Project 2 Digital Alarm Clock

Digital Design I – FALL 2022

Mohamed Noureldin - 900203758
Gehad Ahmed - 900205068
Islam Hassan - 900213579
Reem Said - 900201275

Introduction

In this project, we were required to design and implement a Digital Clock having the Alarm functionality. The project was first designed using block diagrams, followed by Logisim simulations and the designing of Finite State Machines. Then we were asked to implement the design using Verilog for an FPGA board of type BASYS 3.

Specifications

The implemented design uses only 5 LEDs (LD0, LD12, LD13, LD14 and LD15), the seven-segment display and 5 push buttons (Center, Left, Right, Up and Down). The seven-segment display is used to display the time where the two left digits are used to display the hours (from 0 to 23), and the two right digits are used to display the minutes (from 0 to 59).

The digital clock has two modes that the user is able to switch between by clicking the center push-button (BTNC). These two modes are the time or clock mode and the adjust mode. In the time mode, the current time is displayed on the seven-segment display. Moreover, the LD0 is constantly OFF until the time of the clock matches the time set for the alarm indicating the ringing of the alarm. When the time matches the alarm, LD0 turns on and stays on until any push button is clicked to stop the alarm. Since the digital display only displays the hours and minutes, the second decimal place is made to blink once every second to indicate the passage of a second.

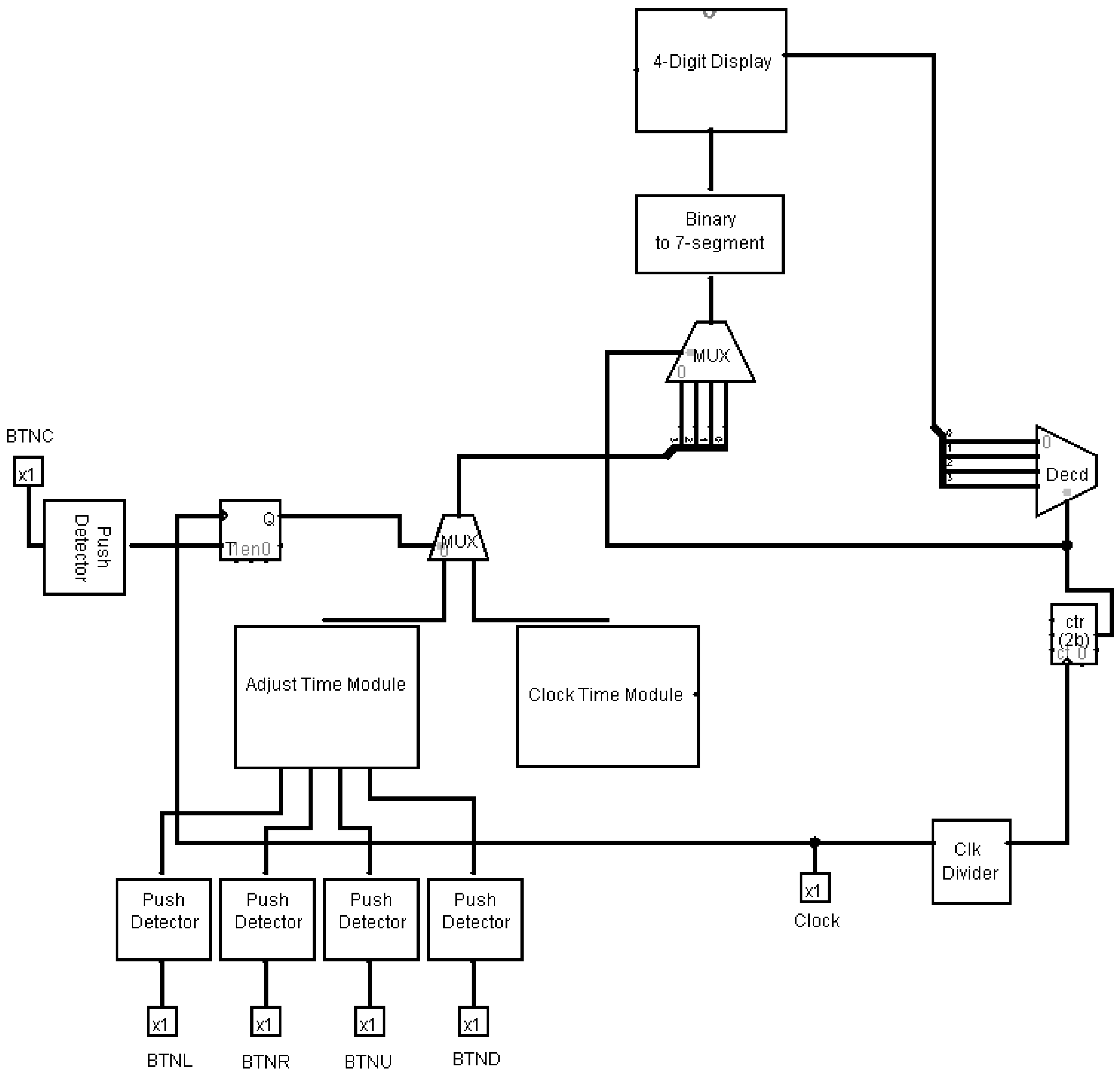
On the other side, the adjust mode is indicated by having LD0 switched on and the decimal point not blinking. When first entering the adjust mode the left two digits are used to display the current time hours when entering the adjust mode and the right two digits are used to display the current time minutes. These displays are changed so that the left two digits are used to display the alarm hours and the right two digits are used to display the alarm minutes when adjusting the alarm time. What is being adjusted is indicated by having a LED ON. LD15 is ON when adjusting the time hours, LD14 is ON when adjusting the time minutes, LD13 is ON when adjusting the alarm hours and LD12 is ON when adjusting the alarm minutes. Clicking on the right or left push button should change what is currently being adjusted. The user is initially assumed to be editing the time hours and each click on the left push button should make him move one left, while each click on the right push button should make him move one right so that the user moves in the circle:

Time Hours <-> Time Minutes <-> Alarm Hours <-> Alarm Minutes
Start

Clicking on the up or down push buttons should then accordingly increment or decrement the selected quantity by one.

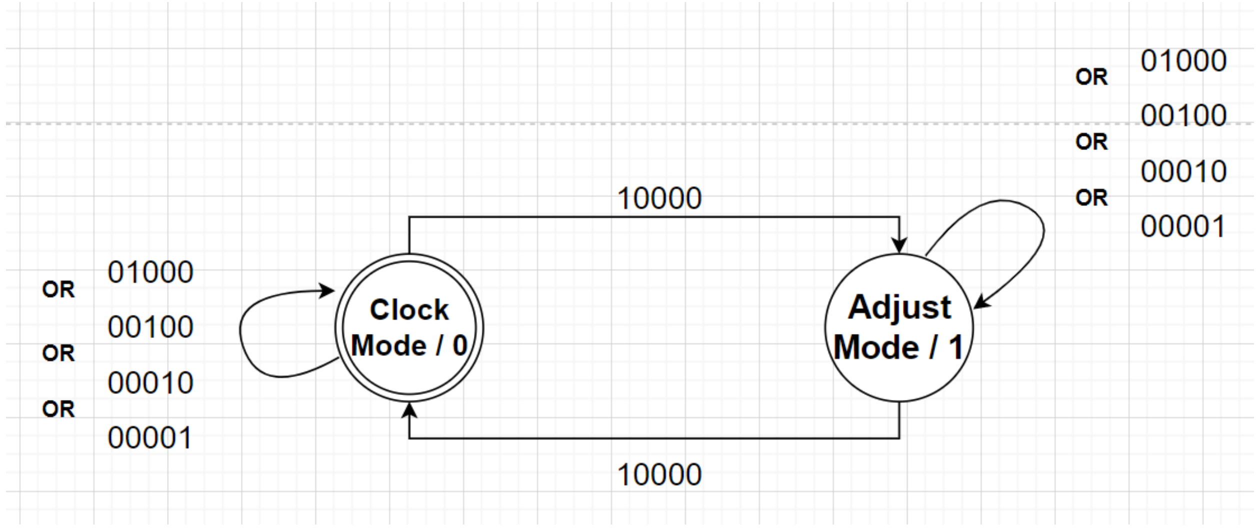
Block Diagrams

As shown in the diagram below, the BTNC is the selection line, through a T-Flip Flop, of the multiplexer that chooses between the clock module and the adjust-time module.The output of both modules is the hour's tens, hour's units digits, minutes' tens digits, and the minutes' units digits. This output is the input of a multiplexer with a selection line from a counter with a frequency 100 Hz. This counter is also a selection line for the decoder that switches the digits displayed across the four seven-segments.

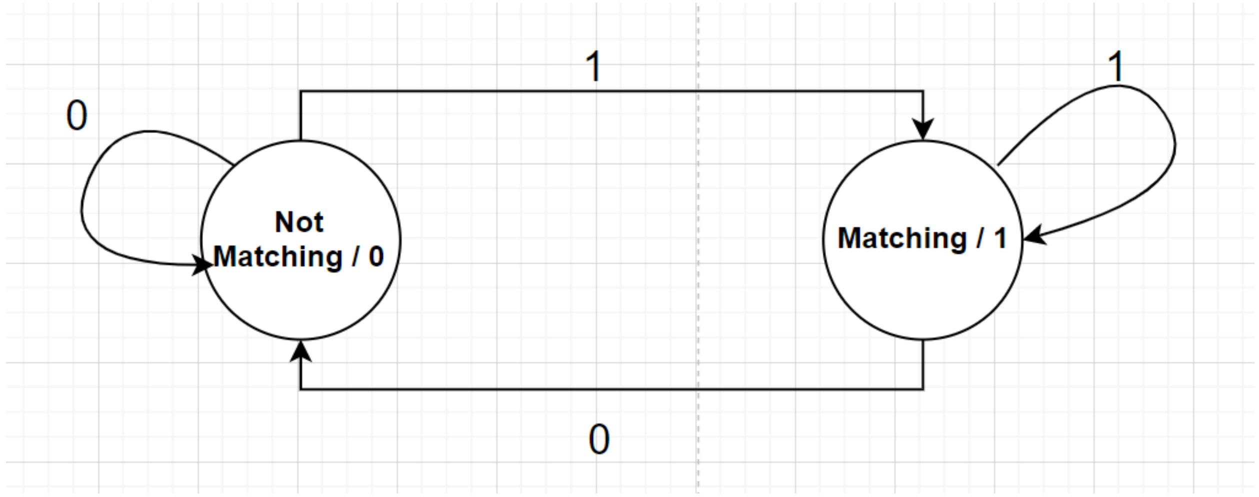


FSM Diagrams

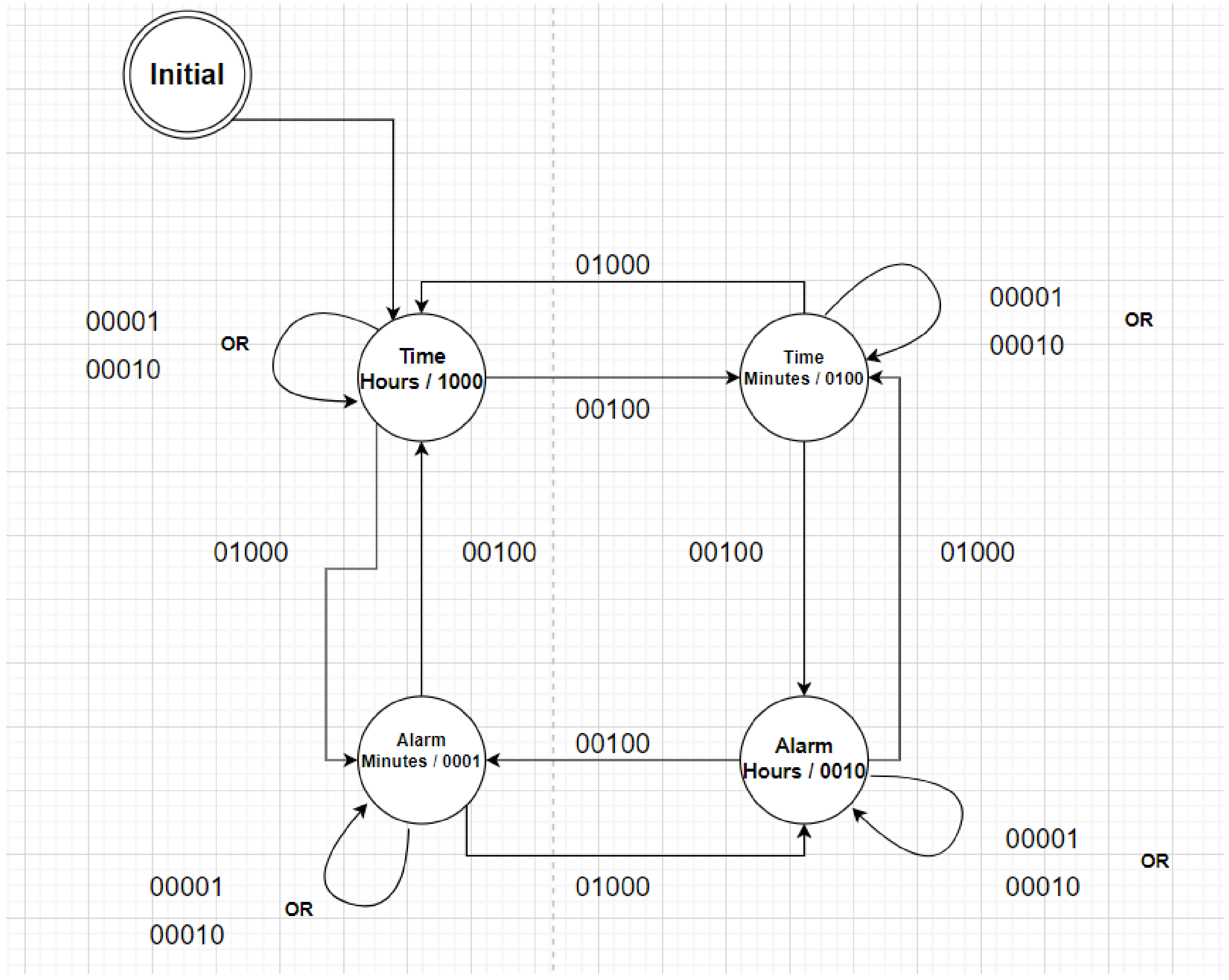
Our FSM design had one main state diagram, and each state had a substrate diagram. The Clock/Adjust mode was our main diagram, and it switches between them using the BTNC button. If any other button is pressed, it remains in the same state.



In the clock mode state, there is a matching/not matching state diagram which compares the time of the clock and the time of the alarm set. The buzzer we installed doesn't go off (output = 1) until the time of the clock is equal to the time of the alarm. When the two timings become equal, the buzzer rings until it is turned off by pressing any button.



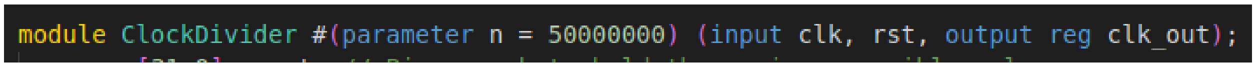
In the adjust mode state, the user can change the clock's time or set the alarm. It starts with the adjust mode on the clock's hours digits, and if the user presses BTNR, it shifts to the clock's minutes, then the alarm minutes, and so on. If they press BTNL, the adjust mode will shift from the clock's hours to the alarm's minutes, the alarm's hours, and so on, until it reaches the clock's hours again. Once the user sets the adjust mode on the digit they want to change, they can press BTNU to increment the number or BTND to decrement. To go out of the adjust mode and into the clock mode again, they just have to press the BTNC button. The user can tell which digit is being adjusted by the LED that is on. For example, if the adjust is on the clock's minutes, then the second LED from the left will be on, and if the Alarm's minutes is the one being adjusted, then the fourth LED will be the one switched on, and the rest will be off. As long as the adjust stays on the same digit, the same LED will stay on and only turn off if the user changes the digit they want to adjust.



Implementation Summary

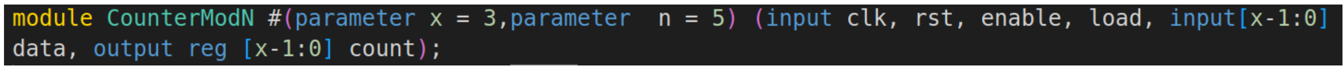
Here are the Verilog Modules we used with their usage in the project:

- **Clock Divider:** a generic clock divider that has a parameter the required frequency division and the output frequency follows this equation (freq_in / 2 n) where n is the input parameter.



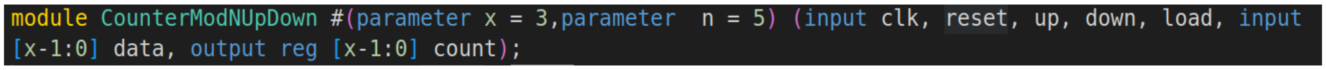
```
module ClockDivider #(parameter n = 50000000) (input clk, rst, output reg clk_out);
```

- **CounterModN:** a counter that takes x number of bits and increments the counter until it reaches n-1 (output the count mod n).



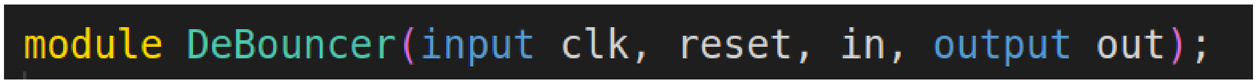
```
module CounterModN #(parameter x = 3,parameter  n = 5) (input clk, rst, enable, load, input[x-1:0]
data, output reg [x-1:0] count);
```

- **CounterModNUpDown:** an up and down counter that takes x number of bits, increments and decrements the counter mod n.



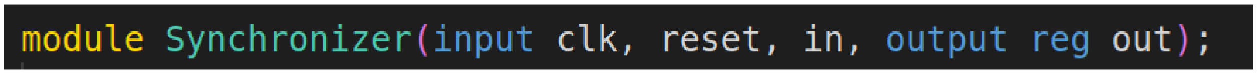
```
module CounterModNUpDown #(parameter x = 3,parameter  n = 5) (input clk, reset, up, down, load, input
[x-1:0] data, output reg [x-1:0] count);
```

- **DeBouncer:** a module for removing unwanted input noise from buttons to help in the detecting of the click. It uses three decoders; the first decoder's output is connected to the second decoder's input, the second decoder's output is connected to the third decoder's input, and the third decoder's output is the output of the debouncer.



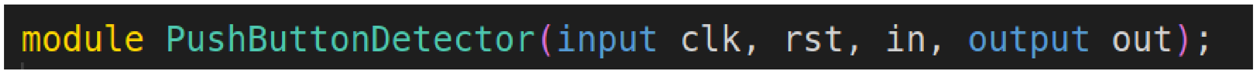
```
module DeBouncer(input clk, reset, in, output out);
```

-
- **Synchronizer:** a module to delay the output so it is synchronized with the clock. It has two decoders; the output of the first decoder is connected to the input of the second decoder, and the output of the second decoder is the output of the synchronizer.



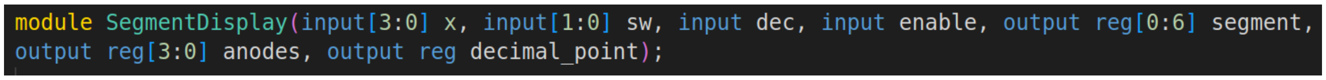
```
module Synchronizer(input clk, reset, in, output reg out);
```

- **PushButtonDetector:** a push button detector that outputs a signal of one clock cycle for the button click. The push button detector has a rising edge detector who's output is connected to the input of the debouncer. The output of the debouncer is the input of the synchronizer. The output of the synchronizer is the output of the push button.



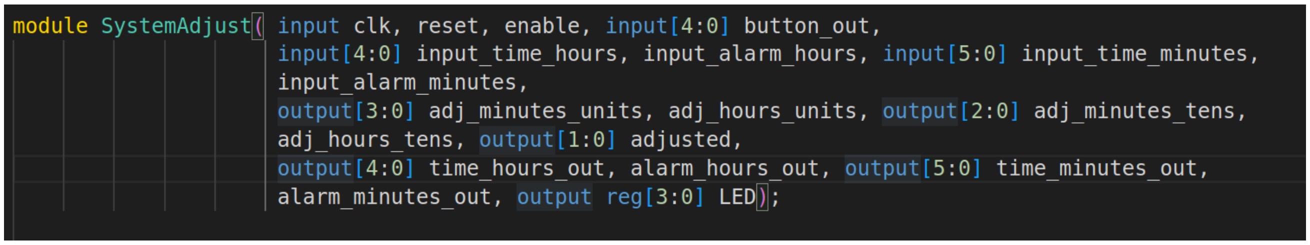
```
module PushButtonDetector(input clk, rst, in, output out);
```

- **SegmentDisplay:** has all the configurations of the segment display, to display the ten digits and the decimal point.



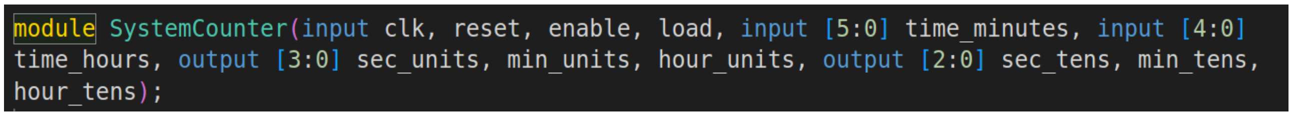
```
module SegmentDisplay(input[3:0] x, input[1:0] sw, input dec, input enable, output reg[0:6] segment,
output reg[3:0] anodes, output reg decimal_point);
```

- **SystemAdjust:** a module used for the adjust mode that takes as input the current time and the current set alarm and outputs the digits to be displayed on the seven-segment display along with the new set time and the new set alarm with a flag indicating which one of them was modified or both.



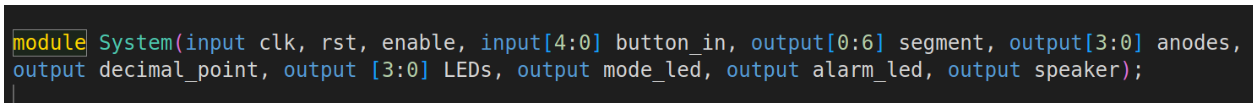
```
module SystemAdjust( input clk, reset, enable, input[4:0] button_out,
                     input[4:0] input_time_hours, input_alarm_hours, input[5:0] input_time_minutes,
                     input_alarm_minutes,
                     output[3:0] adj_minutes_units, adj_hours_units, output[2:0] adj_minutes_tens,
                     adj_hours_tens, output[1:0] adjusted,
                     output[4:0] time_hours_out, alarm_hours_out, output[5:0] time_minutes_out,
                     alarm_minutes_out, output reg[3:0] LED);
```

- **SystemCounter:** a module used for the clock mode that takes as input the load time and outputs the digits to be displayed on the seven-segment display



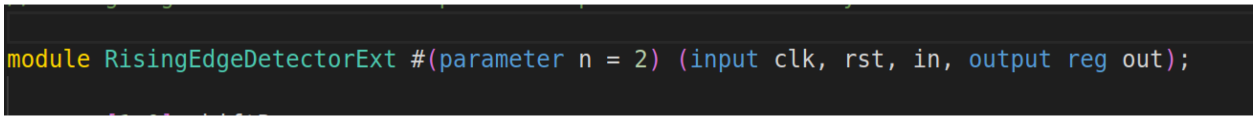
```
module SystemCounter(input clk, reset, enable, load, input [5:0] time_minutes, input [4:0]
time_hours, output [3:0] sec_units, min_units, hour_units, output [2:0] sec_tens, min_tens,
hour_tens);
```

- **System:** a module containing both SystemAdjust and SystemCounter along with the logic to display on the seven segment display and the logic to check for the alarm



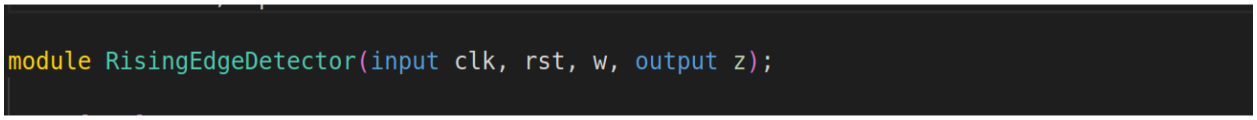
```
module System(input clk, rst, enable, input[4:0] button_in, output[0:6] segment, output[3:0] anodes,
output decimal_point, output [3:0] LEDs, output mode_led, output alarm_led, output speaker);
```

- **RisingEdgeDetectorExt:** a module to give an output of one when the signal changes from 0 to 1 for a count of N(rising edge)



```
module RisingEdgeDetectorExt #(parameter n = 2) (input clk, rst, in, output reg out);
```

- **RisingEdgeDetector:** a module to give an output of one only when the signal changes from 0 to 1 (rising edge)



```
module RisingEdgeDetector(input clk, rst, w, output z);
```

- **RisingEdgeDetector:** a module to give an output of one only when the signal changes from 0 to 1 (rising edge)

Implementation Issues

- Since Verilog is a hardware implementation language, it works with code in a parallel mode which although makes Hardware much faster than Software, it made it harder for us to program since we are all used to the sequential coding paradigm
- Since both the alarm and time mode are working concurrently in the background with only one of them being displayed to the user, we had a problem that we needed to include a flag to indicate that the user is exiting the adjust mode and

another flag to indicate that we are entering the adjust mode so that we can load data to the adjust mode and get output from it

- At first we were checking the alarm time using hours and minutes so the alarm would ring correctly but when turning it off in the same minute the system would recheck the time and find it still equal and so rings again.

Validation Activities

- We propagated through the different modules of the project step by step. We first design the module on Vivado, create testbenches to check the validity of the signal, and then implement the design on the FPGA board.
- For the adjust and clock modules, drawing the FSM preceded any code written to make sure the code follows the behavior we intended.

Contributions

1. Islam Hassan:
    - Designed the FSM on a white board.
    - Worked on the Adjust module.
    - Created a testbench to test the seven-segment display and the clock
2. Reem Saeed:
    - Designed the FSM on draw.io
    - Designed the system block diagram
    - Worked on the System module.
3. Mohamed Sherif:
    - Designed the FSM machine on a white board
    - Worked on the Adjust module
    - Designed the binary counter, adding two features up and down to account for different behaviors inside the system.
4. Gehad Salem:
    - Worked on the System module.
    - Created testbenches to test the System module as whole.
    - Designed the buzzer module.

References

- http://www.sunburst-design.com/papers/CummingsSNUG1998SJ_FSM.pdf
- https://www.fpga4fun.com/MusicBox2.html