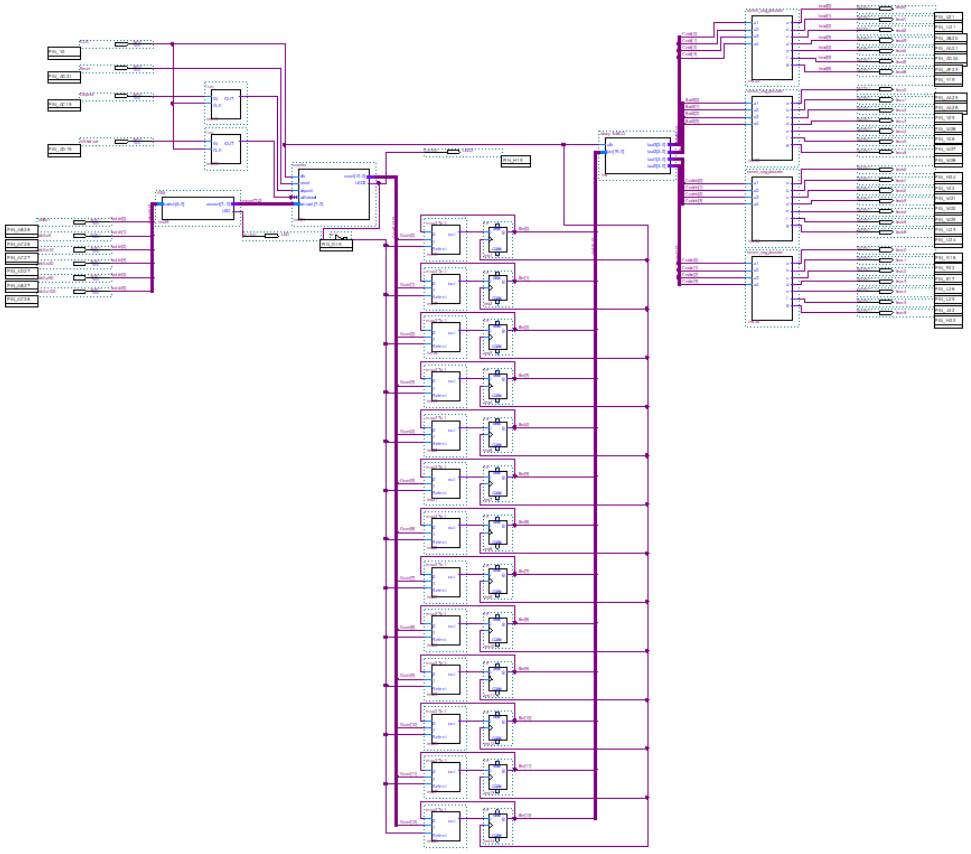


ATM MACHINE – A DIGITAL LOGIC PROJECT

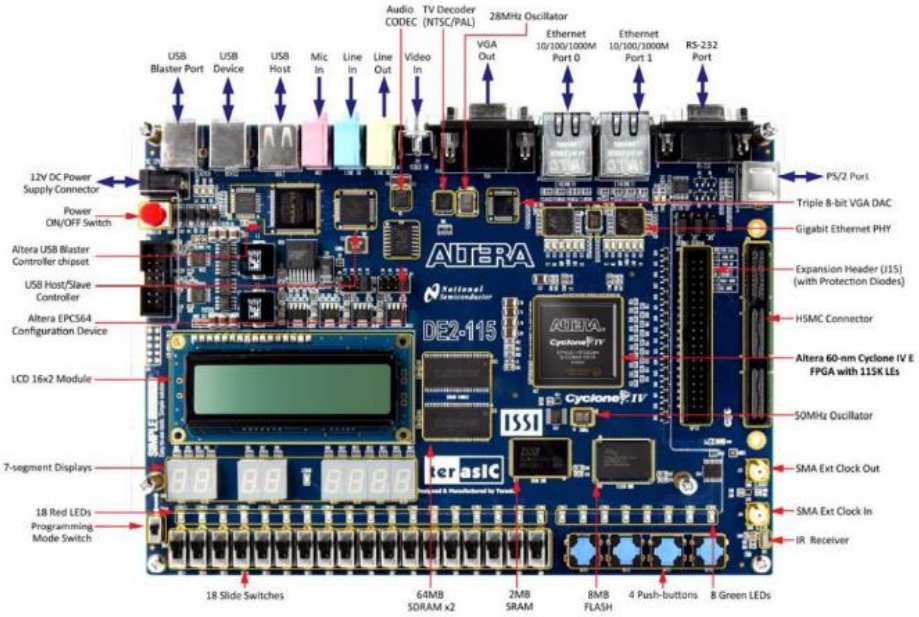
Mark Pyatak

05/03/2024

CPRE 281 Digital Logic

Mark Pyatak
05/03/2024
CPRE 281

I designed and implemented a simple ATM Machine using Quartus Prime and an FPGA Board (DE2-115). This project has many parts, some implemented using Verilog code, and others using block diagrams with logic circuits.

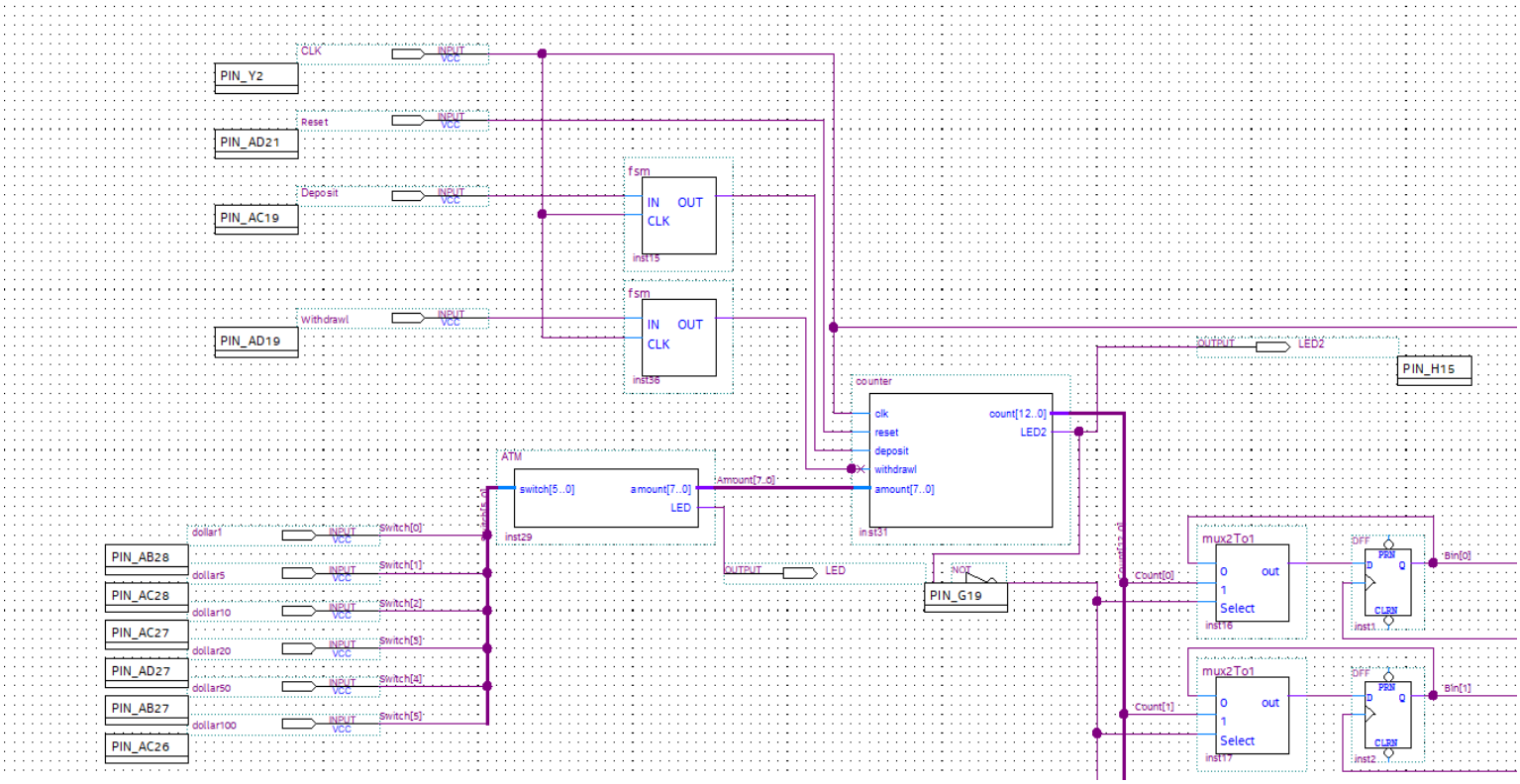


(FPGA BOARD)

When the code is loaded onto the FPGA Board, it initially shows ‘0000’ on four of the seven segment displays. Six different switches can show the amount you want to deposit or withdrawl, but only one type of amount can be used at a time. If more than two amounts are switched on, you cannot deposit or withdraw, and it shows an error LED. Once you have the amount switch set, i.e. the $50 switch, you can use one of two push buttons to deposit or withdrawl the amount. The maximum amount possible on the account is $8192, or 13 bits. If you try to go above this amount, or if you try to withdrawl money you do not have, the error LED lights up and it does not continue with the transaction. Your dollar amount is stored on 13 one-bit register files, that get converted to show on the seven-segment displays. You use a rocker switch to reset the project to ‘0000’.

Mark Pyatak
05/03/2024
CPRE 281

First, I will go over the input pins and initial logic. This zoomed-in section of the final design shows all the input pins and logic that are part of the circuit.



(Inputs and Logic)

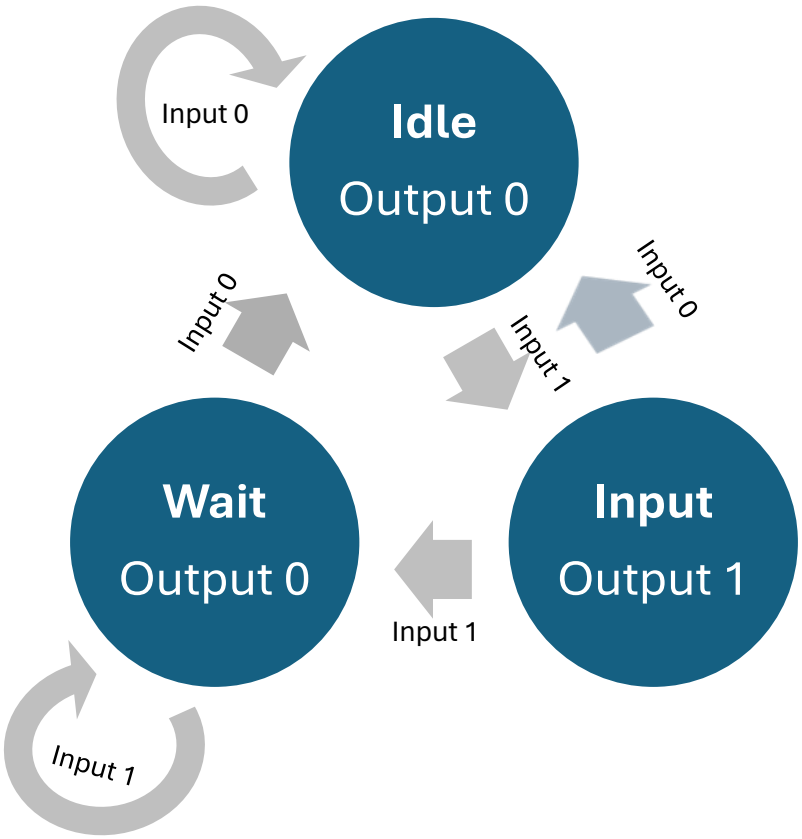
First, the clock is connected to the FPGA's clock to time everything on the positive edge of the clock cycle. This prevents anything from happening out of sync, too early/late, and for the data to be loaded into the register files.

Additionally, three inputs go into the counter module. These are reset, a rocker switch, and deposit/withdrawl pushbuttons. The push buttons first go through the fsm module in order to prevent the holding of the button and to delay the input one cycle of the clock.

Lastly, six switches go into the atm module that are used to show the dollar amount. It goes $1, $5, $10, $20, $50, and $100.

Mark Pyatak
05/03/2024
CPRE 281

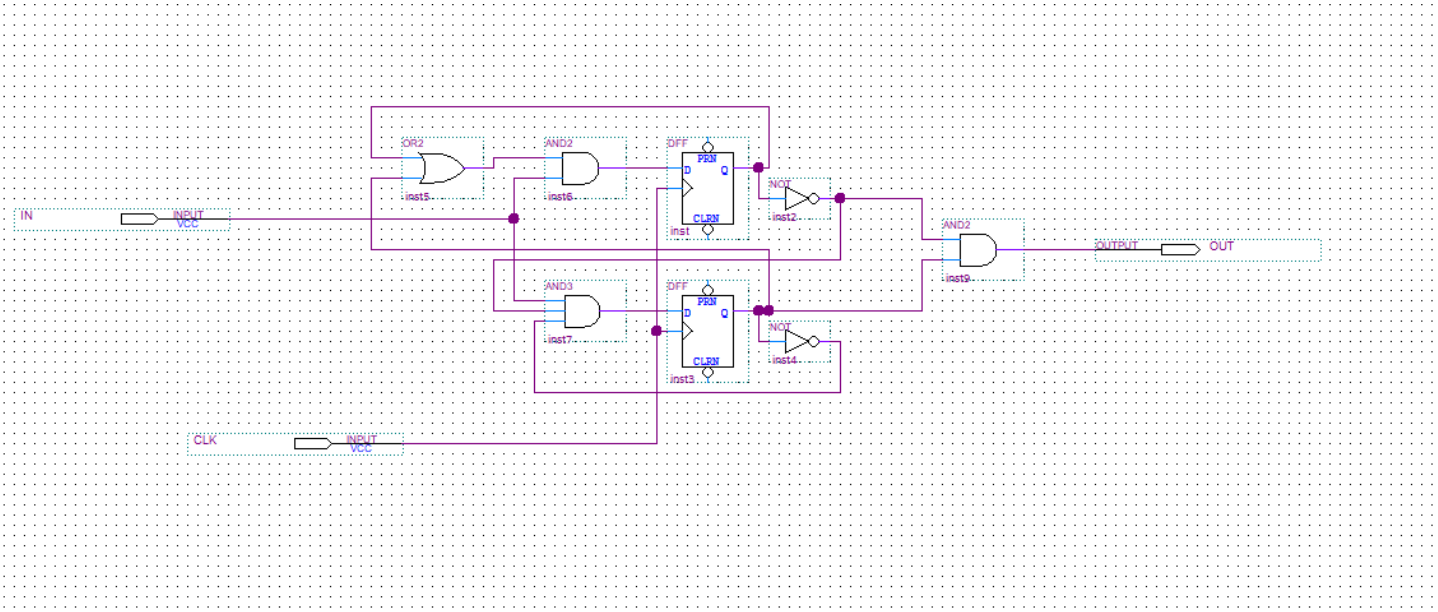
FSM MODULE



This is the finite state machine diagram for the FSM module. While the buttons are not pressed, the machine stays at the Idle state. If the button is pressed, it goes to the Input state and outputs 1. If the button is not pressed further or held down, it goes back to the idle state. If the button is held or pressed very quickly (before the next clock cycle), it goes to the Wait state and waits for the button to not be pressed or held for it to go back to the idle state.

This Finite State Machine is used to both debounce the push button and prevent abuse of the deposit/withdrawl button.

Mark Pyatak
05/03/2024
CPRE 281



(FSM Module Design)

This is the circuit implementation of the State Diagram Above. It has two inputs, the clock and an IN input. It uses D Flip Flops, AND gates, OR gates, and NOT gates to implement the logic for the output, OUT. A symbol was created for this design to simplify and save space on the final design. Two of these circuits are used in the design.

Mark Pyatak
05/03/2024
CPRE 281

ATM MODULE

```
module ATM(
    input [5:0] switch,
    output [7:0] amount,
    output LED
    );

    reg [7:0] bill_amount = 8'b00000000;

    always @(switch)
    begin
        case(switch)
            1: bill_amount = 8'b00000001;//1
            2: bill_amount = 8'b00000101;//5
            4: bill_amount = 8'b00001010;//10
            8: bill_amount = 8'b00010100;//20
            16: bill_amount = 8'b00110010;//50
            32: bill_amount = 8'b01100100;//100
            default: bill_amount = 8'b00000000;
        endcase
    end

    assign amount = bill_amount;

    reg active = 1'b0;

    always @(switch)
    begin
        case(switch)
            0: active = 1'b0;//0
            1: active = 1'b0;//1
            2: active = 1'b0;//5
            4: active = 1'b0;//10
            8: active = 1'b0;//20
            16: active = 1'b0;//50
            32: active = 1'b0;//100
            default: active = 1'b1;
        endcase
    end

    assign LED = active;
endmodule
```

Mark Pyatak
05/03/2024
CPRE 281

This is the code used to create the ATM module. It has one bus, or multiple lines of data, as the input. This is for the six different switches. There are two different outputs. One is an LED. The LED is to show if there is an error with the switches. This happens if there are multiple switches, or dollar amounts, that are used for input. The other output is the dollar amount. This output is an 8-bit bus. This output is binary. For example, $50 would output as ‘00110010’. This output goes to the Counter module.

The LED logic is as follows: if more than one switch is ‘active’ then the LED output is 1, which lights up the LED. Otherwise, if only one switch is active, outputs 0 for the LED which means the LED is not lit up.

Mark Pyatak
05/03/2024
CPRE 281

COUNTER MODULE

```
module counter(
    input clk,
    input reset,
    input deposit,
    input withdrawl,
    input [7:0] amount,
    output [12:0] count,
    output LED2
    );

    reg [12:0] current_count = 0;

    always@(posedge clk) begin
        if(reset)
            current_count <= 0;
        else if(deposit & (count + amount) > count)
            current_count<= count + amount;
        else if(withdrawl & amount <= count)
            current_count<= count - amount;
        else
            current_count <= count;
    end
    assign count = current_count;

    reg set2 = 0;

    always@(posedge clk) begin
        if(reset)
            set2 <= 0;
        else if(deposit & (count + amount) > count)
            set2 <= 0;
        else if(withdrawl & amount <= count)
            set2 <= 0;
        else if(deposit & (count + amount) < count)
            set2 <= 1;
        else if(withdrawl & amount > count)
            set2 <= 1;
        else
            set2 <= LED2;
    end

    assign LED2 = set2;

endmodule
```

Mark Pyatak
05/03/2024
CPRE 281

This is the Verilog code for the Counter Module. It has five inputs and two outputs. One input is from the ATM module, which is the amount bus (8 bits). The other inputs are reset, withdrawl, and deposit. These are buttons or rocker switches that are used to indicate which action is taken with the input dollar amount. The two outputs are a 13-bit bus, and an LED. The bus is the new dollar amount in the account that goes to the register files to be stored. The LED is another error LED that lights up if you try to go beyond the limits of the account ($0000 - $8192).

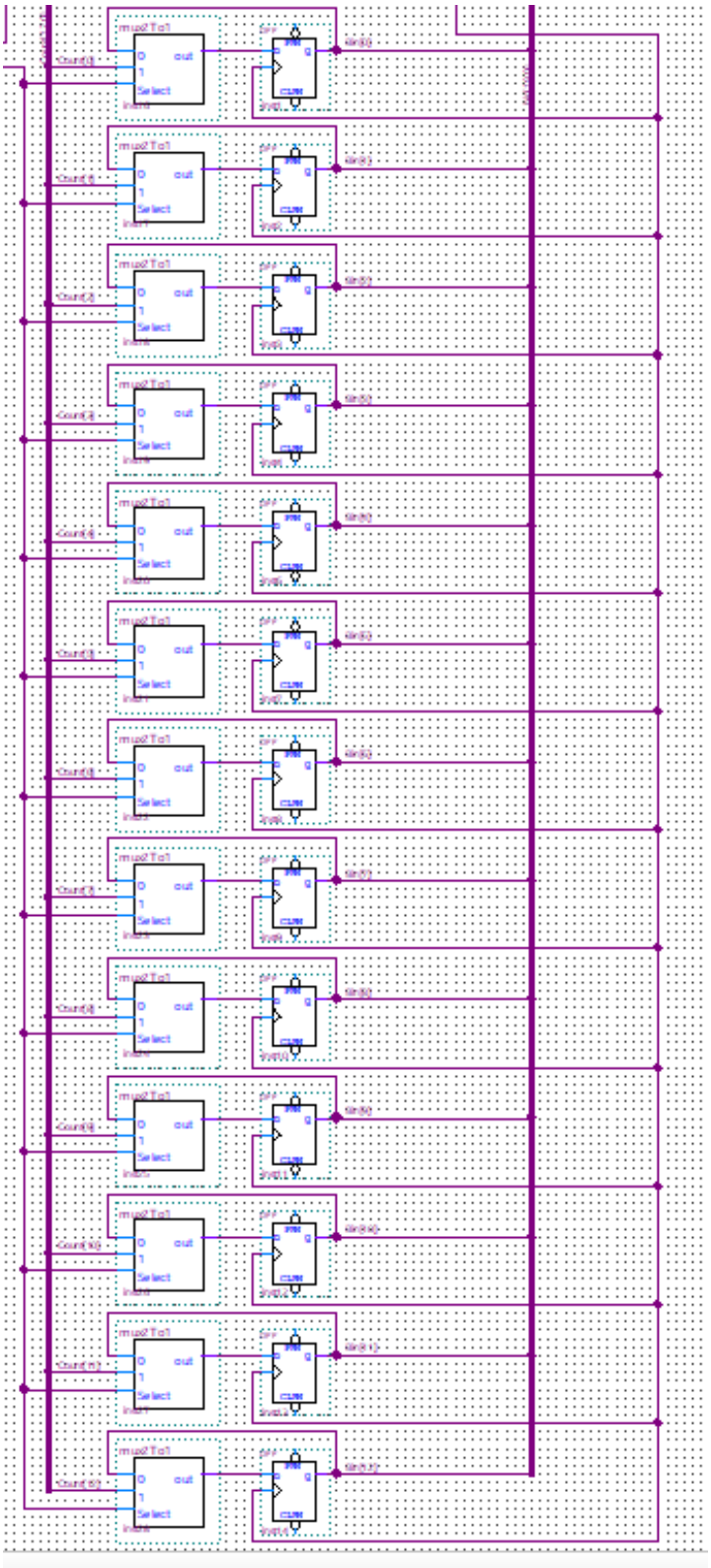
Before creating this code, I attempted to create the same logic with a block diagram. After deciding this would complicate things further (as this would need to have additional register files and an adder for 13 bits, along with logic for the error LED), I implemented the logic with Verilog as this is much simpler and programming is better suited for me.

The logic of the code uses multiple if/else if/else statements to add the input dollar amount to the output counter bus, into the register files.

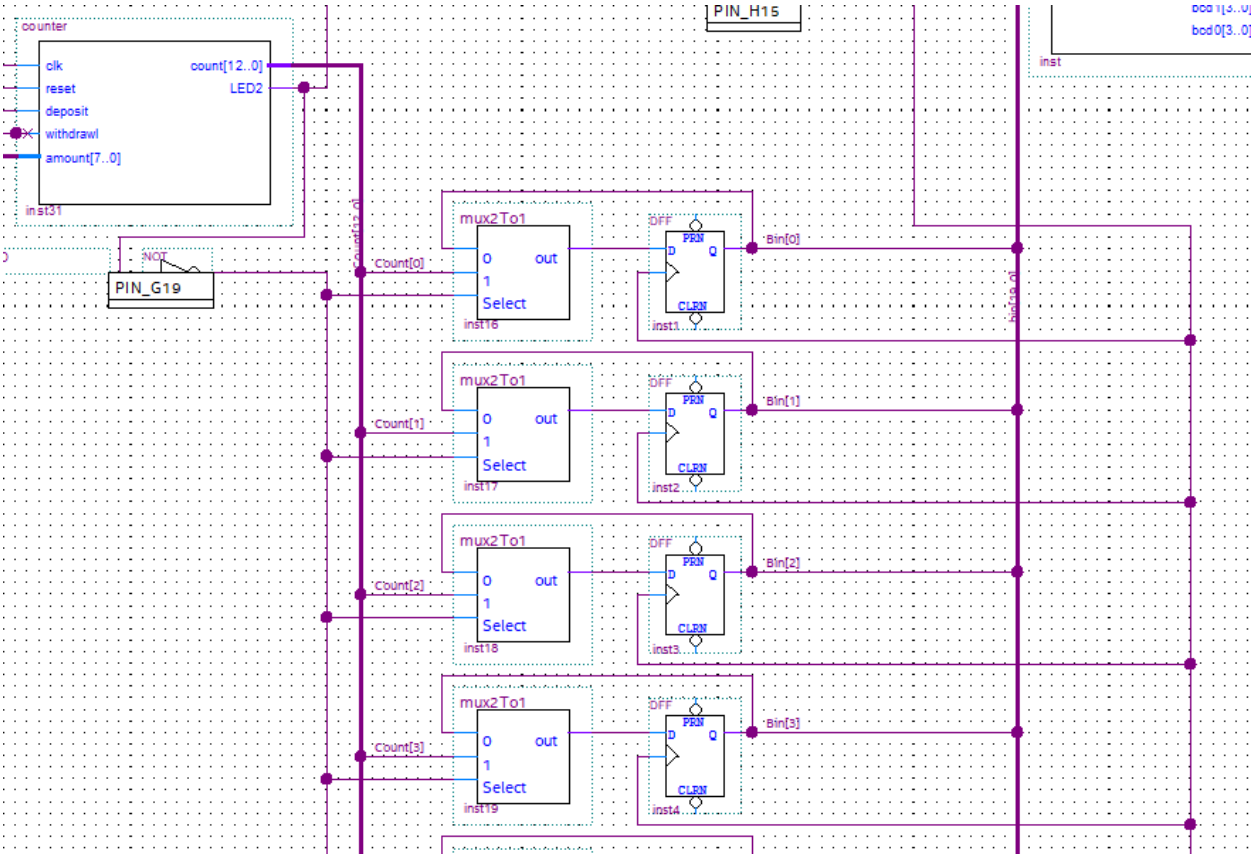
It also uses if/else if/else statements for the logic of the error LED. The LED is lit up if adding the dollar amount would exceed the limit, or $8192, or if it would cause the amount to go negative.

Mark Pyatak
05/03/2024
CPRE 281

REGISTER FILES



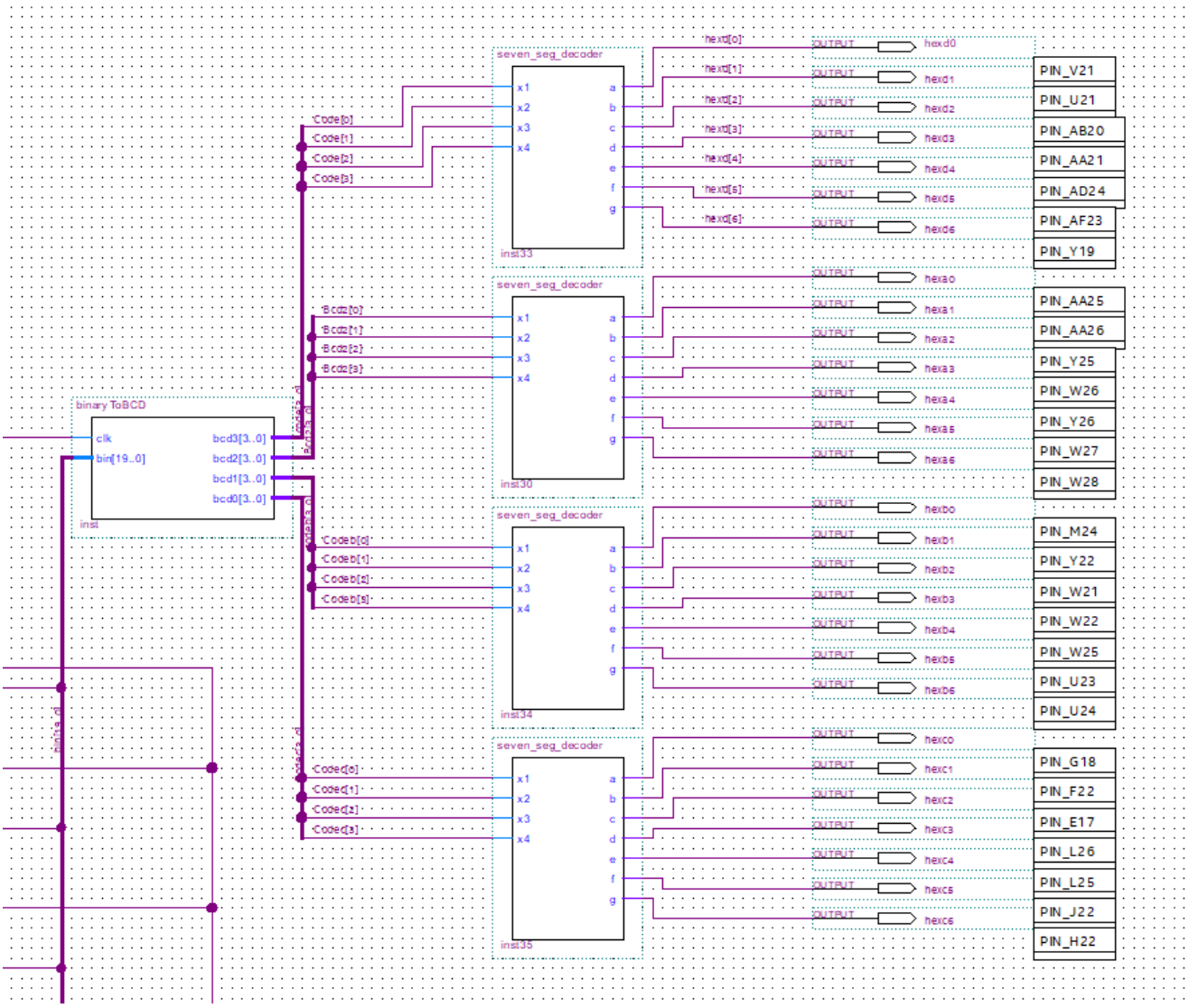
Mark Pyatak
05/03/2024
CPRE 281



There are a total of 13 register files, to store up to $8192 for the display. Each register file uses one two-to-one mux, and a D Flip Flop for storage. If there are no LED errors, the register files will load what the count bus outputs from the Counter module. If there is an error, the register files will not load the data and will instead continue storing the previous data. The register files are connected to a bus and get converted to the seven-segment displays.

Mark Pyatak
05/03/2024
CPRE 281

SEVEN SEGMENT DISPLAY



These are all the outputs for the four seven-segment displays. It uses the data from the register files to output the dollar amount in the account. These seven segment decoders were taken from a previous lab, and are made with Verilog code.