EE 314 Digital Electronics Laboratory
Term Project Final Report

Coin Counter Implementation by Using Verilog HDL on FPGA Development Board

Emre Doğan (*Author*)
Middle East Technical University,
Electrical and Electronics Engineering Department
Ankara, Turkey
dogan.emre@metu.edu.tr

Mert Doğan (*Author*)
Middle East Technical University,
Electrical and Electronics Engineering Department
Ankara, Turkey
mert.dogan@metu.edu.tr

***Abstract*—This report involves the implementation of a coin counter on Verilog HDL by using some filtering, detection algorithms and displaying the total amount of coins with FPGA supported VGA display.**

Keywords—Verilog; FPGA; coin counter; image filtering; edge detection; circle detection; VGA display

I. INTRODUCTION

For the term project of EE314 Digital Electronics Laboratory Course, We were asked to design a coin counter. The system is supposed to take the grey scaled image of coins as input and give the total amount of the money as output by using Verilog, FPGA Development board. A simple block schematic of the overall system is shown in Figure 1.

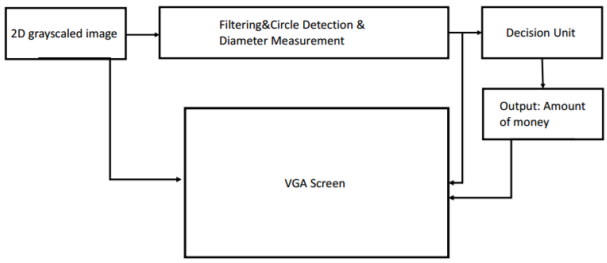

Figure 1.

The simple block schematic of the overall system

Before starting our project, we made a wide literature research and found some fundamental methods to be used in this project:

-Mean filter method

-Sobel edge detection method

-Hough transform for circle detection

-Driving a VGA monitor with an FPGA

After making research in these topics, we first tried to apply these methods in MATLAB, as it would be much easier to implement and simulate the ytem on it. After making the overall system work on MATLAB, we tried to create the same system on Verilog to run it on FPGA. Except a few parts, we could make it work on FPGA with Verilog HDL.

Each part of the project will be explained in detail separately.

II. IMPLEMENTATION WITH VERILOG HDL

A. Converting RGB Image to Grayscale Image

As the camera will give us a color photo, we first need to convert it into grayscale form. For this purpose, MATLAB gives an easy solution with “rgb2gray ()” function that returns the grayscale form of the input image.
Here, after getting the pixel values of the image and converting it into grayscale, we also made a normalization process. To do this we got the highest pixel value and made it equal to the max value 255.
A simple MATLAB code and simulation can be observed below in Figure 2& 3.

```
f=imread('nonoise1.png');
f=rgb2gray(f);
column=240;
row=320;
maxf=max(max(f));
minf=min(min(f));
diff=maxf-minf;

maxg=255;
norm= double(diff)./double(maxg);
for i=1:320;
    for ii=1:240;
        f(i,ii)=f(i,ii)./norm;
    end
end
figure
imshow(f)
fid = fopen('gray.txt', 'wt');
fprintf(fid, '%x\n', f);
fclose(fid);
```

Figure 2.

MATLAB Code to Convert the Image from RGB to Grayscale Form

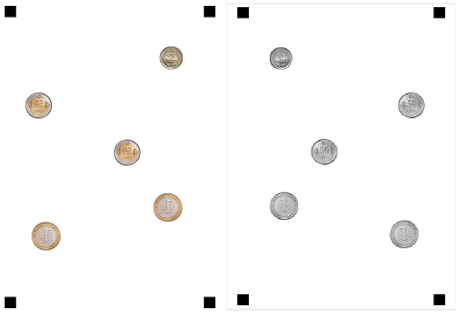

Figure 3. RGB and Gray Scale Converted Images of Sample Coins

In the code above, there is also a part for writing the density values of each pixel in hexadecimals into a ".txt" type file. This document will be helpful for us to process the image in the FPGA board.

B. *Mean Filter*

After getting hex values corresponding to our image in ".txt" file, we can upload these values into FPGA board and process it there. But before using detection algorithms, we wanted to smoothen the image, reducing the amount of intensity variation between one pixel and the next. This application is quite effective to reduce noise in images.
To do this we used a mean filter, which convolves the pixels with a 5x5 matrix with values 1/25 in each row and column shown in Figure 4 represented by K matrix.

$$K = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Figure 4. 5x5 Mean Filter Matrix

For simplicity we decided to use a high pass filter, which makes the lighter areas pure white and the darker areas pure black. A simple Verilog HDL Code for mean filter to be uploaded to FPGA can be observed below in Figure 5.

```
always @(posedge clk_in) begin
  if (x<row*column-1)
     x<=x+1;
  else
     x<=0;

  if(x<2*(row+1))
  filtered[x]<=data[x];


  if((x>row*(column-2)-3)&&(x<row*column))
  filtered[x]<=data[x];


  if((x<row*(column-2)-2)&&(x>2*(row+1)-1))
     begin
  filtered[x]<=(1/25)*(data[x-2*row-2]+data[x-2*row-1]+
  data[x-2*row]+data[x-2*row+1]+data[x-2*row+2]+
  data[x-row-2]+data[x-row-1]+data[x-row]+data[x-row+1]+
  data[x-row+2]+data[x-2]+data[x-1]+data[x]+data[x+1]+
  data[x+2]+data[x+row-2]+data[x+row-1]+data[x+row]+
  data[x+row+1]+data[x+row+2]+data[x+2*row-2]+
  data[x+2*row-1]+data[x+2*row]+data[x+2*row+1]+data[x+2*row+2]);
     end

end
```

Figure 5. Sample Verilog Code for Mean Filter

C. *Sobel Edge Detection Algorithm*

The Sobel detection algorithm determines regions of high spatial frequency that corresponds to edges. Basically, it is used to approximate absolute gradient magnitude for each point of the image.
This operator consists two convolution kernels shown in Figure 6.

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Figure 6. Example Sobel Convolution Kernels for Horizontal and Vertical Axes

These convolution matrices are created such that the system gives the greatest response to the edges on vertical and horizontal axes. These kernels can be applied separately to the

image to get separate measurements for x and y orientation shown in formulas 1&2 below.

$$S_X = \begin{bmatrix} 1 & 0 & -1 \\ 4 & 0 & -4 \\ 1 & 0 & -1 \end{bmatrix} \otimes \mathrm{I} \qquad (1)$$

$$S_Y = \begin{bmatrix} 1 & 4 & 1 \\ 0 & 0 & 0 \\ -1 & -4 & -1 \end{bmatrix} \otimes \mathrm{I} \qquad (2)$$

Where I corresponds to the mean filtered image data.

It can be observed that parameters of Gx and Gy matrices can be changed to adjust the quality of process. In Figure 6, the matrix is created with ±1, ±2 and ±1 where we used ±1, ±4 and ±1 to create this matrix to strengthen the effect of this algorithm.

These separate results can be easily combined as in the formula 3 below.

$$|S_{NET}| = \sqrt{S_X^2 + S_Y^2} \qquad (3)$$

But for simplicity of the system, we can approximate them in such a way,

$$|S_{NET}| = |S_{NET}| = S_X + S_Y \qquad (4)$$

By this approximation done in formula 4, the process is completed faster.

Edge detection algorithm by using Sobel Kernels on Verilog HDL Code can be observed below in Figure 7.

```
module edgeo(outof,clk);
output reg outof;
integer i,ii,iii,iv,v;
integer filee,gradSx,gradSy,gradS,row,column,m;
input clk;
reg [9:0] data [0:76799];
reg [9:0] filtered [0:76799];
reg [9:0] gradOut [0:76799];
initial begin
  column=320;
  row=240;
  $readmemh("filtered.txt", data);

  for(iv=0;iv<column+1;iv=iv+1)begin
    gradOut[iv]<=0;
  end

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

   for (iii=column+1;iii<column*row-column-1;iii=iii+1)
      begin
    gradSy=filtered[iii-column-1]*(1)+filtered[iii-1]*(4)+
    filtered[iii+column-1]*(1)+filtered[iii-column+1]*(-1)+
    filtered[iii+1]*(-4)+filtered[iii+column+1]*(-1);

    gradSx=filtered[iii-column-1]*(-1)+filtered[iii-column]*(-4)+
    filtered[iii-column+1]*(-1)+filtered[iii+column-1]*(1)+
    filtered[iii+column]*(4)+filtered[iii+column+1]*(1);

if (gradSx <0 )
      gradSx=gradSx*(-1);
    if (gradSy <0)
      gradSy=gradSy*(-1);

    gradS=gradSx+gradSy;

    if (gradS<240)
      gradOut[iii]<=0;
    else
      gradOut[iii]<=255;
end
end
```

Figure 7. Sample Verilog Code for Edge Detection Algorithm

In this code, we take the "filtered.txt" file which comes from the mean filter in hexadecimal form. For FPGA Board to read this file, we use "*readmemh*" command meaning "read memory hexadecimal". After this step, we basically apply the classical Sobel Edge Detection Algorithm to the system by making the necessary calculations. Notice that, Sx and Sy are calculated separately and summed up to get net S value approximately.

D. Circle Detection Part

We could not make this part work on Verilog HDL. Related algorithms about circle detection can be seen in MATLAB Simulation part in the following parts.

E. Decision Unit

Due to the problems on circle detection part, the decision unit does not work synchronously with the circuit. But when we assign buttons for each type of coin, we can easily see the increment of total amount of money& total number of coins on FPGA (on 7 segment display).

```
module Filter(bir,elli,yirmibes,clk_in,sseg_temp,sseg_temp2,reset,sseg_temp3,sseg_temp4);
input clk_in;
input bir;
input elli;
input yirmibes;
integer a,b,c,d,e,f,h ;
output reg [6:0] sseg_temp ;
output reg [6:0] sseg_temp2 ;
output reg [6:0] sseg_temp3 ;
output reg [6:0] sseg_temp4 ;
input reset;
initial begin
a=0;
b=0;
c=0;
e=0;
end
always @(posedge bir) begin
a=a+1;
end
always @(posedge elli) begin
b=b+1;
end
always @(posedge yirmibes) begin
c=c+1;
end
always @(posedge clk_in) begin
d = (4*a+2*b+c);
f= (a+b+c) ;
if (d>9 && d<20) begin
e=1;
d=d-10;
end
if (d>19 && d<30) begin
e=2;
d=d-20;
end
if (d>29 && d<40) begin
e=3;
d=d-30;
end
if (d>39 && d<50) begin
e=4;
d=d-40;
end
if (d>49) begin
e=0;
d=d-50;
end
if (f>9 && f<20) begin
h=1;
f=f-10;
end
if (f>19 && f<30) begin
h=2;
f=f-20;
end
if (f>29 && f<40) begin
h=3;
f=f-30;
end
if (f>39 && f<50) begin
h=4;
f=f-40;
end
if (f>49) begin
h=0;
f=f-50;
end
end
```

Figure 8. Sample Verilog Code for Decision Unit-Part1

For our code on Verilog, variable “d” corresponds to the total amount of money where variable “f” corresponds to the total number of coins as it can be seen from Figure 8.

```
always @ (posedge clk_in)
begin
  case(d)
    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(e)
  case(f)
  case(h)
 end
endmodule
```

Figure 9. Sample Verilog Code for Decision Unit-Part2

After the process of decision unit, we need to display our output values on FPGA. With the help of “case” flows, the 7 segment display code for d, e, f and h variables were created as it can be observed in Figure 9.

III. MATLAB SIMULATION

All parts of the system written in Verilog HDL worked successfully on FPGA except the circle diameter detection. But we succeeded to make the overall system work in MATLAB and in this part we will show its code and outputs.

A. Code for Overall System in MATLAB

```
%Taking RGB image and turning it into grayscale form
%normalizing process and writing data of each pixel in hexadecimal form
f=imread('nonoise1.png');
f=rgb2gray(f);
column=240;
row=320;
maxf=max(max(f));
minf=min(min(f));
diff=maxf-minf;

maxg=255;
norm= double(diff)./double(maxg);
for i=1:320;
    for ii=1:240;
        f(i,ii)=f(i,ii)./norm;
    end
end
figure
imshow(f)
fid = fopen('gray.txt', 'wt');
fprintf(fid, '%x\n', f);
fclose(fid);
```

```
%%
%passing the image data through a mean filter and using a threshold pixel
%value such that pixels having lower densities become pure white and the
%ones having greater densities become pure black.
C=zeros(row,column);
for x=3:row-2
 for y=3:column-2
    C(x,y)=f(x-2,y-2)*(1/25)+f(x-1,y-2)*(1/25)+f(x,y-2)*(1/25)+
    f(x+1,y-2)*(1/25)+f(x+2,y-2)*(1/25)+f(x-2,y-1)*(1/25)+
    f(x-1,y-1)*(1/25)+f(x,y-1)*(1/25)+f(x+1,y-1)*(1/25)+f(x+2,y-1)*(1/25)+
    f(x-2,y)*(1/25)+f(x-1,y)*(1/25)+f(x,y)*(1/25)+f(x+1,y)*(1/25)+
    f(x+2,y)*(1/25)+f(x-2,y+1)*(1/25)+f(x-1,y+1)*(1/25)+f(x,y+1)*(1/25)+
    f(x+1,y+1)*(1/25)+f(x+2,y+1)*(1/25)+f(x-2,y+2)*(1/25)+
    f(x-1,y+2)*(1/25)+f(x,y+2)*(1/25)+f(x+1,y+2)*(1/25)+f(x+2,y+2)*(1/25);
  end
end
for xx=3:row-2
  for yy=3:column-2
    if C(xx,yy)<220
        C(xx,yy)=0;
    elseif C(xx,yy)>=220
        C(xx,yy)=255;
    end
  end
end
C(1,:)=255; C(2,:)=255; C(row-1,:)=255; C(row,:)=255; C(:,1)=255;
C(:,2)=255; C(:,column-1)=255; C(:,column)=255;
C=uint8(C);
filterout =int32(C);
figure
imshow(C);
fid2 = fopen('filtered.txt', 'wt');
fprintf(fid2, '%x\n', C);
fclose(fid2);
```

```
%%
%using the black square on the right down as reference so that we do not
%have to put reference radius values for 1TL, 0.50TL and 0.25TL in each
%different image.
for x=1:row;
    for y=1:column;
        if C(x,y)==0;
            refa=x;
            refb=y;
        end
    end
end
ref=0;
for x=1:column-1;
    if (C(refa-3,x)==C(refa-3,x+1) && (C(refa-3,x)==0))
        ref=ref+1;
    end
end
reff=ref/2;
```

```
%%
%edge detection algorithm based on Sobel Kernels and giving the output of
%filtered image in hexadecimal form.
gradOut=zeros(row,column);
for x=1:row
    for y=1:column
        gradOut(x,y)=255;
    end
end
for x=2:row-1
    for y=2:column-1
gradSy=filterout(x-1,y-1)*(-1)+filterout(x,y-1)*(0)+filterout(x+1,y-1)*(1)+
filterout(x-1,y)*(-2)+filterout(x,y)*(0)+filterout(x+1,y)*(2)+
filterout(x-1,y+1)*(-1)+filterout(x,y+1)*(0)+filterout(x+1,y+1)*(1);

gradSx=filterout(x-1,y-1)*(-1)+filterout(x,y-1)*(-2)+
filterout(x+1,y-1)*(-1)+filterout(x-1,y)*(0)+filterout(x,y)*(0)+
filterout(x+1,y)*(0)+filterout(x-1,y+1)*(1)+filterout(x,y+1)*(2)+
filterout(x+1,y+1)*(1);
if gradSx <0
  gradSx=gradSx*(-1);
end
if gradSy <0
  gradSy=gradSy*(-1);
end
gradS=gradSx+gradSy;
if gradS<800
  gradOut(x,y)=255;
else
  gradOut(x,y)=0;
end

    end
end
```

```
for x=1:17
    gradOut(x,:)=255;
end
for x=303:row
    gradOut(x,:)=255;
end
for y=1:26
    gradOut(:,y)=255;
end
for y=215:column
    gradOut(:,y)=255;
end
gradOut=uint8(gradOut);
figure
imshow(gradOut)
fid3 = fopen('edge.txt', 'wt');
fprintf(fid3, '%x\n', gradOut);
fclose(fid3);
```

```
%%
%circle detection part and decision unit for the amount of money and number
%of coins
r25=reff-2;
r50=reff-0.5;
r100=reff+1;
gradOut50=gradOut;
gradOut100=gradOut;
gradOut25=gradOut;

for x=1:row
        for y=1:column
          if gradOut25(x,y)==0
         gradOut25(x,y)=255;
          for theta=0:360
                a=ceil(x+r25*sind(theta));
            if a<=0
                a=1;
            end
                      b=ceil(y+r25*cosd(theta));
                      if b<=0
                b=1;
            end
            if b>column
                b=column;
            end
            if a>row
                a=row;
            end
            gradOut25(a,b)=gradOut25(a,b)-1;
        end
      end
        end
end
for x=1:row
         for y=1:column
     if gradOut25(x,y)==0
       gradOut25(x+1,y)=255;
       gradOut25(x-1,y)=255;
       gradOut25(x,y-1)=255;
       gradOut25(x,y+1)=255;
       gradOut25(x+1,y+1)=255;
       gradOut25(x+1,y-1)=255;
       gradOut25(x-1,y-1)=255;
       gradOut25(x-1,y+1)=255;
     end
     if gradOut25(x,y)==0
       money25=money25+1;
     end
    end
end
for x=1:row...
for x=1:row...
for x=1:row...
for x=1:row...
for x=1:row...
for x=1:row...
figure
imshow(gradOut25);

figure
imshow(gradOut50);

figure
imshow(gradOut100);

coins=money100+money50+money25;
money=1*money100+0.50*money50+0.25*money25;
fprintf('Total Money is = %f tl\n',money);
```

B. *Simulation Results*

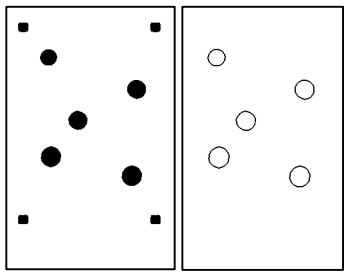

Figure X.

Edge Detected Coins and Corner Square Elimination

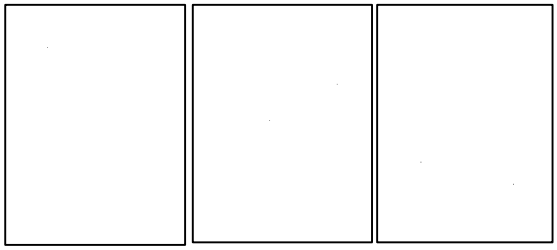

Figure X. Circle Diameter Detection Results

(Showing the centers of 1TLs, 0.5TLs and 0.25TLs separately)

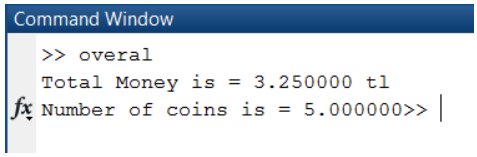

Figure X. Total Amount and Number of Coins

IV. SUMMARY

In this project, we had a deep experience on Verilog HDL. In the experiments, we dealt with much easier systems so after implementing our system on MATLAB, it became a real challenge for us to implement it on Verilog and run it on FPGA. With the help of the internet and our own knowledge, we succeeded to create the overall system with Verilog HDL except the diameter detection part.

Another important point of this project was that for the first time in a lab course, we were asked to use “Image Processing” concepts. It was a good experience for us but we also observed that algorithms due to these concepts are not very suitable to use with FPGA. With Verilog, some parts of the system were compiled in almost 50 minutes where this duration is much smaller in modern simulation platforms like MATLAB.