Alarm Clock Report

ELEC 3500

Chantel Lepage 100999893

Caleb Gryfe 101009798

December 3, 2018

Introduction

In the course we are tasked to create an alarm clock using the knowledge that we have gained throughout the course and to add our own innovation to the clock. The structure is a basic alarm clock that is displayed in 12-hour time, shows a light for am and pm time, allows you to increment the hours and minutes, and has one functioning alarm clock that the user can set. Throughout this lab a series of modules where created and used to create our alarm clock

Design

Diagrams

Seen in the figure below is the top-level schematic of the alarm clock that was implemented.

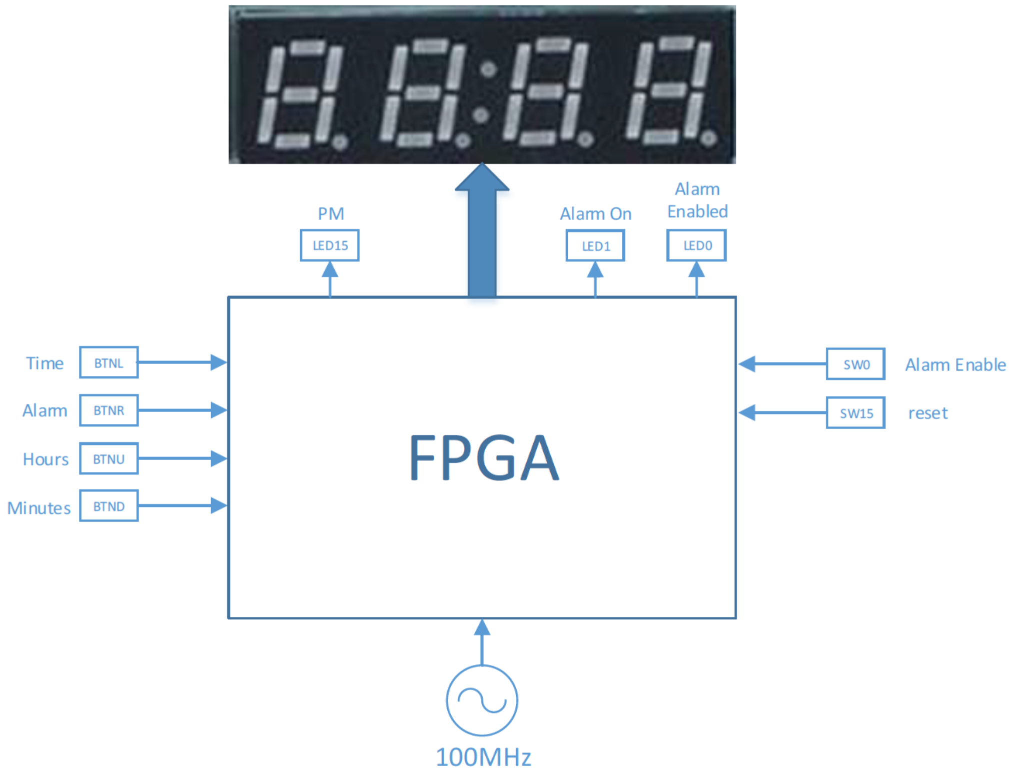
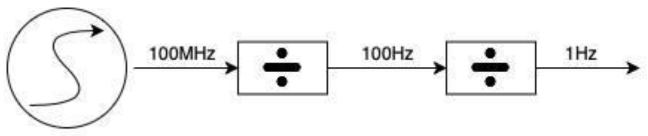


Figure 1: Top Level Schematic for Alarm Clock

Below can be seen the block diagram for the layout of our modules that were implemented on our FPGA.



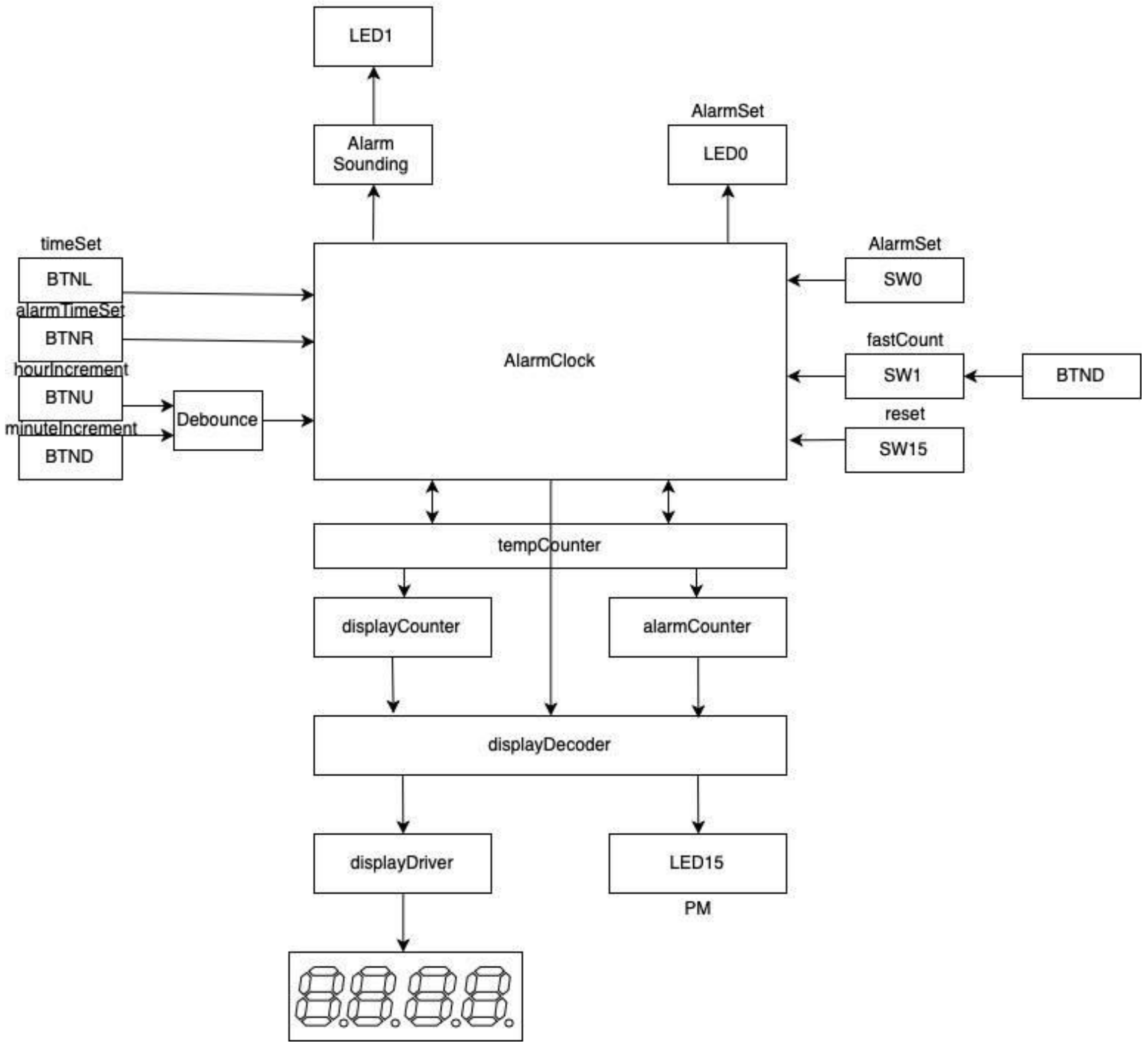


Figure 2: Block Diagram of Modules and Inputs

Code

Master Controller

This is the section of the code that calls all the modules together and allows for them to work as one to create a functioning alarm clock. This code calls the display driver and the seven-segment display output. This consist of all the logic that is used to create a functionality of the alarm clock. The components that are used in this module are as follows:

- If the alarm enable signal is high it turns on the alarm enabled LED0
- If the alarm enable signal is high and the current time minutes and hours counts up to the alarm time minutes and hours the alarm is turned on until the alarm is disabled by turning the alarm enable signal low
- Based on the button presses, control signals need to be generated for the Time block to set the program value (i.e. current time)
- Based on the button presses, control signals need to be generated for the Alarm Time block to set the program value (i.e. alarm time)
- Based on the button presses, control signals need to be generated for the Seven Segment/LED15 Mux block and Time block to select which data to display on the seven segment display.

To create the time comment of the clock a 1 Hz clock signal is used, as the base counter is seconds. Then, this is fed into a block where a 500 Hz refresh rate is generated to ensure the human eye can see the numbers on the display ports on the FPGA. The buttons BTNU and BTND are used to change the hours and minutes when pressed.

For the alarm portion, this was done by creating a display time that is displayed when BTNR is pressed. This can be adjusted using BTNU and BTND respectively. The alarm time is then compared to the current clock time, and the alarm is triggered when then are the same time.

```
`timescale 1ns / 1ps
/*
*** ELEC 3500 Lab 9 --Alarm Clock--
*** Caleb Gryfe && Chantel Lepage
*** Code Submission Date: 03/12/18
*/
module bcd(
input [3:0] A,
output CA,
output CB,
output CC,
output CD,
output CE,
output CF,
output CG);
```

```
reg [6:0] T;
assign CA = T[0];
assign CB = T[1];
assign CC = T[2];
assign CD = T[3];
assign CE = T[4];
assign CF = T[5];
assign CG = T[6];
always@(*)
begin
case(A)
4'd0: T = 7'b1000000;
4'd1: T = 7'b1111001;
4'd2: T = 7'b0100100;
4'd3: T = 7'b0110000;
4'd4: T = 7'b0011001;
4'd5: T = 7'b0010010;
4'd6: T = 7'b0000010;
4'd7: T = 7'b1111000;
4'd8: T = 7'b0000000;
4'd9: T = 7'b0010000;
endcase
end
endmodule
module alarmClock(
input CLK100MHZ,
input reset,
input fastCount,
input hourIncrement,
input minuteIncrement,
input alarmTimeSet,
input alarmSet,
input timeSet,
output [7:0] sevenSegment,
output [7:0] AN,
output [2:0] LED
);
reg [26:0] clock;
reg [26:0] clock2; // f=10Hz
reg [26:0] clock3; // f=5Mhz
reg [16:0] counter; // f=1Hz
reg [16:0] displayCounter;
reg [16:0] alarmCounter;
reg [16:0] tempCounter;
reg [3:0] LED_ON;
wire [5:0] seconds,minutes,hour;
wire [3:0] D0;
wire [3:0] D1;
wire [3:0] D2;
wire [3:0] D3;
wire [3:0] D4;
wire [3:0] D5;
wire [3:0] D6;
wire [3:0] D7;
wire CA;
wire CB;
wire CC;
wire CD;
wire CE;
wire CF;
wire CG;
assign sevenSegment[0]=CA;
assign sevenSegment[1]=CB;
assign sevenSegment[2]=CC;
assign sevenSegment[3]=CD;
assign sevenSegment[4]=CE;
assign sevenSegment[5]=CF;
assign sevenSegment[6]=CG;
assign sevenSegment[7]=0;
initial
begin
displayCounter <= 17'd3600;
alarmCounter <= 17'd3600;
end
always@(posedge CLK100MHZ) // 1Hz from 100MHz
begin
if(clock == 27'd99999999)
clock <= 27'd0;
else
clock <= clock + 27'd1;
end
always@(posedge CLK100MHZ) // 100Hz from 100MHz
begin
if(clock2 == 27'd999999)
clock2 <= 27'd0;
else
clock2 <= clock2 + 27'd1;
end
always@(posedge CLK100MHZ) // 5MHz from 100MHz
begin
if(clock3 == 27'd19) // 100M/20 = 5M
clock3 <= 27'd0;
else
clock3 <= clock3 + 27'd1;
end
always @(posedge CLK100MHZ)
begin
if(reset)
alarmCounter <= 17'd3600;
begin
if(clock == 27'd1)
if(alarmCounter == 17'd86399)
begin
alarmCounter <= 17'd3600;
end
else if(alarmTimeSet && hourIncrement)
// Set alarm time
-----------------------------------------------
begin
if((alarmCounter/3600)%60<24)
begin
if(alarmCounter >= 17'd43199)
alarmCounter <= alarmCounter - 17'd39600;
else
alarmCounter <= alarmCounter + 17'd3600;
end
```

```
end
else if(alarmTimeSet && minuteIncrement)
begin
if((alarmCounter % 17'd3540 == 0) &&
(alarmCounter != 17'd3600))
alarmCounter <= alarmCounter - 17'd3540;
else
alarmCounter <= alarmCounter + 17'd60;
end
// End of Set alarm time
----------------------------------------
else
alarmCounter <= alarmCounter;
end
end
always @(posedge CLK100MHZ)
begin
if(reset)
begin
counter <= 17'd0;
displayCounter <= 17'd3600;
LED_ON[0] = 0;
LED_ON[1] = 0;
LED_ON[2] = 0;
end
else if(fastCount && timeSet)
begin
if (clock2 == 27'd1)
if (counter == 17'd86399)
begin
counter <= 17'd0;
displayCounter <= 17'd3600;
end
else
begin
counter <= counter + 17'd1;
displayCounter = displayCounter + 17'd1;
end
else
begin
counter <= counter;
displayCounter <= displayCounter;
end
end
else
begin
if (clock == 27'd1)
if (counter == 17'd86399)
begin
counter <= 17'd0;
displayCounter <= 17'd3600;
tempCounter <= 17'd3600;
LED_ON[0] = 0;
end
else if(displayCounter == 17'd46799)
begin
displayCounter <= 17'd3600;
LED_ON[0] = 1;
```

```
end
//if counter reacher 43199 (43199+1 / 60/60 = 12) set LED to
on i.e. PM
//Increment Minutes ----------------------------------------
else if(timeSet && minuteIncrement)
begin
counter <= counter + 17'd60;
displayCounter <= displayCounter + 17'd60;
if((counter % 17'd3540 == 0) || (counter % 17'd3541 ==
0) || (counter % 17'd3542 == 0)|| (counter % 17'd3543 == 0)|| (counter
% 17'd3544 == 0)|| (counter % 17'd3545 == 0)|| (counter % 17'd3546 ==
0)|| (counter % 17'd3547 == 0)|| (counter % 17'd3548 == 0)|| (counter
% 17'd3549 == 0)|| (counter % 17'd3550 == 0)|| (counter % 17'd3551 ==
0)|| (counter % 17'd3552 == 0)|| (counter % 17'd3553 == 0)|| (counter
% 17'd3554 == 0)|| (counter % 17'd3555 == 0)|| (counter % 17'd3556 ==
0)|| (counter % 17'd3557 == 0)|| (counter % 17'd3558 == 0)|| (counter
% 17'd3559 == 0)|| (counter % 17'd3560 == 0)|| (counter % 17'd3561 ==
0) || (counter % 17'd3562 == 0)|| (counter % 17'd3563 == 0)|| (counter
% 17'd3564 == 0)|| (counter % 17'd3565 == 0)|| (counter % 17'd3566 ==
0)|| (counter % 17'd3567 == 0)|| (counter % 17'd3568 == 0)|| (counter
% 17'd3569 == 0)|| (counter % 17'd3570 == 0)|| (counter % 17'd3571 ==
0)|| (counter % 17'd3572 == 0)|| (counter % 17'd3573 == 0)|| (counter
% 17'd3574 == 0)|| (counter % 17'd3575 == 0)|| (counter % 17'd3576 ==
0)|| (counter % 17'd3577 == 0)|| (counter % 17'd3578 == 0)|| (counter
% 17'd3579 == 0)|| (counter % 17'd3580 == 0) || (counter % 17'd3581 ==
0) || (counter % 17'd3582 == 0)|| (counter % 17'd3583 == 0)|| (counter
% 17'd3584 == 0)|| (counter % 17'd3585 == 0)|| (counter % 17'd3586 ==
0)|| (counter % 17'd3587 == 0)|| (counter % 17'd3588 == 0)|| (counter
% 17'd3589 == 0)|| (counter % 17'd3590 == 0)|| (counter % 17'd3591 ==
0)|| (counter % 17'd3592 == 0)|| (counter % 17'd3593 == 0)|| (counter
% 17'd3594 == 0)|| (counter % 17'd3595 == 0)|| (counter % 17'd3596 ==
0)|| (counter % 17'd3597 == 0)|| (counter % 17'd3598 == 0)|| (counter
% 17'd3599 == 0)|| (counter % 17'd3600 == 0)&& (counter != 17'd0))
begin
counter <= counter - 17'd3540;
displayCounter <= displayCounter - 17'd3540;
end
end
//Increment Hours ----------------------------------------
else if(timeSet && hourIncrement)
if((counter/3600)%60<24)
begin
counter <= counter + 17'd3600;
displayCounter <= displayCounter + 17'd3600;
if(counter >= 17'd39599)
begin
counter <= counter - 17'd39600;
displayCounter <= displayCounter -
17'd39600;
LED_ON[0] = !LED_ON[0];
end
end
else
begin
counter <= counter;
displayCounter <= displayCounter;
end
//Increment at every clock edge
```

```
else
begin
counter <= counter + 17'd1;
displayCounter <= displayCounter + 17'd1;
end
end
if(alarmSet && (alarmCounter == displayCounter))
begin
LED_ON[3] = 1;
end
if(alarmSet)//if alarmSet switch on turn on LED 0
LED_ON[1] = 1;
else if(!alarmSet) //if alarmSet switch off disable alarm and
lights
begin
LED_ON[1] = 0;
LED_ON[2] = 0;
LED_ON[3] = 0;
end
//Toggle the alarm light when condition is met
if(alarmSet && LED_ON[3]==1)
LED_ON[2] = !LED_ON[2]; //Set LED_ON[2] = 1 for more
consistent results
// Toggle alarm end ------------------------------------
if(alarmTimeSet) //If Alarm set is pressed the alarm time will
display
tempCounter <= alarmCounter;
else
tempCounter <= displayCounter;
end
assign seconds = tempCounter % 60;
assign minutes = (tempCounter/60) % 60;
assign hour = (tempCounter / 3600) % 60;
assign D0 = seconds % 10;
assign D1 = seconds / 10;
assign D2 = minutes % 10;
assign D3 = minutes / 10;
assign D4 = hour % 10;
assign D5 = hour / 10;
assign D6 = 4'd0;
assign D7 = 4'd0;
assign LED[0] = LED_ON[0];
assign LED[1] = LED_ON[1];
assign LED[2] = LED_ON[2];
display_driver
d_d1(CLK100MHZ,reset,D7,D6,D5,D4,D3,D2,D1,D0,CA,CB,CC,CD,CE,CF,CG,AN);
endmodule
```

Display Decoder

The display decoder determines what will be displayed on the output whether it’s the alarm time or the clock time.

```
`timescale 1ns / 1ps
/*
*** ELEC 3500 Lab 9 --Alarm Clock--
*** Caleb Gryfe && Chantel Lepage
*** Code Submission Date: 03/12/18
*/
module displaydecoder(reset, bcd_in, displayOutput);
input reset;
input [3:0]bcd_in;
output reg [6:0]displayOutput;
always @(posedge reset, bcd_in)
begin
if(reset==1)
displayOutput = 7'b0000001;
else
case(bcd_in)
4'd0: displayOutput = 7'b1111110;
4'd1: displayOutput = 7'b0110000;
4'd2: displayOutput = 7'b1101101;
4'd3: displayOutput = 7'b1111001;
4'd4: displayOutput = 7'b0110011;
4'd5: displayOutput = 7'b1011011;
4'd6: displayOutput = 7'b1011111;
4'd7: displayOutput = 7'b1110000;
4'd8: displayOutput = 7'b1111111;
4'd9: displayOutput = 7'b1111011;
default: displayOutput = 7'b0000001;
endcase
end
endmodule
```

Display Driver

The display driver module takes in the inputs from the Seven Segments block and the master controller. This code was previously completed from another lab and was adjusted to be used for this module. This portion provides the output onto the display for the alarm clock.

```
`timescale 1ns / 1ps
/*
*** ELEC 3500 Lab 9 --Alarm Clock--
*** Caleb Gryfe && Chantel Lepage
*** Code Submission Date: 03/12/18
*/
module display_driver(
input CLK100MHZ,
input reset,
input [3:0] D7,
input [3:0] D6,
input [3:0] D5,
input [3:0] D4,
input [3:0] D3,
input [3:0] D2,
input [3:0] D1,
input [3:0] D0,
output CA,
output CB,
output CC,
output CD,
output CE,
output CF,
output CG,
output reg [7:0] AN);
reg [3:0] position;
reg [19:0] counter ;
always@(posedge CLK100MHZ)
begin
if(reset)
counter <= 20'd0;
else
if(counter <= 20'd999999)
counter <= counter + 20'd1;
else
counter <= 20'd0;
end
always@(*)
begin
case(counter[19:17])
3'd0: position = D0;
3'd1: position = D1;
3'd2: position = D2;
3'd3: position = D3;
3'd4: position = D4;
3'd5: position = D5;
3'd4: position = D6;
3'd5: position = D7;
default: position = 4'h0;
endcase
end
always@(*)
begin
case(counter[19:17])
3'd0: AN = 8'b11111110;
3'd1: AN = 8'b11111101;
3'd2: AN = 8'b11111011;
3'd3: AN = 8'b11110111;
3'd4: AN = 8'b11101111;
3'd5: AN = 8'b11011111;
3'd6: AN = 8'b10111111;
3'd7: AN = 8'b01111111;
default: AN = 8'b11111111;
endcase
end
bcd BCD1(position,CA,CB,CC,CD,CE,CF,CG);
endmodule
```

Testing

To test the circuit physical hardware tests where made to determine if the alarm was functioning correctly. These tests where conducted by load the current code to the FPGA to see if it would function appropriately. There were no test benches made as we ran out of time.

Innovation

The innovation that we created for out alarm clock was a fast count function. This would allow for the user to quickly adjust their alarm clock. This is useful for getting from a five-minute mark to the fifty-minute mark in a fraction of the time. The way this was implemented was by using the SW1 switch and increasing the frequency of the clock to allow for the fast count. When SW1 is enable the clock will start to count rapidly until the switch is disabled. The fast count will increase both the hours and minutes respectively.

Hardware Implementation

The code was to run on an FPGA that was provided in the lab. This was done by running a synthesis process, generating the bitstream files and then uploading the code to the FPGA. Through testing and many hours generating bitstream files the we got the alarm clock ready and working to demonstrate to the TA's.

Lessons Learned

Through the course of building the alarm clock we learned a lot about debugging the modules we built. The major one that we had to debug the most was the alarm adjustment. When we initially set up the alarm adjust it would adjust the time properly. Whenever the alarm was adjusted the display would show random numbers and not the ones you wanted. The way we solved this problem was to move the alarm adjust code into the same area as our clock adjust code and it worked perfectly after.

Conclusion

The purpose of this laboratory was to create a fully functional alarm clock using an FPGA and Vivado. Using the previous knowledge gained from prior labs, counters, finite state machines, encoders, registers and logic were implemented to create the alarm clock. To further the functionality of the alarm clock, the innovation of being able to quick adjust the clock was added to the design. Further debugging was required and completed, and the alarm clock served to conclude the overall accumulation of course material.