Digital Design 1 Lab

Student Name: Sherif Sakran

Student ID: 900193609

Lab Project: Vending Machine

Objectives

Building a vending machine to integrate the concepts of Verilog taught in the recent labs, mainly the lab of the digital clock and finite state machine labs.

Tools:

Basys 3 board

Xilinx Vivado software

Results:

I implemented the vending machine, downloaded it on the FPGA, and tested it. It works as expected such that it takes input from the user whether 1, 5, or 10 cents and start pouring once it is 20 or more. The cents paid are shown on the seven-segment display. There is a led to show that the water is pouring. Overall, everything works fine.

Program documentation

1) Receiving the inputs from the user

The program's interactions with the user as the following:

It takes coins (1, 5, or 10 cents) then starts pouring water once it reaches 20.

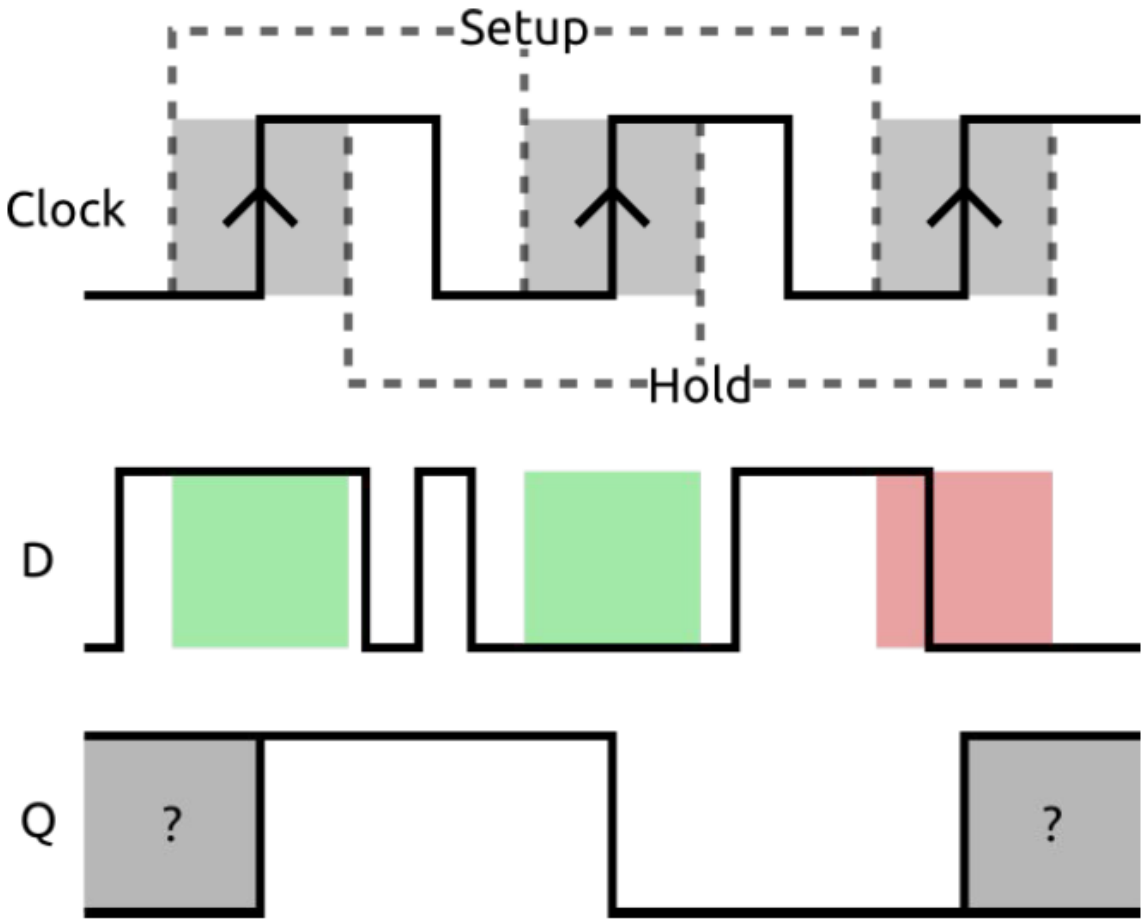
The number of cents is always shown on the display, and if the user entered more than 20, for example (10+5+1+1+1+10) then the display after receiving the 10 after the total was 18, it will show 08 (28-20) and starts pouring water.

If the user pushed stop, then the water stops pouring the remaining coins on the screen.

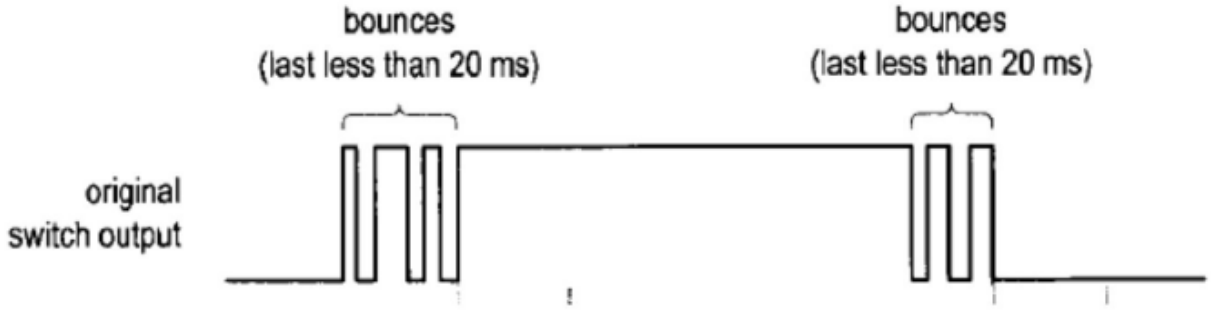
If the machine were not used for 30 seconds whether when pouring or not, it resets.

There is an option for resetting the whole system.

Therefore, it is clear that the user interacts with the system through buttons. Although the program was implemented online using the UART module, it considers the real life where the mechanical buttons are used. And here comes the importance of using the synchronizer and debouncing modules. The first one to avoid the metastable event in which violations to the setup and hold times occur as shown in figure 1.



The second one for solving the debouncing that occurs to the switch before it keeps stable as shown in figure 2.



Also, a rising edge detector module was used, from its name, to detect a rising edge from the input. In other words, to detect a tick when the input changes from 0 to 1.

After the inputs are customized to receive the pushes from the user through the mechanical switches, the actual program and implementation starts here.

In the VendingMAchineMain module which represents the main module, the inputs are the clk of the fpga that has a frequency of 100 Mhz, the RxD that is used for the UART module, and the outputs are an enable vector to choose which one of the segment display will display a specific number, LEDs vector to show the display the number after being received from the BCD module as (a to g) bits, and the out that shows the led of the pouring that I have connected to the second led on the FPGA in the constraint file.

Wires are used to obtain the pushes of the user after being received from the UART, synchronizer, debouncer, then risingEdgeDetector modules.

2) Handling the program

We need to do the following:

The number to be responsive for the pushes and to be displayed. In order to display the number, mux and decoder are used. In order for the decoder to work, it receives the output form a binary counter that counts from 0 to 3 to show one of the four digits on the screen of the FPGA. A clock divider is used to make a clock of 100 Hz which the human eye will not recognize when the display of the 4 seven-segment displays from one to another. Let us assume we already have the number that we need to display, so we need to take every digit of it which are units and tens, then pass it to the BCD module that will produce the representation readable by the seven-segment display. The remaining part now is basically how I obtained the number. Here is the main work done for the program in the VendingMachine module.

3) The VendingMachine module

We can divide the “state” of the program using the Moore method into 30 states: for the cents from 0 to 9, we have two states such that one is when the output is 0 and the other when it is 1. And a state for each of the 10 to 19 cents. But here I used a different methodology, it could be interpreted as a Mealy representation. Inputs are the clk of the FPGA, the three inputs for the coins, one for the stop, and one last for the reset. Cases could be from 0 up to 29, so the output states are saved in a vector stores up to 32 (5 bits). An output out is used for the led of the pouring. Since both outputs (out and state) change inside an always block, they are declared as registers. We need the following:

When the user pushes the first button, the cents are increased by one, and so on with the other two buttons for the 5 and 10 cents. And here we have the following always block do the that. The always block will be entered with every positive edge for the clock or negative edge of the reset. This is the clock of the FPGA of the 100 MHz. When the reset is used, then we need to reset the whole system such that the coins displayed to be zero and the water is not pouring, and that's why we have the first to lines of the block. The rstCounter will be explained later why it is used. All statements are non-blocking because basically there are no dependence here, so we do not need to block one statement until the other is implemented. If the user enters one cent, then the display needs to add one more cent to the previous value but we need to consider the case when the added cent causes the total to be at least 20, or in other words when the previous value is basically 19. In that case, we need to show a zero and start pouring (out to be 1) with the 20 cents formed. If that was not the case, then the state will be updated with the addition of that one cent. The same applies for the 5 and 10 cents. The use of the if statement [if (out ==1) out <=1] is to handle the case when the water is pouring (out =1) then the user adds more coins. In that case, the program needs to stop pouring and continue adding the coins. When the stop button is pressed, then we need to just stop pouring, so in that case we make out equal to 0. Till this point, we have a vending machine that receives the input from the user and increase the number of cents whenever the users add more coins (push switches) then will pour once the cents reaches 20 and display the remaining. If the user pushed stop, then it will stop pouring, and if they pushed reset the whole system will reset. We need to add the feature that after 30 seconds the system will reset (whether it was pouring or not). And here comes the idea of making a counter that counts from 0 to 30 and increase every second.

*Untitled - Notepad

File Edit Format View Help

```
      always @(negedge rst, posedge clk)
   begin
      if(rst)
          begin
              state <= 0;
              out<=0;
              rstCounter<=1;
          end

              else if(in1) // if(in1 &&out!=1)
          begin
              if (out==1)
                  out<=0;
              rstCounter<=1;
              if (state+1==20)
                  begin
                  state <= 1'b0;
                  out <= 1;
                  end
              else
                      state <=state +1;
          end
          else if(in5 )
          begin
              if (out==1)
              out<=0;
              rstCounter<=1;
              if (state+5>=20)
                  begin
                  state <= state - 15;
                  out <= 1;
                  end
          else
                  state <=state +5;
          end
          else if(in10 )
          begin
              if (out==1)
              out<=0;
              rstCounter<=1;
              if (state+10>=20)
                  begin
                      state <= state - 10;
                      out <= 1;
                  end
              else
                  state <=state +10;
          end
          else if(inStop)
          begin
              rstCounter<=1;
              out <= 0;
          end

          else if (counter==30)
          begin
              state <= 0;
              out<=0;
              rstCounter<=1;
          end

          else
              rstCounter<=0;
   end
```

```
always @(posedge clk1sec, posedge rstCounter)
if (rstCounter)
    counter<=0;
else if (clk1sec)
     counter<=counter+1;
```

By now, it might be clear that another always block will be used to implement that counter that will detect a pass of 30 seconds. It will take the output of the clk divider of a one-Hz clock (a positive edge every second). Then every second, we will increase the conter by one. So by now, we can measure a 30-seconds period. So returning back to the previous always block, we will have an if statement when the counter reaches 30, so the system will reset which is as same as what happens when a reset button is pressed. That clarifies making the counter start from 1 when the system resents in order to start counting the 30 seconds from the beginning. Note that I did not directly make the counter equal to 1 because basically it is used in the other always block, and no registers could be used in two different always block, so here comes the need to the rstCounter as an aget between the two blocks. But that means the system will reset every 30 seconds, and we need it to reset only when the user does not push any thing for 30 seconds. In other words, every interaction between the user and the machine will make the counter to start over, and that's why I made the rstCounter to be 1 in every if statement that represents an interaction (a push). If no interactions occur, the the counter keeps counting, and for the counter to count, the rstCounter must be 0, so here comes the use of the last condition (else) when no interactions occur, then the rstCounter is equal to 0 in order to continue counting. The feature of resetting after no interactions for 30 seconds is achieved.

We need to return back to the assumption that we already have the digits that will displayed on the FPGA and check how we got them. As I mentioned, the VendingMachine module outputs the state which represetns the number of conins, and that is basically the number we need to display. However, the seven-segment display requires digists to display, and that is why I splitted the number into digits in the main module (VendingMachineMain) after receiving the number itself from the call of the VendingMachine module that we just covered.

Then, the output from the BCD enters the multiplexer to choose which one will be displayed on the seven-segment at the very moment. The first two digits are always set to 0 because basically the input coins display more that 19.

References

Lab 7, 8 manual, slides, and projects.