Fadahunsi Adeife

Section 4

CPR E 281

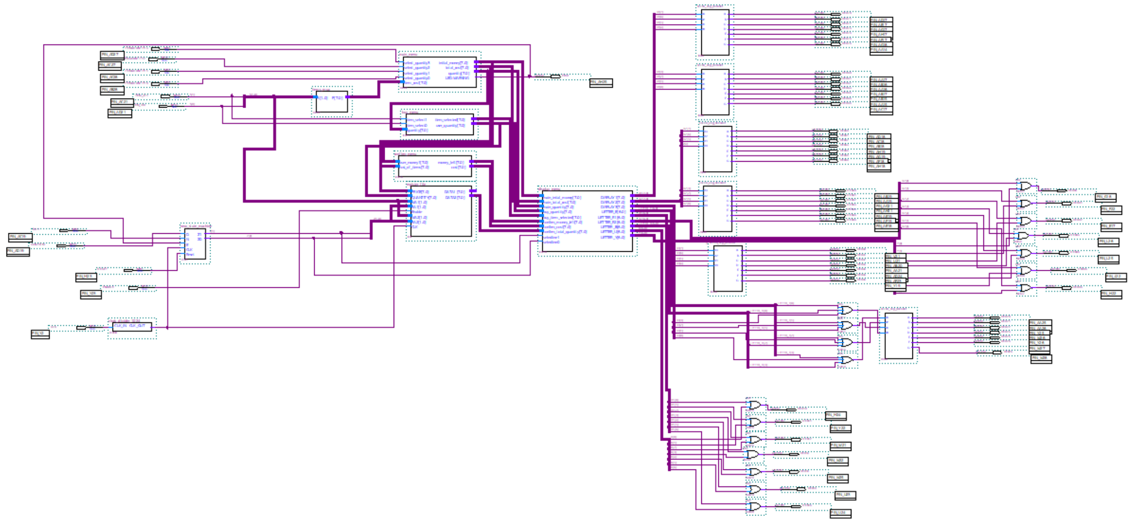
372469158

FINAL PROJECT REPORT

I completed project option number 3 - Vending Machine. This report contains details of all the workings and derivations required to build the circuit and Verilog modules contained in the project. It contains a general overview of every top-level module, followed by a multilevel description of all the inner workings of each module such as the register file, arithmetic logic unit, and finite state machine.

TOP-LEVEL DIAGRAM:

The design of the vending machine is shown in the picture below



The design has the following inputs;

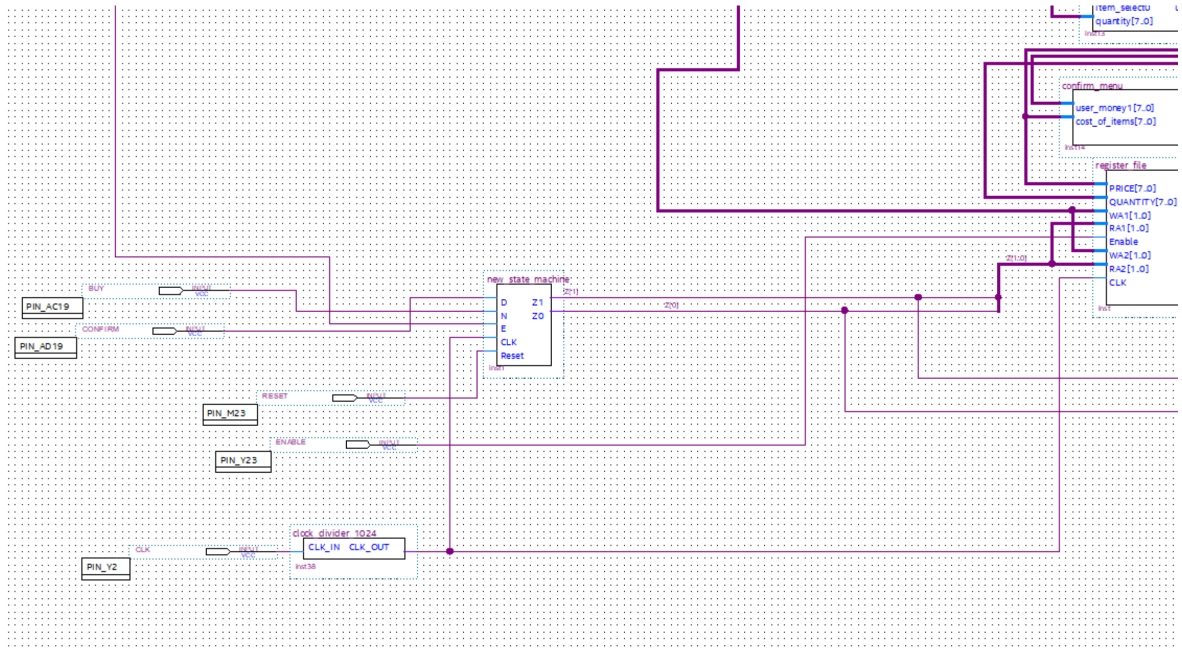
- The ITEMQUANTITY3, ITEMQUANTITY2 ITEMQUANTITY1, ITEMQUANTITY0 are the four bits input the user uses to select the quantity.
- The ITEM_ID1, and ITEM_ID0 are the two bits input the user uses to select the item.
- BUY input, which is a push button for the user to buy a quantity of an item
- CONFIRM input, which is a push button for the user to confirm the quantity of the item to be bought
- RESET, which would reset finite state machine to the menu screen which is its first state.
- CLK, which is set to the 50Mhz clock from the FPGA and connected to clock_divider_1024.
- ENABLE, which would enable or disable the functioning of the register file.
- SUB, this is to set the operation in the confirm_menu to Subtraction by setting the switch to one.

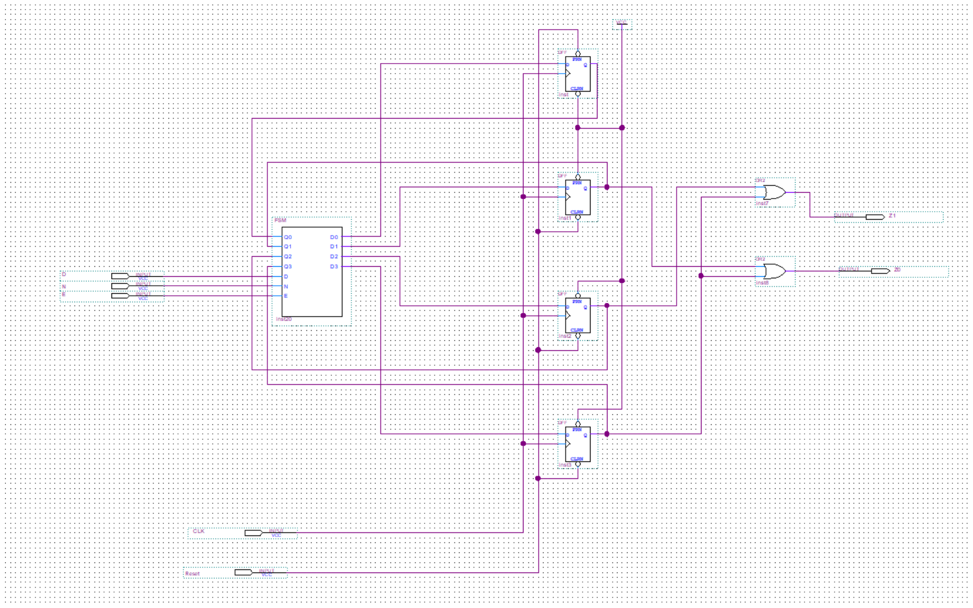
The design has the following outputs;

- Error, which is an output from the main menu which calculates the total cost of transaction gives an Error output of 1 if the user attempts to make an unaffordable transaction
- 7 bits output at displays HEX7, HEX6, HEX5, HEX4, HEX3, HEX2, HEX1, and HEX0 to display the menu screen, buy screen, confirm screen and error screen.

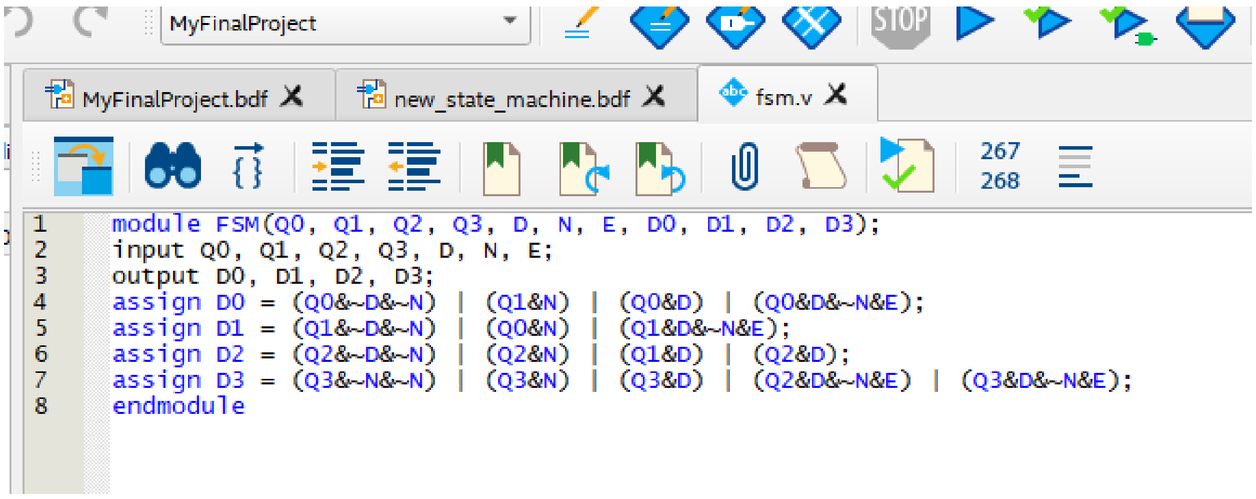
FINITE STATE MACHINE:

The finite state machine is consisted of D-flip flops to store the value of the present states that are used as inputs to determine the next state. The state machine also includes combinational circuits to express the relationship between the current and the next state in the FSM. The following shows a sketch.





The FSM Block Component in the above picture is a Verilog code to write the expressions for the next state variables. The following picture shows the Verilog Code;



```
module FSM(Q0, Q1, Q2, Q3, D, N, E, D0, D1, D2, D3);
input Q0, Q1, Q2, Q3, D, N, E;
output D0, D1, D2, D3;
assign D0 = (Q0&~D&~N) | (Q1&N) | (Q0&D) | (Q0&D&~N&E);
assign D1 = (Q1&~D&~N) | (Q0&N) | (Q1&D&~N&E);
assign D2 = (Q2&~D&~N) | (Q2&N) | (Q1&D) | (Q2&D);
assign D3 = (Q3&~N&~N) | (Q3&N) | (Q3&D) | (Q2&D&~N&E) | (Q3&D&~N&E);
endmodule
```

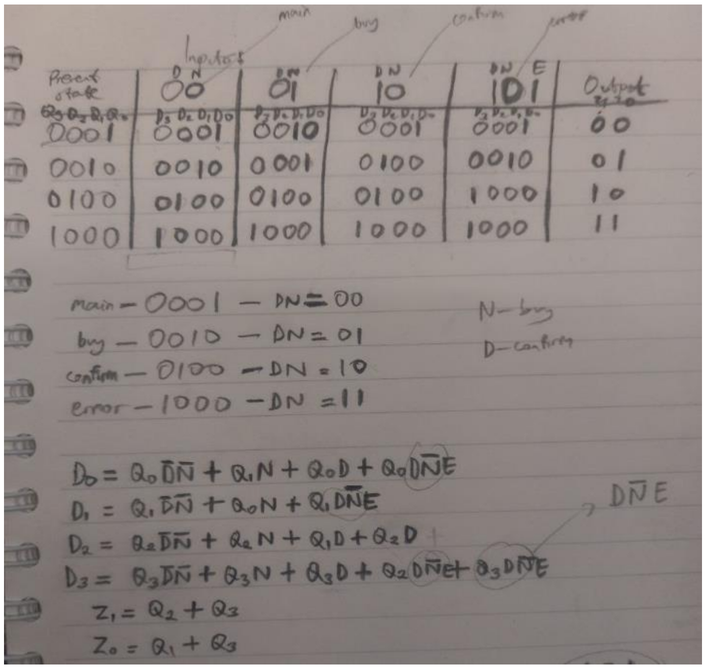
The state assigned table is shown below with the use of one-hot encoding except for the condition of Error which is 101, and this is when the user presses the confirm button when Errror is 1.

0001 shows the menu screen, 0010 shows the buy menu when the buy button is pressed, 0100 shows the confirm screen when the confirm button is pressed, and 1000 shows the error screen when the user presses the confirm button while error is one. Output of the FSM is two bits which transitions through the states depending on the input of the user. The inputs of the are D, N and E, CLK and RESET.

D is the Confirm signal

N is the Buy signal

E is the Error signal


Inputs
main
buy
confirm
error

Present state $Q_3Q_2Q_1Q_0$	DN 00 $D_3D_2D_1D_0$	DN 01 $D_3D_2D_1D_0$	DN 10 $D_3D_2D_1D_0$	DN E 101 $D_3D_2D_1D_0$	Output Z_1Z_0
0001	0001	0010	0001	0001	00
0010	0010	0001	0100	0010	01
0100	0100	0100	0100	1000	10
1000	1000	1000	1000	1000	11

main – 0001 – DN = 00

buy – 0010 – DN = 01

confirm – 0100 – DN = 10

error – 1000 – DN = 11

N – buy

D – confirm

$$D_0 = Q_0\overline{D}\overline{N} + Q_1N + Q_0D + Q_0D\overline{N}E$$

$$D_1 = Q_1\overline{D}\overline{N} + Q_0N + Q_1D\overline{N}E$$

$$D_2 = Q_2\overline{D}\overline{N} + Q_2N + Q_1D + Q_2D$$

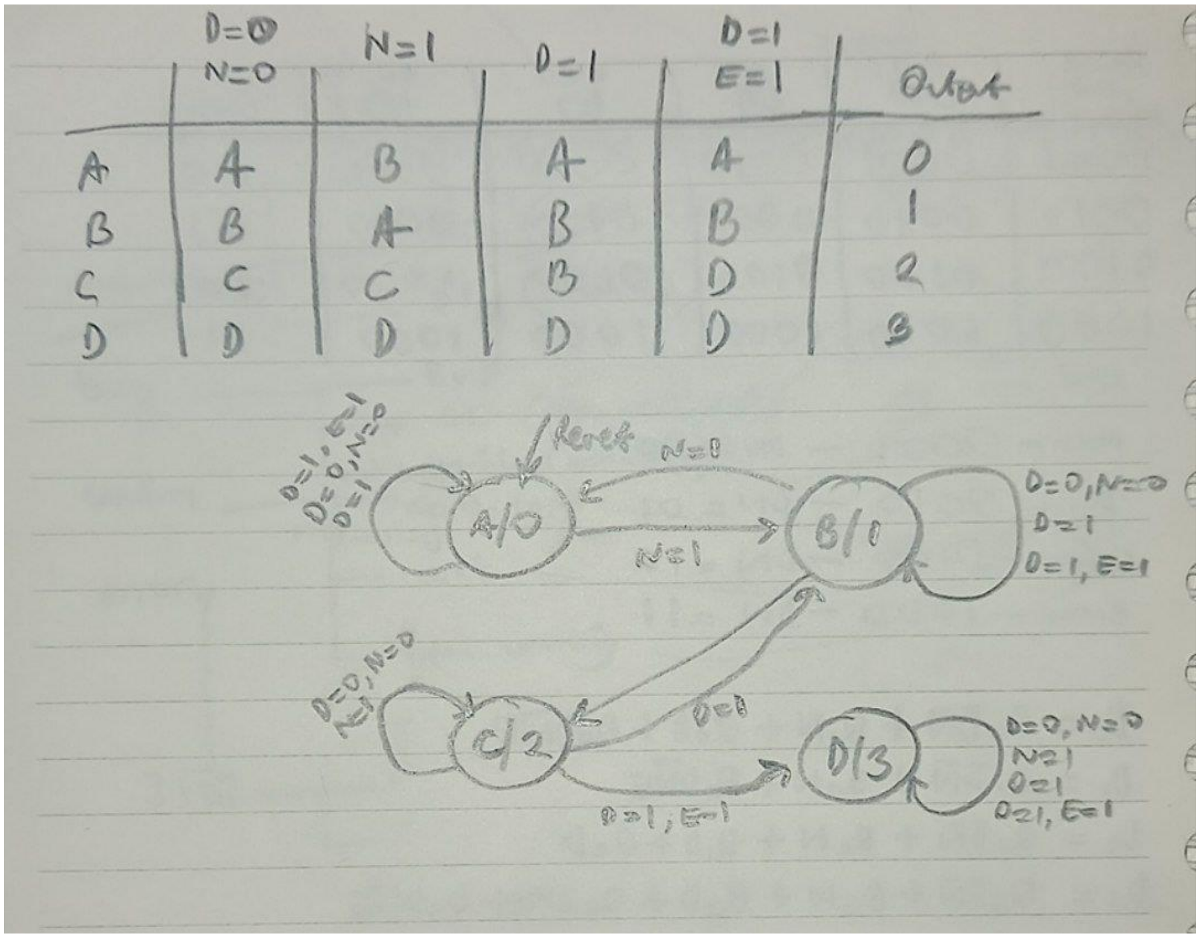
$$D_3 = Q_3\overline{D}\overline{N} + Q_3N + Q_3D + Q_2D\overline{N}E + Q_3D\overline{N}E$$

$D\overline{N}E$

$$Z_1 = Q_2 + Q_3$$

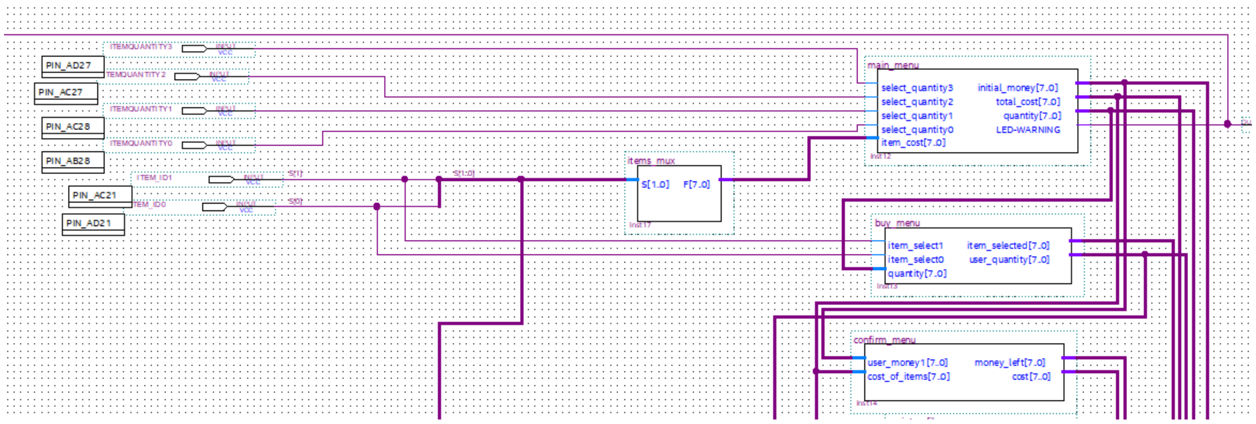
$$Z_0 = Q_1 + Q_3$$

The State diagram of the Finite State Machine is shown below;



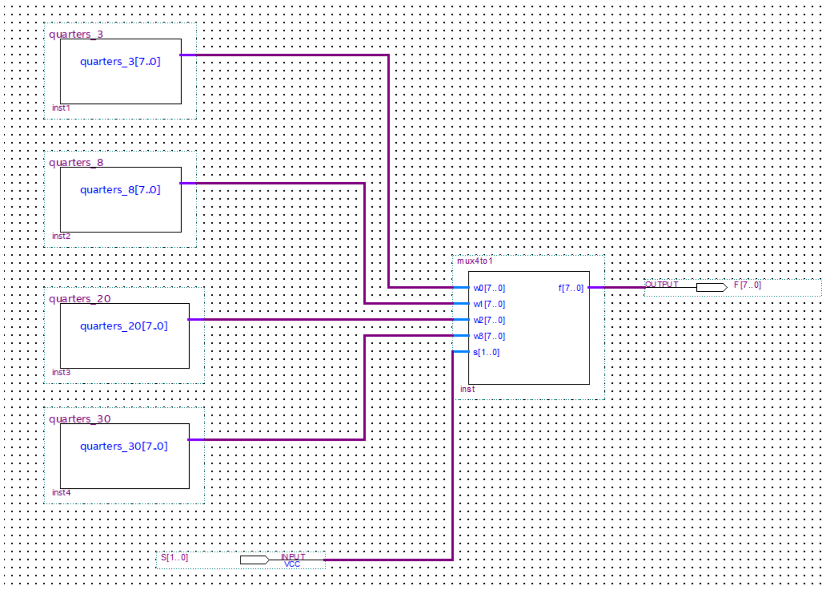
	D=0 N=0	N=1	D=1	D=1 E=1	Output
A	A	B	A	A	0
B	B	A	B	B	1
C	C	C	B	D	2
D	D	D	D	D	3

ALU OR ARITHMETIC LOGIC UNIT:

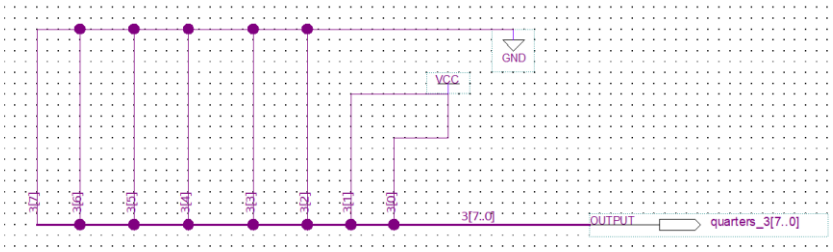


- **ITEMS-MUX**

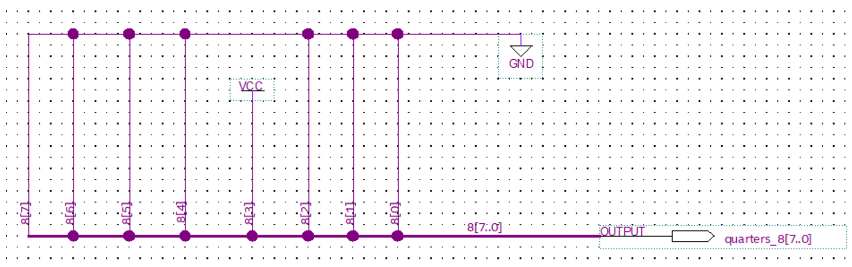
It starts with the **items_mux** which is a schematic diagram that contains the hard code of the 8 binary bits of the number of quarters for each item and acts as a 4 to 1 multiplexer that chooses which item ID to select based on the user input from ITEM_ID1, and ITEM_ID0 and give that as its output. The following picture shows the Block diagram of items_mux.



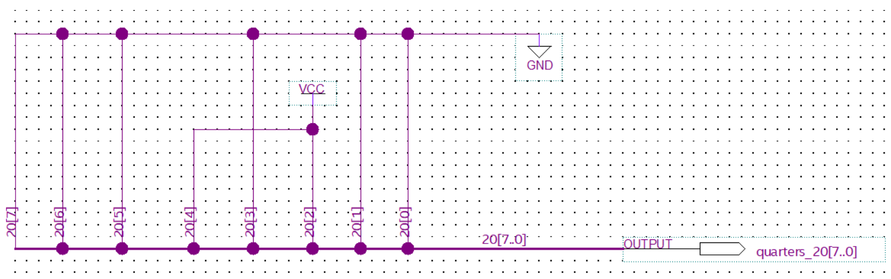
ITEM ID 0 is 3 quarters



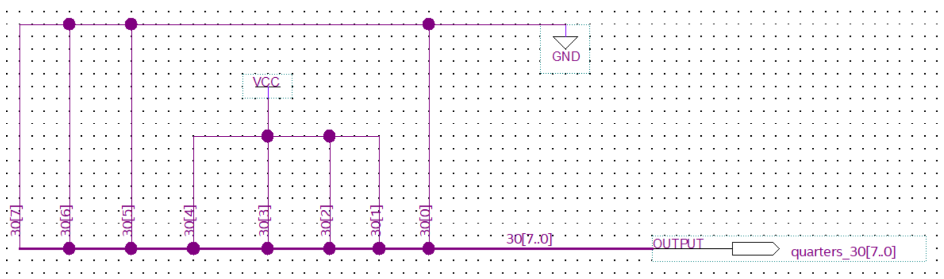
ITEM ID 1 is 8 quarters



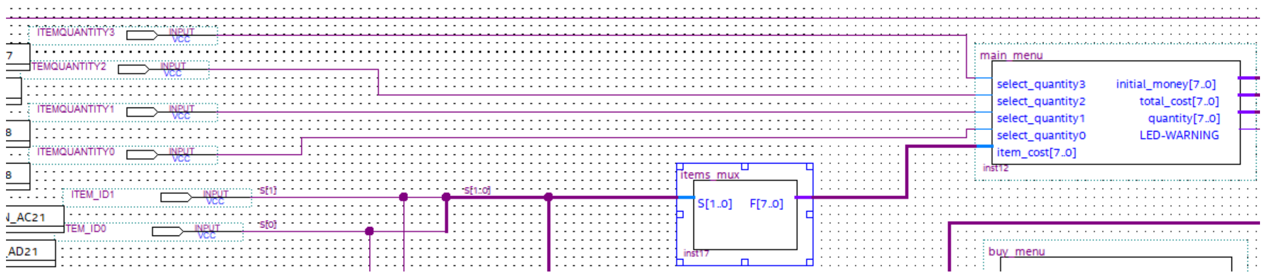
ITEM ID 2 is 20 quarters



ITEM ID 3 is 30 quarters

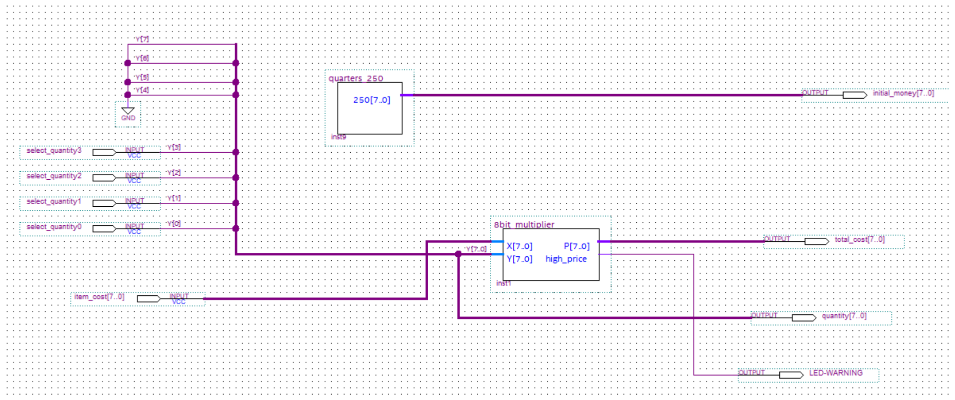


- MAIN-MENU

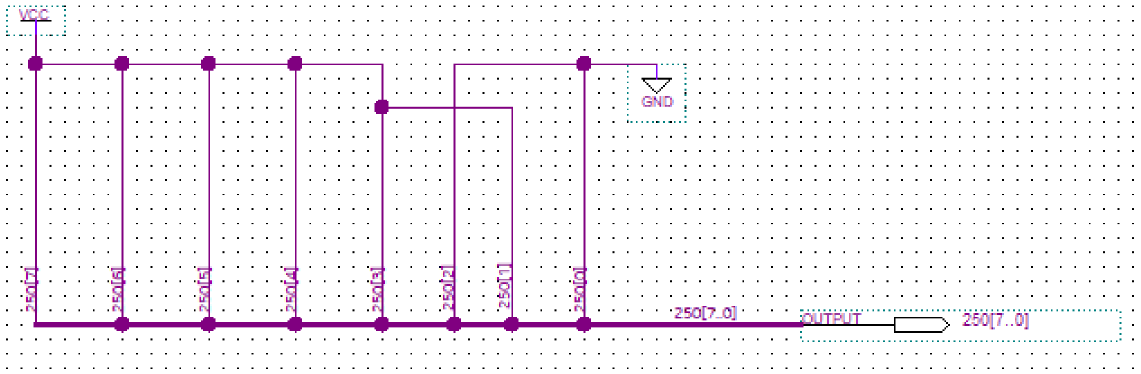


The inputs of the main menu are the 4 bits input from the user to select the quantity of the item to purchase and the 8 bits single cost of the item from the items_mux block diagram.

The sketch of the main-menu is shown below;

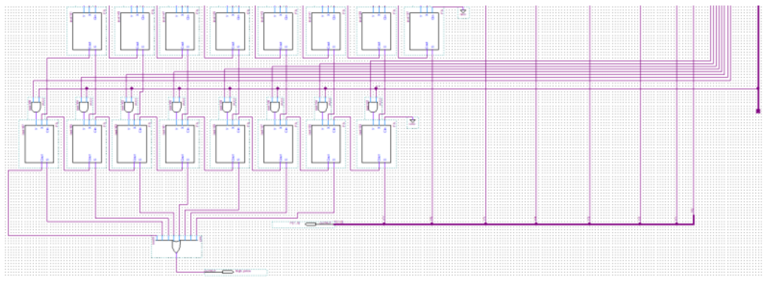


The money the user starts with is 250 quarters. Here in the main menu, the 250 output is hard coded as the following;

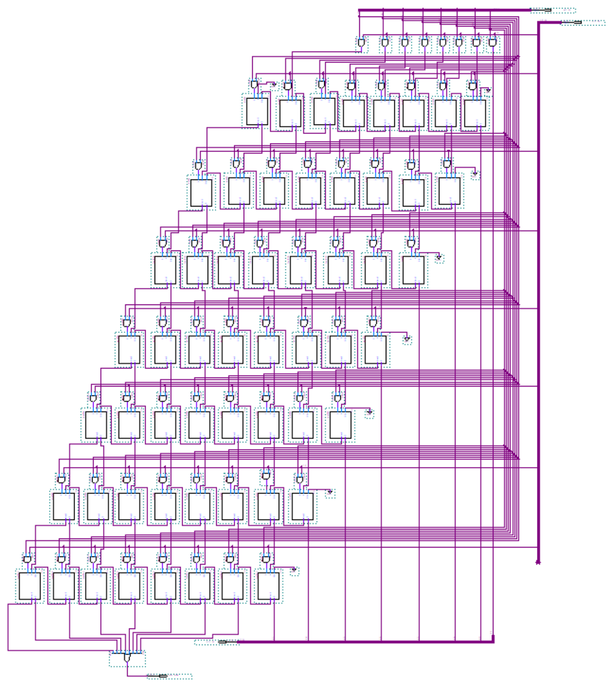


The four bits, select_quantity3, select_quantity2, select_quantity1, select_quantity0 are inputs from the user as usual to select the number of items to be purchased. The bits would enter a bus so that additional bits that are set to zero would enter 8 bits Multiplier with the cost of item selected from the **items_mux** block by the user as the other input of the Multiplier. This is where the transaction cost is calculated and where unaffordable transactions are checked. The output **LED_WARNING** would be connected to a LED light in the top-level diagram. This would serve as a warning to the user whether the current cost is higher than 250 when he is in the main menu (menu screen) and buy screen. However, the user would only be taken to the ERROR screen when he presses the confirm button to attempt the transaction in the buy screen. These transitions are processed by our FSM.

The 8-bit Multiplier unit is implemented as follows, and will output an 8-digit binary number. Since we're assigning the number to an 8-bit register, the register will only take the first eight bits of the result. Thus, the unaffordable transactions are checked by using an OR gate to the other 8 bits output where in this case, an overflow would cause an unaffordable transaction. This is shown below;

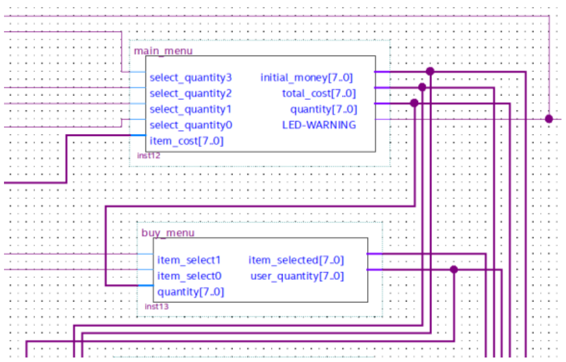


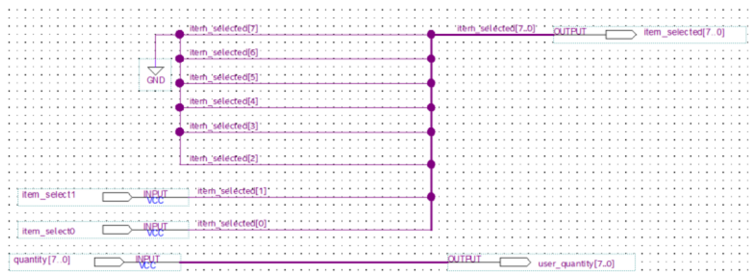
The 8 bits Multiplier is shown below;



- **BUY-MENU**

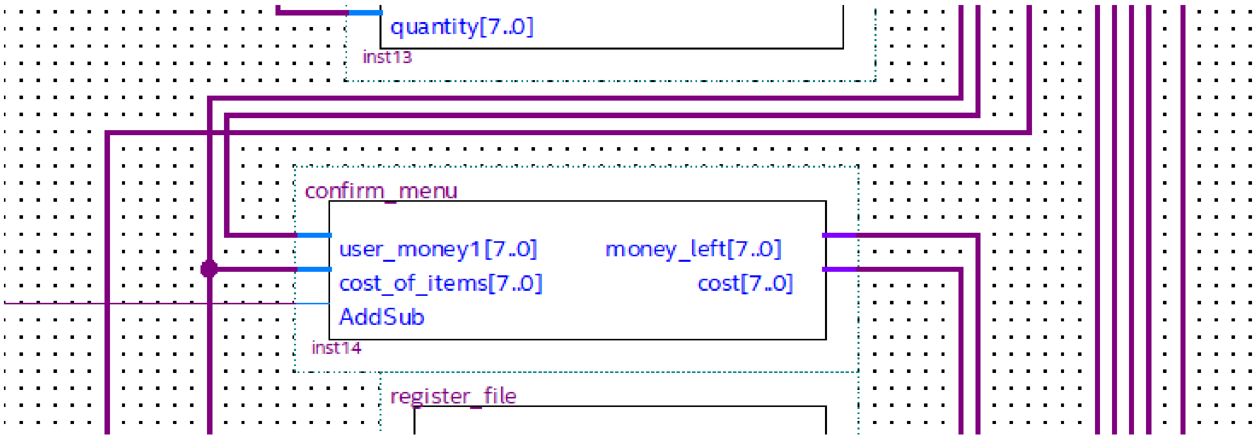
In my circuit, I implemented main_menu, buy_menu, confirm_menu and display_menu blocks. This was done for organization and less wiring in my Top-Level Diagram. However, this block does not have much functionality to it except it would be useful for choosing my outputs in the display_menu. Our buy screen is supposed to show the quantity chosen by the user, the item ID and the buy symbol in the HEX displays. Thus, I made this block with an input of the user select lines to choose an item ID and converted that 2 bits to 8-bits to fit my display. The quantity goes straight to another quantity to fit my display_menu.



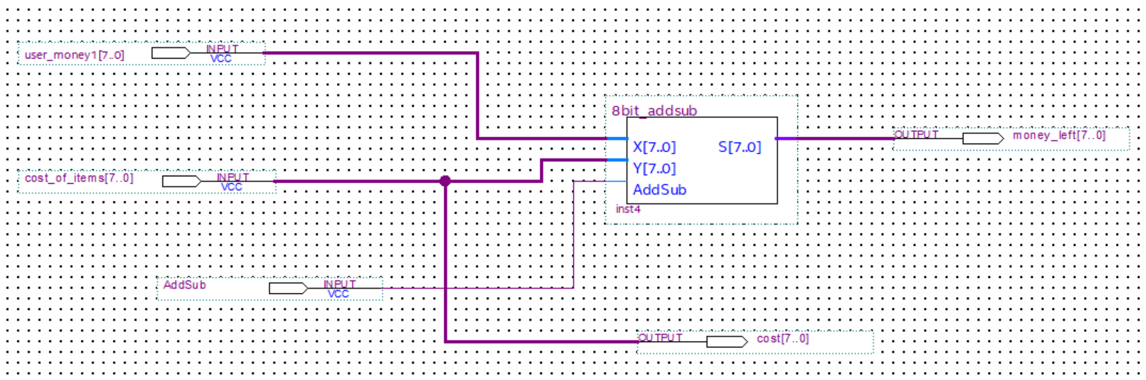


- **CONFIRM_MENU**

The **confirm_menu** is the last of the **ALU**. It is designed to calculate the money the user will have after purchasing items. The inputs are the money the user starts with which is 250 from the main_menu and the cost of items calculated.

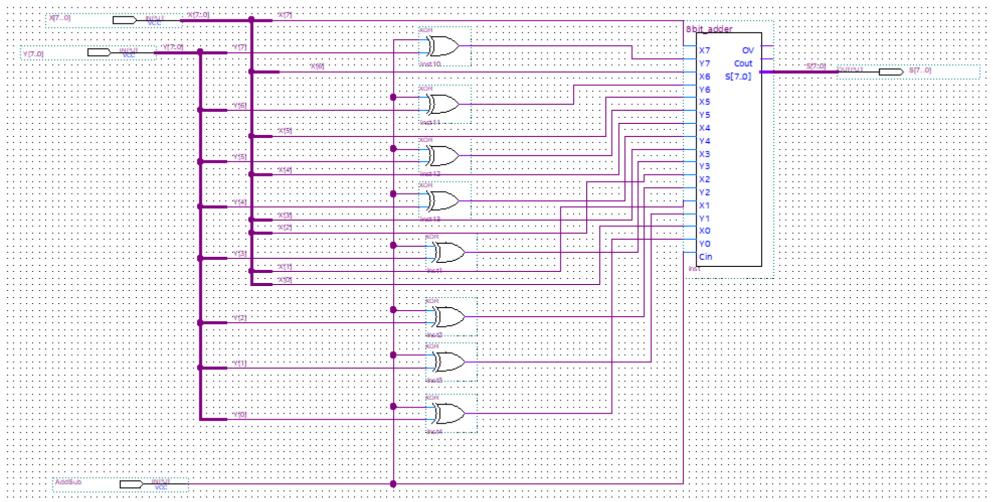


The circuit is as follows;

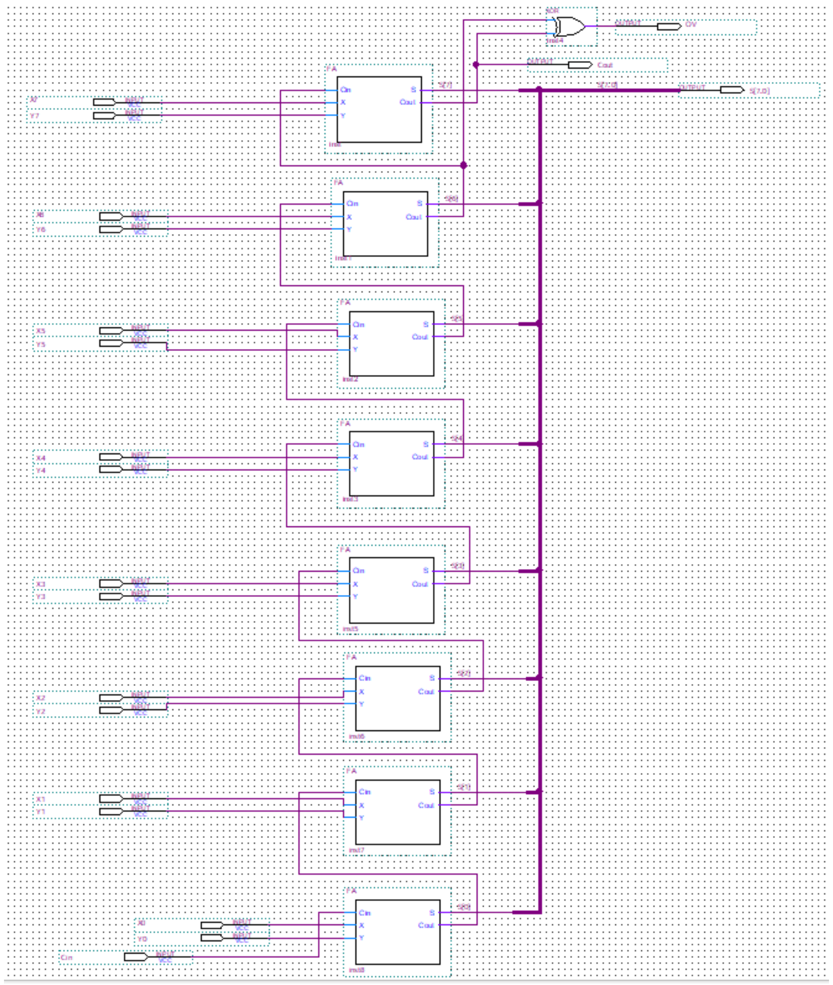


The 8bit_addsub is designed as follows;

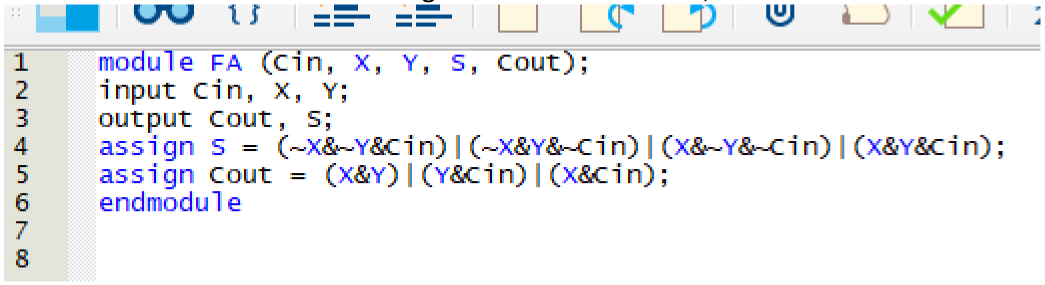
This unit uses eight-bit adder to perform the operation of the two 8-bit inputs.



Eight-bit adder: The unit that performs the addition is shown below, which used a full adder as a sub unit



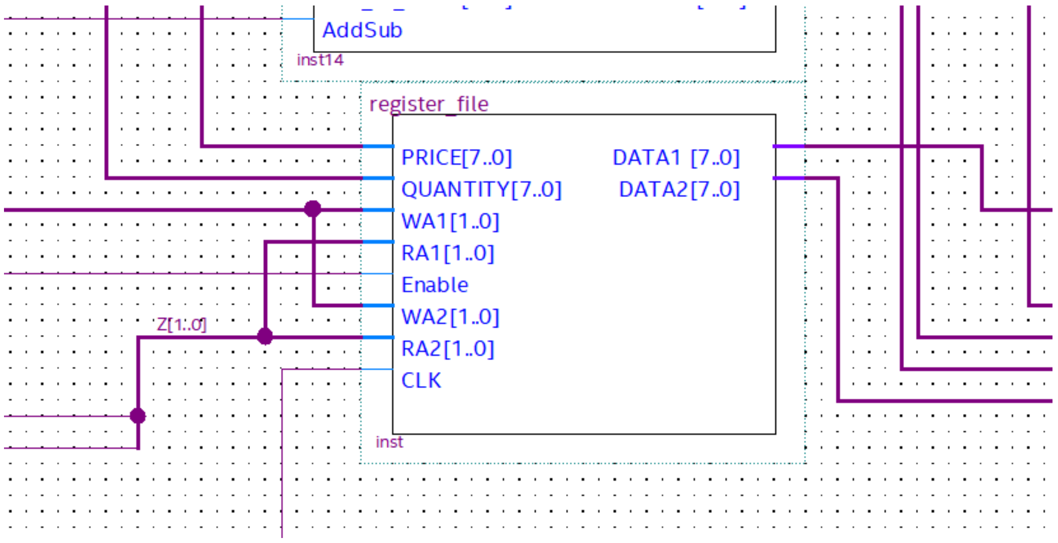
The full adder was written in Verilog which is shown below;



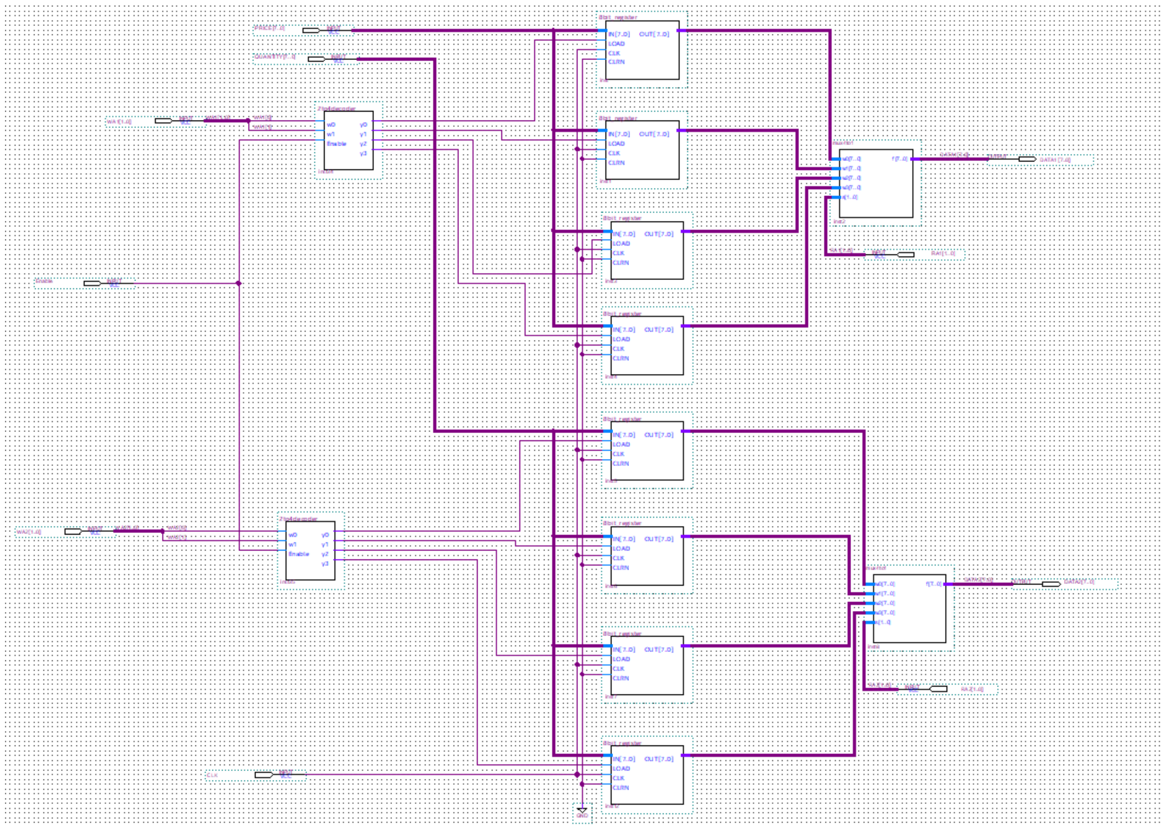
```
module FA (Cin, X, Y, S, Cout);
input Cin, X, Y;
output Cout, S;
assign S = (~X&~Y&Cin)|(~X&Y&~Cin)|(X&~Y&~Cin)|(X&Y&Cin);
assign Cout = (X&Y)|(Y&Cin)|(X&Cin);
endmodule
```

Register File: We had to create an 8 eight-bit register file
The register file is useful to store the values needed to perform the next operation. The first four registers store the menu items and the next four stores the number of items of the ID that the user has purchased.
The inputs for this unit are the following;

- WA1 and WA2 will specify which address to write into.
- Price of 8 bits which will have the 8 bits loaded to the register specified by the address WA1
- Quantity of 8 bits which will have the 8 bits loaded to the register specified by the address WA2
- Enable will determine whether or not to perform the operation specified by the FSM.
- The clock here is attached to a clock_divider_1024 to have the operation done automatically once the perform is requested.
- RA1 and RA2 will specify the address from we need to read and display the value on the board.



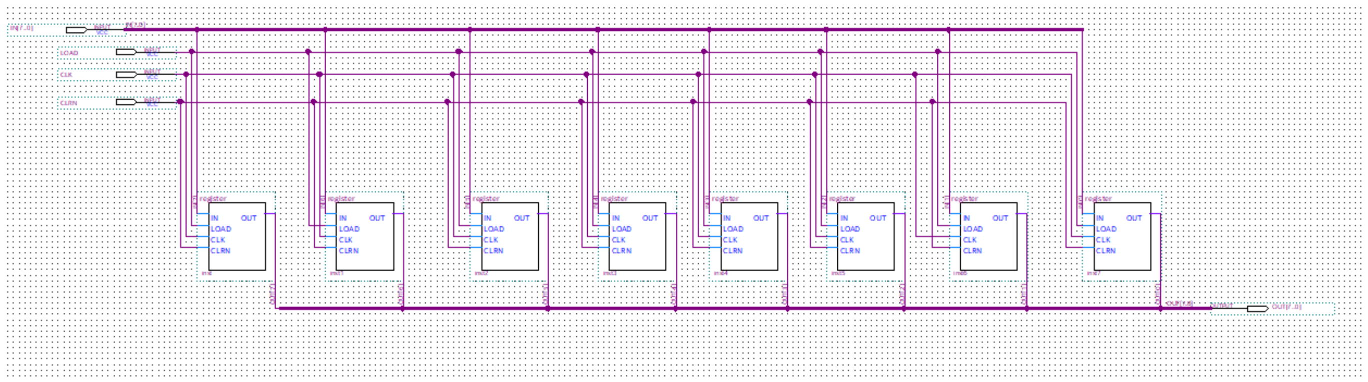
The outputs of the unit is the data from the register of which the address is specified by the corresponding RA.



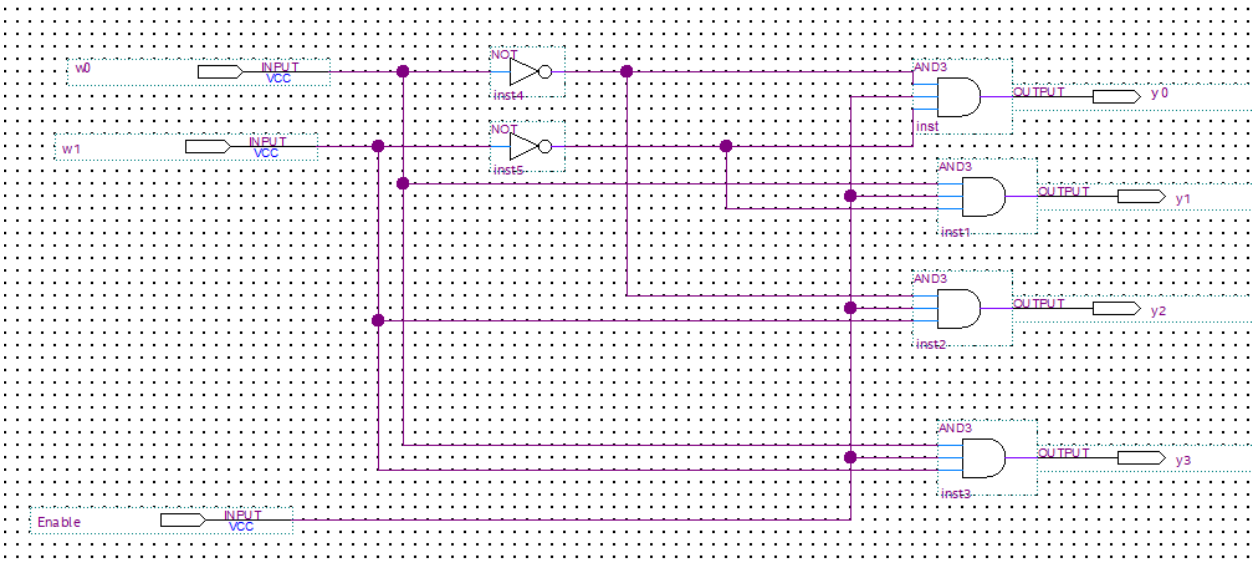
The interior design includes the following

- Two 2 to 4 decoders that will determine the address of the register we need to assign a value into.
- 8 eight-bit registers that will load new values if the enable key is high.
- Two eight-bit 4 to 1 multiplexers are used to determine the output which is selected by the FSM.

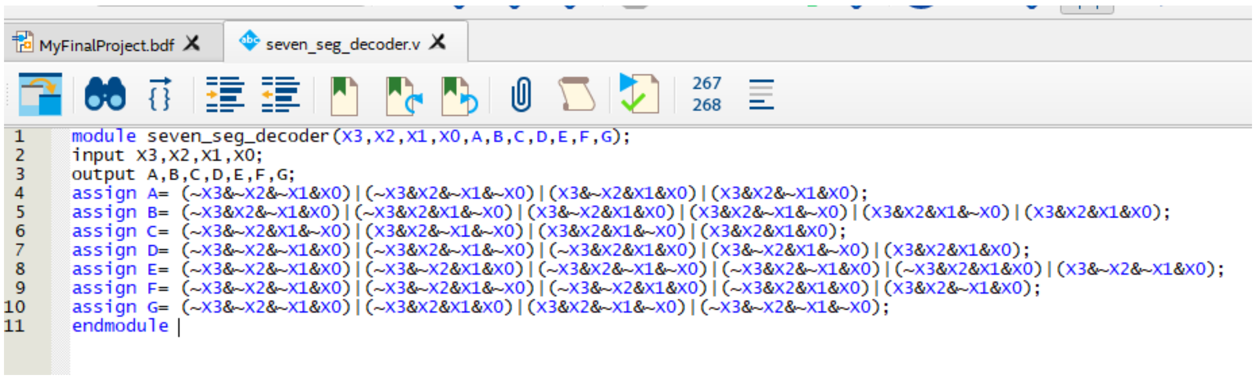
8-bit Register:



The **2 to 4 decoder** used is shown below;



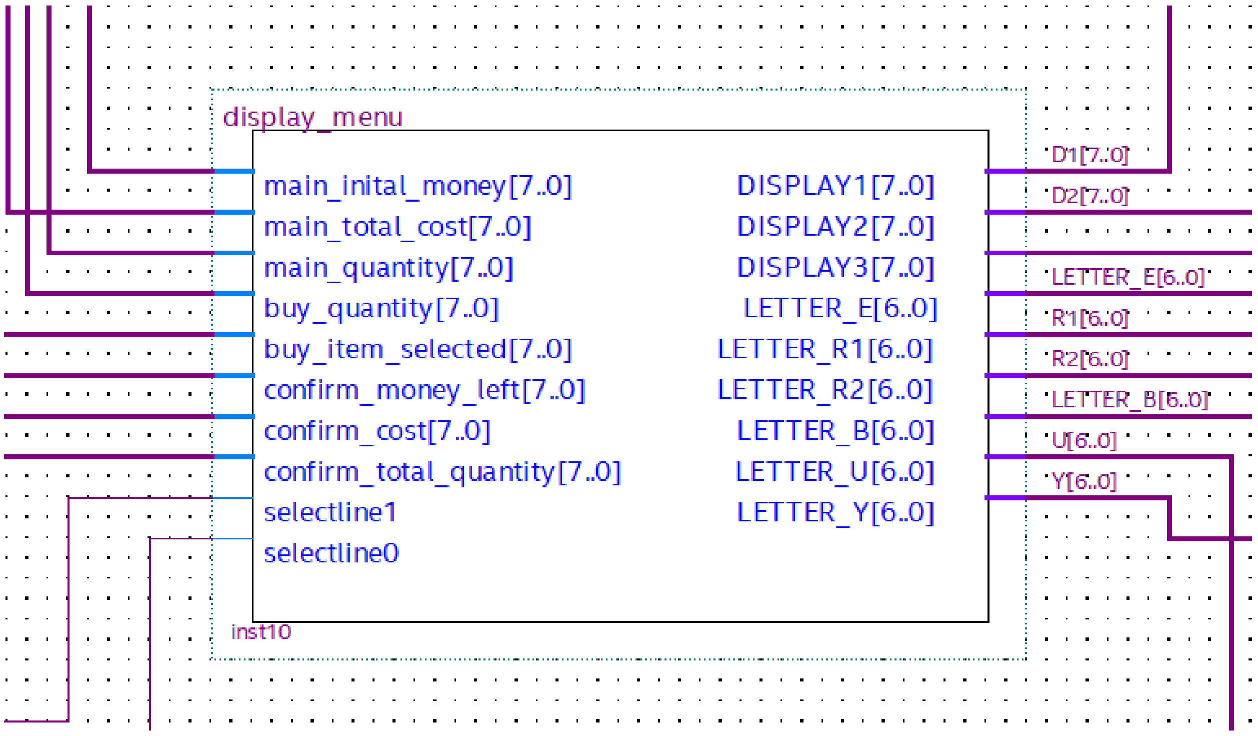
The seven-seg-decoder used in **display_menu**;



```
module seven_seg_decoder(X3,X2,X1,X0,A,B,C,D,E,F,G);
input X3,X2,X1,X0;
output A,B,C,D,E,F,G;
assign A= (~X3&~X2&~X1&X0)|(~X3&X2&~X1&~X0)|(X3&~X2&X1&X0)|(X3&X2&~X1&X0);
assign B= (~X3&X2&~X1&X0)|(~X3&X2&X1&~X0)|(X3&~X2&X1&X0)|(X3&X2&~X1&~X0)|(X3&X2&X1&~X0)|(X3&X2&X1&X0);
assign C= (~X3&~X2&X1&~X0)|(X3&X2&~X1&~X0)|(X3&X2&X1&~X0)|(X3&X2&X1&X0);
assign D= (~X3&~X2&~X1&X0)|(~X3&X2&~X1&~X0)|(~X3&X2&X1&X0)|(X3&~X2&X1&~X0)|(X3&X2&X1&X0);
assign E= (~X3&~X2&~X1&X0)|(~X3&~X2&X1&X0)|(~X3&X2&~X1&~X0)|(~X3&X2&~X1&X0)|(~X3&X2&X1&X0)|(X3&~X2&~X1&X0);
assign F= (~X3&~X2&~X1&X0)|(~X3&~X2&X1&~X0)|(~X3&~X2&X1&X0)|(~X3&X2&X1&X0)|(X3&X2&~X1&X0);
assign G= (~X3&~X2&~X1&X0)|(~X3&X2&X1&X0)|(X3&X2&~X1&~X0)|(~X3&~X2&~X1&~X0);
endmodule
```

DISPLAY OUTPUT

- **DISPLAY_MENU**



There are four states that the Vending Machine needs to display. The first state is the menu screen which is named as the **main_menu** in my project. The menu screen shows the total cost of the item, how much the user has, and the copies of the item the user has or wants to purchase, e.g FA OF 05.

In my **display_menu**, the total cost of item is the **main_total_cost**, the amount the user has is the **main_initial_money**, and the copies of item the user has is **main_quantity**.

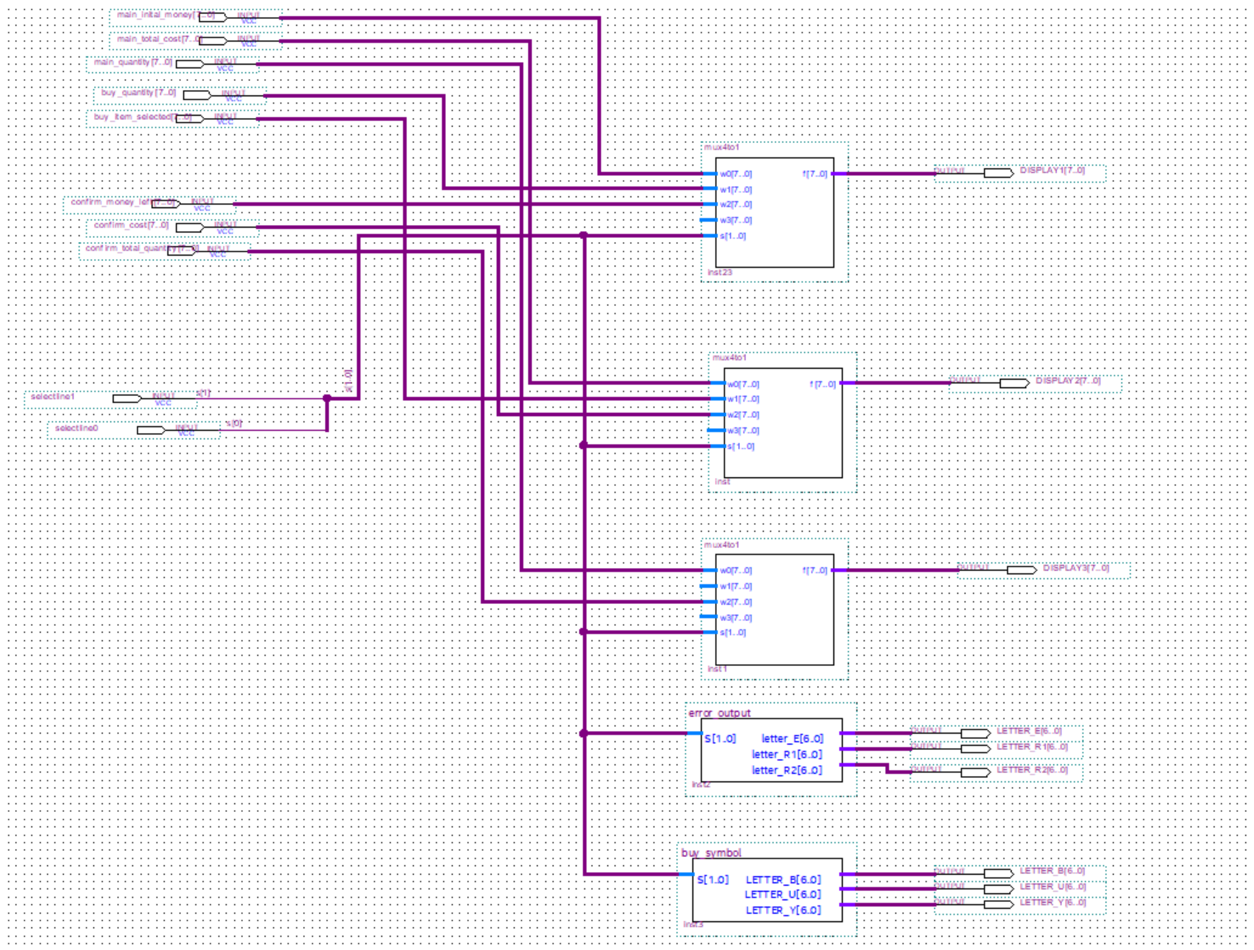
The second state is the buy screen which is named as the **buy_menu** in my project. The buy screen shows the quantity of the item the user is attempting to buy, the item ID, and it displays buy in the HEX displays.

In my **display_menu**, the quantity of the item the user is attempting to buy is the **buy_quantity**, the item ID is the **buy_item_selected** and the buy symbol is implemented in the block design.

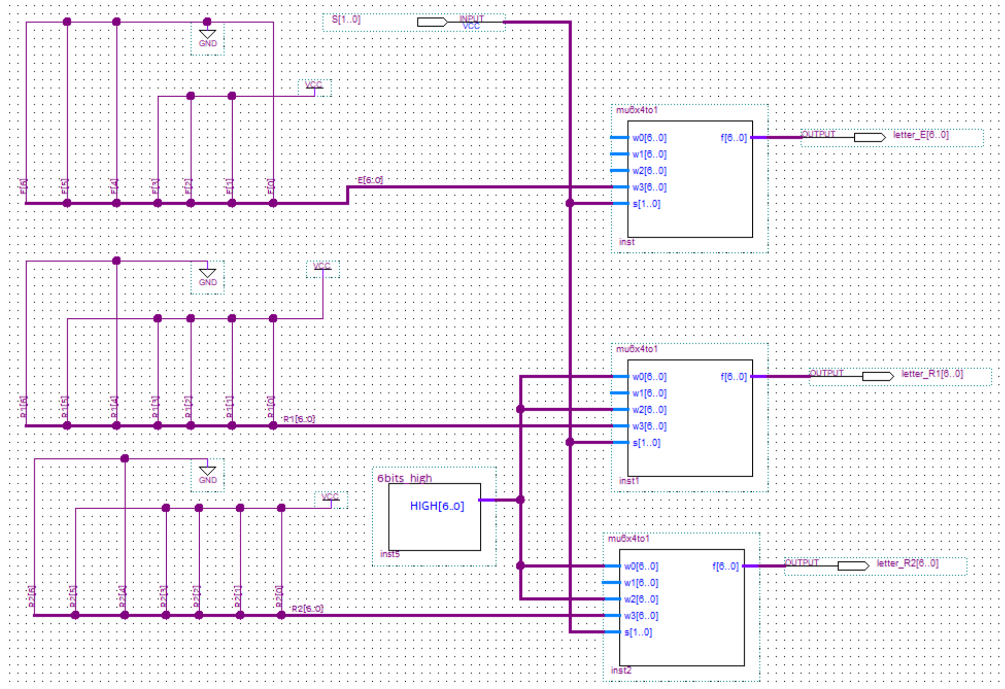
The third state is the confirm state, in my **display_menu,** the amount of money left with the user is the **confirm_money_left,** the cost of the transaction is **confirm_cost** and the total quantity bought by the user is **confirm_total_quantity**

The last state is the Error state which is when the user attempts to confirm an unaffordable transaction and this is hard coded and implemented in the block diagram

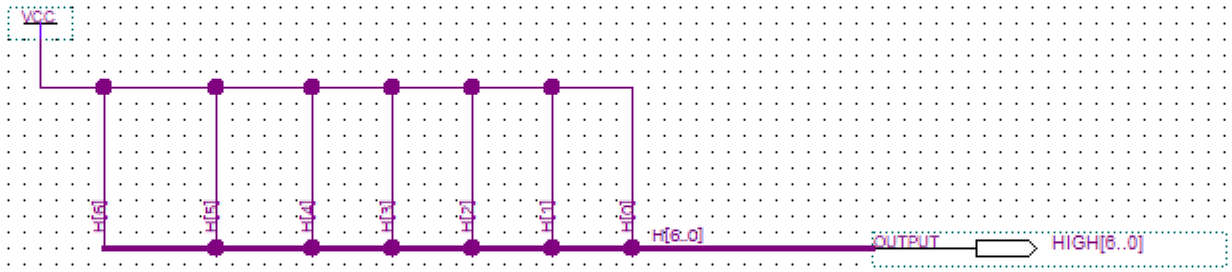
These states are simultaneously displayed and determined by the select lines which is the output of the FSM.



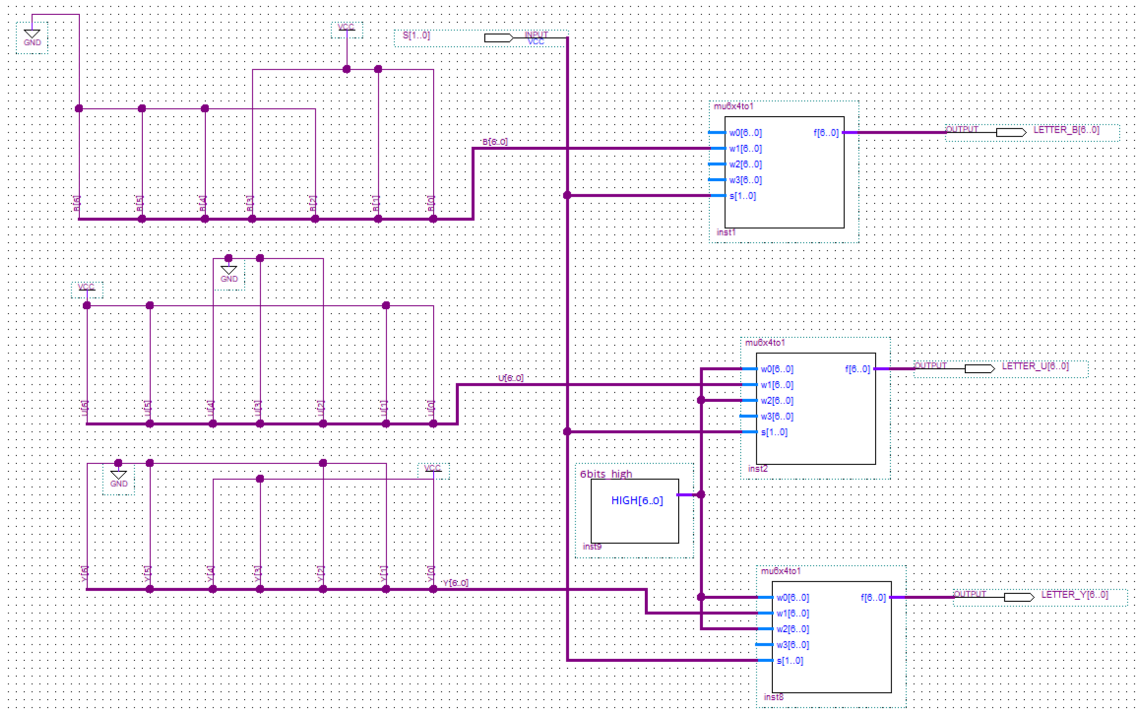
The error output is hard coded as follows;



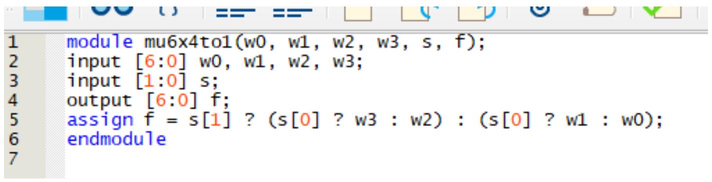
6bits_high is a 7bits all high output which I implemented to turn off the display of some HEX displays at different states. This was to allow for It is hard coded as follows;



The buy display symbol on the HEX display is hard coded as follows;

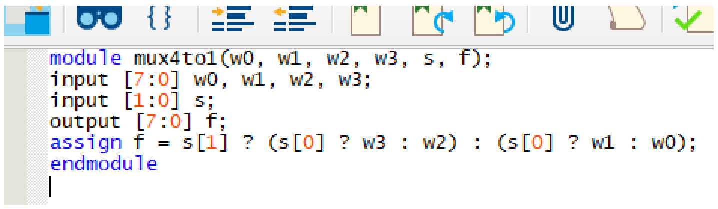


The mu6x4to1 is a 6-bits 4 to 1 Multiplexer. It is written in Verilog and shown below;



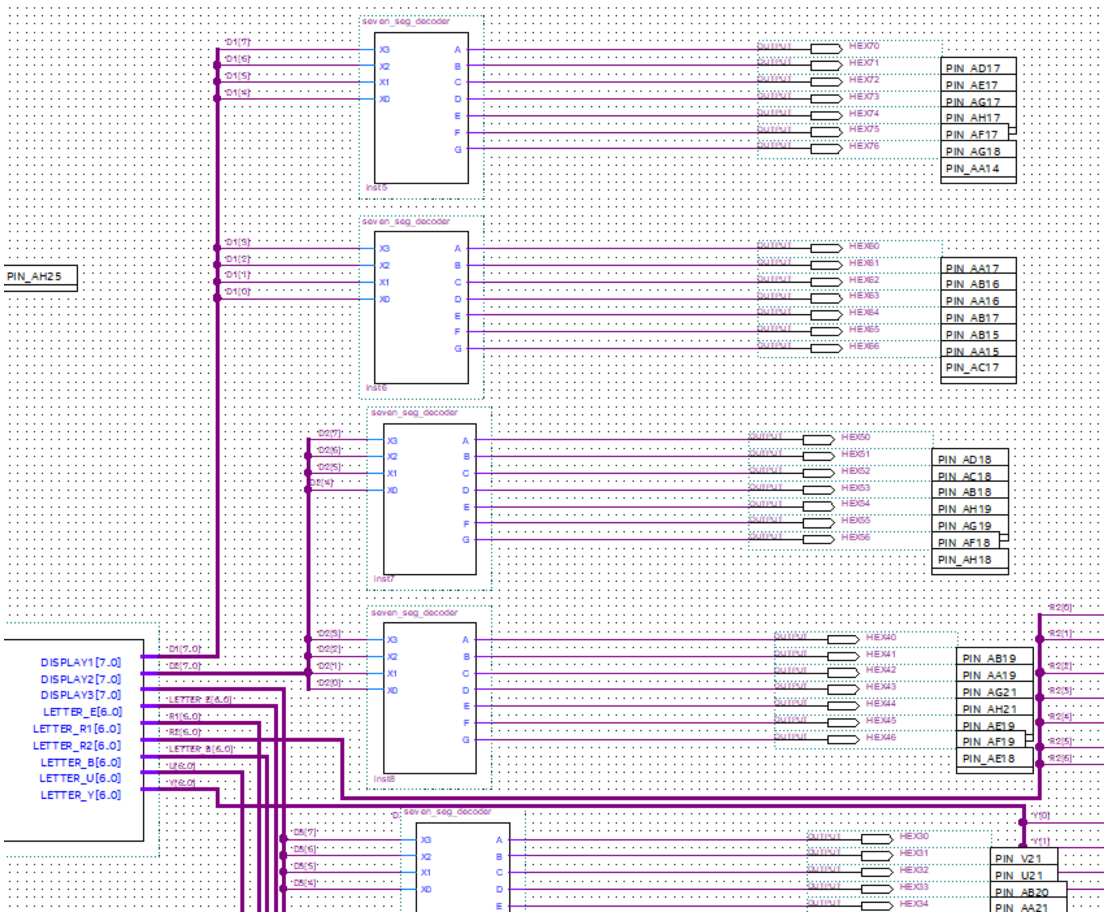
```
module mu6x4to1(w0, w1, w2, w3, s, f);
input [6:0] w0, w1, w2, w3;
input [1:0] s;
output [6:0] f;
assign f = s[1] ? (s[0] ? w3 : w2) : (s[0] ? w1 : w0);
endmodule
```

The mux4to1 is a 7bits 4 to 1 Multiplexer. It is written in Verilog and shown below;

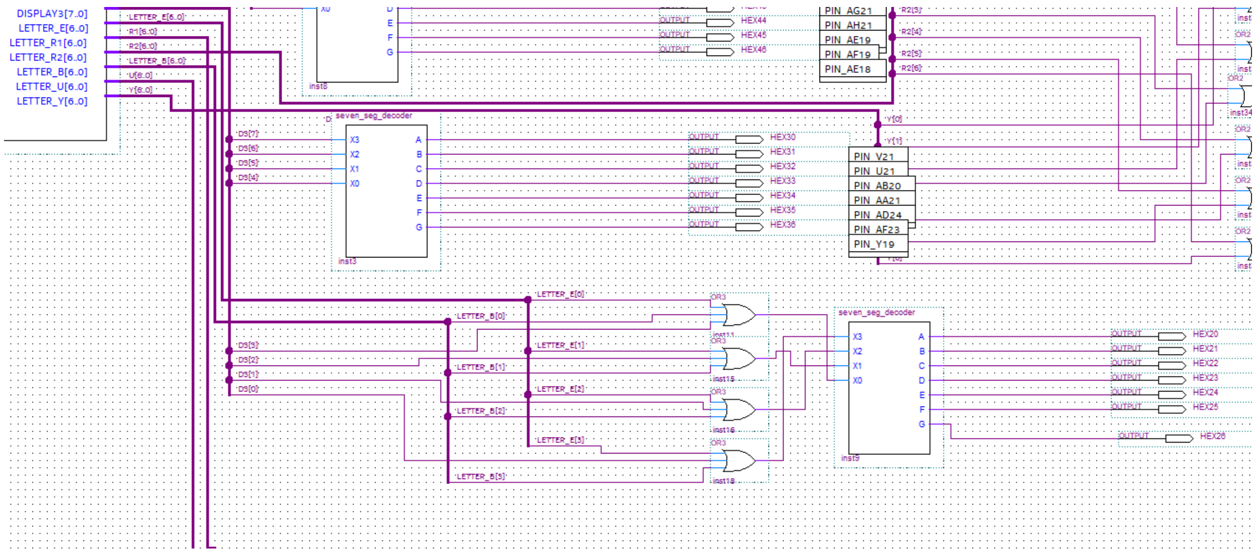


```
module mux4to1(w0, w1, w2, w3, s, f);
input [7:0] w0, w1, w2, w3;
input [1:0] s;
output [7:0] f;
assign f = s[1] ? (s[0] ? w3 : w2) : (s[0] ? w1 : w0);
endmodule
```

The outputs of the **display_menu** are shown below;

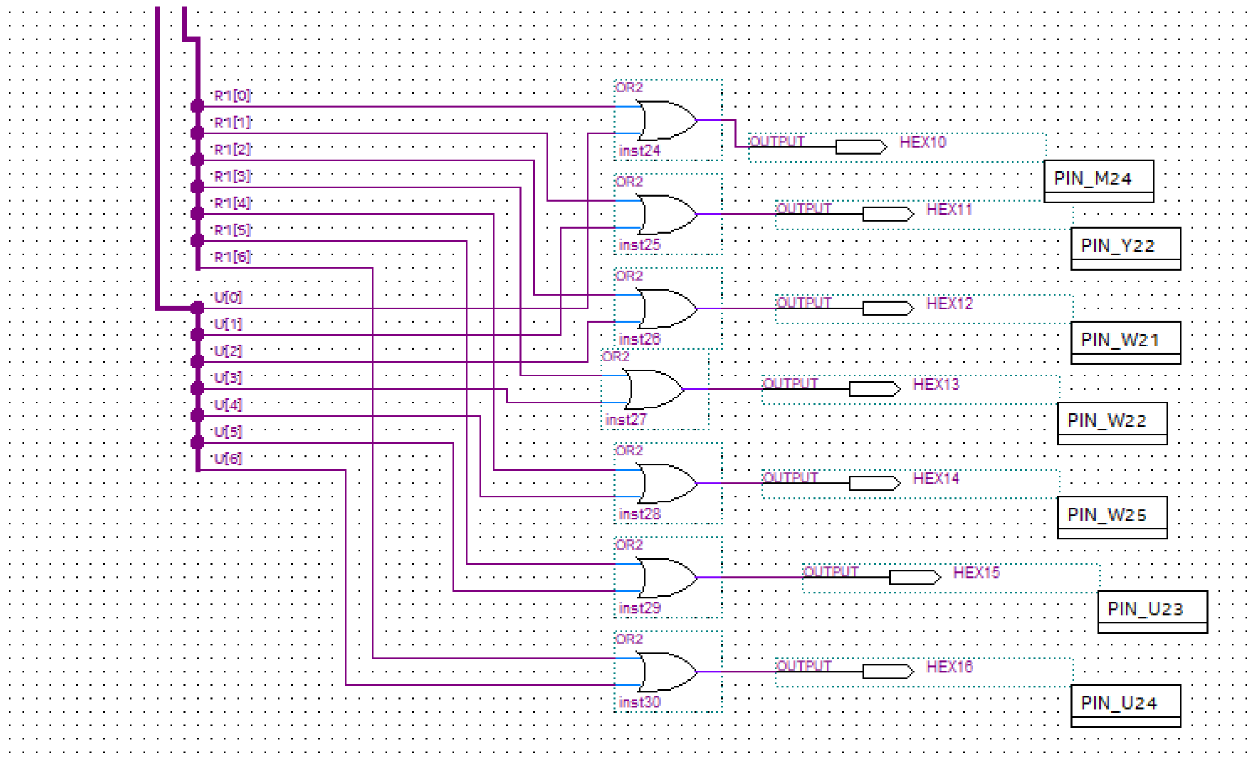


The DISPLAY1 is displayed in HEX7 and HEX6 while DISPLAY2 is displayed in HEX5 and HEX4.



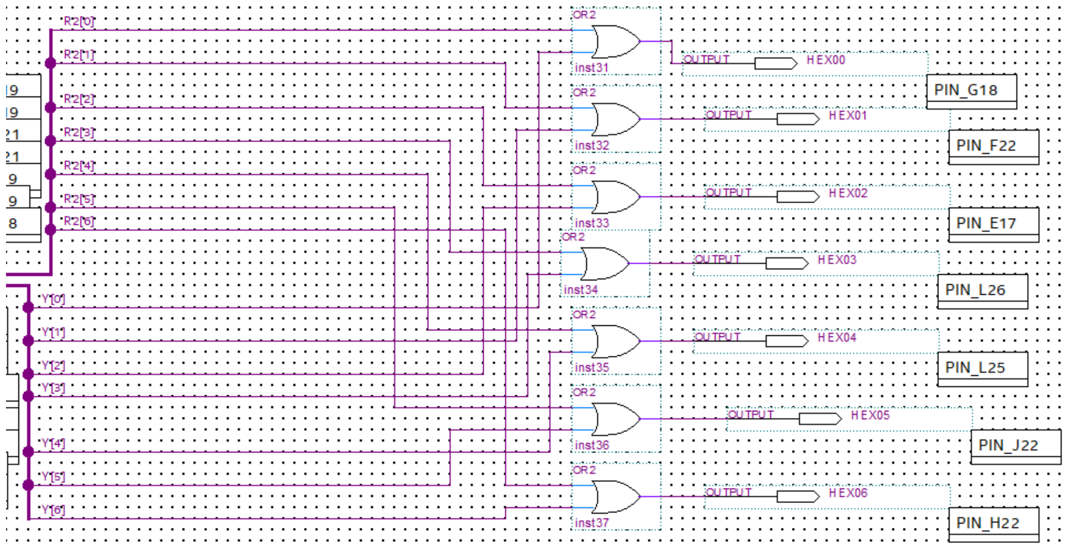
DISPLAY3 is displayed on HEX3 and HEX2 while ‘b’ from the buy display and ‘E’ from the error display are also displayed on HEX2. Thus, the use of the OR gates was to display when either of the three cases of bits are one for HEX2 which is selected by the multiplier in the display_menu block diagram.

The same concept of the use of the OR gates is used again in the following picture for the display on HEX1.



When either LETTER_R1 or LETTER_U is active, it is displayed on the HEX1. They would never both be active at the same time because they are in different states of the multiplexer. LETTER_R1 is from the error output and LETTER_U is from the buy symbol.

The same concept of the use of the OR gates is used again in the following picture for the display on HEX0.

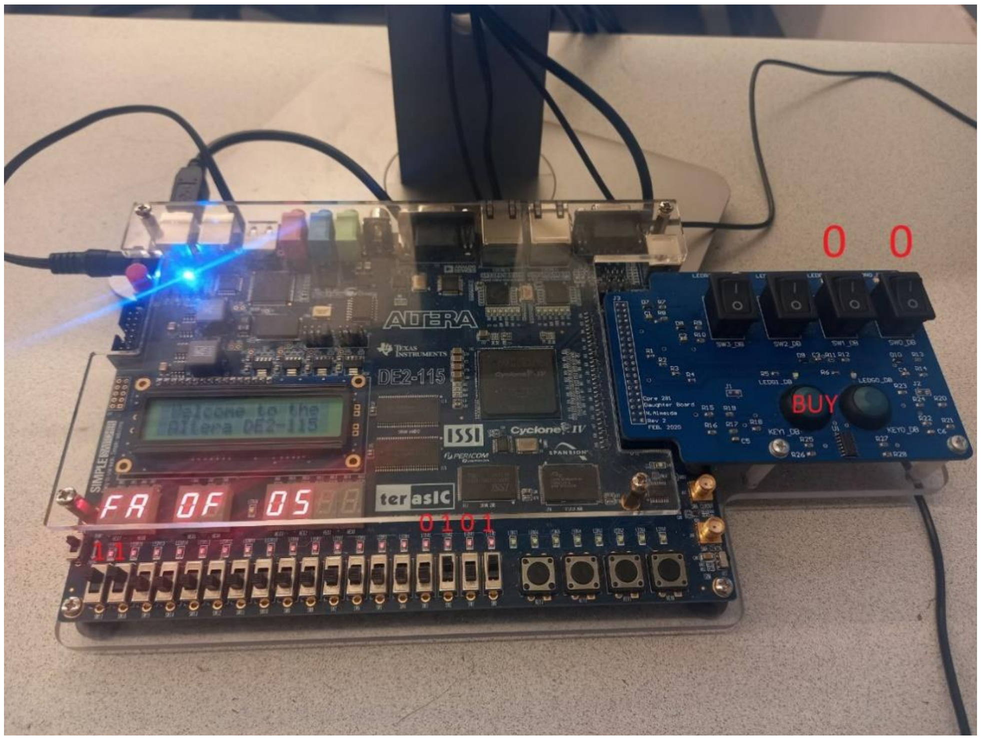


When either LETTER_R2 or LETTER_Y from their states is active, it is displayed on the HEX0.

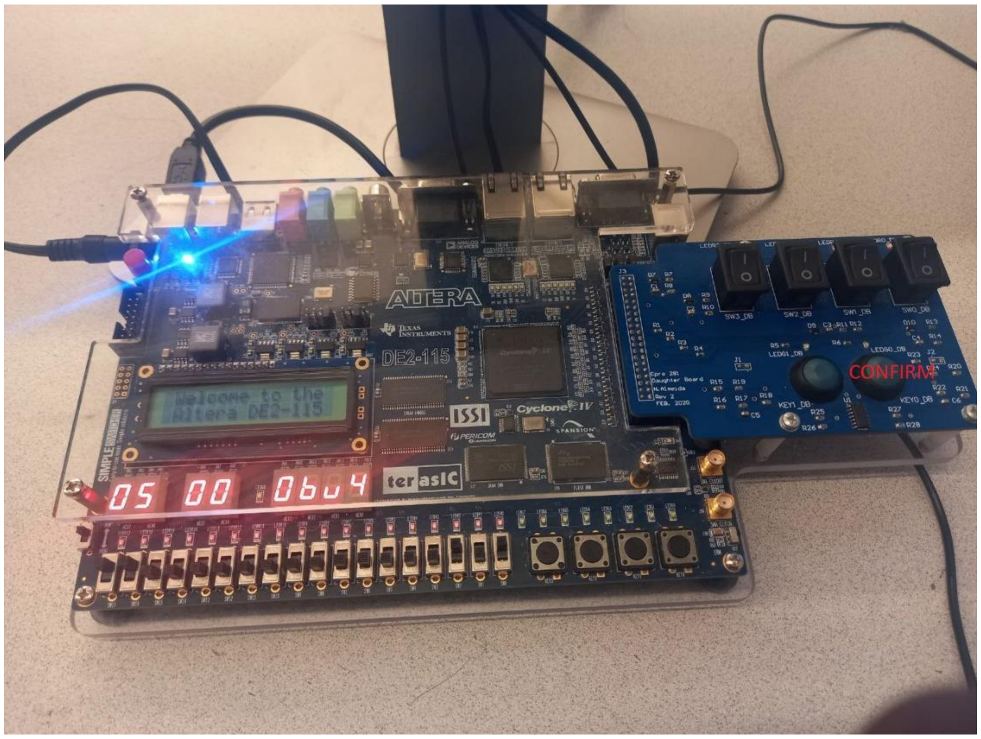
DEMONSTRATION AND RESULT

The following is a demonstration of the vending machine;

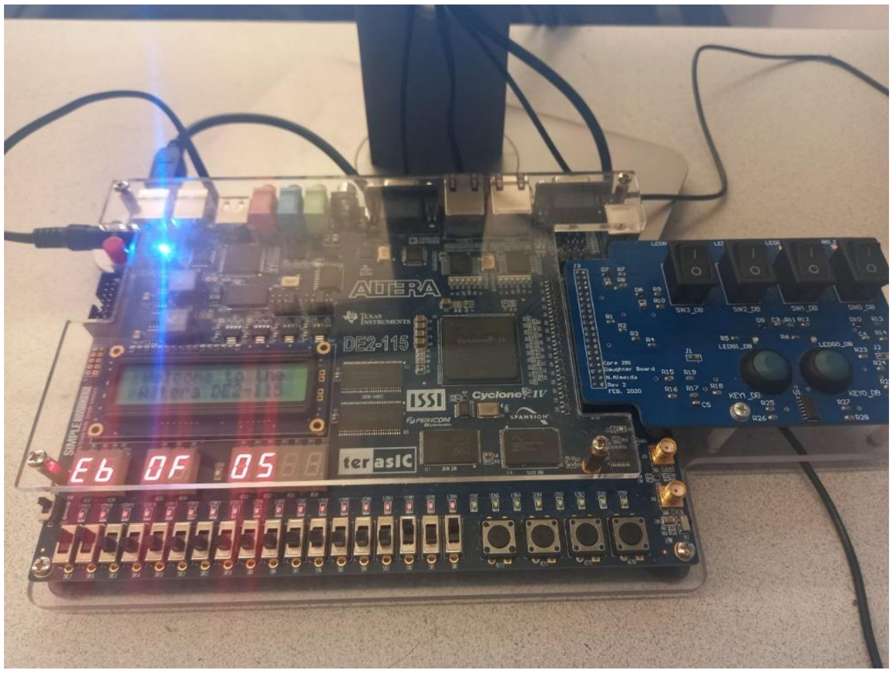
- User selects item ID 0 and 5 as quantity



- User presses the BUY button

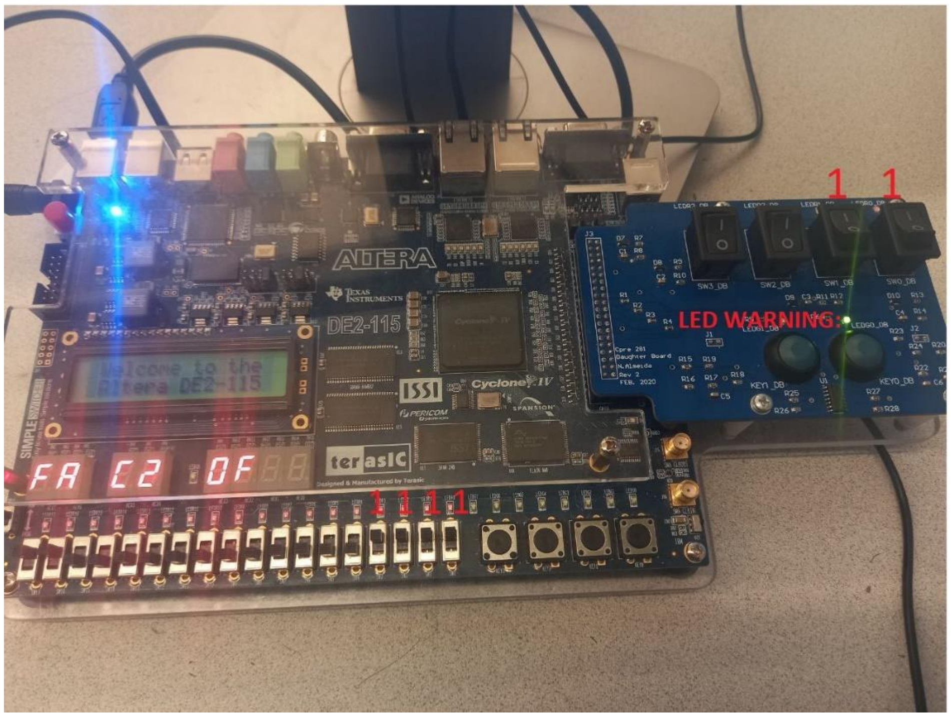


- User presses the CONFIRM button

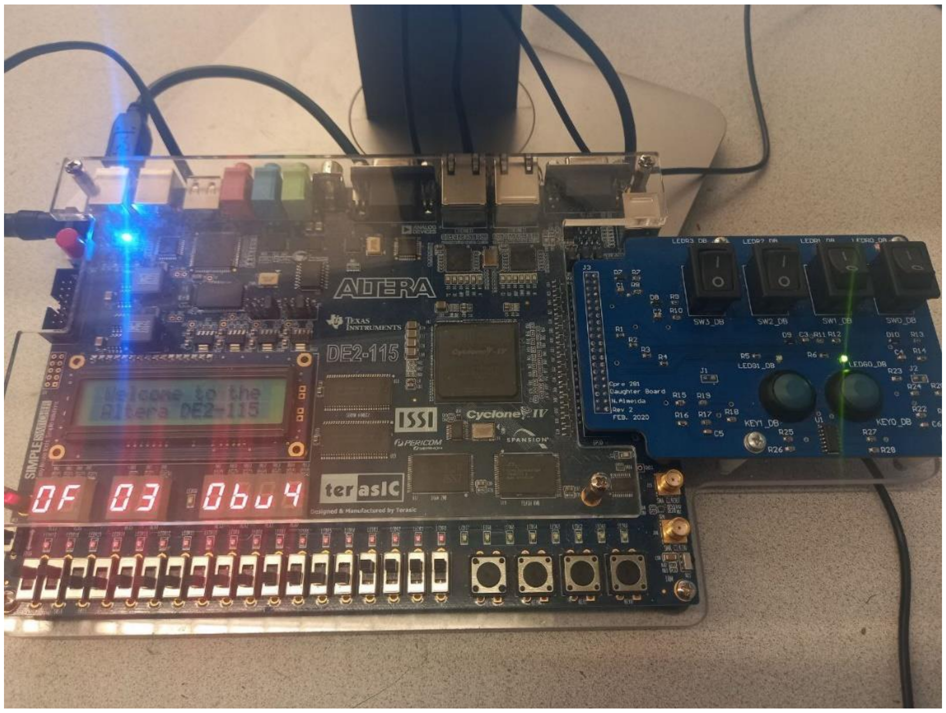


Another experiment:

- User selects Item 3 with quantity 15



- User presses BUY button



- User attempts transaction by pressing the CONFIRM button