Tecnológico de Monterrey
Escuela de Ingeniería y Ciencias

TE3059 Embedded Systems

Project Title: Camera Location

Final Project

Gabriela Hernandez López A01634320

Isaac Benjamin Ipenza Retamozo A01228344

Perla Vanessa Jaime Gaytán A00344428

November 26th, 2020.

Revision History

Rev. #	Date	Changes By	Description
0.1	01/11/20	Whole Team	Last sketching of the project
0.2	05/11/20	Isaac Ipenza	Began structuring of the present document
0.3	10/11/20	Vanessa Jaime	Began structuring Project Description section
0.4	12/11/20	Gabriela Hernández	Began structuring Design Description section
0.5	16/10/20	Whole Team	As the teacher gave some advice towards the project, things needed to be changed in both the implementation and in this document regarding content and structuring
0.6	18/10/20	Isaac Ipenza	Once things were agreed between the team's members edited this document to began final version
0.7	20/11/20	Vanessa Jaime	Documented theoretical background
0.8	22/10/20	Isaac Ipenza	Documented design description
0.9	22/10/20	Gabriela Hernández	Documented test results and test strategy
1	27/10/20	Vanessa Jaime	Completed the theoretical background section including some aspects around the camera module
1.1	28/10/20	Whole Team	Final revision and editing of the document. Conclusions were added

Table of Contents

Table of Figures

1. Introduction

This document describes the project that we developed as part of the final evaluation of the course. The main goal of this document is to describe the main details of what we implemented, the general objectives of the project, the description of the project, a theoretical background and some system requirements. We also described the design of this project, some tests and results that we achieve, the verification process and some general conclusions of the development of this project. Overall, this project is a representation of how to implement a camera with facial detection and location through an DE10-Standard.

a. Scope

The main scope of this document is to present the high-level details for the final project that we designed and implemented. This document won't present any technical design specification, it is made in order to present the main objectives of the idea that we have for the final project and how it can be implemented. Some scope representation will shown

b. Intended Audience

This text is meant to be read and fairly understood by students who have a little experience with Verilog, C and C++ language. Basic knowledge of FPGA boards and hardware technology is required as well. Any person interested in using face detection on a DE10-Standard board.

c. Terminology

Acronym	Description
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
HLS	High Level Synthesis
HPS	Hard Processor System
I^2C	Inter-Integrated Circuit
LBP	Local Binary Patterns
LBPH	Local Binary Patterns Histograms
LCD	Liquid Crystal Display
LED	Light Emitting Diode
LSB	Least Significant Bit
Mbps	Megabits per second
MMCM	Mixed-Mode Clock Manager
PC	Personal Computer
PL	Programmable Logic
PS	Processing System
RGB	Red Blue and Green
RMII	Reduced Media Independent Interface
Rx	Reception
SFP	Small Form-Factor Pluggable Transceptor
SoC	System on Chip
SPI	Serial Peripheral Interface
TFT	Thin Film Transistor

Table 1 Acronyms

d. Related Documents

Inside our project proposal the essential feature is centered around face detection. For this we based the foundation on the following document, provided by Tersasic and OpenCV. To have an idea of how interconnect the 7-segment displays we used the SoC-FPGA Design Guide by Kashani-Akhavan and Beuchat. Also, the DE10-Standard CD-ROM and DE10-Standard User Manual are required. The links to these documents are linked below.

DE10-Standard OpenCV User Manual. (2017). Terasic Inc. Available at: https://bit.ly/2JpL0M7

DE10-Standard Standard CD-ROM. (2017). Terasic Inc. Available at: https://bit.ly/39nO6ew

Kashani-Akhavan, S. & Beuchat R. (2016). *SoC-FPGA Design Guide.* Available at: https://bit.ly/33qFZKv

2. Project Description

The project consists of doing a facial detection drawing the contour of it in a circle and the eyes on two ellipses. After the face has been detected, on the LCD will appear *"Face detected"*, if not, it will appear *"Face not detected"*. The camera should be attached to the DE10-Standard board in order to track its location. Through the g-sensor integrated in the board, we can visualize how the coordinates are changing along with how we move the camera attached to the board. These coordinates will be appearing on the board, the Y coordinate will also appear on the 7-segments displays in order to see how it´s value changes. On the other hand, the LEDs on the board are also changing together with the Y coordinate, that is why we decided to display this coordinate on the 7-segments displays.

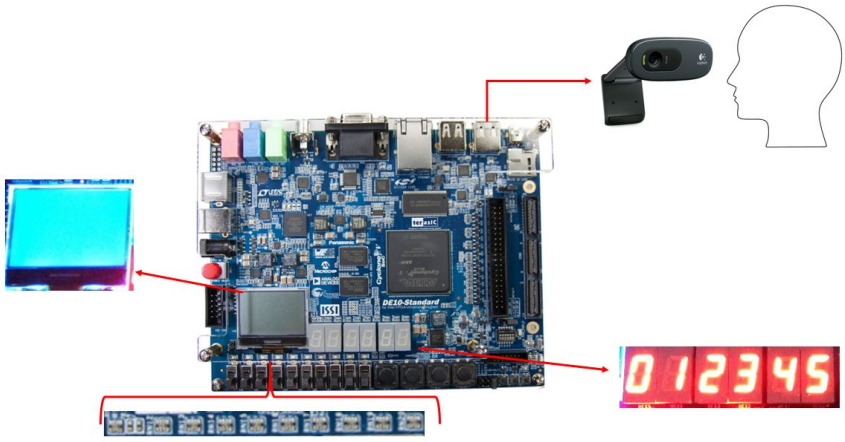

Figure 1. Basic functions of our implementation.

We found intriguing to work with face recognition, and capture our desire to include it somehow in our project, and that is the main reason why we decided to build up this project. After a discussion between the members of our team, we came up with this idea of doing an emulation of how some cameras with facial recognition worked. Our main idea was to build up some kind of an intelligent device (Figure 2) which will do certain tasks depending on the face that has been recognized, later on, we realized that the program only does face detection and not face recognition. After this, we decided to carry on with face detection, but change the goal of our project in order to include it. At the end, we came up with the idea of emulating a face recognition with only face detection and including also the g-sensor to detect what direction the camera is pointing on.

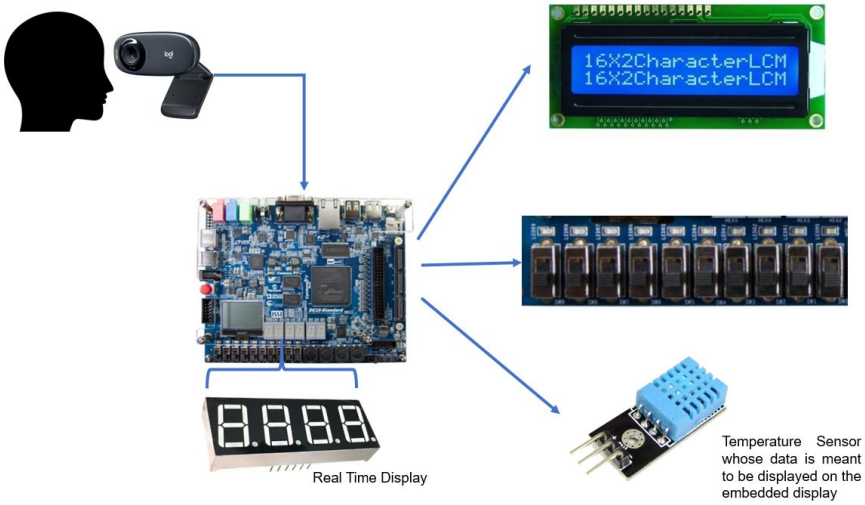

Figure 2. First idea of implementation.

a. System Block diagram

The following block diagram is based on the functionality of the project once the codes are up in the board. The camera, keyboard and mouse are connected to the DE10-Standard in order to use them to control the linux interfaz and run the camera program. These devices are connected via usb to the HPS. Inside the HPS we have connected the G-sensor, LCD and VGA. Also, all the codes related to the functionality of the camera and the devices listed before are on the HPS. The control of which LEDs are turned on and off and the 7-segment displays control are also on the HPS. The bridge between HPS and FPGA is made in order to use the LEDs and 7-segment displays which are connected to the g-sensor.

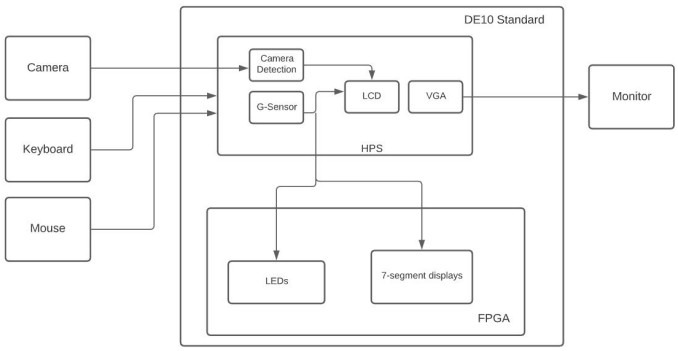

Figure 3. System Block Diagram.

b. Project Goals

The main goals for this project are:

1. Understand how a face can be detected.
2. Combine the LCD code with the face detection which will show when the face is being detected.
3. Understand the code of the g-sensor and implement it to show it in the LCD.
4. Implement the Y-coordinate on the 7-segment displays.
5. Understand the LEDs' code and adapt it to the Y-coordinate.

3. Theoretical Background

In this section it will be described the background theory of this project of every component that we needed to do some research about.

a. Accelerometer

The usage of accelerometers is very common nowadays, for example, when using the compass app on our smartphones, somehow, it knows which direction is pointing. Another common example is for the apps that showed the constellations on the sky, how do they know where you are pointing? The reason for this is that smartphones and many other technological devices have an accelerometer inside of them. Another common use for accelerometers are cars. In comparison with smartphones, cars only use a two-axis to determine the moment of impact. The sensitivity of both applications must be very high in order to do the measurements more easily.

An accelerometer (figure 4) is a small electromechanical device made up of axis-based motion sensing which is used to measure acceleration forces. The piezoelectric effect is the most common form of accelerometer which uses microscopic crystal structure which becomes stressed due to accelerative forces. After they are stressed, they create a voltage which can be determined as the velocity and orientation. The capacitance is another form of accelerometer which senses changes in capacitance between microstructures located next to the device. The changes on an accelerative force creates capacitance which creates voltages for interpretation. **[9]**

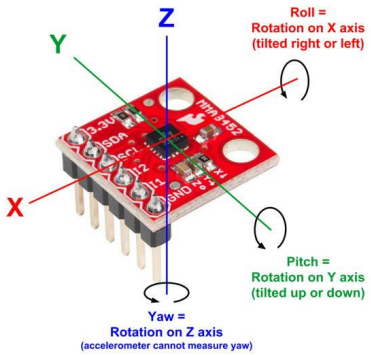

Figure 4. Accelerometer device.

The DE10-Standard comes with a digital accelerometer sensor module (ADXL345) which is commonly known as g-sensor. This sensor can only be managed through the HPS. It measures the acceleration in three directions which are referred to as x-axis, y-axis and z-axis. This accelerometer supports two serial communication: SPI and I^2C. This sensor is small, thin and extremely useful due its ultralow power assumption and high-resolution measurement. **[2][8]**

b. LEDs and 7-segment displays

It is extremely common to find LEDs in every technological device that we have in our day-to-day life. With them you can make any combination of colors you might need or want to do. They are also very helpful to manage the functionality of electronic devices since they are easy to see and visualize how a microcontroller is working. LEDs are the simplest thing to manage in electronics since they have only two wires: anode (positive) and cathode(negative). Usually, in the LEDs used in laboratories, the longer lead is the positive one and the other one is the negative (as shown in Figure 5) , but they can also be differentiated because they have a plain side which indicates where the negative lead is which can be shown in Figure 6. **[5]**

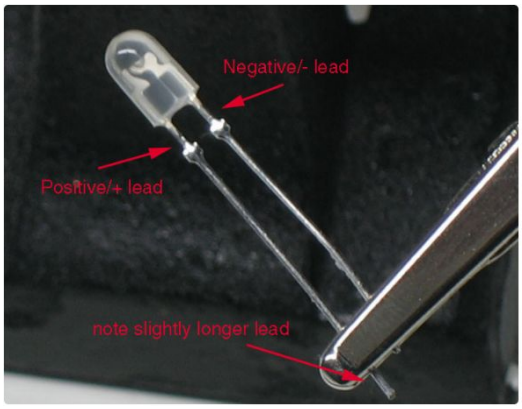

Figure 5. Common LED.

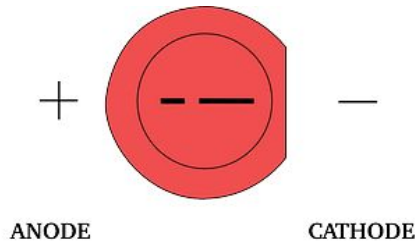

Figure 6. Down side of a LED.

Another component commonly used in electronics is the seven segment display where decimal numbers from 0 to 9 can be represented. This component usually uses LEDs or LCDs to light up the desired number. You can find them on many electronic devices such as: microwaves, ovens, calculators, radios, washing machines, and plenty more. It consists of 7

illuminated segments, as shown in Figure 8, that can be on at the same time to represent the desired number. There it is also the Decimal Point on them which can be light on to represent numbers with fractions. There are two types of 7 segment displays: common anode and common cathode. On the common anode all the negative terminals of the 8 LEDs are connected together which makes them power on with a high logic level. On the other hand, the common anode has all the positive terminals connected together, therefore each segment power on with a low logic level. **[1]**

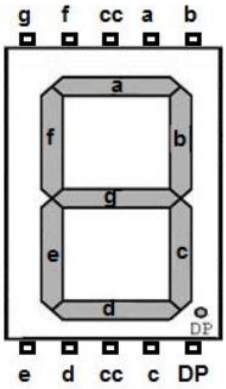

Figure 7. 7-segment display pinout.

The DE10-Standard counts with ten user-controllable LEDs and six 7- segment displays connected to the FPGA. Each of them are driven directly and individually by the Cyclone V SoC FPGA. The LEDs can be turned on using a high logic level and turned off using a low logic level. On the other side we have the 7-segments, where they all have common anode, therefore can be turned on using low logic level and turned off using high logic level. **[8]**

c. LCD

Most of the laptops, televisions, smartphones, or any technological device than we can encounter nowadays have a flat screen with LCDs. Before LCDs, these screens used to power-hungry beasts which demanded a lot of energy. This technology was frequently used on calculators and digital watches. This element is helpful to display different messages, images, or any illustration we want to. LCDs allowed that the displays can be thinner replacing CRT technology. They even replace LED and gas-display displays because they consume less power than them. **[13]**

An LCD is made up of millions of pixels and its number is commonly referred to as the quality (4K is made up of 4096x2160 pixels). Inside of each pixel there are three subpixels which are commonly called as RGB. With the combination of each subpixel, different colors can be produced, therefore it can make millions of different colors. These pixels are controlled (switched on and off) electronically while using liquid crystal to rotate polarized light. Another layer of the LCD is made of polarizing glass filter which is placed in front and behind all the pixels. LCDs can be made either with a passive matrix or an active matrix display grid. The active matrix LCD is also known as TFT display. The representation of the layers can be appreciated on figure 8. **[12]**

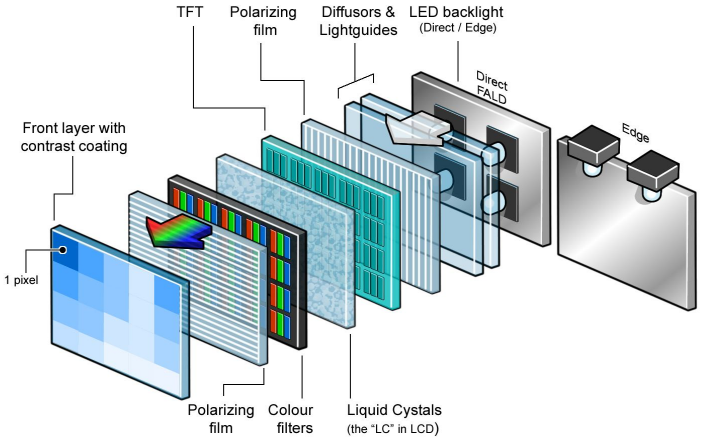

Figure 8. LCD layers.

The DE10-Standard board has an LCD module with 128x64 pixels. It supports two serial communication: SPI and I ^{2}C. This SPI is used in order to connect to the HPS. It is incorporated in order to display anything that the user may need. **[8]**

d. Face Detection Algorithm: LBPH Face Detection.

The face detection algorithms are methods that allow us to reach the process of face recognition. The face detection consists in the detection of regions in the skin map and this process is based on the Object detection process that uses the Haar feature-based cascade classifiers. This is an effective object detection method proposed by Paul Viola and Michael Jones in 2001, it is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. The basic algorithms of face and eye detection

needs a lot of positive images of faces and negative images without faces to train the classifier, then we need to extract features from it (some shown in Figure 9), for this, Haar feature is used where a single value obtained by subtracting sum of pixels under the white rectangle from sum of pixels under the black rectangle.

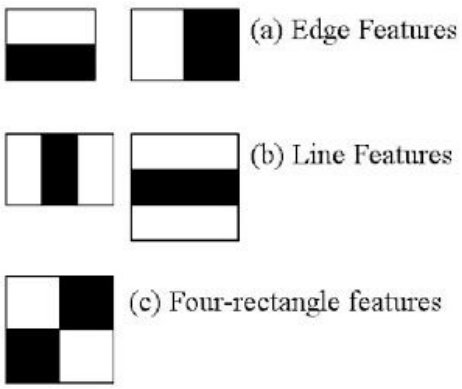

Figure 9. Types of features.

All possible sizes and locations of each kernel are used to calculate lots of features. For each feature calculation, it needs to find the sum of the pixels under white and black rectangles. To solve this, they introduced the integral image. However large your image, it reduces the calculations for a given pixel to an operation involving just four pixels. The concept of Cascade of Classifiers consist of applying the classifiers process on regions where there can be a face instead of applying it to more regions that there is no face by applying all the 6000 features on a window. This Cascade of Classifiers process grouped the features into different stages of classifiers and applied one-by-one, this allow us to apply the classifiers process to the first stage and if fails, the whole window is discarded, but if this stage passes it continue the process with the next stages, and if in a windows all stages are passed, this is a face region. **[4]**

The OpenCV library comes with a very new Face Recognizer class for face recognition, and it is currently available in three algorithms that are: Eigen Face Recognizer, Fisher Face Recognizer and Local Binary Patterns Histograms Face Recognizer. The Face Detection example code that comes with the OpenCV of the DE-10 Standard documentation consist in

the LBPH Face Detection with the Haar feature-based cascade classifiers, and it is a code released under the BSD licenses, therefore we are able to use it in our project.

To do face recognition, some face images are needed. Two options are available for this, either you create your own dataset or you can start with one of the availables face databases. The three examples of known databases are the AT&T FaceDatabase, the Yale FaceDatabase A and the Extended Yale FaceDatabase B; however the example code that OpenCV provides is a pretrained model.

The Local Binary Patterns methodology has its roots in 2D texture analysis. The basic idea of this method is to summarize the local structure in an image by comparing each pixel with its neighborhood. It takes a pixel as a center and threshold its neighbor against; if the intensity of the center pixel is greater-equal its neighbor, then denote it with 1 and 0 if not. It will end up with a binary number for each pixel like shown in Figure 10.A more formal description of the LBP operator in the algorithm of LBPH method can be given as shown in Equation 1.

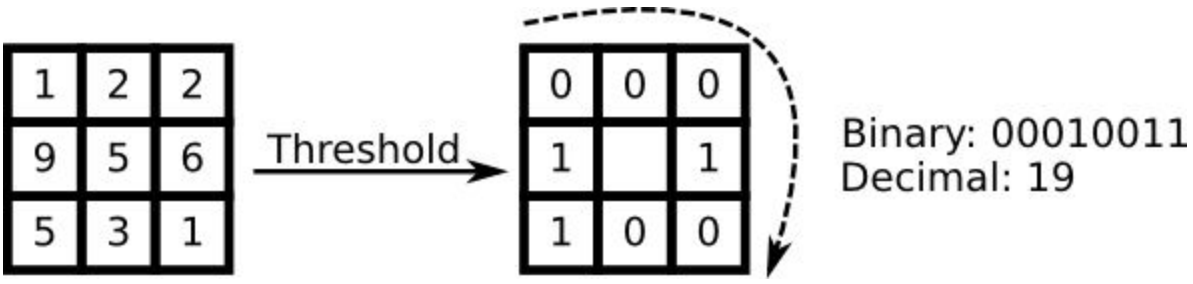

Figure 10. Example of the LBP methodology

$$LBP(x_c, y_c) = \sum_{p=0}^{P-1} 2^p s(i_p - i_c) \qquad \textbf{Eq. 1}$$

In Eq. 1. we consider the (x_c, y_c) as the central pixel with an intensity i_c, i_p being the intensity of the neighbor pixels and s is the sign function. This description enables you to capture very fine grained details in images, in fact the authors were able to compete with state of the art results for texture classification. Soon after the operator was published it was noted that a fixed neighborhood fails to encode details differing in scale. So the operator was extended to use a variable neighborhood and the idea is to align an arbitrary number

of neighbors on a circle(Figure 11) with a variable radius, which enables to capture the following neighborhoods:

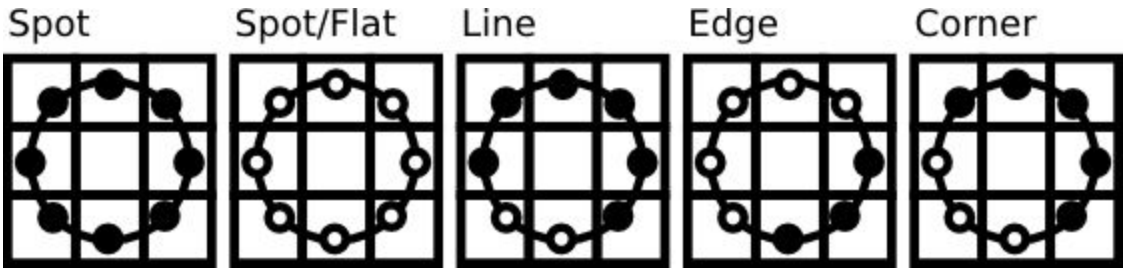

Figure 11. Types of the Circular LBP

The LBP operator is robust against monotonic gray scale transformations. We can easily verify this by looking at the LBP image of an artificially modified image. The representation proposed by Timo Ahonen is to divide the LBP image into m local regions and extract a histogram from each. The spatially enhanced feature vector is then obtained by concatenating the local histograms, and these histograms are what we have as the LBPH in this algorithm. **[9]**

This Algorithm comes in order to solve the problem that it comes with real life codes with some other algorithms. Some of them depend more of the numbers in images per person in the database and that it cannot guarantee a perfect light setting in the images of a person. On the other hand, if the number of images per person is just one, the covariance estimation for the subspace may be wrong and the other two algorithms will not recognize the face of the person. Extensive experiments clearly show the superiority of the proposed scheme over all considered methods on FERET tests which include testing the robustness of the method against different facial expressions, lighting and aging of the subjects. In addition to its efficiency, the simplicity of the proposed method allows for very fast feature extraction. **[3]**

4. System Requirements

This project implements two applications. The first one is based on face detection with a camera and the LCD where a message is being displayed. The second one is the implementation of a HPS-FPGA bridge that allows us to display in the seven segment the g-sensor value of the Y-coordinate. In this second application the 3 coordinates of the g-sensor are being shown on the LCD. To reach the whole object of this project the requirements are listed on the table 2.

State	ID	Name	Description	Importance level	Priority	Comments
Active	FR-01	Detection of elements external to the device.	The device must be able to detect the external devices that are connected to it, in this case it will be the camera, the mouse and the keyboard.	Mandatory	6	Not all cameras are compatible with the DE-10standard card.
Active	FR-02	Video capture	The device must be able to capture and display on the monitor the video received by the camera.	Mandatory	7	
Active	FR-03	Face detection	The system must be able to detect that a face is being captured by the camera.	Mandatory	8	
Inactive	FR-04	Face recognition	The system must be able to recognize the person of the face that is being detected	Deseable	9	It is unreachable due the code provided.

Inactive	FR-05	Internal connection in the HPS for face detection data	The system must have the necessary internal connections to use the LCD to display whether the face is being detected or not.	Deseable	10	Due all the face detection code is in c++, we were unable to convert it because of lack of time.
Inactive	FR-06	Face Detection LCD Message Display	The device must be able to display a message on the LCD of whether a face was detected or not.	Deseable	11	
Active	FR-07	Coordinate capture through the board's g-sensor	The device must be able to receive the data when using the g-sensor and read through the hps.	Mandatory	1	
Active	FR-08	Display the g-sensor’s coordinates on the LCD.	The device must be able to display the three coordinates given by the g-sensor on the LCD.	Mandatory	2	
Active	FR-09	Connection between FPGA and HPS for g-sensor data	The system must be able to have the necessary internal connections to connect the HPS with the 7-segment display and the LEDs connected to the FPGA.	Mandatory	4	
Active	FR-10	Show Y-coordinate values in the 7 segment displays	The device must be able to display the Y-coordinates on the 7-segment displays.	Mandatory	5	

Active	FR-11	Activation of the LEDs according to the Y-coordinates values	The device must be able to change the led that is on, depending on the value of the Y coordinate. The value considered as the middle point, will light the central LEDs	Mandatory	3	

Table 2 Project Requirements.

5. Design Description

This project was designed based on an initial proposal that we made, which with time was reviewed and adapted to the possibilities and times that we had to complete this project. The design was done understanding how the g-sensor (accelerometer) and the OpenCV face_detection code works. We decided to use some of the output elements that we put in the initial proposal, but now to display the information that we received from the input devices that we chose. We took the g-sensor and the camera as our input devices. Our output devices are the LCD, the LEDs and the 7-segment displays. A VGA monitor, a mouse and a keyboard were needed in order to run the face detection program. The connection between this element was done with the help of the qsys tool, although this greatly facilitates the process of developing the design, it was necessary to have the basic and key knowledge of each element to be used, so that we really have the certainty that we use this tool correctly.

We began the project implementation by checking the demos provided by the System CD. The g-sensor and one project related to the use of LEDR were available so we continued by building the necessary C program based on those examples. Since we had already worked with the GHDR project also provided by the System CD we decided to use it for this first step because it had the LEDRs already instantiated and ready to go. It represented no bigger issue and we faced no trouble while implementing this. Since we wanted first to display the coordinates on to the LCD, nothing was done on the FPGA side besides what was already done. The bigger deal began once we decided to display the Y coordinate on the 7-segment display because we had to do some changes in the qsys file in order to assign the new components a base address and also generate the necessary header files.

Six PIOs were added with the help of the Qsys tool as Figure 12 shows, each with 7 bits representing the 7 segments that built a character. Once added to the System Contents we connected the clk, rest and s1 each to the main clock, the clock reset and to mm_bridge and fpga_master modules respectively. After that we generated the header file where all the connections between the FPGA and the HPS are being declared. This header was made as shown in Figure 13 and then moved to our HPS folder in order to use it on the C code. We also needed to connected the inputs

for the 7-segments displays (seg 0 to seg 5) to the real 7-segments display as shown in Figure 14. Figure 15 shows the address map generated by Qsys.

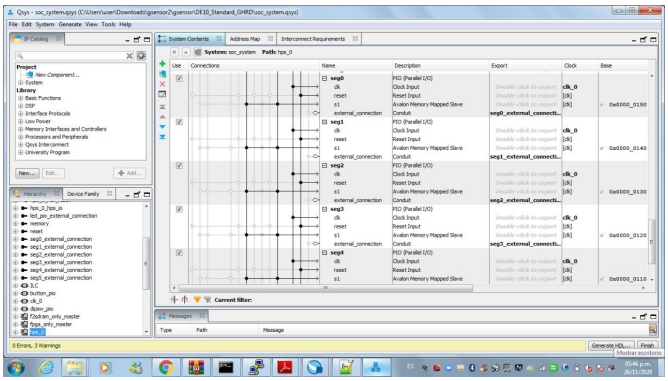

Figure 12. Create the 7-segments displays on Qsys.

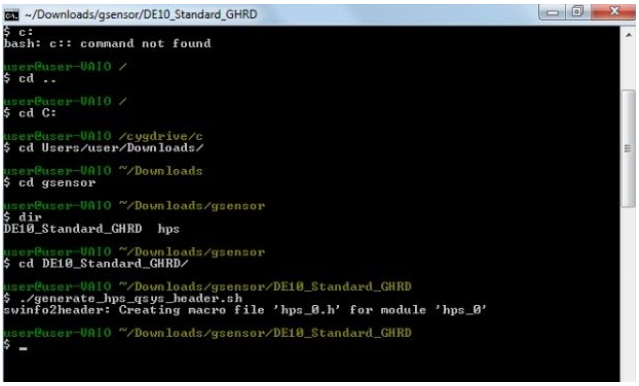

Figure 13. Generate the header file.

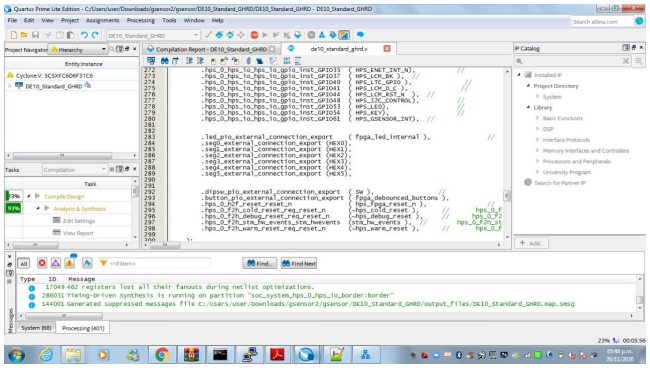

Figure 14. Mapping the inputs of the file with the 7-segments displays.

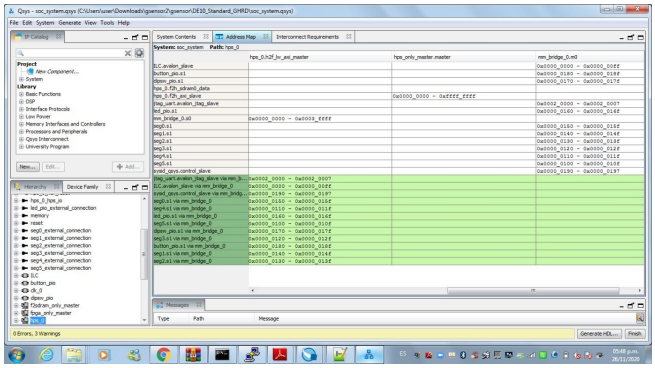

Figure 15. Qsys Address Map.

a. Intricacies of the of the hardware architecture

The main intricacies in this project were when we needed to implement the FPGA-HPS bridge and to handle the obtained data to display it on the correct output devices that were: the LCD screen, the 7segments display and the LEDs. Something that had to be rethought in the process of carrying out the project was specifically the way we would obtain the data of whether a face was detected or not. Even though the connection and the concept of how to implement were known, its implementation was complicated by the language in which the OpenCV codes were.

Despite the project being much more complex than any previous activity or laboratory carried out, the fact of using the Qsys tool that is available in quartus facilitates a large part of the process of developing it. Although we did not achieve the entire objective of the project proposal, we fully understand how the development of internal connections is generated (Figure 16) for this and other applications as well as the bridge that was generated between FPGA and HPS to reach more complex applications.

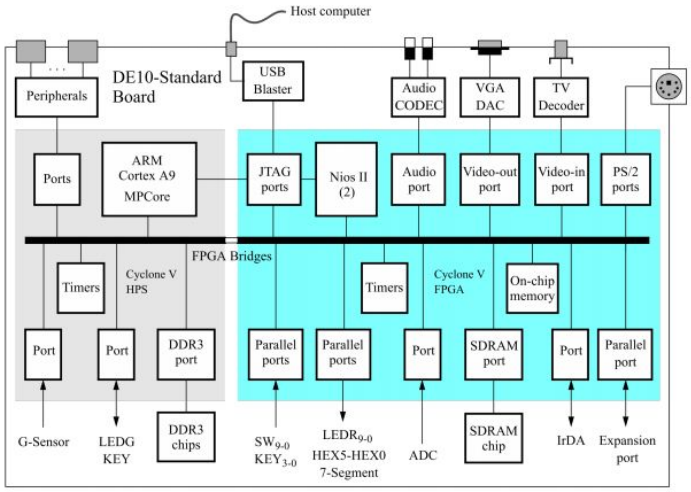

Figure 16. FPGA and HPS bridge on a DE10-Standard Board.

b. Hardware architecture at signal level

Figure 17 shows our result in Quartus for our Hardware architecture at signal level.

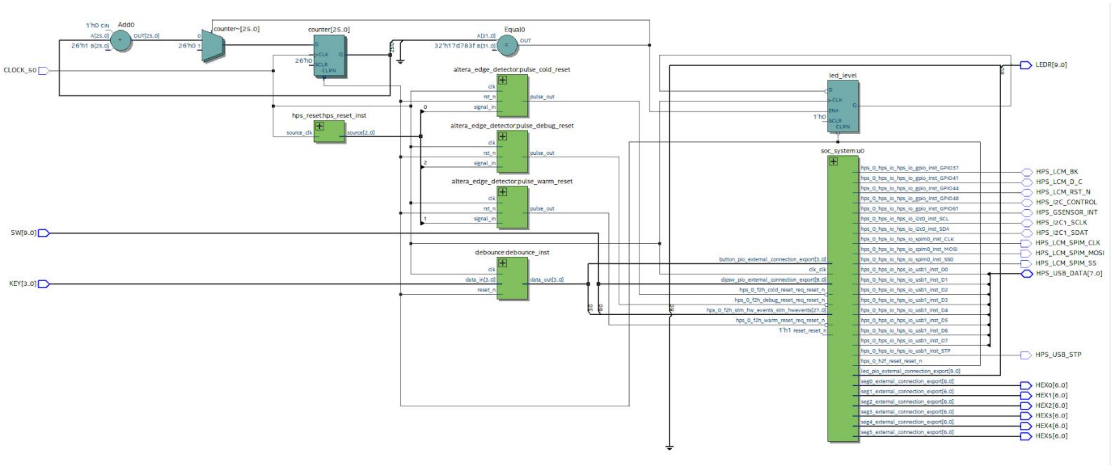

Figure 17. Hardware architecture at signal level.

c. Hardware Inputs / Outputs

The necessary elements in this project for a correct communication with the external world, and to have our input signal and where we send our output signals are:

- Monitor
- Camera
- Keyboard
- Mouse
- LCD screen
- 7 segments display
- Leds

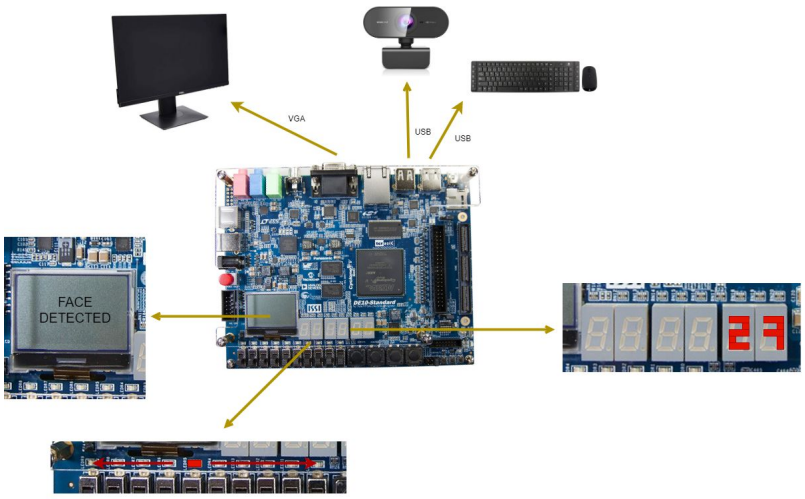

Figure 18. Inputs and outputs of the project.

d. Code for LCD, g-sensor, LEDs and 7-segments displays.

For the implementation of this project we program our HPS with a C code. This code is a combination of some demos found on the DE10-Standard Standard CD-ROM being adapted to our project.

The first part of it is to declare all the requiere libraries shown in Figure 19 which are the ones used in all the demos and the one required in order to use the LEDs and 7-segments

displays. In the next part of code shown in Figure 20 are all the declarations of constants that are needed in the following parts. From the Line 75 to the Line 101 in Figure 21 are two functions made to do the initial configuration and to read the data from the g-sensor. The following function shown in Figure 22 was made to return the hexadecimal value of which segment is turning on in each 7-segment display and the last return is just in case some bad information was entered. The Figure 23 and Figure 24 are all the variables needed in the main code.

```
#include "terasic_os_includes.h"
#include "LCD_Lib.h"
#include "lcd_graphic.h"
#include "font.h"
#include <errno.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <linux/i2c-dev.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "hwlib.h"
#include "ADXL345.h"
#include "ADXL345.c"
#include <sys/mman.h>
#include "socal/socal.h"
#include "socal/hps.h"
#include "socal/alt_gpio.h"
#include "hps_0.h"              //declaration of the LEDs and 7 segments displays
```

Figure 19. Inclusion of libraries at the main code.

```
#define HEX_DISPLAY_CLEAR (0x7F)
#define HEX_DISPLAY_ZERO (0x40)
#define HEX_DISPLAY_ONE (0x79)
#define HEX_DISPLAY_TWO (0x24)
#define HEX_DISPLAY_THREE (0x30)
#define HEX_DISPLAY_FOUR (0x19)
#define HEX_DISPLAY_FIVE (0x12)
#define HEX_DISPLAY_SIX (0x02)
#define HEX_DISPLAY_SEVEN (0x78)
#define HEX_DISPLAY_EIGHT (0x00)
#define HEX_DISPLAY_NINE (0x18)
#define HEX_DISPLAY_A (0x08)
#define HEX_DISPLAY_B (0x03)
#define HEX_DISPLAY_C (0x46)
#define HEX_DISPLAY_D (0x21)

#define HW_REGS_BASE ( ALT_STM_OFST )
#define HW_REGS_SPAN ( 0x04000000 )
#define HW_REGS_MASK ( HW_REGS_SPAN - 1 )
```

Figure 20. Declaration of some constants.

```
bool ADXL345_REG_WRITE(int file, uint8_t address, uint8_t value){
  bool bSuccess = false;
  uint8_t szValue[2];

  // write to define register
  szValue[0] = address;
  szValue[1] = value;
  if (write(file, &szValue, sizeof(szValue)) == sizeof(szValue)){
      bSuccess = true;
  }
  return bSuccess;
}

bool ADXL345_REG_READ(int file, uint8_t address,uint8_t *value){
  bool bSuccess = false;
  uint8_t Value;
  // write to define register
  if (write(file, &address, sizeof(address)) == sizeof(address)){

    // read back value
    if (read(file, &Value, sizeof(Value)) == sizeof(Value)){
      *value = Value;
      bSuccess = true;
    }
  }
  return bSuccess;
}
```

Figure 21. Initial configuration of g-sensor.

```
int hex_num(int num){
    switch(num){
      case 0: return HEX_DISPLAY_ZERO;
      case 1: return HEX_DISPLAY_ONE;
      case 2: return HEX_DISPLAY_TWO;
      case 3: return HEX_DISPLAY_THREE;
      case 4: return HEX_DISPLAY_FOUR;
      case 5: return HEX_DISPLAY_FIVE;
      case 6: return HEX_DISPLAY_SIX;
      case 7: return HEX_DISPLAY_SEVEN;
      case 8: return HEX_DISPLAY_EIGHT;
      case 9: return HEX_DISPLAY_NINE;
      return HEX_DISPLAY_CLEAR;
    }
}
```

Figure 22. Function used to identify the number to be displayed on the 7-segments.

```
int main(int argc, char *argv[]) {

  void *virtual_base;
  int fd;
  int X;
  int Y_c;
  int Z;

  char cad_X[5];
  char cad_Y[3];
  char cad_Z[3];

  int cad_Y_1;
  int cad_Y_2;
  int cad_Y_3;
  int cad_Y_4;

  LCD_CANVAS LcdCanvas;
  int file;
  const char *filename = "/dev/i2c-0";
  uint8_t id;
  bool bSuccess;
  const int mg_per_digi = 4;
  uint16_t szXYZ[3];
  int cnt=0, max_cnt=0;
```

Figure 23. Variable declarations.

```
int cnt=0, max_cnt=0;
int led_mask;
int seg0_mask;
int seg1_mask;
int seg2_mask;
int seg3_mask;
int seg4_mask;
int seg5_mask;

void *h2p_lw_led_addr;
void *h2p_lw_seg0_addr;
void *h2p_lw_seg1_addr;
void *h2p_lw_seg2_addr;
void *h2p_lw_seg3_addr;
void *h2p_lw_seg4_addr;
void *h2p_lw_seg5_addr;
```

Figure 24. Variable declarations.

The following code shown in Figure 25 is handling some errors that can occured opening files or buses. In Figure 26 the initialization and read of the g-sensor is shown. In Figure 27 shows how the address of each 7-segment display and each LED are handled to use them later on. The next part shown in Figure 28 the LCD is initialized and handles the error in case there is no frame buffer.

```
printf("===== gsensor test =====\r\n");
if (argc == 2){
  max_cnt = atoi(argv[1]);
}
if( ( fd = open( "/dev/mem", ( O_RDWR | O_SYNC ) ) ) == -1 ) {
  printf( "ERROR: could not open \"/dev/mem\"...\n" );
  return( 1 );
}

virtual_base = mmap( NULL, HW_REGS_SPAN, ( PROT_READ | PROT_WRITE ), MAP_SHARED, fd, HW_REGS_BASE );

if( virtual_base == MAP_FAILED ) {
  printf( "ERROR: mmap() failed...\n" );
  close( fd );
  return( 1 );
}
// open bus
if ((file = open(filename, O_RDWR)) < 0) {
    /* ERROR HANDLING: you can check errno to see what went wrong */
    perror("Failed to open the i2c bus of gsensor");
    exit(1);
}
```

Figure 25. Error handlers.

```
// init
// gsensor i2c address: 101_0011
int addr = 0b01010011;
if (ioctl(file, I2C_SLAVE, addr) < 0) {
    printf("Failed to acquire bus access and/or talk to slave.\n");
    /* ERROR HANDLING; you can check errno to see what went wrong */
    exit(1);
}
  // configure accelerometer as +-2g and start measure
  bSuccess = ADXL345_Init(file);
  if (bSuccess){
      // dump chip id
      bSuccess = ADXL345_IdRead(file, &id);
      if (bSuccess)
          printf("id=%02Xh\r\n", id);
  }
```

Figure 26. Reading g-sensor.

```
h2p_lw_led_addr =virtual_base + ( ( unsigned long  )( ALT_LWFPGASLVS_OFST + LED_PIO_BASE ) & ( unsigned long)( HW_REGS_MASK ) );
h2p_lw_seg0_addr= virtual_base +( ( unsigned long  )(ALT_LWFPGASLVS_OFST + SEG0_BASE) );
h2p_lw_seg1_addr= virtual_base +( ( unsigned long  )(ALT_LWFPGASLVS_OFST + SEG1_BASE) );
h2p_lw_seg2_addr= virtual_base +( ( unsigned long  )(ALT_LWFPGASLVS_OFST + SEG2_BASE) );
h2p_lw_seg3_addr= virtual_base +( ( unsigned long  )(ALT_LWFPGASLVS_OFST + SEG3_BASE) );
h2p_lw_seg4_addr= virtual_base +( ( unsigned long  )(ALT_LWFPGASLVS_OFST + SEG4_BASE) );
h2p_lw_seg5_addr= virtual_base +( ( unsigned long  )(ALT_LWFPGASLVS_OFST + SEG5_BASE) );

led_mask = 0x000;
```

Figure 27. Assigning addresses.

```
LcdCanvas.Width = LCD_WIDTH;
LcdCanvas.Height = LCD_HEIGHT;
LcdCanvas.BitPerPixel = 1;
LcdCanvas.FrameSize = LcdCanvas.Width * LcdCanvas.Height / 8;
LcdCanvas.pFrame = (void *)malloc(LcdCanvas.FrameSize);

if (LcdCanvas.pFrame == NULL){
    printf("failed to allocate lcd frame buffer\r\n");
```

Figure 28. LCD initialization.

The next section is where the “real code” begins. Before this, all the code was either for initialization, declaration or handling some errors that can occur. In this section is where we are reading from the g-sensor, writing on the LCD, turning on the LEDs, and displaying the Y-coordinate on the 7-segments displays. The else shown in Figure 29 is used to do it all if the lcd frame buffer is available. It initializes the LCD with our virtua base and turns on the LCD backlight. The while statement in the same figure mentioned before is looping when the buffer still exists and the count on each reading is not more than what the HPS can handle. In this loop, first it clears the screen, and then clear all the 7-segments displays and turn off all the leds. The next if is to check if the g-sensor data is ready to be read. Inside of this is where the buffer is updating and the count in each cycle is being made. In Figure 30 shows how to obtain the X, Y and Z coordinates. These coordinates are printing on the UART to keep control of them. After this the DRAW_PrintString is called which updates the internal frame with all the messages wanted to print on the LCD, in this case, we are printing the coordinates. After this the DRAW_Refresh function is being called in order to update the LCD with the desired data.

We are using the LED1 to LED9, therefore there are 9 LEDs being used, we took the middle in LED 5, we arrange some parameters to turn on the LEDs depending on where the Y-coordinates is changing as shown in Figures 31 and 32. The next step is turning on the 7-segments displays. The first thing shown in Figure 33 that we did was converting each number to a one digit number. For example, if the Y-coordinate is given 1524, in order to represent that one we needed in an one digit integer, therefore, we divide that number by 1000, taking account that the biggest number is not more that the limit of 9999. For that 5, we took the original number, rest what we obtained in the step before and then divide this by 100. In this example will be cad_Y_3 = (1524 - 1 * 10000)/100 = (524)/100. We did the same for the following two digits. After we have all the one digit numbers of our original number, we did some if statements because we did not wanted to turn a 7-segments display in 0 if it is not relevant, for example, if the Y-coordinate is 26, we did not wanted that the 7-segments display will turn on as 0 0 2 6. In case the Y-coordinate was negative, we did another else if statement where we convert the number to positive and implement the same as it was positive but we activate the display 5 with a minus symbol, this can be shown in

Figure 34. At the end of Figure 35 it shows how the mask for the 7-segments displays and the led mask is updating their addresses. The function usleep is made to delay. Tha last part (Figure 36) of the code is to close everything and clean the LCD.

```
}else{

  LCDHW_Init(virtual_base);
  LCDHW_BackLight(true); // turn on LCD backlight

  LCD_Init();

  while(bSuccess && (max_cnt == 0 || cnt < max_cnt)){
      DRAW_Clear(&LcdCanvas, LCD_WHITE); // clear screen
      seg0_mask = HEX_DISPLAY_CLEAR;
      seg1_mask= HEX_DISPLAY_CLEAR;
      seg2_mask= HEX_DISPLAY_CLEAR;
      seg3_mask= HEX_DISPLAY_CLEAR;
      seg4_mask= HEX_DISPLAY_CLEAR;
      seg5_mask= HEX_DISPLAY_CLEAR;

      if (ADXL345_IsDataReady(file)){
          bSuccess = ADXL345_XYZ_Read(file, szXYZ);
          if (bSuccess){
              cnt++;
```

Figure 29. Starting loop as long as g-sensor exists.

```
X = (int16_t)szXYZ[0]*mg_per_digi;
Y_c = (int16_t)szXYZ[1]*mg_per_digi;
Z = (int16_t)szXYZ[2]*mg_per_digi;
sprintf(cad_X, "%d", X);
sprintf(cad_Y, "%d", Y_c);
sprintf(cad_Z, "%d", Z);
printf("[%d]X=%f mg, Y=%f mg, Z=%f mg\r\n", cnt,X, Y_c, Z);

DRAW_PrintString(&LcdCanvas, 15, 5, "X=", LCD_BLACK, &font_16x16);
DRAW_PrintString(&LcdCanvas, 40, 5, cad_X, LCD_BLACK, &font_16x16);
DRAW_PrintString(&LcdCanvas, 15, 5+16, "Y=", LCD_BLACK, &font_16x16);
DRAW_PrintString(&LcdCanvas, 40, 5+16, cad_Y, LCD_BLACK, &font_16x16);
DRAW_PrintString(&LcdCanvas, 15, 5+32, "Z=", LCD_BLACK, &font_16x16);
DRAW_PrintString(&LcdCanvas, 40, 5+32, cad_Z, LCD_BLACK, &font_16x16);
DRAW_Refresh(&LcdCanvas);
```

Figure 30. Reading and displaying on the LCD the coordinates.

```
if(Y_c<-770){
  led_mask = 0x100; //LED 9
}
else if(-770<=Y_c && Y_c<-550){
  led_mask = 0x80; //LED 8
}
else if(-550<=Y_c && Y_c<-330){
  led_mask = 0x40; //LED 7
}
else if(-330<=Y_c && Y_c<-110){
  led_mask = 0x20; //LED 6
}
else if(-110<=Y_c && Y_c<110){
  led_mask = 0x10; //LED 5
}
```

Figure 31. Turning on LEDs according to the Y-coordinate.

```
else if(110<=Y_c && Y_c<330){
  led_mask = 0x08; //LED 4
}
else if(330<=Y_c && Y_c<550){
  led_mask = 0x04; //LED 3
}
else if(550<=Y_c && Y_c<771){
  led_mask = 0x02; //LED 2
}
else{
  led_mask = 0x01; //LED 1
}
```

Figure 32. Turning on LEDs according to the Y-coordinate.

```
cad_Y_4 = Y_c/1000;
cad_Y_3 = (Y_c - cad_Y_4*1000)/100;
cad_Y_2 = (Y_c - cad_Y_4*1000 - cad_Y_3*100 )/10;
cad_Y_1 = (Y_c - cad_Y_4*1000 - cad_Y_3*100 - cad_Y_2*10);
if(cad_Y_4 > 0 ){
  seg4_mask= hex_num(cad_Y_4);
  seg3_mask= hex_num(cad_Y_3);
  seg2_mask= hex_num(cad_Y_2);
  seg1_mask= hex_num(cad_Y_1);
}
else if(cad_Y_3 > 0){
  seg3_mask= hex_num(cad_Y_3);
  seg2_mask= hex_num(cad_Y_2);
  seg1_mask= hex_num(cad_Y_1);
}
else if (cad_Y_2 > 0){
  seg2_mask= hex_num(cad_Y_2);
  seg1_mask= hex_num(cad_Y_1);
}
else if (cad_Y_1 >0){
  seg1_mask= hex_num(cad_Y_1);
}
```

Figure 33. Y-coordinate positive shown in 7-segments.

```
else if(Y_c < 0 ){
  Y_c = Y_c * -1;
  cad_Y_4 = Y_c/1000;
  cad_Y_3 = (Y_c - cad_Y_4*1000)/100;
  cad_Y_2 = (Y_c - cad_Y_4*1000 - cad_Y_3*100 )/10;
  cad_Y_1 = (Y_c - cad_Y_4*1000 - cad_Y_3*100 - cad_Y_2*10);
  seg5_mask= 0x3F ;
  if(cad_Y_4 > 0 ){
    seg4_mask= hex_num(cad_Y_4);
    seg3_mask= hex_num(cad_Y_3);
    seg2_mask= hex_num(cad_Y_2);
    seg1_mask= hex_num(cad_Y_1);
  }
  else if(cad_Y_3 > 0){
    seg3_mask= hex_num(cad_Y_3);
    seg2_mask= hex_num(cad_Y_2);
    seg1_mask= hex_num(cad_Y_1);
  }
```

Figure 34. Y-coordinate negative shown in 7-segments.

```
                else if (cad_Y_2 > 0){
                  seg2_mask= hex_num(cad_Y_2);
                  seg1_mask= hex_num(cad_Y_1);
                }
                else if (cad_Y_1 >0){
                  seg1_mask= hex_num(cad_Y_1);
                }
              }
              *(uint32_t *)h2p_lw_led_addr = led_mask;
              *(uint32_t *)h2p_lw_seg0_addr = seg0_mask;
              *(uint32_t *)h2p_lw_seg1_addr = seg1_mask;
              *(uint32_t *)h2p_lw_seg2_addr = seg2_mask;
              *(uint32_t *)h2p_lw_seg3_addr = seg3_mask;
              *(uint32_t *)h2p_lw_seg4_addr = seg4_mask;
              *(uint32_t *)h2p_lw_seg5_addr = seg5_mask;
              usleep(1000*1000);

          }
      }
  }

  free(LcdCanvas.pFrame);
```

Figure 35. Updating addresses.

```
    free(LcdCanvas.pFrame);
  }

  close( fd );

  return( 0 );
}
```

Figure 36. Close and clear all.

e. OpenCV Face Detection Code.

For the implementation of this project we program our HPS with a C code. The code required for Face Detection can be found on the Linux interface that comes with the board. To access this code, a vga monitor, keyboard and a mouse is required in order to use the interface. On the desktop of this interface it shows the OpenCv folder which contains all the demos for face detection. This code located on the facedetection folder is on C++.

From the Line 6 to the Line 26, shown in Figure 37, there are all the declarations of global variables and inclusions of libraries required to this code. The next part shown in Figure 38 and 39 is the function used to display on the monitor the eyes on 2 circles and the face on an ellipse. Figure 40 shows the beginning of the main code which handles some errors that can occured by loading the libraries. The last part of the main includes a while that enters in a loop as long as the camera keeps transmitting something shown in Figure 41.

```
6  #include "opencv2/objdetect.hpp"
7  #include "opencv2/videoio.hpp"
8  #include "opencv2/highgui.hpp"
9  #include "opencv2/imgproc.hpp"
10 #include <iostream>
11 #include <thread>
12 #include <mutex>
13 #include <sys/time.h>
14
15 using namespace std;
16 using namespace cv;
17
18 /** Function Headers */
19 void detectAndDisplay( Mat frame );
20
21 /** Global variables */
22 String face_cascade_name = "lbpcascade_frontalface.xml";
23 String eyes_cascade_name = "haarcascade_eye_tree_eyeglasses.xml";
24 CascadeClassifier face_cascade;
25 CascadeClassifier eyes_cascade;
26 String window_name = "Capture - Face detection";
```

Figure 37. Inclusion of libraries.

```
void detectAndDisplay( Mat frame )
{
    std::vector<Rect> faces;
    Mat frame_gray;

    cvtColor( frame, frame_gray, COLOR_BGR2GRAY );
    equalizeHist( frame_gray, frame_gray );

    //-- Detect faces
    face_cascade.detectMultiScale( frame_gray, faces, 1.1, 2, 0, Size(80, 80) );

    for( size_t i = 0; i < faces.size(); i++ )
    {
        Mat faceROI = frame_gray( faces[i] );
        std::vector<Rect> eyes;

        //-- In each face, detect eyes
        eyes_cascade.detectMultiScale( faceROI, eyes, 1.1, 2, 0 |CASCADE_SCALE_IMAGE, Size(30, 30) );
        if( eyes.size() == 2)
        {
            //-- Draw the face
            Point center( faces[i].x + faces[i].width/2, faces[i].y + faces[i].height/2 );
            ellipse( frame, center, Size( faces[i].width/2, faces[i].height/2 ), 0, 0, 360, Scalar( 255, 0, 0 ), 2, 8, 0 );
```

Figure 38. Display the image on the monitor.

```
        for( size_t j = 0; j < eyes.size(); j++ )
        { //-- Draw the eyes
            Point eye_center( faces[i].x + eyes[j].x + eyes[j].width/2, faces[i].y + eyes[j].y + eyes[j].height/2 );
            int radius = cvRound( (eyes[j].width + eyes[j].height)*0.25 );
            circle( frame, eye_center, radius, Scalar( 255, 0, 255 ), 3, 8, 0 );
        }
    }
}

char str[100];
static struct timeval last_time;
struct timeval current_time;
static float last_fps;
float t;
float fps;
gettimeofday(&current_time, NULL);
t = (current_time.tv_sec - last_time.tv_sec) + (current_time.tv_usec - last_time.tv_usec) / 1000000.;
fps = 1. / t;
fps = last_fps * 0.8 + fps * 0.2;
last_fps = fps;
last_time = current_time;
sprintf(str, "%2.2f", fps);
//cout << str << endl;
putText(frame, str, Point(20, 40), FONT_HERSHEY_DUPLEX, 1, Scalar(0, 0, 255));
//-- Show what you got
imshow( window_name, frame );
```

Figure 39. Display the image on the monitor.

```
int main( void )
{
    VideoCapture capture;
    Mat frame;

    //-- 1. Load the cascade
    if( !face_cascade.load( face_cascade_name ) ){
      printf("--(!)Error loading face cascade\n"); return -1;
    }
    if( !eyes_cascade.load( eyes_cascade_name ) ){
      printf("--(!)Error loading eyes cascade\n"); return -1;
    }

    //-- 2. Read the video stream
    capture.open( -1 );
    if ( ! capture.isOpened() ) {
      printf("--(!)Error opening video capture\n"); return -1;
    }
```

Figure 40. Declaration of main variables.

```
    while ( capture.read(frame) )
    {
        if( frame.empty() )
        {
            printf(" --(!) No captured frame -- Break!");
            break;
        }

        //-- 3. Apply the classifier to the frame
        detectAndDisplay( frame );

        //-- bail out if escape was pressed
        int c = waitKey(10);
        if( (char)c == 27 ) { break; }
    }
    return 0;
}
```

Figure 41. Loop to display the image made by the camera.

6. Tests and Results

In this section three important aspects will be described when carrying out this project, the first aspect of these consists of the strategy used to carry out the tests of the project, once the strategy has been defined in a second section, the cases that will be taken into account will be explained when carry out the tests. In this it is also defined what is the expected result in each of the cases and then there will be a final section, which consists of the results that were obtained when applying each of the tests.

a. Test Strategy

As the overall project is based on the functionality of the g-sensor embedded in our board, the first test strategy was to send any data retrieved from it to our terminal so we could so the type of data we would be getting for future purposes. Once we checked this, we knew what the format was like so we proceeded to work around the lcd which was our initial goal. Once we got this, the x, y and z coordinates were being displayed through the LCD and to corroborate this data this was also still being sent to the terminal so we could be sure it was the correct data.

Once we got this, it was time for us to work with the LEDR embedded on the board. Since the Y coordinate was chosen to be the one for reference, the LEDR would be lightning up according to the position of this starting from the LEDR in the middle and to the right if it was a coordinate with a value below -111 and to the left it was above 110. This was tested by tilting the board so the Y coordinate would get affected and once we checked the 9 ranges were matching with the assigned LEDR we were done with this part.

The last step added was the Y coordinate to be displayed on to the 7 segments displays available. This was easy to test since we had two ways of corroborating this: the LCD and the terminal. As soon as this was implemented and ready to go this was tested once again by tilting the FPGA board and checking if the data being displayed matched the data we were getting on the terminal and both the LCD and LEDs were still working as they did before.

b. Test Cases

There are basically 9 different cases that correspond each to one of the LEDR we used and are explained below:

Case (Range of Y coordinate value)	What's expected?
Y < -770	● Light up LEDR1 ● Display Y coordinate on the 7 segment, terminal and LCD ● Display the - symbol on the first 7 segment available on the board
-770 ≤Y < 550	● Light up LEDR2 ● Display Y coordinate on the 7 segment, terminal and LCD ● Display the - symbol on the first 7 segment available on the board
-550 ≤ Y < -330	● Light up LEDR3 ● Display Y coordinate on the 7 segment, terminal and LCD ● Display the - symbol on the first 7 segment available on the board
-330 ≤ Y < -110	● Light up LEDR4 ● Display Y coordinate on the 7 segment, terminal and LCD ● Display the - symbol on the first 7 segment available on the board
-110 ≤Y < 110	● Light up LEDR5 ● Display Y coordinate on the 7 segment, terminal and LCD ● If value is below 0, use the first segment available to display the - symbol
110 ≤ Y < 330	● Light up LEDR6 ● Display Y coordinate on the 7 segment, terminal and LCD
330 ≤ Y < 550	● Light up LEDR7 ● Display Y coordinate on the 7 segment, terminal and LCD
550 ≤ Y < 770	● Light up LEDR8 ● Display Y coordinate on the 7 segment, terminal and LCD
770 < Y	● Light up LEDR9 ● Display Y coordinate on the 7 segment, terminal and LCD

Table 3. Test cases.

c. Test Results

Figure 42 shows how the components are where there is no code running. The first test result that we had was to test the g-sensor where the coordinates are shown only on the terminal as shown in Figure 43. The next step was to turn on the LCD, clear the 7-segments displays and turn on all the LEDs to verify they have a correct configuration and this is shown in Figure 44. After that, we did the changes in order to program all the components as we wanted to. The system worked as expected: the g-sensor sent the three coordinates and these were displayed onto the LCD. Because of hardware limitations, only the Y coordinate data was sent to the 7 segment displays as shown in Figures 45 and 46. The LEDR followed the Y coordinate values as expected (Figures 45 and 46) and this data was also shown on the terminal as shown in Figure 43. The camera detected the face and displayed 2 circles around the eyes and one around the face this is shown on Figure 47.

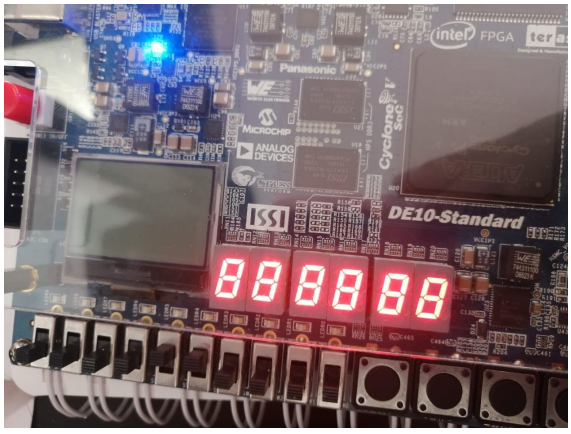

Figure 42. HPS configuration with no running program.

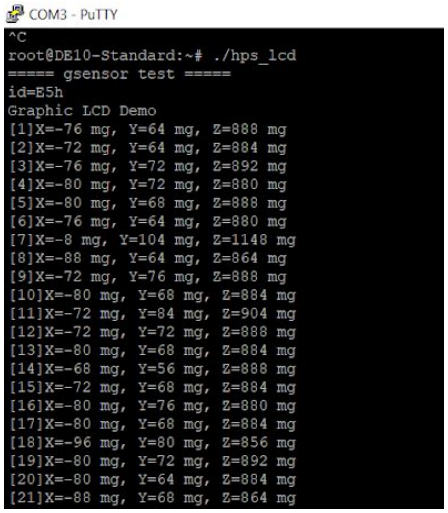

Figure 43. Terminal of the board showing g-sensor coordinates.

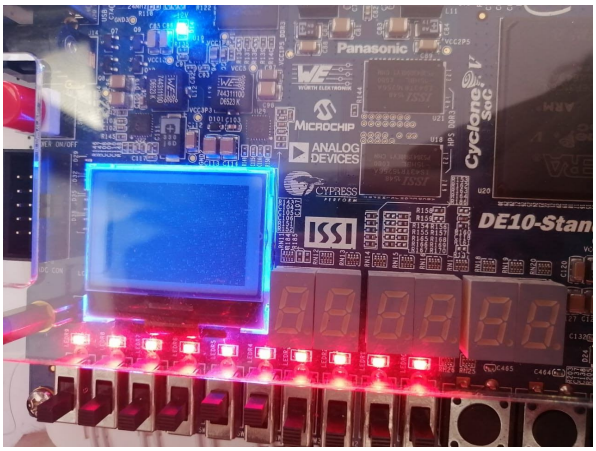

Figure 44. Initialization of the components.

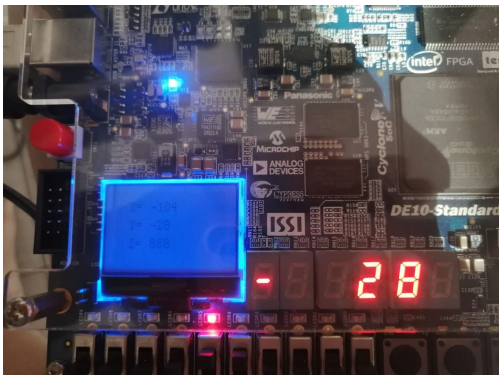

Figure 45. Negative Y-coordinate detected.

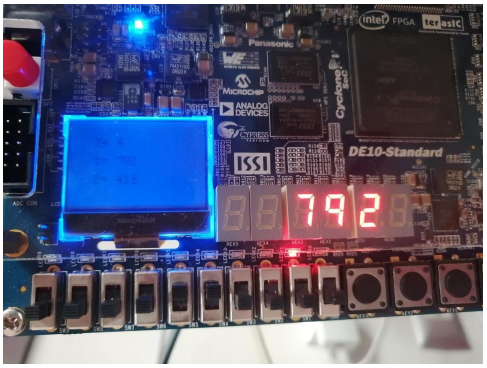

Figure 46. Positive Y-coordinate detected.

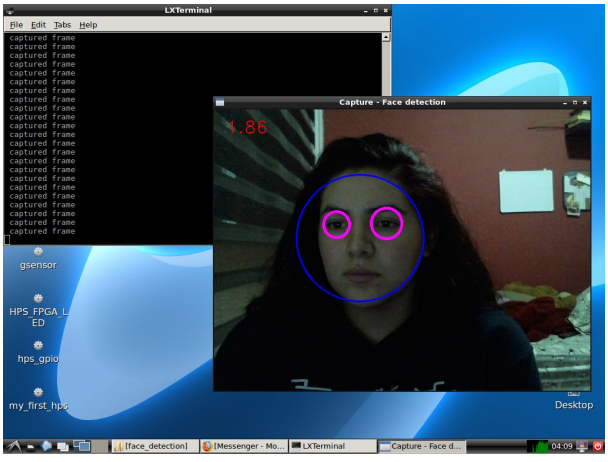

Figure 47. Face detection program running.

Video: https://youtu.be/4eXkdlZXlP8

7. Verification

The verification process changed as our project changed too. For this reason, we have to do a new acceptance criteria and new requirements. This actualization is shown in Table 4 as well as the results and a comparison between our project before and our project now.

Requirement		Acceptance Criteria		Test Result	Requirement met?
Before	**Now**	**Before**	**Now**		
Temperature Sensor	G-sensor	The temperature sensor is working correctly.	The G-sensor is working correctly and displaying the values on the terminal.	Figure 43.	Yes.
		The temperature is shown in the LCD Display.	The coordinates are shown in the LCD.	Figures 45 and 46.	Yes.
Real time clock	7-segment diplay	The real time clock is working.	The 7-segments displays are working and configurable for the HPS.	Figure 44.	Yes.
		The real time clock shows correctly on the seven segments displays.	The Y-coordinate is shown on the 7-segments displays and differentiates from negative and positive.	Figures 45 and 46.	Yes.
Switches	LEDs	The switches work correctly and the leds are working properly with them.	The LEDs are working and configurable for the HPS.	Figure 44.	Yes.
		The task is shown correctly on the LCD depending on the person.	The LEDs change depending on the Y-coordinate of the G-sensor.	Figures 45 and 46.	Yes.
Camera		The system is recognizing the camera.		Figure 47	Yes.
		The face is correctly detected by the camera.		Figure 47	Yes.

Table 4. Verification results.

8. Conclusions

a. Perla Vanessa Jaime Gaytán

The development of this project was very helpful to me in order to understand in a better way all the concepts that we saw in the theory class. I believe that it was challenging at some point where we have to do a lot of research and combine all the demo projects that came with the board. It was interesting to include some components that we did not have any idea how they would work as the 7-segments displays and the camera project. The part of the 7-segments display was at some point difficult to implement in our project in the meaning of developing the bridge between the FPGA and the HPS. For me, it was the part that had more tricks that we needed to understand completely in order to do the correct mapping on Quartus. The camera project was fascinating to work on because we were really interested in learning how the detection process was developed on this code. Since we left this part to the end, and our complete project was developed in C, we were not able to change all the code that we already had to C++, which was the language that the camera project was. I think that I will continue doing this project in C++ in the near future because I really enjoy the research of it and the initial idea. For me it was an important part of the course to do the complete development of this project and to make it work somehow.

b. Gabriela Hernandez López

Throughout the making of this project I'd say what I enjoyed the most was getting to work with new tools and features of the Quartus Software, It's really true that one never stops learning things and in this case it was really clear since I've already worked with this and never imagined the amount of things that could be done within it. I also enjoyed working with the HPS side of the FPGA board and getting to know and implement the FPGA-to-HPS bridge. I find the HPS side to be truly helpful and in some ways easier to work with, at least for this project it was. I feel this seems evident especially if one is working with real time applications, in this case for use retrieving the data from the g-sensor really helped us to begin the development of the rest of our project's design.

Since I was the one working with the camera initially because the other members of the team didn't have one still it was kind of hard to explain to them what was going on and how was the system working but once Vanessa got hers everything went better because she could actually see and experiment with the camera project. However, as we saw time wasn't going to be enough we had to skip this part eventually but I'd love to keep working with this and learn more about face recognition since it's something that many devices work with nowadays.

c. Isaac Benjamin Ipenza Retamozo

During this project, I learn mainly what kinds of application I can do with the bridge made between the FPGA and the HPS in the DE-10 Standard; in this project we have to change the object because we had a problem with the Face Detection part and the language of OpenCV and with the knowledge we have of the FPGA and HPS, we decide to do a different application between this. Although, we were doing this new application that was with the G-sensor, Leds and 7segments that we document in this report, something that I have learn doing this project is the huge quantity of documentation of different elements or different source that we can achieve search in the Internet, and I have notice this because I search different possibilities with the problem that we have with OpenCV, I found several pages in Internet with different codes for other kinds of application and how they are already done and we are free to use them in the future. This project has two interesting things for me that are the two main thing that later on I will be investigate them, the first one is the Qsys tools, this is a tool that help us to make in an easier way the structure of the project, and now I want know more about it, and what other application I can do with this and with different element that I can use together; and the second thing is how a Face detection and a Face recognition is done, I found in internet different code of other algorithms to do this recognition that I want to try them in the future, this because I thing that in the future this will be the main way the people will be able to enter to a concert, cinema, mall, any store, we will no need to have money or a ticket with us, just our faces that will be connected with a database with all our information, like now that we can pay with our phones or have a digital ticket on it, but in the future I think that this can be done just with Face recognition.

9. References

[1] *7 Segment Display Pinout | Working Understanding of 7 Segment Display.* (2019). ElectronicsForu. Available at: https://bit.ly/36eC2ul

[2] *Accelerometer SPI Mode Core for DE-Series Boards.* (2014). ALTERA.

[3] Ahonen, T., Hadid, A., & Pietikainen, M. (2004). *Face recognition with local binary patterns*. In Computer vision-eccv (pp. 469-481). Springer.

[4] *Cascade Classifier*. (2020). Open Source Computer Vision. Available at: https://bit.ly/2V88t7i

[5] Cooper, T. (2013). *All About LEDs.* Available at: https://bit.ly/2VhDci9

[6] *DE10-Standard CD-ROM.* (2017). Terasic Inc. Available at: https://bit.ly/36eUC5B

[7] *DE10-Standard OpenCV User Manual.* (2017). Terasic Inc. Available at: https://bit.ly/3mjg4Mf

[8] *DE10-Standard User Manual*(2017). Terasic Inc. Available at: https://bit.ly/36cDH3l

[9] *Face Recognition with OpenCV*.(2020). Open Source Computer Vision. Available at: https://bit.ly/2KCtftF

[10] Goodrich, R. (2013). *Accelerometers: What They Are & How They Work.* Live Science. Available at: https://bit.ly/3maRgWG

[11] Kashani-Akhavan, S. & Beuchat R. (2016). *SoC-FPGA Design Guide.* Available at: https://bit.ly/3q7zVjH

[12] Rose, M. (2019). *LCD (Liquid Crystal Display).* WhatIs. Available at: https://bit.ly/39naGE8

[13] Woodford, C. (2020). *LCDs (liquid crystal displays).* Available at: https://bit.ly/33n25xF