g = (current >> 16) & 0xFF;

	//convert to greyscale
	if (greyscale) {
		int Y = in_r/3  + in_g/3  + in_b/3 ;
		in_r = Y;
		in_b = Y;
		in_g = Y;
	}

	pixel_val output = in_r | (in_b << 8) | (in_g << 16);
	return output;
}
```

Appendix C: Image loading code for processing system in python

```
#generate images for user
image_dim = 250
#create empty contiguous arrays
out_image_1 = mmu.cma_array(shape = (image_dim,image_dim,4),dtype= np.uint8)
out_image_2 = mmu.cma_array(shape = (image_dim,image_dim,4),dtype= np.uint8)
out_image_3 = mmu.cma_array(shape = (image_dim,image_dim,4),dtype= np.uint8)
out_image_4 = mmu.cma_array(shape = (image_dim,image_dim,4),dtype= np.uint8)
out_image_5 = mmu.cma_array(shape = (image_dim,image_dim,4),dtype= np.uint8)

#read images from files and copy to contiguous arrays
for i in range (1,6):
  image_name = username + '_' + str(i)+'.png'
  try:
    print(image_name)
    ima=scipy.misc.imread(name= image_name, mode='RGBA')
    print('read')
    print(i)
    if i == 1:
      for j in range (0,image_dim):
        for k in range (0,image_dim):
          out_image_1[j][k] = ima[j][k]
    elif i == 2:
      for j in range (0,image_dim):
        for k in range (0,image_dim):
          out_image_2[j][k] = ima[j][k]
    elif i == 3:
      for j in range (0,image_dim):
        for k in range (0,image_dim):
          out_image_3[j][k] = ima[j][k]
    elif i == 4:
      for j in range (0,image_dim):
        for k in range (0,image_dim):
          out_image_4[j][k] = ima[j][k]
    elif i == 5:
      for j in range (0,image_dim):
        for k in range (0,image_dim):
          out_image_5[j][k] = ima[j][k]
  except FileNotFoundError:
    continue

#store physical addresses of arrays to be used in PL
output_adr_list = [ out_image_1.physical_address, out_image_2.physical_address,
out_image_3.physical_address, out_image_4.physical_address, out_image_5.physical_address]
output_adr = output_adr_list[0]
```

Appendix D: Change user functions

```
def face_login(img, username):

    print('change user')
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
    eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
    nose_cascade = cv2.CascadeClassifier('nose.xml')
    right_side =[]
    left_side =[]
    colour = 0

    faces = face_cascade.detectMultiScale(gray, 1.3, 5)
    for (x,y,w,h) in faces:
        cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
        roi_gray_top = gray[y:y+int(h/2), x:x+w]
        roi_gray_bottom = gray[y+int(h/2):y+h, x:x+w]
        roi_color_top = img[y:y+int(h/2), x:x+w]
        roi_color_bottom = img[y+int(h/2):y+h, x:x+w]
        roi_color = img[y:y+h, x:x+w]
        eyes = eye_cascade.detectMultiScale(roi_gray_top)
        nose = nose_cascade.detectMultiScale(roi_gray_bottom,1.3,5)
        centers= []
        nose_centers = []
        for (ex,ey,ew,eh) in eyes:
            centers.append((x+int(ex+0.5*ew), y+int(ey+0.5*eh)))
            #creates rectangle with 'colour'
        for(mx,my,mw,mh) in nose:
            nose_centers.append((x+int(mx+0.5*mw), y+int(my+0.5*mh)))
        centers = np.array(centers)
        centers.flatten()
        nose_centers = np.array(nose_centers)
        nose_centers.flatten()
    try:
        a = cv2.norm(centers[0], centers[1])
        face_eyes_ratio = a
        b = cv2.norm(nose_centers[0], centers[0])
        c = cv2.norm(nose_centers[0], centers[1])
    except IndexError:
        a = 1
        b = 1
        c = 1
        print('undetected')
    cosine = (b**2+c**2-a**2)/(2*b*c)
    print("cosine", cosine)

    if ((cosine > 0.01) & (cosine < 0.32)):
        username = "Ramon"

        valid_login = 1
    elif ((cosine > 0.32) & (cosine < 0.70)):
        username = "Tharusha"
        valid_login = 1
    else:
        valid_login = 0
```

```
    print("cosine", cosine)
    print(username)
    change_user(valid_login, username)

def change_user (valid_login, username):

    retry = 1
    while ((valid_login != 1) & (retry == 1)):
        username = input('Username: ')
        password = input ('Password: ')
        for i in range (0, len(stored_usernames)):
            if ((username == stored_usernames[i]) & (password == stored_passwords[i])):
                print ("Welcome, ", username)
                valid_login = 1
        if valid_login == 0:
            print ("Invalid username & password")
            retry_check = input ('Retry? [y/n] ')
            if retry_check == 'y':
                retry = 1
            else:
                print("Using default case")
                retry = 0
                username = 'default'

    for i in range (1,6):
        image_name = username + '_' + str(i)+'.png'
        try:

            ima=scipy.misc.imread(name= image_name, mode='RGBA')
            print('Loading...')
            print(image_name)
            if i == 1:
                for j in range (0,image_dim):
                    for k in range (0,image_dim):
                        out_image_1[j][k] = ima[j][k]
            elif i == 2:
                for j in range (0,image_dim):
                    for k in range (0,image_dim):
                        out_image_2[j][k] = ima[j][k]
            elif i == 3:
                for j in range (0,image_dim):
                    for k in range (0,image_dim):
                        out_image_3[j][k] = ima[j][k]
            elif i == 4:
                for j in range (0,image_dim):
                    for k in range (0,image_dim):
                        out_image_4[j][k] = ima[j][k]
            elif i == 5:
                for j in range (0,image_dim):
                    for k in range (0,image_dim):
                        out_image_5[j][k] = ima[j][k]
```

```
    except FileNotFoundError:
        continue

output_adr_list = [out_image_1.physical_address, out_image_2.physical_address, out_image_3.physical_address, out_image_4.physical_address, out_image_5.physical_address]
output_adr = output_adr_list[0]
return
```