EE314 Term Project- FPGA Based Point of Sale Terminal (July 2021)

Erdem Şimşek, Beste DUMLUPINAR, Ömer ŞEN, Mustafa YILDIRIM, Senanur KEKEÇ

***Abstract*— This term project paper is based on designing a basic POS, point of sale, terminal with FPGA board. It was expected to construct a screen on the computer that includes twelve items with their images, barcodes, and unit costs in that design. Then, a shopping list displays the selected items, which is at most six, names and its quantities, which is at most four. At the end of the shopping list, there is also a full-cost display with seven segments. Moreover, there are two different selection types; the first one works with entering the barcode of an item and its quantity with assigned buttons on FPGA. The second one is navigation through the items again with assigned buttons for going up, left, right and select operations. While selecting through barcode, items with entered barcode values up to that point have hovered until all four digits are completed. At the end, only one item with the entered barcode is hovered. In terms of selection with navigation, there is a yellow highlight around the current item which goes around the items as customer navigate with assigned buttons. These selections work is SW-1 is HIGH. If SW-2 is HIGH, one can navigate in the shopping list and delete the selected item which has a red cursor on its left. While this cancel operation total cost is updated as well as the list. Hence, a POS terminal is designed with expected requirements as explained.**

I. Introduction

As the term project of Digital Electronics Laboratory Course, we were expected to design a point of sale terminal. POS terminal is an application in which the customers can choose a product and a quantity for the product by means of buttons on the terminal. In our project, we are also required to implement interactive modes for the POS.

In our design, we have used the FPGA board and Verilog hardware language to implement the POS application, and the VGA interface to display it. Since we had not had enough knowledge of Verilog and FPGA implementation, we have researched a lot to be able to decide the path and meet the requirements. We have learnt a lot about the usage of FPGA and Verilog programming while working on the project.

We have divided the project into sub-blocks. These blocks are display of the items in main menu and chosen products in the shopping list and VGA synchronization, selecting a product and its quantity via barcode reading, selecting a product and its quantity navigating through the main menu, lastly filling and cancelling items in the shopping list.

II. RESULTS

A. Display

To display images, various indicators, numbers, and text on the screen using VGA, we used counters, namely *counterx* and *countery* registers, for synchronization of horizontal and vertical pixels. Firstly, by dividing 50 MHz of the FPGA Board to 25 MHz, every positive edge of 25 MHz, duty cycle *counterx* and *countery* is increased. Then, *counterx* scan the one horizontal axis by increasing one, at the end of the line countery is increased and it starts to scan the next line. In this way, every pixel on the 640x480 screen can be scanned.

```
always @(posedge clk25)
  begin
    if(counterx == 799)
    begin
      counterx <= 0;
      if(countery == 524)
        countery <= 0;
      else
        countery <= countery+1'b1;
    end
    else
      counterx <= counterx+1'b1;


  if (countery >= 490 && countery < 492)
    vsync <= 1'b0;
  else
    vsync <= 1'b1;

  if (counterx >= 656 && counterx < 752)
    hsync <= 1'b0;
  else
    hsync <= 1'b1;
  end
```

Fig. 1. Always block that is used for scanning every pixels on the 640x480 screen.

By defining variables and assigning specific areas with counterx and countery, the positions of images, lines, words, or numbers can be determined. The screen color at idle state is black. To determine the color for the indicators, Defined variables are assigned to VGA's RGB (Red, Green, Blue) input pins. For different combinations of the RGB inputs, different colors can be obtained.

To display an image, the *readmemh* function is used. This function reads the *.list* files of the desired images and the data in the .list files are assigned to the images' registers. The data in the. list files are stored as 8-bit hexadecimal. Every pixel on the image is converted to hexadecimal on MATLAB and the readmemh function reads these hexadecimal codes.

```
$readmemh("imagelist/avo.list",avo);
```

Fig. 2. The readmemh function for displaying images.

By defining an area with counterx and countery, the place of the image is determined. State counts the array elements of the area with clk25 cycles. Then, every pixel of the image stored in the images' registers are assigned to the color register by the help of counting state. In other words, the state points the elements of the image array one by one. At the end of the code, color's 8 bit (3 bit red, 3 bit green and 2 bit blue) are assigned to VGA's RGB input pins.

```
always @ (posedge clk25) begin
  if (counterx > 199 && counterx < 300 && countery > 139 && countery < 240) begin
    state <= (counterx-200)*100+countery-140;
    color <= avo[{state}];
  end
```

Fig. 3. Always block for definition of area to display items' images.

```
wire R=color[0]||color_text0[0]||color_text1[0]||color_text2[0]||
wire R1=color[1]||color_text0[1]||color_text1[1]||color_text2[1]||
wire R2=color[2]||color_text0[2]||color_text1[2]||color_text2[2]||
wire G=color[3]||color_text0[3]||color_text1[3]||color_text2[3]||
wire G1=color[4]||color_text0[4]||color_text1[4]||color_text2[4]||
wire G2=color[5]||color_text0[5]||color_text1[5]||color_text2[5]||
wire B=color[6]||color_text0[6]||color_text1[6]||color_text2[6]||
wire B1=color[7]||color_text0[7]||color_text1[7]||color_text2[7]||

// vga input rgb pins are assigned in here
always@(posedge clk25) begin
    vgaR <= R;
    vgaR1 <= R1;
    vgaR2 <= R2;
    vgaG <= G;
    vgaG1 <= G1;
    vgaG2 <= G2;
    vgaB <= B;
    vgaB1 <= B1;
end
```

Fig. 4. Assignments of color bits to VGA's RGB input.

B. *Selection of Items by Barcode*

In this part of the project, we built up selecting a product with barcode mod. Initially we constructed a *buttonvalue* register which contains 3 bits to store barcode numbers. To determine which decimal, we were dealing with, we created a 4 bits *checkfordecimal* register. By the help of previous two registers, we were able to put the button values to the correct decimal places. To put first number to thousands digit, we multiplied first barcode input number with 1000, second barcode number with 100, third barcode number with 10, and last barcode number with 1 and summed up all of them. To prevent the failure that assigning all digits with first digit due to quickness of clock, we added *checkbutton* register which becomes one when the button on the board is pressed and became 0 when the finger on the button was pulled out. Since all if's require *checkbutton==0* condition, previously mentioned failure is prevented.

```
 always@(posedge clk5) begin

if(button_1 == 0)begin
buttonvalue = 3'b001;
end

else if(button_2 == 0)begin
buttonvalue = 3'b010;
end

else if(button_3 == 0)begin
buttonvalue = 3'b011;
end

else if(button_4 == 0)begin
buttonvalue = 3'b100;
end

else if ((button_1&&button_2&&button_3&&button_4)) begin
buttonvalue = 3'b000;
end

end
```

Fig. 5. Always block for *buttonvalue* assignment.

```
else if ((buttonvalue == 3'b000) && (sw1==0) && (sw2 ==0)) begin
checkbutton <= 0;
quantity <= 7'b0000000;
end
```

Fig. 6. Else if statement for quantity check.

```
always@(posedge clk5) begin

if(((sw1==0) && (sw2==0)) && (checkfordecimal[3]== 0) && (checkbutton==0) && ( buttonvalue!=3'b000 )) begin

barcode <= barcode + 1000*buttonvalue;
checkfordecimal[3] <= 1;
checkbutton <= 1;
end


else if(((sw1==0) && (sw2==0)) && (checkfordecimal[2]==0) && (checkbutton==0) && ( buttonvalue!=3'b000 )) begin

barcode <= barcode+(100*buttonvalue);
checkfordecimal[2] <= 1;
checkbutton <= 1;
end


else if(((sw1==0) && (sw2 ==0)) && (checkfordecimal[1]==0) && (checkbutton == 0) && ( buttonvalue!=3'b000 )) begin

barcode <= barcode+(10*buttonvalue);
checkfordecimal[1] <= 1;
checkbutton <= 1;
end
```

Fig. 7. Always block for barcode assignment with *checkfordecimal*.

```
wire avokado_e1 =((counterx>196 && counterx<200) && (countery>136 && countery<243)) ;
wire avokado_e2 =((counterx>299 && counterx<303) && (countery>136 && countery<243));
wire avokado_e3 =((countery>239 && countery<243) && (counterx>196 && counterx<303));
wire avokado_e4 =((countery>136 && countery<140) && (counterx>196 && counterx<303));

wire avokado_hl_corners;
assign avokado_hl_corners = avokado_e1 || avokado_e2 || avokado_e3 || avokado_e4 ;

wire cilek_e1 =((counterx>306&&counterx<310) && (countery>136&&countery<243));
wire cilek_e2 = ((counterx>409&&counterx<413) && (countery>136&&countery<243));
wire cilek_e3 = ((countery>239&&countery<243) && (counterx>306&&counterx<413));
wire cilek_e4 = ((countery>136&&countery<140) && (counterx>306&&counterx<413));

wire cilek_hl_corners;
assign cilek_hl_corners = cilek_e1 || cilek_e2 || cilek_e3 || cilek_e4 ;
```

Fig. 8. Border assignments of items.

According to the barcode numbers, the products are highlighted. By using cases of barcode, the situation of *hl_{fruitnames}* and highlight according to it can be checked. The line at the border of the products is drew by defining areas with *counterx* and *countery* variables. Those lines named as *{fruitnames}_hl_corners.* Then, they are AND'ed as *hl_{fruitnames}* and *{fruitnames}_hl_corners* to the wire created as *theonewire* wire. This wire controls pixels of borders of fruits.

```
case (barcode)

32'd4000:begin
hl_avokado= 1;  hl_cilek= 1; hl_domates= 1; hl_patates= 1;
hl_ananas= 0; hl_elma= 0;  hl_seftali= 0;  hl_muz= 0;
hl_coconut= 0; hl_hiyar= 0; hl_karpuz= 0;hl_nar= 0; end
32'd4100:  begin
hl_avokado= 0;  hl_cilek= 0; hl_domates= 1; hl_patates= 1;
hl_ananas= 0; hl_elma= 0; hl_coconut= 0; hl_hiyar= 0;
 hl_karpuz= 0;hl_nar= 0; hl_seftali= 0;  hl_muz= 0; end
```

Fig. 9. Case block for selection with entering barcode.

When the first 4 input were provided, the barcode number was completed, and quantity of the product was gotten. At this point, a 7 bits register was created which is *quantity* to store internal number of the product and quantity of it. First 4 bits holds internal number of the product, and last 3 bits holds quantity of it which is taken as fifth input at the mode. To restart the process the information is needed to be kept in the *quantity* register and reset it. To keep the information, the information was assigned to the *quantity* register to the register called *sepet{number}*, which will be explained later. After the transfer of the info, the *quantity* was made all zero when the finger was pulled back from the button; so that, register became ready for the next iteration.

```
else if( ((sw1==0) && (sw2==0)) && (checkfordecimal==4'b1111) && (checkbutton == 0) && ( buttonvalue!=3'b000 )) begin


case(barcode)

32'd4432: begin
quantity <= 7'b0001000 + buttonvalue;
end

32'd4431: begin
quantity <= 7'b0010000 + buttonvalue;
end
32'd4133: begin
quantity <= 7'b0011000 + buttonvalue;
end
32'd4132: begin
quantity <= 7'b0100000 + buttonvalue;
end
```

Fig. 10. Case block for the assignment of quantity according to barcode.

C. *Selection of Items by Navigation in Main Menu*

When switch 1 is high, shopping in the main menu via buttons is activated. We have utilized three of the buttons for navigating between the products. The remaining button is used to select the item that is currently hovered. The indication of the currently hovered item is done by highlighting the surrounding of the item with a yellow square. Therefore, the customer can understand which item she/he is choosing. It has been decided to use the leftmost button KEY[3] and the rightmost button KEY[0] of FPGA board for going in the left and right directions respectively. In the vertical direction, only one button KEY[1] was used, which goes down. If the currently hovered item is in the last row of the product list, the customer can go back to the first row of the list with the help of KEY[1]. The last button KEY[2] is used to select the currently hovered item. If a customer presses this button, the item is selected and added to the shopping list which is not displayed yet. After an item is selected, the function of the buttons is changed. Then, the customer needs to choose a quantity for the selected product using the same buttons. As indicated in the project description, at most four pieces from the selected product can be bought. It has been decided to use KEY[0] (rightmost button) as 1 and it is increased up to four going to the left where KEY[3] (leftmost button) being 4 in terms of the quantity. After the quantity is chosen, the customer is able to continue to navigate in the main menu and select a new product directly.

D. Shopping List

In this part of the project a shopping list was constructed. Initially six different 7 bits registers were created as *sepet{number}*. Every register represents one segment of the shopping list which can store 6 products maximum. To fill them one by one, *sepetcount* six bits register was utilized which assigns quantity information in an order. At the same time, in order not to assign one quantity value to all *sepet* register *sepetcheck* was used which works in the same logic as *checkbutton* register. *Sepetcheck* becomes zero when *quantity* register is reset. *Zero==sepetcheck* is required condition to fill the segment of the shopping list.

```
else if((sw2==0)&&(sepetcheck == 0)&&(quantity!=7'b0000000)&&(sepetcount==6'b011111)) begin

sepet6 <= quantity;
sepetcount[5] <= 1;
sepetcheck <= 1;
end

else if ((quantity == 7'b0000000)&&(sw2==0)) begin
sepetcheck <= 0;
end



else if((sw2==1)&&((button_1&&button_2&&button_3&&button_4)==1))begin
deletecheck <=0;
end
```

Fig. 11. Else if statements for sepetcheck.

```
////Prices of every segment of shopping list
always@(posedge clk5) begin
case(sepet1[6:3])

4'b0001:begin
 sepet1value = 800*sepet1[2:0];
 end

4'b0010:begin
 sepet1value = 100*sepet1[2:0];
 end
```

Fig. 12. Case statements for the assignments of *sepetxvalue*s.

```
always@(posedge clk5) begin
if((sw2==0)&&(sepetcheck == 0)&&(quantity != 7'b0000000)&&(sepetcount==6'b000000)) begin

sepet1 <= quantity;
sepetcount[0] <= 1;
sepetcheck <= 1;
end

else if((sw2==0)&&(sepetcheck == 0)&&(quantity!=7'b0000000)&&(sepetcount==6'b000001)) begin

sepet2 <= quantity;
sepetcount[1] <= 1;
sepetcheck <= 1;
end
```

Fig. 13. Assignment of *quantity* to each *sepet*x.

```
always@(posedge clk5) begin
total <= sepet1value + sepet2value + sepet3value + sepet4value + sepet5value + sepet6value ;
end
```

Fig. 14. Always block for adding up *sepetxvalue*s to obtain total price.

A register was created which was *sepet_cursor* to indicate chosen segment of the shopping list, first one, second one or last one etc. Also, *sepetcounter* was created to keep the information how many segments of the shopping list were filled. According to the information of *sepetcounter*, a range of motion of cursor was decided. The button3 was assigned as delete or cancel button. It deletes the information the segment that cursor indicates. If the position of the deleted item is known and how many items stored in the shopping list, *sepetcounter* holds it, the items can be shifted below the deleted one, if there is any. All the cases were considered of shifting with respect to how many segments were filled and deleted segment. Every shifting case was considered by *case* block. The cursor was constructed as a red bar that was taking place of the left of the shopping list. The cursor moves with respect to the value of *sepet_cursor* which holds number between 1 and 6. The cursor is not displayed when there are no items in the list(*sepet_counter==0*).

```
else if ((sw2==1)&&(sepet_cursor==3'b001)&&(button_1==0)&&(deletecheck==0)) begin

case(sepetcounter)

3'b001: begin

sepet1 <= 7'b0000000;
sepetcount <= 6'b000000;
deletecheck <=1;
end

3'b010: begin

sepet1 <= sepet2;
sepet2 <= 7'b0000000;
sepetcount <= 6'b000001;
deletecheck <=1;

////
end
```

Fig. 15. Case block for deleting the selected item and updating the list.

Moreover, a total price for the items in the shopping list was established. First 4 bits of *Sepet{number}* indicates the item and last 3 bits indicates quantity of it. Each price of the segment was calculated by multiplying *sepet{number}value* with the prize of stored item. The prize is decided according to the first four bit of *sepet{number}*. Finally, to obtain total, all *sepet{number}value*s were added up.

III. Conclusion

It can be concluded that, FPGA boards are useful devices that enable us to perform various operations by using its features including VGA interface, clock, and pins. Due to this ability of the FPGAs, we could construct a simple and straightforward POS device that can be used in shopping places, since it displays items and enables customer to select them or cancel the selected one or ones. However, working on FPGA and designing a project or task requires exclusive research on its features and enough competence on hardware programming language.

APPENDIX

Fig. 5. The screenshot of the designed POS when SW-1 is HIGH and selecting is by navigation.

Fig. 6. The screenshot of the designed POS when SW-1 is HIGH and selecting is by barcode entering, all digits are entered.

Fig. 7. The screenshot of the designed POS when SW-1 is HIGH and selecting is by barcode entering, just one digit entered.

Fig. 8. The screenshot of the designed POS when SW-2 is HIGH, the shopping list is available.

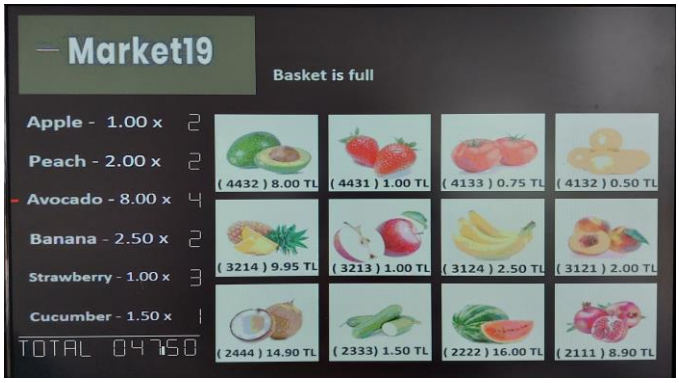

Fig. 9. The screenshot of the designed POS when SW-2 is HIGH, at most six items can be selected.

REFERENCES

[1] A. Bulut, “FPGA Project”, Feb. 11, 2020. [Online]. Available: https://github.com/anilcanbulut/FPGA-Project

[2] D. Meads, “Quartus Projects”, Nov. 9, 2020. [Online]. Available: https://github.com/dominic-meads/Quartus-Projects/blob/main/VGA_face/smiley_test.v

[3] G. Pacchiella, “Implementing VGA Interface with Verilog”, Jan. 23, 2018. [Online]. Available: https://ktln2.org/2018/01/23/implementing-vga-in-verilog/

[4] Intel, “How to Program Your First FPGA Device”, March 24, 2017. [Online]. Available: https://software.intel.com/content/www/us/en/develop/articles/how-to-program-your-first-fpga-device.html

[5] M. Das, “Image from FPGA to VGA”, Instructables Circuits. [Online]. Available: https://www.instructables.com/Image-from-FPGA-to-VGA/

[6] R. Singh, “Simple VGA Design Example for Telesto”, Numato Lab, March 14, 2018. [Online]. Available: https://numato.com/kb/simple-vga-design-example-for-telesto/