METU EE314 Laboratory Spring 2016-2017 Term Project Coin Counter

Berk İSKENDER, Asena M. SARICI,
2031920, *berk.iskender@metu.edu.tr*, 2031284,
sarici.asena@gmail.com

Abstract*: This report includes explanation, related codes, simulations and achieved experimental values for the EE 314 Coin Counter term project. The purpose of this report is to provide information about codes and overall project to the reader. Respectively, expected results and the obtained ones are explained. Moreover, basic knowledge and research for the VGA utilization and methods that are used in the establishment of the project is included.

INTRODUCTION

In this term project, we are required to design a coin counter using Verilog and FPGA boards. We are supposed to use an input to FPGA, a 2D Grayscaled image of coins on a white sheet of paper marked with black squares of 1 cm edges. This document is composed of proposed solution, image loading and filtering techniques that are used for the detection of the circular shaped object in the image. Also, codes, block diagrams and images explaining the working principles of the theoretical background, output and simulation results of the algorithms and experiments are included. These algorithms focus on taking the grayscale image as a .hex input, applying threshold filter, doing edge detection using 3x3 Sobel mask, CHT (Circular Hough Transform) for detecting the circles and giving the output of this process to the VGA display. However, since the preliminary report is based on the filtering techniques and theoretical background related to them, VGA partis not included and left for the final report. The overall block diagram of the project is shown in Figure 1.

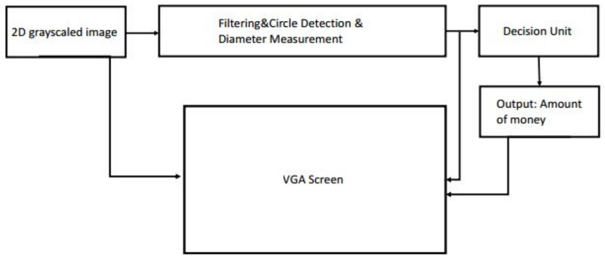


Fig 1. Block Diagram of the overall Project

PROPOSED SOLUTION

As a solution for the given problem, a detailed research on filtering techniques, uploading an image to FPGA [1], circle and edge detection algorithms, Hough Transform for square and circle detection, serial input processing for VGA are performed. We decided to use MATLAB for converting the image into a suitable form such that it is readable by FPGA. Then a filter design is found suitable for the uploaded file. The next step is proposed as finding circles fitting into a range of diameter and counting their numbers using circle and square detection such that square pixel length will give a proportionality with the real dimensions of the image and the coins and counting process can be performed afterwards. Meanwhile, edge filtered array line is back converted to a file such that MATLAB can get the file and convert the pixel values written in the file back to image and this image is observable in the VGA screen. For this, it is proposed to use a suitable code later on in FPGA such that the monitor will understand where to stop and start a new line while showing the image. During the project, programming tools like MATLAB and Quartus, VGA interface, different images of coins and some other objects for filtering will be used. . Codes represented as images are added in appendix part of the report as text.

I. GRAYSCALE CONVERSION

In order to simplify the filtering, edge and circle detecting and identification operations, input image has to be converted to a grayscale form which is taken as RGB output with equal R, G, B values on VGA. In order to achieve this goal simple MATLAB code below is used:

```
Image=imread('nonoise1.png');
GS=rgb2gray(Image);
```

II. RESIZING AN IMAGE

All of the further iterations in verilog and MATLAB codes depend on the image resolution which is predefined as 320*240 in the project. Resizing provides the project with the ability to run on images with various resolution values. However, we need to beware of the loss of information while doing this operation which may cause several deteriorations in the identification and detection which is explained in further parts. The MATLAB code for this part is provided below:

```
PartI=imresize((PartI),[320 240]);
M_col=ones(size(PartI,1),size(PartI,2))*255;
```

III. HEXADECIMAL FORMAT CONVERSION

Verilog does not have any built in functions to take a .jpg, .png etc. format file as an input directly. Pixel values of the image has to be converted to the binary or hexadecimal text format (.hex) and read by the $readmemh or $readmemb functions. MATLAB code to obtain the image pixel values as outputin hexadecimal format:

```
fid = fopen('coinsafter.hex', 'wt');
```

```
fprintf(fid, '%x\n', a);
disp('Text file write done');disp('');
fclose(fid);
```

To get the .hex extension file as input, built-in verilog function “$readmemh” should be used. After storing the .hex file in the same folder address with the verilog code, following Verilog code should be executed in initial case:

```
reg [9:0] data [0:76799]; //data register array takes hexadecimal input and store it as 0-255 binary and it has 76800 (320*240) elements

// Same size register arrays to hold the pixel values after required operations (threshold, sobel filtering)
reg [9:0] thresholded [0:76799];
reg [9:0] finalGrad [0:76799];

initial begin
column=320;
 row=240;

// Obtaining the text file as input
$readmemh("coins", data);

//Attaining black or white pixel values using a threshold value 100
for(a=0;a<row*column;a=a+1)begin
if (data[a]<=100)
        thresholded[a]<=0;

if (data[a]>100)
        thresholded[a]<=255;

end
```

IV. FILTERING TECHNIQUES

A. Thresholding

Thresholding is utilized to increase the contrast in the image. It assigns 0 or 255 to the pixel values that is beyond a threshold value. By doing so, filtering operations become easier such as gradient calculations using Sobel Mask. A moderate threshold value should be chosen in order to distinguish white paper from the objects on top of it. To obtain thresholding functionality following verilog code is used:

```
//Attaining black or white pixel values    using   a   threshold value 100
for(a=0;a<row*column;a=a+1)begin
if (data[a]<=100)
        thresholded[a]<=0;

if (data[a]>100)
        thresholded[a]<=255;

end
```

This code traverses the whole image (row*column) image and applies thresholding on each pixel.

B. Background on Edge Filtering Algortihms

For edge filtering, high contrast image points can be found by intensity difference computations in the images. These areas of high intensity form the borders of different objects using the surrounding masks which are matrices representing some derivative operations. These masks are multiplied by the image vectors so that the pixel value difference at each point with respect to its surrounding is obtained.

There are different type of masks that can be used for edge detection. These include Prewitt, Sobel which are 3 by 3 masks and Roberts which is a 2 by 2 mask. The difference between Sobel and Prewitt is that the center estimate of Sobel is twice more intense. This can be observed from Figure 2.

$$\text{Prewitt:}\quad M_x = \begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \quad ; \quad M_y = \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -1 & -1 \\ \hline \end{array}$$

$$\text{Sobel:}\quad M_x = \begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \quad ; \quad M_y = \begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array}$$

$$\text{Roberts:}\quad M_x = \begin{array}{|c|c|}\hline 0 & 1 \\ \hline -1 & 0 \\ \hline \end{array} \quad ; \quad M_y = \begin{array}{|c|c|}\hline 1 & 0 \\ \hline 0 & -1 \\ \hline \end{array}$$

Fig 2. Masks Used in Image Filtering

Another filtering technique is Canny Edge Detector which first smooths the intensity of the image and after that highlights the contours.

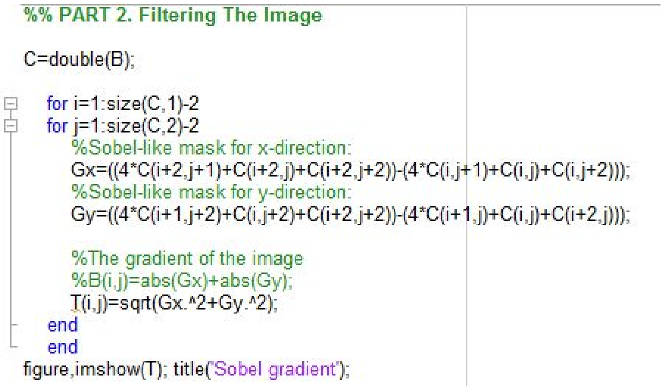
After some research, it is observed that masks are the basics for the filtering operation. The response of a mask to a certain region in an image is proportional to how similar that neighbourhood looks like to the mask.[2]. We can design a mask for tasks we want to perform like edge detection or special pattern detection like corners or circles etc. Usually masks used for image processing are 3x3 or higher and the elements in the sets need to be orthogonal.

In our filter design, we take Sobel Filter Design as base and using the gradient technique showed in Figure 2 we propose a filter with element values composed of 1’s and 4’s instead of 1’s and 2’s in Sobel Filter. This way we aim to strengthen the difference of the boundaries in the image. The matrix can be divided to its weight as proposed in Figure 2 to keep the intensity constant but we do not consider it here. Moreover the 3x3 choice is for the simplicity of the implementation on FPGA for later use of the filter.

Then the rest of the procedure will be conducted on FPGA.

V. MATLAB COMPLETE FILTERING AND DECISION UNITS IMPLEMENTATIONS

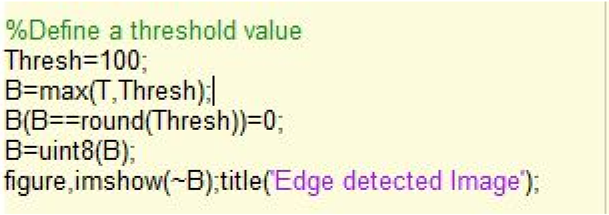
Before implementing the algorithm in Verilog and hence FPGA, firstly, we conducted the procedure in MATLAB for the ease. As a result we get the filtered image as in Figure 5.a.



```
%% PART 2. Filtering The Image

C=double(B);

    for i=1:size(C,1)-2
    for j=1:size(C,2)-2
        %Sobel-like mask for x-direction:
        Gx=((4*C(i+2,j+1)+C(i+2,j)+C(i+2,j+2))-(4*C(i,j+1)+C(i,j)+C(i,j+2)));
        %Sobel-like mask for y-direction:
        Gy=((4*C(i+1,j+2)+C(i,j+2)+C(i+2,j+2))-(4*C(i+1,j)+C(i,j)+C(i+2,j)));

        %The gradient of the image
        %B(i,j)=abs(Gx)+abs(Gy);
        T(i,j)=sqrt(Gx.^2+Gy.^2);
    end
    end
figure,imshow(T); title('Sobel gradient');
```

Fig 3. MATLAB code used for Filtering the Image

Same threshold (100) is selected so that we can see the edges clearly. Figure 4 shows the code used for this procedure.



```
%Define a threshold value
Thresh=100;
B=max(T,Thresh);
B(B==round(Thresh))=0;
B=uint8(B);
figure,imshow(~B);title('Edge detected Image');
```

Fig 4. Thresholding the Output Image in MATLAB

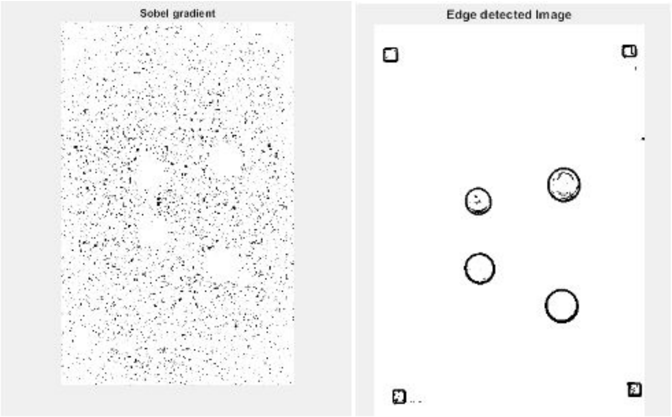


Fig. 5.a On left, the filtered image before thresholding. 6.b On right, filtered image after thresholding

Decision Unit Using Hough Transform

Hough transform is utilized for detecting analytical shapes in an image. For our purpose it will be used for detecting circles which have radii lying in a specified interval representing different coins. Then using accumulation, the number of circles in a radius class is counted. This will give us the total amount of money on the grayscaled picture. For this type of filter, a grayscale image should be used as an input and to avoid unnecessary circle detections which do not correspond to real circular objects, a threshold mask and an edge detection mask can be used. For this purpose we used Sobel filter as stated above. This transform has an algorithm that aims to detect if an edge exists for that particular pixel value. If that pixel is proved to be an edge, its parameters are measured and related variable for that candidate parameter is increased. In this procedure, voting algorithm is implemented. Local maxima corresponding to parameters are found which are "voted" most by the algorithm and used in the transform. This maxima is found in the accumulator space which is described by the circle expression in the 2D space. [3]

Using this voting method and the filtered image we obtained after Sobel Filtering, following MATLAB code is created:

```
    for theta=-pi/6:pi/6
```

//For the square detection at the corners of the images. The lenght found from the squares will be used as the reference point for the classification of the coins.

```
        R_theta = abs(round(y*sin(theta)+x*cos(theta)));

          if R_theta>0
            Voter(R_theta) = Voter(R_theta) + 1 ;
           if Voter(R_theta)>maximum_R_theta
              maximum_R_theta=Voter(R_theta);
           end
          end
```

//After the voting process the starting edge of the squares from the right is detected. After this by simply looking at the 3 columns and adding the pixel values and by rounding and finding the mean of the value, the edge length is found. This value is the reference length to be used.

```
      while(Voter(sqr_length)~=maximum_R_theta)
        sqr_length=sqr_length+1;
      end
      sqr_length=sqr_length-2
      LengthFinder=zeros(1,3);
      for i=sqr_length-1:sqr_length+1
        for j=1:Final_y
          LengthFinder(p)=LengthFinder(p)+Edge_better(j,i);
        end
      for i=1:3
        if LengthFinder(i)>Length
          Length=LengthFinder(i);
        end
      end
      Length=round(Length/2)
```

//For different diameters of the coins this length is multiplied with the appropriate diameter (real) value.

```
      for theta=0:pi/60:2*pi
        for R_theta=0:radius_100+10
          m = round(x - R_theta*cos(theta));
          n = round(y - R_theta*sin(theta));
            if (m>0 && n>0)
              if(R_theta==radius_25-1 )
              Voter25(m,n) = Voter25(m,n) + 1 ;
      if Voter25(m,n)>maxi25
                maxi25=Voter25(m,n);
```

…..//similar for different diameters, on a circle the values of the pixels are looked and if this value is the opposite to the selected center, the value of

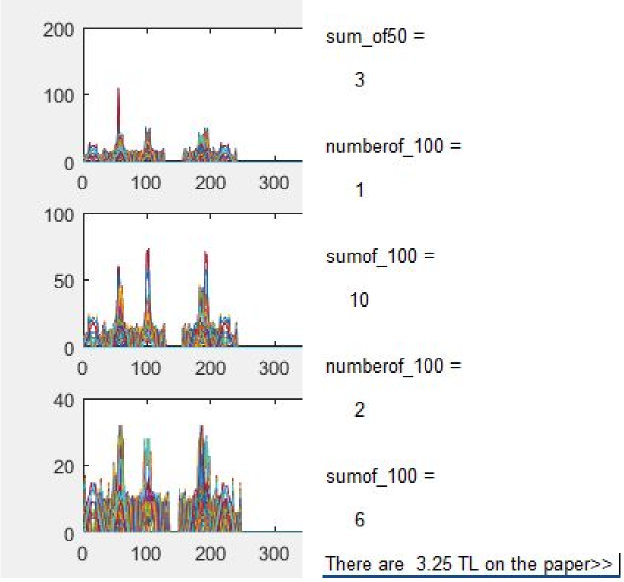
the Voter is increased.

```
        if maximum25<A_binary25(i,j)
                maximum25=A_binary25(i,j);
                y=j;
                x=i;
```

//here, another thresholding is applied so that we can find the peaks of the highest probable places where the center of the coins may lay. (25kr here)

```
        for k = x-20 : x+20
            for j = y-20 : y+20
                A_binary25(k,j) =0;
```

//here, the detected circle is deleted so that it is not going to affect the remaining process of coin finding.



```
sum_of50 =

    3

numberof_100 =

    1

sumof_100 =

    10

numberof_100 =

    2

sumof_100 =

    6

There are  3.25 TL on the paper>>
```

Figure 6.Coin center probabilistic distribution and the result of the coin count

The complete code is provided in the Appendix part of the report.

VI. VERILOG IMPLEMENTATION OF CIRCLE EDGE DETECTION ALGORITHM

The following code which implements the same procedure in Verilog to detect edges. The output is also taken as .hex format into the PC environment and printed out to visualize the result before proceeding into the next parts. In Verilog, traversing is done inside the for loop and verilog has a different operator than MATLAB while taking the different powers of a variable or constant which is denoted as "**". Comments are denoted with the symbol "//".

//Initialization of the finalGrad register array which stores the ultimate circle detected image

```
        for(d=0;d<column+1;d=d+1)begin
            finalGrad[d]<=0;
        end
            for(e=column*row-column-1;e<row*column;e=e+1)begin
                finalGrad[e]<=0;
            end
```

// Traversing on the image using a single for loop with the required parameters and the modified Sobel Mask

```
        for (c=column+1;c<column*row-column-1;c=c+1)
            begin
        yGrad=thresholded[c-column-1]*(1)+thresholded[c-1]*(4)+thresholded[c+column-1]*(1)+thresholded[c-column+1]*(-1)+thresholded[c+1]*(-4)+thresholded[c+column+1]*(-1);
        xGrad=thresholded[c-column-1]*(-1)+thresholded[c-column]*(-4)+thresholded[c-column+1]*(-1)+thresholded[c+column-1]*(1)+thresholded[c+column]*(4)+thresholded[c+column+1]*(1);

                if (xGrad <0 )
                xGrad=xGrad*(-1);
                if (yGrad <0)
                yGrad=yGrad*(-1)
```

//Verilog has ** operator to take the corresponding power of the variable

// The operation is findind the norm of xgrad and ygrad and store it in the sumGrad

```
            sumGrad=xGrad**2+yGrad**2;
            sumGrad=sumGrad**0.5;
```

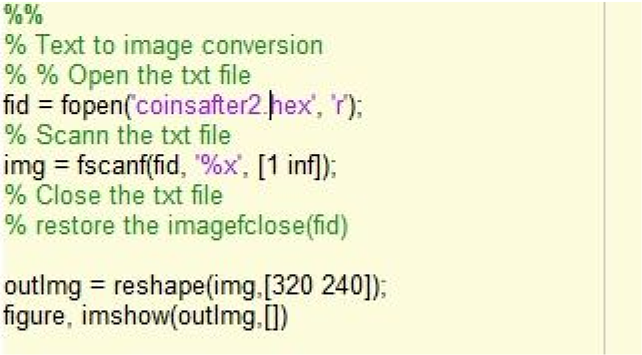
// Final output gradient is thresholded with the norm 100 to highlight the detected circular shapes

```
            if (sumGrad<100)
            finalGrad[c]<=255;
              else
            finalGrad[c]<=0;
              end
```

// For the visual purposes, output register values are also taken as text to see the result before proceeding further

```
            file2 = $fopen("coinsafter","w");
            for(b=0;b<column*row;b=b+1)
              begin
            $fdisplay(file2,"%d",finalGrad[b]);
              end
        end
```

After this step. since the compilation process takes a very long time for parallel processing as well as writing the data in a file is not synthesizable in Quartus, ModelSim is used for compilation. Later, to understand whether the filtering process is successful or not, a MATLAB code shown in Figure 7. to print out the verilog FPGA results is used



```
%%
% Text to image conversion
% % Open the txt file
fid = fopen('coinsafter2.hex', 'r');
% Scann the txt file
img = fscanf(fid, '%x', [1 inf]);
% Close the txt file
% restore the imagefclose(fid)

outImg = reshape(img,[320 240]);
figure, imshow(outImg,[])
```

Figure 7. MATLAB code for text to image conversion

For the given “nonoise1” picture, the filtered image in Figure 8. is obtained.

We could not manage to implement the Decision Unit in FPGA, therefore we showed the results of the MATLAB Circle Decision Unit that is explained in part V.

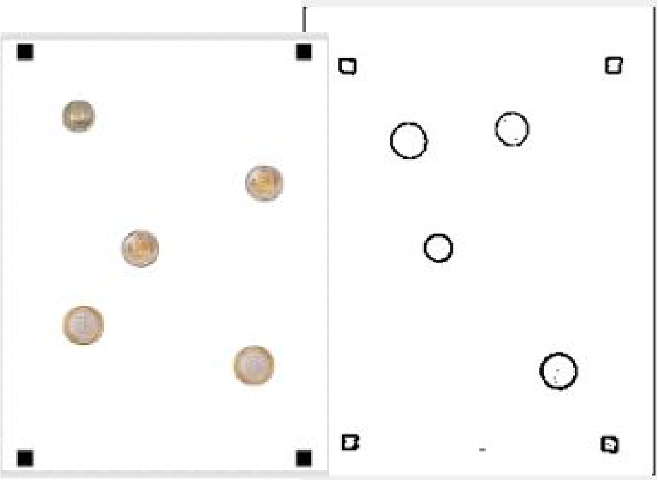


Figure 8. Printed filtered verilog output .hex text file of the nonoise1 (left) which is implemented on FPGA

VI. VGA MODULE

In the project definition it is stated that the results of our decision unit (circle detection and coin identification) should be printed out on a VGA monitor in 320*240 dimensions. To fulfill this requirement, certain procedure should be followed according to the VGA protocol. Sufficient information is provided concerning the pin assignments and required clock frequencies in the user manual of DE-SOC1 FPGA.

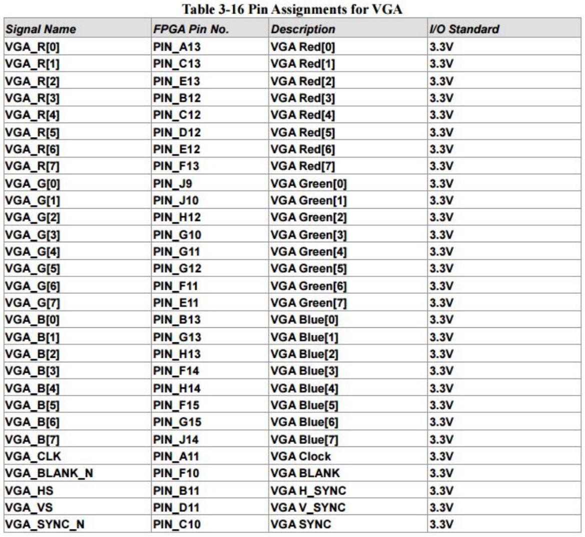


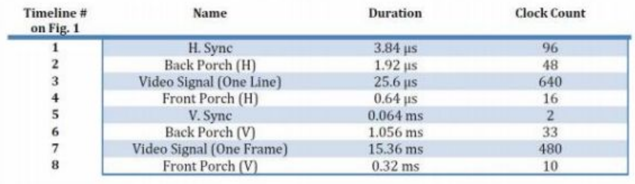
Table 3-16 Pin Assignments for VGA

Signal Name	FPGA Pin No.	Description	I/O Standard
VGA_R[0]	PIN_A13	VGA Red[0]	3.3V
VGA_R[1]	PIN_C13	VGA Red[1]	3.3V
VGA_R[2]	PIN_E13	VGA Red[2]	3.3V
VGA_R[3]	PIN_B12	VGA Red[3]	3.3V
VGA_R[4]	PIN_C12	VGA Red[4]	3.3V
VGA_R[5]	PIN_D12	VGA Red[5]	3.3V
VGA_R[6]	PIN_E12	VGA Red[6]	3.3V
VGA_R[7]	PIN_F13	VGA Red[7]	3.3V
VGA_G[0]	PIN_J9	VGA Green[0]	3.3V
VGA_G[1]	PIN_J10	VGA Green[1]	3.3V
VGA_G[2]	PIN_H12	VGA Green[2]	3.3V
VGA_G[3]	PIN_G10	VGA Green[3]	3.3V
VGA_G[4]	PIN_G11	VGA Green[4]	3.3V
VGA_G[5]	PIN_G12	VGA Green[5]	3.3V
VGA_G[6]	PIN_F11	VGA Green[6]	3.3V
VGA_G[7]	PIN_E11	VGA Green[7]	3.3V
VGA_B[0]	PIN_B13	VGA Blue[0]	3.3V
VGA_B[1]	PIN_G13	VGA Blue[1]	3.3V
VGA_B[2]	PIN_H13	VGA Blue[2]	3.3V
VGA_B[3]	PIN_F14	VGA Blue[3]	3.3V
VGA_B[4]	PIN_H14	VGA Blue[4]	3.3V
VGA_B[5]	PIN_F15	VGA Blue[5]	3.3V
VGA_B[6]	PIN_G15	VGA Blue[6]	3.3V
VGA_B[7]	PIN_J14	VGA Blue[7]	3.3V
VGA_CLK	PIN_A11	VGA Clock	3.3V
VGA_BLANK_N	PIN_F10	VGA BLANK	3.3V
VGA_HS	PIN_B11	VGA H_SYNC	3.3V
VGA_VS	PIN_D11	VGA V_SYNC	3.3V
VGA_SYNC_N	PIN_C10	VGA SYNC	3.3V

Figure 9. Pin Assignment Table used during VGA implementaion

VGA functions properly with the clock frequency 25 MHz. Out default clock frequency on the FPGA is 50 MHz. Therefore, frequency division should be done in order to obtain 25 MHz clock as VGA Clock. Moreover, FPGA has Hsync, Vsync, VGA Blank and VGA Syncinputs to synchronize itself with the input image data. Hsync and Vsync are the active low signals which signal the end of the line and end of the frame, respectively. They require spesific time periods in which they have to give low signals to the VGA to tell it to start a new frame or new line.

Their time periods are stated strictly for proper operation in Table 1.



Timeline # on Fig. 1	Name	Duration	Clock Count
1	H. Sync	3.84 µs	96
2	Back Porch (H)	1.92 µs	48
3	Video Signal (One Line)	25.6 µs	640
4	Front Porch (H)	0.64 µs	16
5	V. Sync	0.064 ms	2
6	Back Porch (V)	1.056 ms	33
7	Video Signal (One Frame)	15.36 ms	480
8	Front Porch (V)	0.32 ms	10

Table 1: Timing.

Clock count means while doing iterative operation to print out pixels from the registers required signals should be given for a period that takes that number of steps.

As it can be seen form the Figure 10 and 11, VGA protocol also has back porch and front porch regions in which the input RGB values should all be set to 0. This is done to prevent the pop up of electron beam while it is moving from the end of the previous frame to the beginning of the next one.

So while implementing the code RGB values has to be set to zero for the clock counts of Hsync, Vsync, back porch and front porch when the steps are in their region. A simple model which represents the total VGA “screen” is shown below.

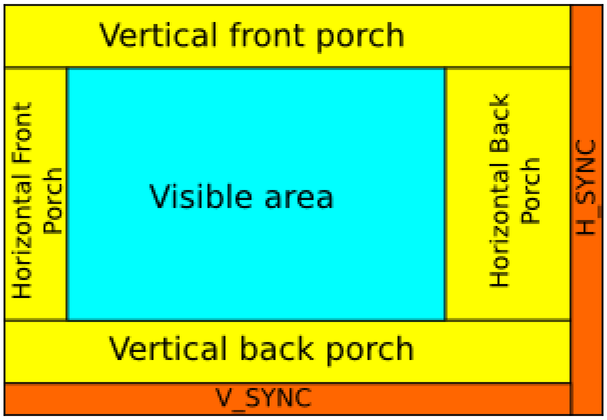
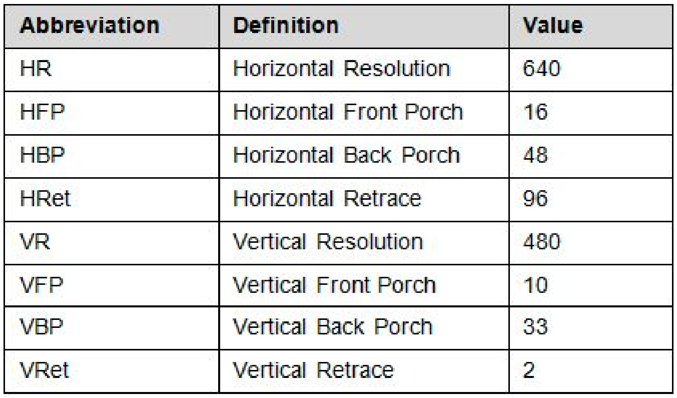


Figure 10. Complete VGA screen



Abbreviation	Definition	Value
HR	Horizontal Resolution	640
HFP	Horizontal Front Porch	16
HBP	Horizontal Back Porch	48
HRet	Horizontal Retrace	96
VR	Vertical Resolution	480
VFP	Vertical Front Porch	10
VBP	Vertical Back Porch	33
VRet	Vertical Retrace	2

Figure 11. Table for total pixel values for each region

VI. PRINTING THE IMAGE AND COUNTER

For this part two different implementation methods are followed as :

A. First Method: Parallel transfering

In this method, whole pixels of the image are taken into the register at once using $readmemh and printed out by traversing. This method has drawbacks. Due to the parallel computing of all pixels at once, compilation takes a lot of time for one 320*240 image (40-50 minutes) and printing out two different images on the screen were taking hours to compile and giving errors during analysis and compilation. However, the method is implemented for one image and the code and experimental result is shown below:

```
    module vgason(clk_in, Hsync, Vsync, Red, Green, Blue, clk);

       input clk_in;
       output Hsync, Vsync, Red, Green, Blue;
       output reg clk;

       reg horizontal, vertical;
       reg [9:0] xPos; reg [8:0] yPos;
       reg [9:0] data [0:76799];
       reg [7:0] Blue; reg [7:0] Green;
       reg [7:0] Red;
       reg[9:0] row; reg[9:0] column;
       reg [1:0] counter;

       initial
      begin
       $readmemh("coins", data);

       column=240; row=320;
       clk=0; xPos=0; yPos=0; counter=0;
          end

//division to 25MHz from 50 MHz
       always @ (posedge clk_in)
         begin
       counter = counter+1;
       if (counter ==1)
         begin
// inverting the vga clock once in two clocks
       clk=~clk; counter=0;
         end
      end


//counters for x and y positions
       always @ (posedge clk)
      begin
       if (xPos<800)
       xPos=xPos+1;
       else
         begin
       xPos=0;
           if (yPos<525)
           yPos=yPos+1;
              else
              yPos=0;
              end

//Hsync Vsync Generators
       if (xPos>656 && xPos<752)
       horizontal = 0;
           else
           horizontal = 1;
              if (yPos>490 && yPos<492)
              vertical = 0;
                 else
                 vertical = 1;

//assignment of pixels to 0 in porchs and vsync, hsync and to the register values in the active visible region of VGA screen with an indent of 25 pixels vertically and horizontally

       if((xPos>640 && xPos<800) || (yPos>480 && yPos<525))
           begin
                            Red<=8'h 0;
                            Green<=8'h 0;
                            Blue<=8'h 0;
            end
       end


       if   ((yPos<row+25) &&
       (yPos>25)&&(xPos<column+25)&&(xPos>25))
             begin

       Red<=data[row*(xPos-25)+yPos-25];
                  Green<=data[row*(xPos-25)+yPos-25];
                  Blue<=data[row*(xPos-25)+yPos-25];
             end
          else begin
                  Red<=8'h 0;
                  Green<=8'h 0;
                  Blue<=8'h 0;
              end
       end

//assignment of Hsync and Vsync output from the generated ones in the code

       assign Hsync=horizontal;
       assign Vsync=vertical;

     endmodule
```

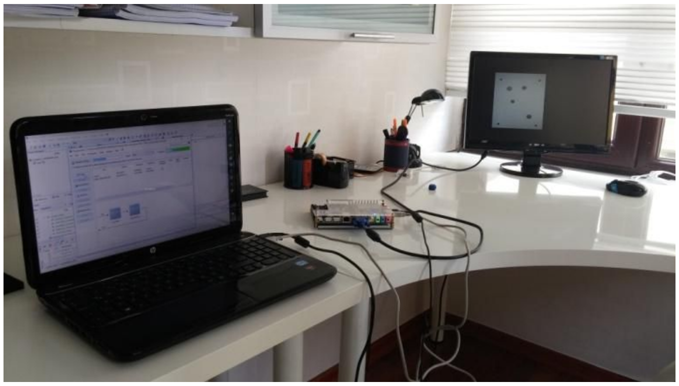


Figure 12. VGA implementation using one output image

B. Second Method: Serial transffering using ROM

In this method, every register corresponding to one pixel is taken as input and printed out afterwards. By doing so, compilation takes very small amount of time because FPGA only arranges one input output system instead of implementing it for all pixels simultaneously. Compilation time has dropped to 3-4 minutes and it did not change with respect to image number that is printed on the screen. By all means this method prevails compared to the first one.

Using serial transferring, two images (grayscaled one and the circle detected) are shown at the top and the total number of money is printed at the bottom left corner. The money output changes when the total coin count increases or decreases.

The complete code is provided in the appendix part. Since the clock divisions and the traversing pattern is the same for this part, all important functional parts of the code and the experimental results are shown below:

//registers for storing position data

```
wire [16:0] pos; wire [11:0] pos25;
wire [11:0] pos50; wire [11:0] pos100;
wire [11:0] pos_result;
```

//registers that store the current position pixel data at every iteration

```
assign pos = county2*240+countx2;
assign pos_result =(county2-420)*50+countx2;
```

//ROMs for two 320*240 images

```
reg [7:0] TOTAL_ROM[0:76799];
reg [7:0] TOTAL_ROM2[0:76799];
```

//registers for printing out amount of total money

```
reg[7:0]q;  reg[7:0]l;
reg[7:0]result0; reg[7:0]result25;
reg[7:0]result50; reg[7:0]result75;
reg[7:0]result100; reg[7:0]result125;
reg[7:0]result150;
reg[7:0]result175; reg[7:0]result200;
reg[7:0]result225; reg[7:0]result250;
reg[7:0]result275; reg[7:0]result300;
reg[7:0]result325; reg[7:0]result350;
```

//ROMs for 40*40 coin counters

```
reg[7:0] ROM_0[0:1599]; reg [7:0] ROM_25[0:1599];
reg[7:0] ROM_50[0:1599];reg [7:0] ROM_75[0:1599];
reg[7:0] ROM_100[0:1599];reg[7:0] ROM_125[0:1599];reg[7:0]
ROM_150[0:1599];
reg[7:0] ROM_175[0:1599];reg[7:0] ROM_200[0:1599];
reg[7:0] ROM_225[0:1599];reg[7:0] ROM_250[0:1599];reg[7:0]
ROM_275[0:1599];
reg[7:0] ROM_300[0:1599];
reg[7:0] ROM_325[0:1599];reg[7:0] ROM_350[0:1599];
```

//variables to store amount of each money type

```
ceyrek=0;
tam=0;
yarim=0;

$readmemh("0TL.hex", ROM_0);
$readmemh("25kr.hex", ROM_25);
$readmemh("50kr.hex", ROM_50);
$readmemh("75kr.hex", ROM_75);
$readmemh("1TL.hex", ROM_100);
$readmemh("1.25TL.hex", ROM_125);
$readmemh("1.5TL.hex", ROM_150);
$readmemh("1.75TL.hex", ROM_175);
$readmemh("2TL.hex", ROM_200);
$readmemh("2.25TL.hex", ROM_225);
$readmemh("2.50TL.hex", ROM_250);
$readmemh("2.75TL.hex", ROM_275);
$readmemh("3TL.hex", ROM_300);
$readmemh("3.25TL.hex", ROM_325);
$readmemh("3.5TL.hex", ROM_350);
$readmemh("nonoise1.hex", TOTAL_ROM);
$readmemh("data_in_fpga.hex", TOTAL_ROM2);
```

//Taking pixels one by one in to the related registers

```
always @(posedge clk_in)
  begin
q=TOTAL_ROM[pos];l=TOTAL_ROM2[pos];
result0=ROM_0[pos_result];
result25=ROM_25[pos_result];
result50=ROM_50[pos_result];
result75=ROM_75[pos_result];
result100=ROM_100[pos_result];
result125=ROM_125[pos_result];
result150=ROM_150[pos_result];
result175=ROM_175[pos_result];
result200=ROM_200[pos_result];
result225=ROM_225[pos_result];
result250=ROM_250[pos_result];
result275=ROM_275[pos_result];
result300=ROM_300[pos_result];
result325=ROM_325[pos_result];
result350=ROM_350[pos_result];
```

//Increasing the coin counts according to the external inputs using push buttons

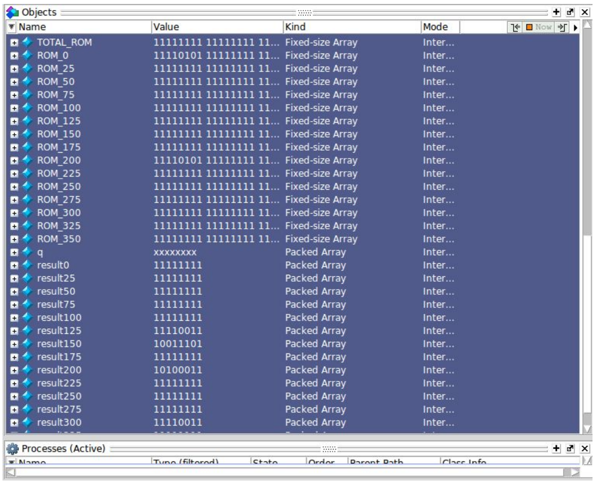
```
always@(posedge a)
  begin
ceyrek=ceyrek+1;
  end
```

//...(repeated for 3 different coin inputs)

//Printing out the images according to their positional if conditions

```
always @(posedge clk_in)
  begin
    if(countx2<column*2 && county2<row && countx2>column)
     begin
            R<=q; B<=q; G<=q;
     end
    else if(countx2<column && county2<row)
      begin
            R<=l; B<=l; G<=l;
      end
else if(countx2<40 && countx2>0 && county2>430 && county2<470)
            begin
              if((ceyrek*25+yarim*50+tam*100)==0)
                begin
                R<= result0; G<= result0; B<= result0;
                end
            end
//...This conditional statement is repeated until the sum of ceyrek*25+yarim*50+tam*100) reaches 3.5 TL.
      endmodule
```

To illustrate that the code is working without using an FPGA, we also implemented in modelsim and using a testbench we obtained the results.



Name	Value	Kind	Mode
TOTAL_ROM	11111111 11111111 11...	Fixed-size Array	Inter...
ROM_0	11110101 11111111 11...	Fixed-size Array	Inter...
ROM_25	11111111 11111111 11...	Fixed-size Array	Inter...
ROM_50	11111111 11111111 11...	Fixed-size Array	Inter...
ROM_75	11111111 11111111 11...	Fixed-size Array	Inter...
ROM_100	11111111 11111111 11...	Fixed-size Array	Inter...
ROM_125	11111111 11111111 11...	Fixed-size Array	Inter...
ROM_150	11111111 11111111 11...	Fixed-size Array	Inter...
ROM_175	11111111 11111111 11...	Fixed-size Array	Inter...
ROM_200	11110101 11111111 11...	Fixed-size Array	Inter...
ROM_225	11111111 11111111 11...	Fixed-size Array	Inter...
ROM_250	11111111 11111111 11...	Fixed-size Array	Inter...
ROM_275	11111111 11111111 11...	Fixed-size Array	Inter...
ROM_300	11111111 11111111 11...	Fixed-size Array	Inter...
ROM_325	11111111 11111111 11...	Fixed-size Array	Inter...
ROM_350	11111111 11111111 11...	Fixed-size Array	Inter...
q	xxxxxxxx	Packed Array	Inter...
result0	11111111	Packed Array	Inter...
result25	11111111	Packed Array	Inter...
result50	11111111	Packed Array	Inter...
result75	11111111	Packed Array	Inter...
result100	11111111	Packed Array	Inter...
result125	11110011	Packed Array	Inter...
result150	10011101	Packed Array	Inter...
result175	11111111	Packed Array	Inter...
result200	10100011	Packed Array	Inter...
result225	11111111	Packed Array	Inter...
result250	11111111	Packed Array	Inter...
result275	11111111	Packed Array	Inter...
result300	11110011	Packed Array	Inter...

Figure 13. Output of the VGA code using testbench on ModelSim

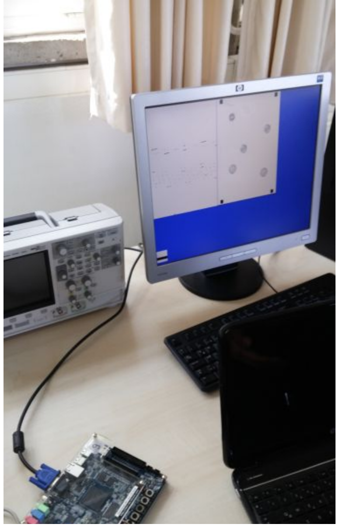


Figure 13. Output of the several images on VGA using ROM (original one on the right, detected one on the left and count amount on the bottom left)

Experimental problem: The only problem that was encountered in this method was accidentally we printed out the transpose of the edge detected and total number of coins images which originated from the MATLAB .hex conversion and had nothing to do with the verilog code which worked totally correct.

We corrected the grayscaled image from the MATLAB but we did not have enough time to correct it for all images before the demonstration. However, by correcting it for the grayscaled image we showed that it can be easily done in 5 minutes for all image conversions and the correct result can be obtained.

VII.SECOND EXTERNAL COINCOUNTER USING THE LED DISPLAY AND 7 SEGMENT DISPLAY

Another counter is implemented in addition to the one implemented in the VGA to guarantee the functionality. This counter takes inputs externally and according to the number of each different type of coins, 3 different 7-segment display code show the total amount and LEDs show the total amount of money in terms of binary 25kr amount.

The code and the experimental results are shown below:

```
reg [9:0] x1, x2, x3;
output reg [7:0] sseg_temp;
output reg [7:0] sseg_temp1;
output reg [7:0] sseg_temp2;

integer s1;
integer s50;
integer s25;

input clk;

initial
   begin
led=0; s1=0; s50=0; s25=0;
    end

always @(posedge a)
    begin
s1=s1+1; x1=x1+3'b100;
     end
```

```
        always @(posedge b)
            begin
        s50=s50+1; x2=x2+2'b10;
            end

        always @(posedge c)
            begin
        s25=s25+1; x3=x3+1'b1;
            end

        always @ (*)
            begin
         case(s1)
        4'd0 : sseg_temp = 7'b1000000; //to display 0
        4'd1 : sseg_temp = 7'b1111001; //to display 1
        4'd2 : sseg_temp = 7'b0100100; //to display 2
        4'd3 : sseg_temp = 7'b0110000; //to display 3
        4'd4 : sseg_temp = 7'b0011001; //to display 4
        4'd5 : sseg_temp = 7'b0010010; //to display 5
        4'd6 : sseg_temp = 7'b0000010; //to display 6
        4'd7 : sseg_temp = 7'b1111000; //to display 7
        4'd8 : sseg_temp = 7'b0000000; //to display 8
        4'd9 : sseg_temp = 7'b0010000; //to display 9
        default : sseg_temp = 7'b0111111; //dash
         endcase

//Same steps are repeated for the 25kr and 50kr
        case(s50)
        ...
        case(s25)
        ...
        led=x1+x2+x3;
         end
        endmodule
```

CONCLUSION

In this project, we were asked to construct a coin counter that identifies the shape of a coin and distinguish its type (25kr, 50kr, 1TL) To achieve this, firstly the input is converted to grayscale. Then it is converted to .hex extension. Both of these steps are done in order to obtain the system in a form to be used as an input for filtering operations. After grayscaling and format conversion, filtering operations take place on the image and firstly, a contrast enhancing method (thresholding) is used. It can be easily seen that without using a threshold, it is nearly impossible to use Sobel Mask effectively on the image to detect edges. Thresholding eases the detection of transitions between different objects in images using a 3x3 Sobel filter.

After the detection of edges in the image, the method that we tried to implement in order to identify the coins was Circular Hough Transform. Although this method is implementable on MATLAB, it is quite hard to get a functional result for it using verilog language. The reasons behind this fact is that verilog does not have predefined cosine or sine functions which are used in the voting system for detecting candidate center pixels. Also, using real values are not implementable on the FPGA which are necessary while computing the sine and cosine function outputs. A solution for this problem might be using a Taylor series expansion for few terms and multiplying it with a large constant to avoid real outputs. However, as the code gets complex (several number of nested loops) while constructing the voting algorithm, compilation period gets longer and after some limitations, code becomes quite difficult to compile and function properly for FPGA.

Lastly, this project aims to make participants familiar to the VGA environment and its input output relationships and different regions with characteristic properties.

To sum up, the project aims to achieve the goal of familiarizing the participants with the decent verilog knowledge, simple image processing methods and VGA output protocol.

REFERENCES

[1] Dhanabal.R, Sarat kumar Sahoo, Bharathi V, “FPGA BASED IMAGE PROCESSING UNIT USAGE IN COIN DETECTION AND COUNTING” in *International Conference on Circuit, Power and Computing Technologies [ICCPCT]*, 2015.

[2] L.G. Shapiro and G.Stockman, “Filtering and Enhancing Images,” in Computer Vision, Mar 2010.

[3] J. Borovicka, “Circle Detection Using Hough Transforms Documentation COMS30121 - Image Processing and Computer Vision”

[4] Edge Detection on Images Available at :https://www.mathworks.com/help/coder/examples/edge-detection-on-images.html?prodcode=ML. [Online] Accessed: May 20, 2017.

[5] Hough transform for circles[Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/26978-hough-transform-for-circles: May 20, 2017.

[6] Examples [Online]. Available: http://fpgacenter.com/examples/vga/img/constant.png: May 20, 2017.

APPENDIX

LED & 7 SEGMENT COUNTER

```
module coincounter();
integer xGrad;
integer yGrad;
integer sumGrad;
integer m;
integer a,b,k,delete1,delete2,to_right,shift,Coin_diameter,count,temp,u,t;
reg [9:0] data [0:76799];
reg [9:0] thresholded [0:76799];
reg [9:0] finalGrad [0:76799];
integer row;
integer column;
integer a,b,c,d,e;
integer file2;
initial begin
  column=320;
  row=240;
  $readmemh("coins", data);
                for(a=0;a<row*column;a=a+1)begin
                if (data[a]<=100)
                                thresholded[a]<=0;

                if (data[a]>100)
                                thresholded[a]<=255;

                end
reg [9:0] x1, x2, x3;
output reg [7:0] sseg_temp;
output reg [7:0] sseg_temp1;
output reg [7:0] sseg_temp2;
integer s1;
integer s50;
integer s25;
input clk;
initial
begin
led=0;
s1=0;
s50=0;
s25=0;
end
always @(posedge a)
begin
s1=s1+1;
x1=x1+3'b100;
end
always @(posedge b)
begin
s50=s50+1;
x2=x2+2'b10;
end
always @(posedge c)
begin
s25=s25+1;
x3=x3+1'b1;
end
```

```
always @ (*)
 begin
  case(s1)
   4'd0 : sseg_temp = 7'b1000000; //to display 0
   4'd1 : sseg_temp = 7'b1111001; //to display 1
   4'd2 : sseg_temp = 7'b0100100; //to display 2
   4'd3 : sseg_temp = 7'b0110000; //to display 3
   4'd4 : sseg_temp = 7'b0011001; //to display 4
   4'd5 : sseg_temp = 7'b0010010; //to display 5
   4'd6 : sseg_temp = 7'b0000010; //to display 6
   4'd7 : sseg_temp = 7'b1111000; //to display 7
   4'd8 : sseg_temp = 7'b0000000; //to display 8
   4'd9 : sseg_temp = 7'b0010000; //to display 9
   default : sseg_temp = 7'b0111111; //dash
  endcase

    case(s50)
   4'd0 : sseg_temp1 = 7'b1000000; //to display 0
   4'd1 : sseg_temp1 = 7'b1111001; //to display 1
   4'd2 : sseg_temp1 = 7'b0100100; //to display 2
   4'd3 : sseg_temp1 = 7'b0110000; //to display 3
   4'd4 : sseg_temp1 = 7'b0011001; //to display 4
   4'd5 : sseg_temp1 = 7'b0010010; //to display 5
   4'd6 : sseg_temp1 = 7'b0000010; //to display 6
   4'd7 : sseg_temp1 = 7'b1111000; //to display 7
   4'd8 : sseg_temp1 = 7'b0000000; //to display 8
   4'd9 : sseg_temp1 = 7'b0010000; //to display 9
   default : sseg_temp1 = 7'b0111111; //dash
  endcase

    case(s25)
   4'd0 : sseg_temp2 = 7'b1000000; //to display 0
   4'd1 : sseg_temp2 = 7'b1111001; //to display 1
   4'd2 : sseg_temp2 = 7'b0100100; //to display 2
   4'd3 : sseg_temp2 = 7'b0110000; //to display 3
   4'd4 : sseg_temp2 = 7'b0011001; //to display 4
   4'd5 : sseg_temp2 = 7'b0010010; //to display 5
   4'd6 : sseg_temp2 = 7'b0000010; //to display 6
   4'd7 : sseg_temp2 = 7'b1111000; //to display 7
   4'd8 : sseg_temp2 = 7'b0000000; //to display 8
   4'd9 : sseg_temp2 = 7'b0010000; //to display 9
   default : sseg_temp2 = 7'b0111111; //dash
  endcase

  led=x1+x2+x3;

 end
end module

module vgaout(clk_in, a, b, c, clk_25mhz, h_sync, v_sync, vga_blank,
vga_sync, R, B, G, vga_clk);
input clk_in;
input a;
input b;
input c;
//clk_in 50 Mhz
//clk25MHz 25MHZ
//25Kr counter
reg[1:0] ceyrek;
//50Kr counter
reg[1:0] yarim;
//1tl counter
reg[1:0] tam;
output vga_sync;
output reg clk_25mhz;
reg [9:0] countx2;
reg [9:0] county2;
output reg[7:0] R;
output reg[7:0] B;
output reg[7:0] G;
output reg h_sync;
output reg v_sync;
output reg vga_blank;
output reg vga_clk;
input wire [1:0] kr25;
input wire [1:0] kr50;
input wire [1:0] tl1;
reg h_sync2, v_sync2, vga_blank2;
wire clrx, clry;
assign vga_sync=1;
reg[3:0] counter;
//initializations of the position indexes
initial
begin
countx2=0;
county2=0;
counter=0;
clk_25mhz=0;
end

//25 MHz clock division

always @ (posedge clk_in)
begin
counter = counter+1;
if (counter ==1)
begin
clk_25mhz=~clk_25mhz;
counter=0;
end
end
always  @(posedge clk_25mhz)
begin
//Position indexes for pixels
if (countx2<800)
countx2=countx2+1;
else
begin
countx2=0;
if (county2<525)
county2=county2+1;
else
county2=0;
end
//Vsync Hsync arrangements

if (countx2>656 && countx2<752)
h_sync = 0;
else
h_sync = 1;
if (county2>490 && county2<492)
v_sync = 0;
else
v_sync = 1;

//VGA blank assignment for the active part of VGA

vga_blank <= ((countx2 < 640) && (county2 < 480));

end

parameter row=320;
parameter column=240;
parameter total=76800;

//registers for storing position data

wire [16:0] pos;
wire [11:0] pos25;
wire [11:0] pos50;
wire [11:0] pos100;
wire [11:0] pos_result;

//registers that store the current position pixel data at every iteration

assign pos = county2*240+countx2;

assign pos_result =(county2-420)*50+countx2;

//ROMs for two 320*240 images

        reg [7:0] TOTAL_ROM[0:76799];
        reg [7:0] TOTAL_ROM2[0:76799];

//registers for printing out amount of total money

reg[7:0]q;
reg[7:0]l;
reg[7:0]result0;
reg[7:0]result25;
reg[7:0]result50;
reg[7:0]result75;
reg[7:0]result100;
reg[7:0]result125;
reg[7:0]result150;
reg[7:0]result175;
reg[7:0]result200;
reg[7:0]result225;
reg[7:0]result250;
reg[7:0]result275;
reg[7:0]result300;
reg[7:0]result325;
reg[7:0]result350;

//ROMs for 40*40 coin counters

reg [7:0] ROM_0[0:1599];
reg [7:0] ROM_25[0:1599];
reg [7:0] ROM_50[0:1599];
reg [7:0] ROM_75[0:1599];
reg [7:0] ROM_100[0:1599];
reg [7:0] ROM_125[0:1599];
reg [7:0] ROM_150[0:1599];
reg [7:0] ROM_175[0:1599];
reg [7:0] ROM_200[0:1599];
reg [7:0] ROM_225[0:1599];
reg [7:0] ROM_250[0:1599];
reg [7:0] ROM_275[0:1599];
reg [7:0] ROM_300[0:1599];
reg [7:0] ROM_325[0:1599];
reg [7:0] ROM_350[0:1599];

//initializations

initial
begin

//variables to store amount of each money type

        ceyrek=0;
        tam=0;
        yarim=0;

//Reading hexadecimal extension files into the ROMs

$readmemh("0TL.hex", ROM_0);
$readmemh("25kr.hex", ROM_25);
$readmemh("50kr.hex", ROM_50);
$readmemh("75kr.hex", ROM_75);
$readmemh("1TL.hex", ROM_100);
$readmemh("1.25TL.hex", ROM_125);
$readmemh("1.5TL.hex", ROM_150);
$readmemh("1.75TL.hex", ROM_175);
$readmemh("2TL.hex", ROM_200);
$readmemh("2.25TL.hex", ROM_225);
$readmemh("2.50TL.hex", ROM_250);
$readmemh("2.75TL.hex", ROM_275);
$readmemh("3TL.hex", ROM_300);
$readmemh("3.25TL.hex", ROM_325);
$readmemh("3.5TL.hex", ROM_350);
$readmemh("nonoise1.hex", TOTAL_ROM);
$readmemh("data_in_fpga.hex", TOTAL_ROM2);

end

//Taking pixels one by one in to the related registers

always @(posedge clk_in)
begin
        q=TOTAL_ROM[pos];
        l=TOTAL_ROM2[pos];
        result0=ROM_0[pos_result];
        result25=ROM_25[pos_result];
        result50=ROM_50[pos_result];
        result75=ROM_75[pos_result];
        result100=ROM_100[pos_result];
        result125=ROM_125[pos_result];
        result150=ROM_150[pos_result];
        result175=ROM_175[pos_result];
        result200=ROM_200[pos_result];
        result225=ROM_225[pos_result];
        result250=ROM_250[pos_result];
        result275=ROM_275[pos_result];
        result300=ROM_300[pos_result];
        result325=ROM_325[pos_result];
        result350=ROM_350[pos_result];

end

//Increasing the coin counts according to the external inputs using push buttons

always@(posedge a)
begin
ceyrek=ceyrek+1;
end

always@(posedge b)
begin
yarim=yarim+1;
end

always@(posedge c)
begin
tam=tam+1;
end

//Printing out the images according to their positional if conditions

always @(posedge clk_in)
begin
if(countx2<column*2 && county2<row && countx2>column)
begin

R<=q;
B<=q;
G<=q;

end
```

```
else if(countx2<column && county2<row)
begin

R<=l;
B<=l;
G<=l;

end
else if(countx2<40 && countx2>0 && county2>430 && county2<470)
            begin
            if((ceyrek*25+yarim*50+tam*100)==0)
            begin
            R<= result0;
            G<= result0;
            B<= result0;
            end
            if((ceyrek*25+yarim*50+tam*100)==25)
            begin
            R<= result25;
            G<= result25;
            B<= result25;
            end

            if((ceyrek*25+yarim*50+tam*100)==50)
            begin
            R<= result50;
            G<= result50;
            B<= result50;
            end

            if((ceyrek*25+yarim*50+tam*100)==75)
            begin
            R<= result75;
            G<= result75;
            B<= result75;
            end

            if((ceyrek*25+yarim*50+tam*100)==100)
            begin
            R<= result100;
            G<= result100;
            B<= result100;
            end

            if((ceyrek*25+yarim*50+tam*100)==125)
            begin
            R<= result125;
            G<= result125;
            B<= result125;
            end

                if((ceyrek*25+yarim*50+tam*100)==150)
            begin
            R<= result150;
            G<= result150;
            B<= result150;
            end

            if((ceyrek*25+yarim*50+tam*100)==175)
            begin
            R<= result175;
            G<= result175;
            B<= result175;
            end

            if((ceyrek*25+yarim*50+tam*100)==200)
            begin
            R<= result200;
            G<= result200;
            B<= result200;
            end

            if((ceyrek*25+yarim*50+tam*100)==225)
            begin
            R<= result225;
            G<= result225;
            B<= result225;
            end

            if((ceyrek*25+yarim*50+tam*100)==250)
            begin
            R<= result250;
            G<= result250;
            B<= result250;
            end

            if((ceyrek*25+yarim*50+tam*100)==275)
            begin
            R<= result275;
            G<= result275;
            B<= result275;
            end

            if((ceyrek*25+yarim*50+tam*100)==300)
            begin
            R<= result300;
            G<= result300;
            B<= result300;
            end

            if((ceyrek*25+yarim*50+tam*100)==325)
            begin
            R<= result325;
            G<= result325;
            B<= result325;
            end

            if((ceyrek*25+yarim*50+tam*100)==350)
            begin
            R<= result350;
            G<= result350;
            B<= result350;
            end

            else
            begin

            R<= 0;
            G<= 0;
            B<= 0;

end
end
endmodule
```

FILTER

```
module coincounter();

integer xGrad;
integer yGrad;
integer sumGrad;

integer m;
integer a,b,k,delete1,delete2,to_right,shift,Coin_diameter,count,temp,u,t;

reg [9:0] data [0:76799];
reg [9:0] thresholded [0:76799];
reg [9:0] finalGrad [0:76799];

integer row;
integer column;

integer a,b,c,d,e;
integer file2;

initial begin
  column=320;
  row=240;

  $readmemh("coins", data);
                  for(a=0;a<row*column;a=a+1)begin
                  if (data[a]<=100)
                                    thresholded[a]<=0;

                  if (data[a]>100)
                                    thresholded[a]<=255;

                  end

  for(d=0;d<column+1;d=d+1)begin
    finalGrad[d]<=0;
  end

  for(e=column*row-column-1;e<row*column;e=e+1)begin
    finalGrad[e]<=0;
  end

  #100000 for (c=column+1;c<column*row-column-1;c=c+1)begin

yGrad=thresholded[c-column-1]*(1)+thresholded[c-1]*(4)+thresholded[c+column-1]*(1)+thresholded[c-column+1]*(-1)+thresholded[c+1]*(-4)+thresholded[c+column+1]*(-1);

xGrad=thresholded[c-column-1]*(-1)+thresholded[c-column]*(-4)+thresholded[c-column+1]*(-1)+thresholded[c+column-1]*(1)+thresholded[c+column]*(4)+thresholded[c+column+1]*(1);
if (xGrad <0 )
      xGrad=xGrad*(-1);
    if (yGrad <0)
      yGrad=yGrad*(-1);

    sumGrad=xGrad**2+yGrad**2;
    sumGrad=sumGrad**0.5;

    if (sumGrad<100)
      finalGrad[c]<=255;
    else
      finalGrad[c]<=0;
end

file2 = $fopen("coinsafter","w");
  #1000000 for(b=0;b<column*row;b=b+1)begin
     $fdisplay(file2,"%d",finalGrad[b]);
  end

end
endmodule
```

DECISION UNIT

```
Image=imread('nonoise1.png');
GS=rgb2gray(Image);
figure('Name','orig'), imshow(GS);

PartI=double(GS);
X=[-1 0 1;-2 0 2; -1 0 1]; % X edge detection
Y=[-1 -2 -1;0 0 0; 1 2 1]; % Y edge detection
Blur_filt=[2 2 2;2 0 2;2 2 2];
SecondFilter=[0 -2 0;-2 9 -2;0 -2 0];
PartI=imresize((PartI),[320 240]);
M_col=ones(size(PartI,1),size(PartI,2))*255;
After_Blur_filt=PartI;


for i=1:size(PartI,1)
    for j=1:size(PartI,2)
        if PartI(i,j)>250
            PartI(i,j)=0;
        else
            PartI(i,j)=255;
        end
    end
end
figure('Name','Thresholded Image'), imshow(PartI,[]);


Sobelled=PartI;
for i=1:size(PartI,1)-2
    for j=1:size(PartI,2)-2
        Gx=sum(sum(X.*PartI(i:i+2,j:j+2)));
        Gy=sum(sum(Y.*PartI(i:i+2,j:j+2)));
        Sobelled(i,j)=sqrt(Gx.^2+Gy.^2); % magnitude
    end
end
Edge_better=PartI;
for i=1:size(PartI,1)-2
    for j=1:size(PartI,2)-2


        Edge_better(i,j)=sum(sum(SecondFilter.*Sobelled(i:i+2,j:j+2)));


    end
end


Sobelled =im2bw(Sobelled);

figure('Name','filtreli');
imshow(Sobelled)
Edge_better =im2bw(Edge_better);

figure('Name','thin');
imshow(Edge_better)


%%
[Final_y,Final_x]=size(Edge_better);
Voter=zeros(1000,1);

maximum_R_theta=0;
for y=1:size(PartI,1)-2
    for x=1:size(PartI,2)/3
        if Edge_better(y,x)==1
            for theta=-pi/6:pi/6

                    R_theta = abs(round(y*sin(theta)+x*cos(theta)));

                        if R_theta>0
                            Voter(R_theta) = Voter(R_theta) + 1 ;
                           if Voter(R_theta)>maximum_R_theta
                               maximum_R_theta=Voter(R_theta);
                           end
                        end


            end
        end
    end
end
```

```
figure;
plot(Voter)
maximum_R_theta;
sqr_length=1;
while(Voter(sqr_length)~=maximum_R_theta)
   sqr_length=sqr_length+1;
end
sqr_length=sqr_length-2 %this is my distance
p=1;
LengthFinder=zeros(1,3);
for i=sqr_length-1:sqr_length+1
   for j=1:Final_y
      LengthFinder(p)=LengthFinder(p)+Edge_better(j,i);

   end
   p=p+1;
end
i=1;
Length=1;

for i=1:3
   if LengthFinder(i)>Length
      Length=LengthFinder(i);
   end
end
Length
Length=round(Length/2)

radius_25=round(Length-2);
radius_50=round(radius_25*1.15);
radius_100=round(radius_25*(2.55/2));
%%


Voter25=zeros(1000,1000);
Voter50=zeros(1000,1000);
Voter100=zeros(1000,1000);


maxi25=0;
maxi50=0;
maxi100=0;


for y=1:size(PartI,1)-1
   for x=1:size(PartI,2)-1
      if Edge_better(y,x)==1
         for theta=0:pi/60:2*pi
            for R_theta=0:radius_100+10
               m = round(x - R_theta*cos(theta));
               n = round(y - R_theta*sin(theta));
                  if (m>0 && n>0)
                     if(R_theta==radius_25-1 )
                     Voter25(m,n) = Voter25(m,n) + 1 ;
                     end
                     if(R_theta==radius_50-2 )
                     Voter50(m,n) = Voter50(m,n) + 1 ;
                     end
                    % if(R<radius_100+0.5 && R>radius_100-0.5)
                     if(R_theta==radius_100+2)
                     Voter100(m,n) = Voter100(m,n) + 1 ;
                     end
                     if Voter25(m,n)>maxi25
                       maxi25=Voter25(m,n);
                     end
                     if Voter50(m,n)>maxi50
                       maxi50=Voter50(m,n);
                     end
                     if Voter100(m,n)>maxi100
                       maxi100=Voter100(m,n);
```

```
                    end
                  end
                         end
                      end
                   end
                end
             end

%%
subplot(3,1,1)
plot(Voter25);
subplot(3,1,2)
plot(Voter50);
subplot(3,1,3)
plot(Voter100);

binary25 = Voter25 > 5*maxi25/6;
binary50 = Voter50 > 10*maxi50/11;
binary100 = Voter100 > 10*maxi100/11;


numb_of_25 = 0;
maximum25=0;
sum_of25=0;
for i=1:1000
      for j=1:1000
         sum_of25=sum_of25+binary25(i,j);
         if maximum25<binary25(i,j)
            maximum25=binary25(i,j);
            y=j;
            x=i;
         end
      end
end
while(sum_of25>0)
   numb_of_25 = numb_of_25+1

   sum_of25
   Index25_x(numb_of_25) = x;
   Index25_y(numb_of_25) = y;
 % roi =[x-20:x+20;y-20:y+20];
   for k = x-20 : x+20
      for j = y-20 : y+20
         A_binary25(k,j) =0;


      end
   end
   sum_of25=0;
   maximum25=0;
   for i=1:1000
      for j=1:1000
         sum_of25=sum_of25+binary25(i,j);
         if maximum25<binary25(i,j)
            maximum25=binary25(i,j);
            y=j;
            x=i;
         end
      end
   end
end

number_of_50 = 0;
maximum50=0;
sum_of50=0;
for i=1:1000
      for j=1:1000
         sum_of50=sum_of50+binary50(i,j);
         if maximum50<binary50(i,j)
            maximum50=binary50(i,j);
```

```
                y=j;
                x=i;
            end
        end
end
while(sum_of50>0)
    number_of_50 = number_of_50+1

    sum_of50
    Index50_x(number_of_50) = x;
    Index50_y(number_of_50) = y;
  % roi =[x-20:x+20;y-20:y+20];
    for k = x-20 : x+20
        for j = y-20 : y+20
            binary50(k,j) =0;
        end
    end
    sum_of50=0;
    maximum50=0;
    for i=1:1000
        for j=1:1000
            sum_of50=sum_of50+ binary50(i,j);
            if maximum50<binary50(i,j)
                maximum50=binary50(i,j);
                y=j;
                x=i;
            end
        end
    end
end


 numberof_100 = 0;
maximum100=0;
sumof_100=0;
for i=1:1000
        for j=1:1000
            sumof_100=sumof_100+ binary100(i,j);
            if maximum100<binary100(i,j)
                maximum100=binary100(i,j);
                y=j;
                x=i;
            end
        end
    end
while(sumof_100>0)
    numberof_100 = numberof_100+1

    sumof_100
    Index100_x(numberof_100) = x;
    Index100_y(numberof_100) = y;
  % roi =[x-20:x+20;y-20:y+20];
    for k = x-20 : x+20
        for j = y-20 : y+20
            binary100(k,j) =0;
        end
    end
    sumof_100=0;
    maximum100=0;
    for i=1:1000
        for j=1:1000
            sumof_100=sumof_100+binary100(i,j);
            if maximum100<binary100(i,j)
                maximum100=binary100(i,j);
                y=j;
                x=i;
            end
        end
    end
end
```

```
%%
Amount_of_money=double(+number_of_50/2+numb_of_25/4+
numberof_100*1);
fprintf('There are  %3.2f TL on the paper',Amount_of_money)
%%
```

%% PART I. Reading the Image in MATLAB

```
U=imread('nonoise1.png'); % An image is taken
B=rgb2gray(U);  % The image is converted to grayscale

fid = fopen('nonoise1.hex', 'wt'); % A document for the pixel data is opened
count=fprintf(fid, '%x\n', B);
disp('Text file write done');disp(' ');
fclose(fid);
% fid=fopen('coinsafter.hex','wt'); % An empty document is created for FPGA
use
% fclose(fid);
% fid=fopen('diameter','wt'); % An empty document is created for FPGA use
% fclose(fid);
%% PART 2. Filtering The Image

C=double(B);

    for i=1:size(C,1)-2
    for j=1:size(C,2)-2
        %Sobel-like mask for x-direction:
        Gx=((4*C(i+2,j+1)+C(i+2,j)+C(i+2,j+2))-(4*C(i,j+1)+C(i,j)+C(i,j+2)));
        %Sobel-like mask for y-direction:
        Gy=((4*C(i+1,j+2)+C(i,j+2)+C(i+2,j+2))-(4*C(i+1,j)+C(i,j)+C(i+2,j)));

        %The gradient of the image
        %B(i,j)=abs(Gx)+abs(Gy);
        T(i,j)=sqrt(Gx.^2+Gy.^2);
    end
    end
figure,imshow(T); title('Sobel gradient');

%%

%Define a threshold value
Thresh=100;
B=max(T,Thresh);
B(B==round(Thresh))=0;
B=uint8(B);
figure,imshow(~B);title('Edge detected Image');

%%
% Text to image conversion
% % Open the txt file
fid = fopen('coinsafter2.hex', 'r');
% Scann the txt file
img = fscanf(fid, '%x', [1 inf]);
% Close the txt file
% restore the imagefclose(fid)

outImg = reshape(img,[320 240]);
figure, imshow(outImg,[])
```