# CPRE 281 - Final Project

By: *Akhilesh Nevatia* (400542190)(Lab12) (akhilnev@iastate.edu)
*Kausshik Manojkumar* (313899602)(Lab 12) (kausshik@iastate.edu)

## Explanation of the Block Diagram File:

**OVERALL CIRCUIT OF THE PROJECT:**

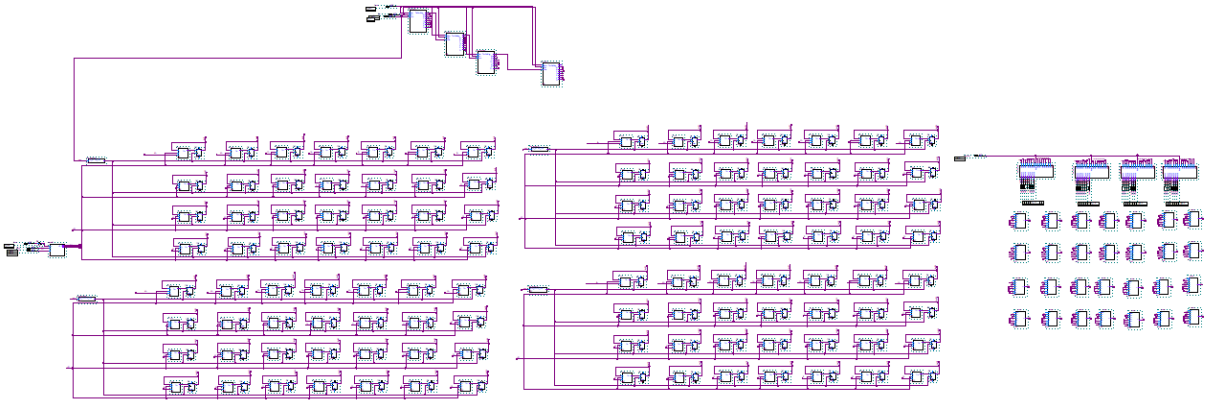


Fig 1 (above) : shows the overarching design of our circuit for the stopwatch

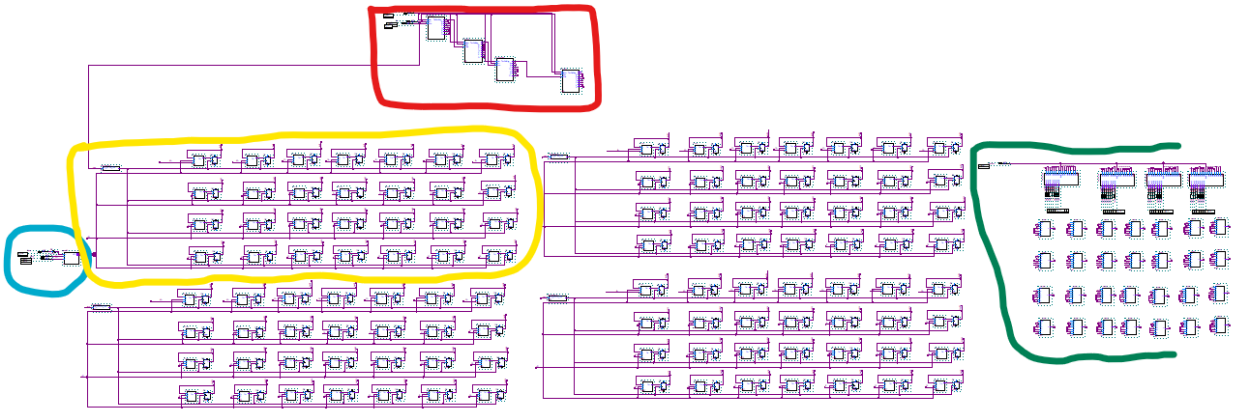


Fig 2 (above) : Marks the various regions of interest of the Stopwatch bdf file

The part highlighted in RED : are the four counters that count the seconds and minutes of the stopwatch ( format of stopwatch : MM : SS)

The part highlighted in GREEN : is the output selection region, it is controlled by a switch (SW0_DB) (choose between displaying time or lapped times/laps)

The part highlighted in BLUE : is the decoder that choose which lap we want to store at a given instant (when the clock is running or is paused and the stopwatch's time is being displayed) and what lap we want to display (when displaying laps)

The part highlighted in YELLOW : 1 of 4 28-bit registers that make up the register file. This section stores the information necessary to display one lap. The 4 7-bit sequences we see are used to store the corresponding HEXADECIMAL display's output so we need the 4 7-bit registers to store the information for one lap.

**OVERVIEW OF COUNTERS USED AND THEIR WORKING:**

**Overview:**

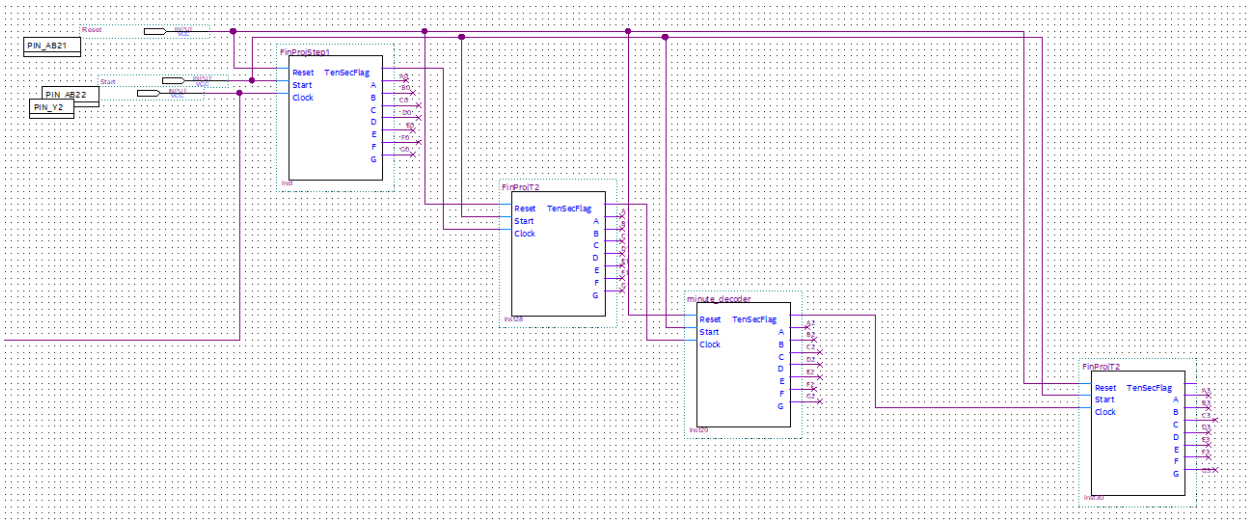



Fig 3 (above) : Circuit diagram/logic for the counting part of the stopwatch

The stopwatch counts 4 four main things, the seconds (0-9) (called: FinProjStep1), ten seconds (0-5) (called: FinProjT2), minutes(0-9) (called: minute_decoder), and ten minutes (0-5) (called: FinProjT2). These four main things are represented by each of these blocks in the above diagram from left to right.
Each of these four blocks have three inputs : RESET, START, CLOCK.
Each of these four blocks have eight outputs : a FLAG, and 7 outputs of the 7 segment display.

**Working:**

The START button effectively functions as a start/stop or pause play button. It tells the counter when to start or stop counting.
The RESET button clears all outputs and counting and starts the counter afresh at 0, so it becomes 00:00.

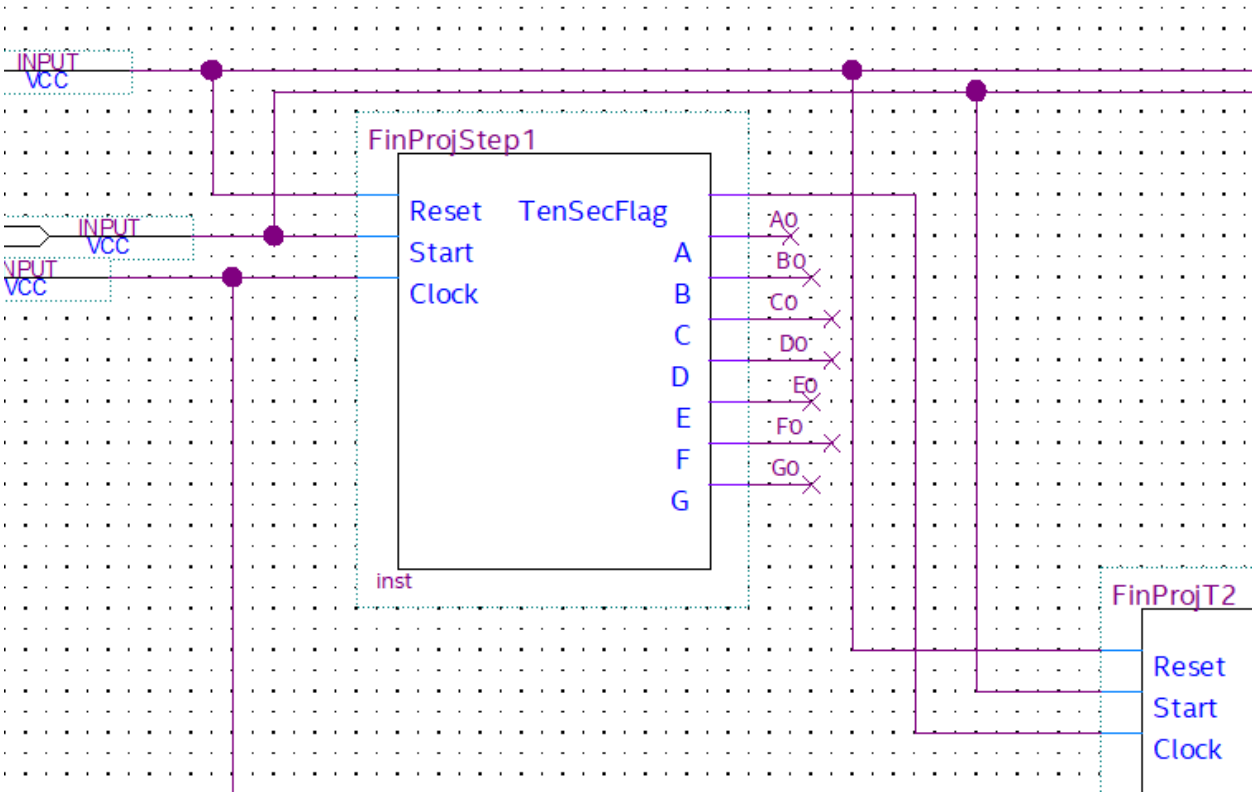



Fig 4 (above) : shows the FinProjStep1 block that counts from 0-9 seconds

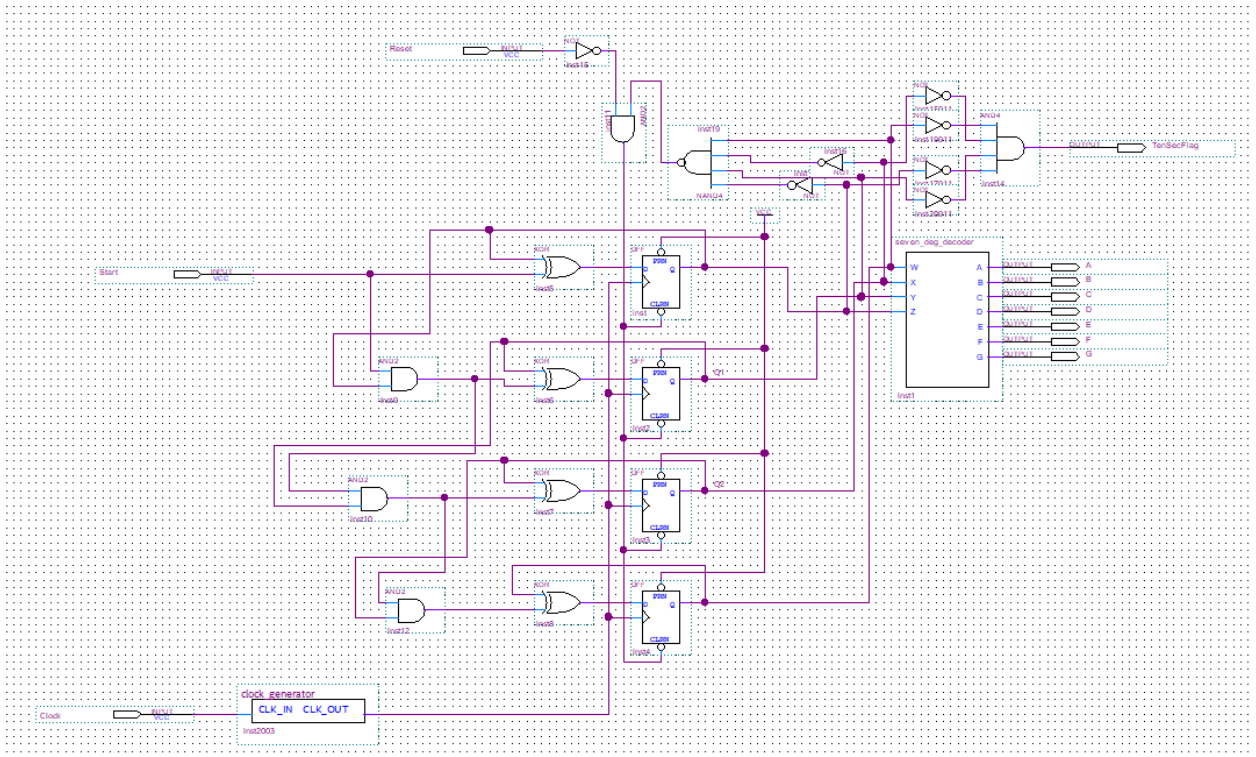


Fig 5 (above) : the FinProjStep1 block

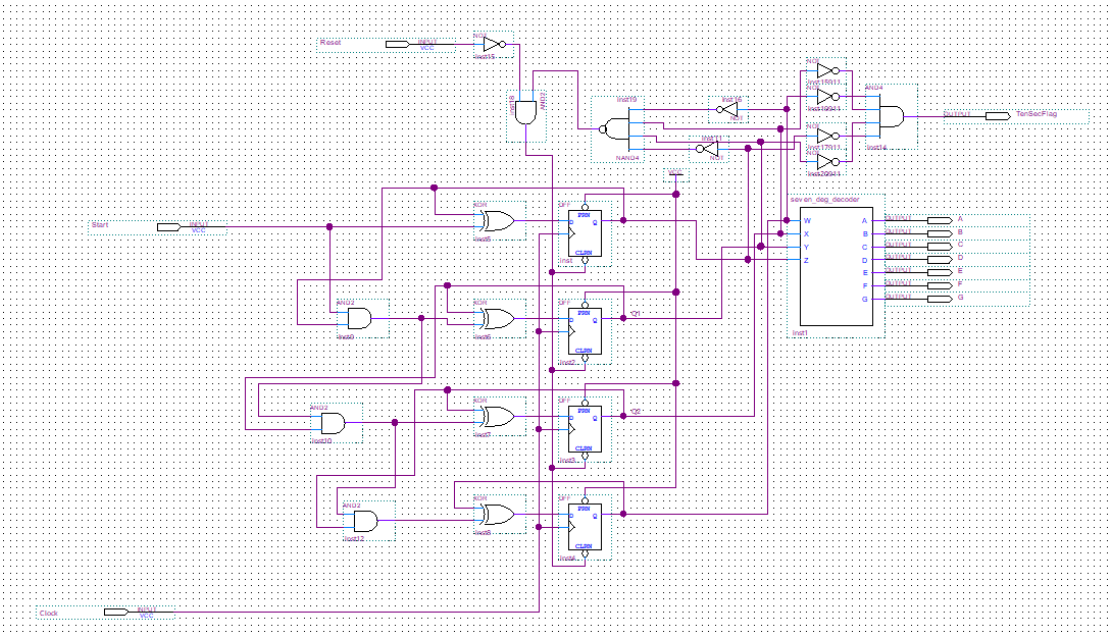


Fig 6 (above) : the FinProjT2 block, counts from 0-5 (used for ten second and ten minute counters)

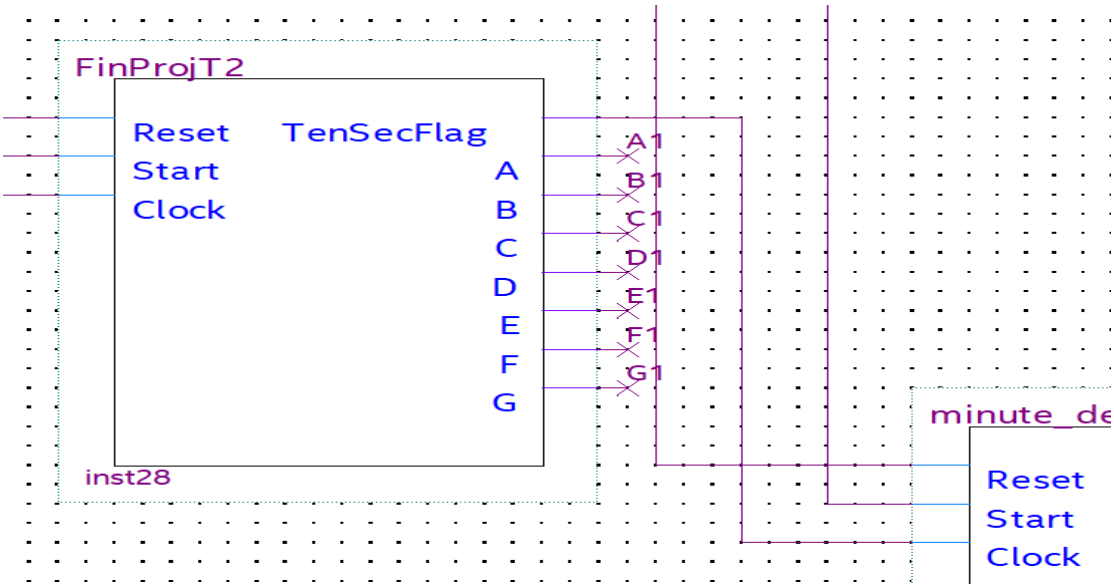



Fig 7 (above) : the FinProjT2 bsf, the FLAG output goes to the next counter's clock

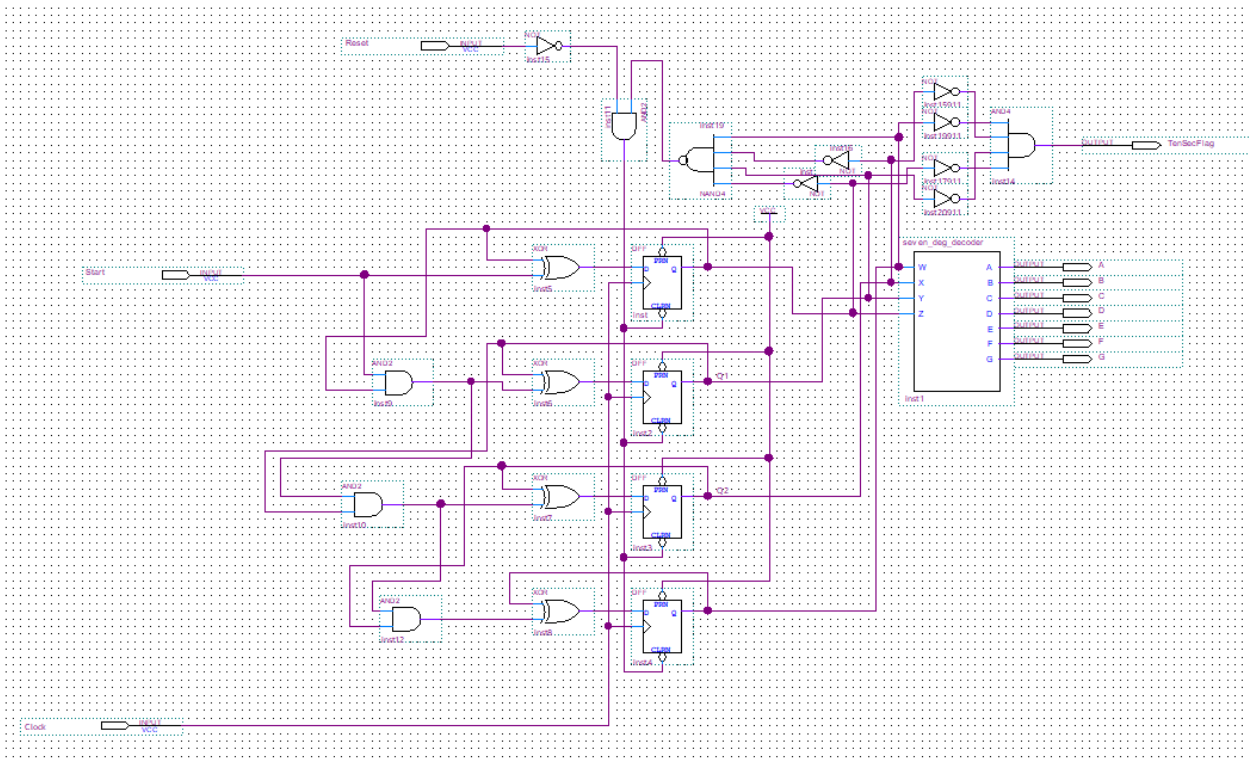


Fig 8 (above) : the minute_decoder block, counts from 0-9, same as FinProjStep1 block but just without the clock_generator

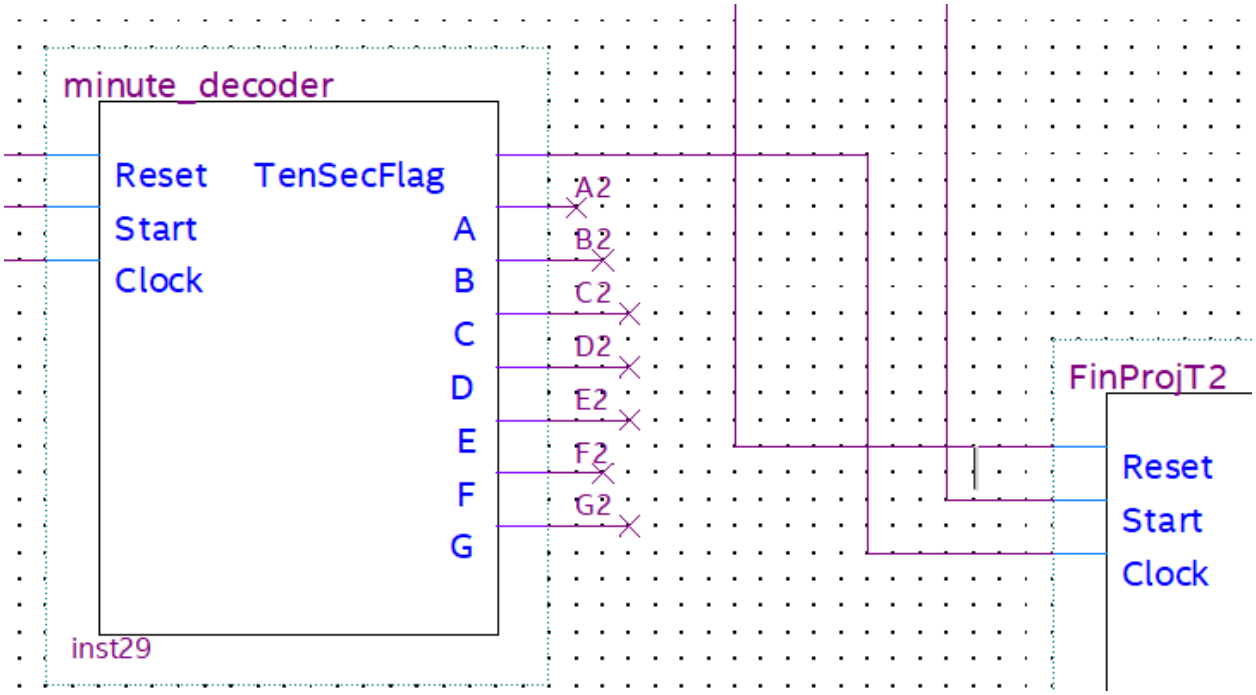



Fig 9 (above) : the minute_decoder symbol, the FLAG connects to the next counter's clock

The CLOCK button is the major key component of the stopwatch. The initial pin is assigned to the PIN_Y2 on the board that is a 50 MHz clock, it then goes into the FinProjStep1's block and meets its CLOCK_GENRATOR, that effectively slows the clock down to count at *almost* every one second. Now look at Fig 4, the clock enters the FinProjStep1 block, and is met by the clock_generator to be slowed down. This block then begins to count up.

Look at Fig 5, when this block reaches 10, it triggers the NAND gate to output a 0, which in-turn goes to the DFFs' CLRN and assigns it to zero, hence clearing it. Now in the same diagram, when the number resets to 0, we make sure to output a 1 to the FLAG (here : TenSecondFlag) which then acts as the clock for the next block and ensures that counts up.

***This idea of incrementing the next necessary counter's count by 1 when the previous counter has reset back to its initial state, which is told to us by the FLAG output of each counter, is the heart of the functionality of our stopwatch.***

Look at Fig 5 and 7, the FinProjStep1 and minute_decoder blocks, both of them are modulo 16 counter (given to us in Lab10 by Prof. Stoytchev). The main purpose of this block was to count from 0-9 and go back to 0, basically like a modulo 10 counter. Instead of making new counters we decided to clear the DFFs of the counter as and when the counter reached 10, essentially sending it back to 0. While doing that we also output a 1 (one) when we have reset the counts to zero. This '1' that we are outputting using the FLAG, acts as the clock for the next counter an signals the next counter to count up by one. The only difference between these is that the minute_decoder is triggered by the FinProjT2 block while the other is has its own clock and effectively dictates the working of the other counters

Look at Fig 6, the FinProjT2 block, this is a modulo 16 counter (given to us in Lab10 by Prof. Stoytchev). The main purpose of this block was to count from 0-5 and go back to 0, basically like a modulo 6 counter. Instead of making new counters we decided to clear the DFFs of the counter as and when the counter reached 6, essentially sending it back to 0. While doing that we also output a 1 (one) when we have reset the counts to zero. This '1' that we are outputting using the FLAG, acts as the clock for the next counter an signals the next counter to count up by one.

The RESET button : its input is given to a NOT gate first and is put in every block with an AND gate to the internal reset functionality of the block, that is when either RESET is 1 (inside block it becomes 0) or the counter is resetting to zero on its own, the DFFs are cleared.

**DECODER LOGIC:**

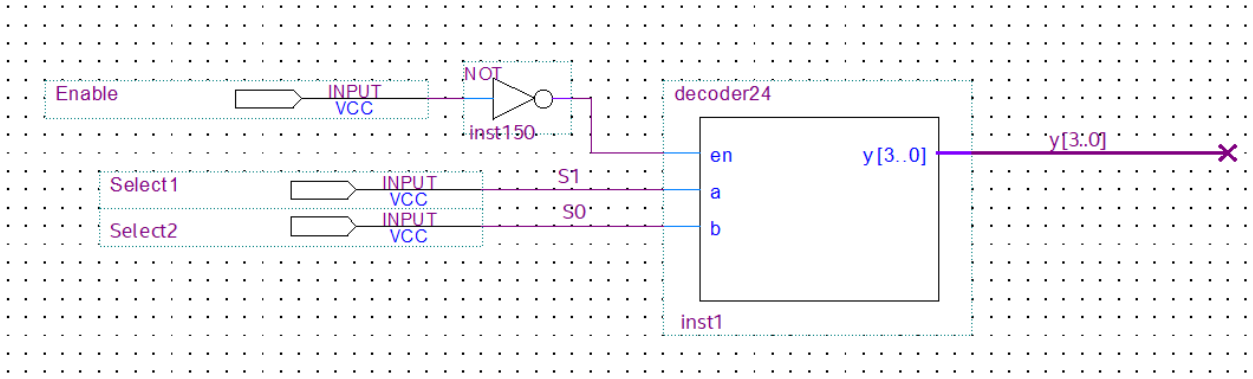



Fig 10 (above) : decoder symbol

```
module decoder24(en,a,b,y);

   input en,a,b;


   output reg [3:0]y;


   always @(en,a,b)
     begin

        if(en==1)
          begin
            if(a==0 & b==0) y=4'b0001;
            else if(a==0 & b==1) y=4'b0010;
            else if(a==1 & b==0) y=4'b0100;
            else if(a==1 & b==1) y=4'b1000;
          end
       else
          begin
          y = 4'b0000;
          end
     end
endmodule
```

Fig 11 (above) : decoder logic in verilog

The decoder we used here takes in two inputs of two slide switches and one enable input (mapped to a push-button, hence NOT gate). The two Select1 and Select2 inputs determine which lap we are going to store the current stopwatch time in and selects the appropriate set of registers to store them (00 is lap1 01 is lap2 10 is lap3 and 11 is lap4). So this decoder gives us a bus output y[3..0] and at a given time only one of y3, y2, y1, y0 will be one.

Using this idea, we used the lines y3, y2, y1, y0 as the select lines for our multiplexers that dictate if a set of registers load a new value (lap) or store the old one).

**REGISTERS/MEMORY ELEMENTS:**

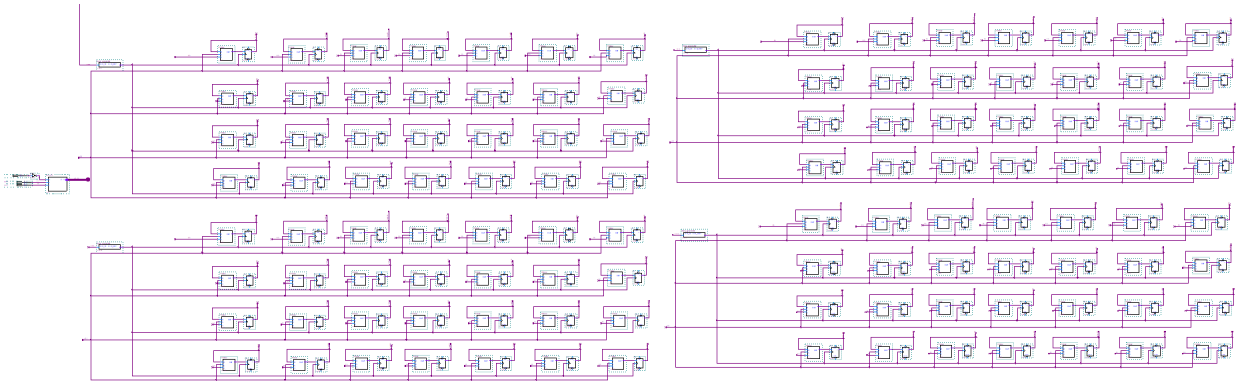


Fig 12 (above) : all the registers

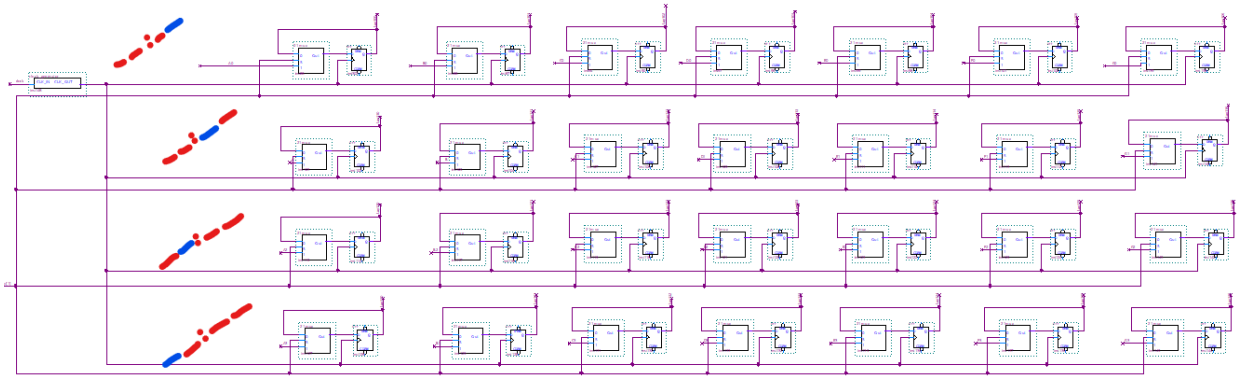


Fig 13 (above) : shows one of the four register systems that stores one lap (here : Lap2)

Look at Fig 11, it tells us which part of the time (the seconds, ten seconds, minutes, and ten minutes) the specific set of 7 registers stores. The mulitplexers in front of the DFFs take the decoder lines y3, y2, y1, y0 as their select lines, they hold on to the current value if 0, or the load a new value from the Counter Region of 1 and store that. The clock for all DFFs comes from the clock_generator (the block that slows down the inbuilt clock of the ALTERA board)

**OUTPUT LOGIC:**

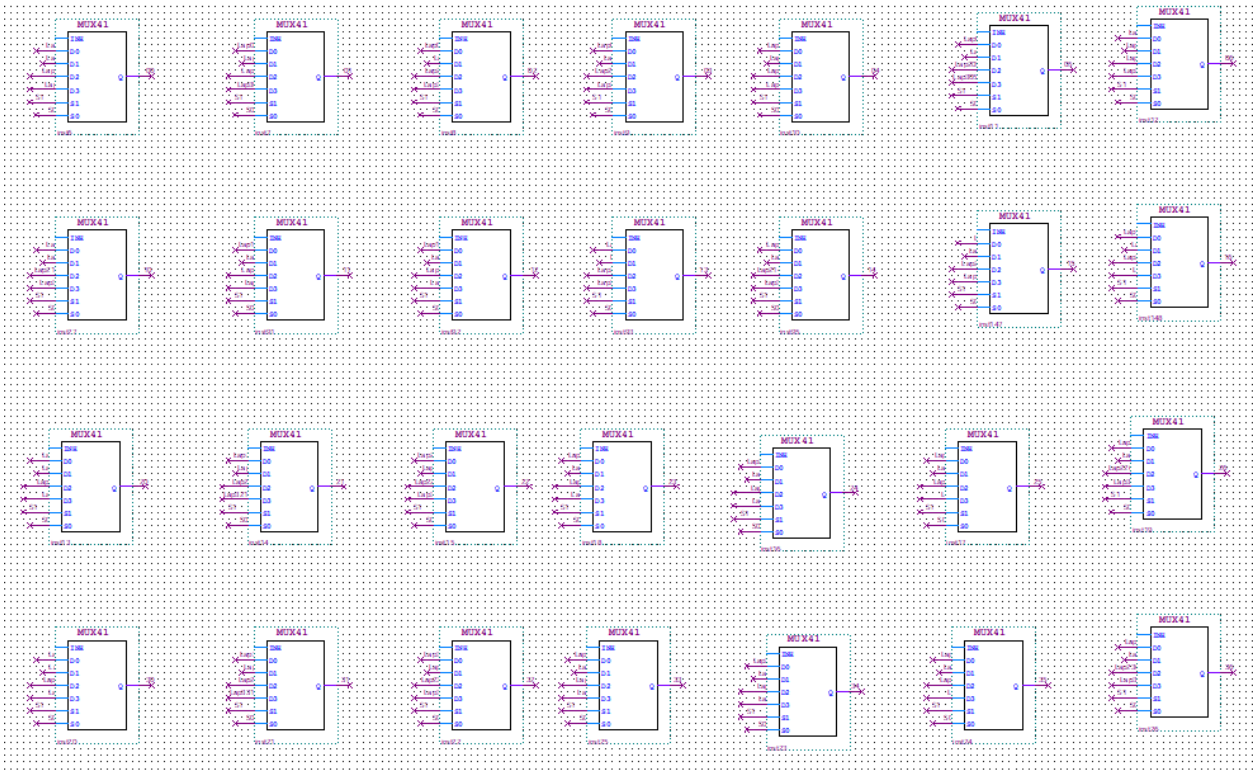


Fig 14 (above) : Output Mulitiplexers, they determine which lap is outputted

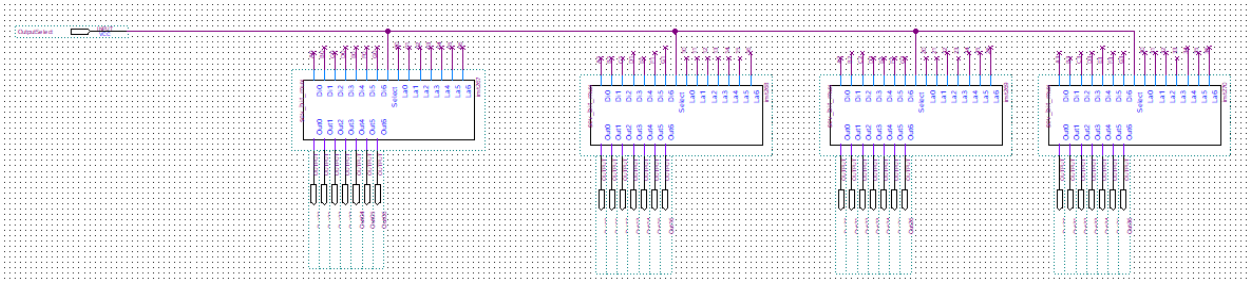


Fig 15 (above) : Four 14-bit 2to1 MUX

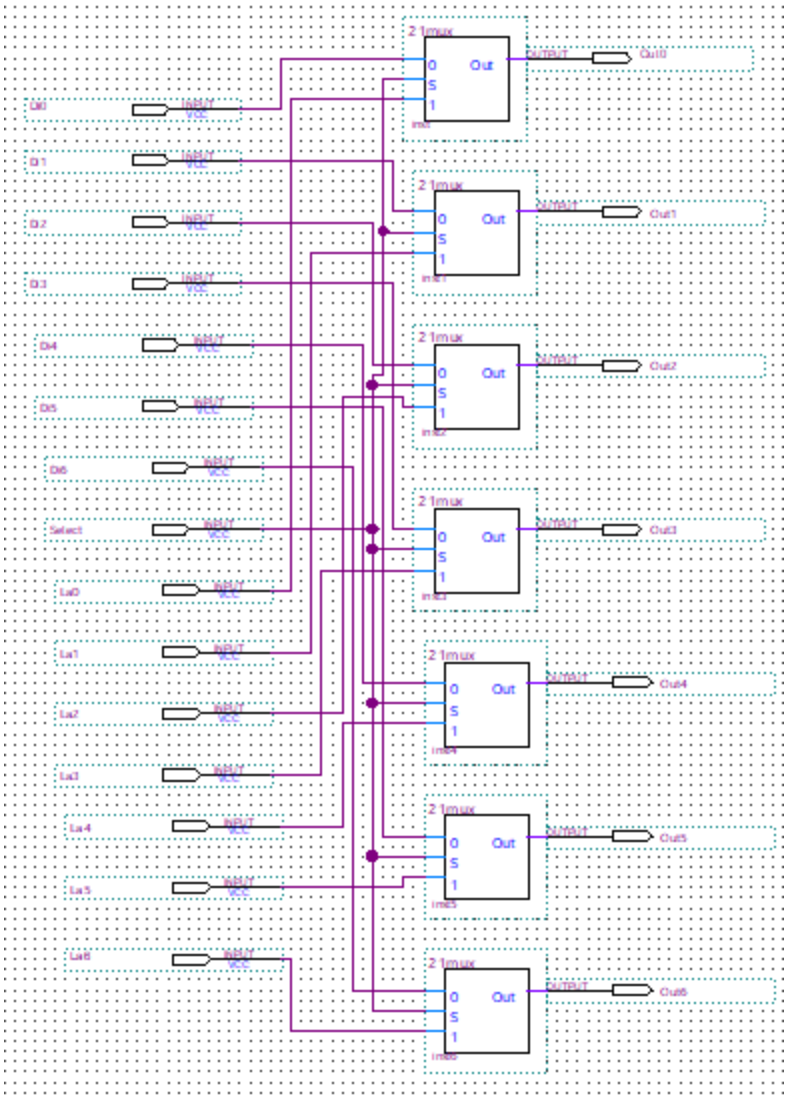



Fig 16 (above) : seven_bit_mux bdf, 7 21MUXes

The system above is a group of 4to1 MUXes (mulitplexers), their select lines are the same as the input lines of the decoder. They decide which of the four laps get displayed. Every row in the system above corresponds to the 7 segment display value for that particular place (seconds, ten seconds, minutes, ten minutes). The output of these MUXes then goes to another set of 14-bit MUXes (basically 7 2to1 MUXes).

Look at Fig 15, This is where the outputs of each row of the system of multiplexers come to, this is basically a set of 7 2to1 MUXes that govern what gets displayed on the board. If the Output Select Line is turned on, it shows the lapped time for the corresponding lap that we want to see, the stopwatch keeps running in the background in the mean time.

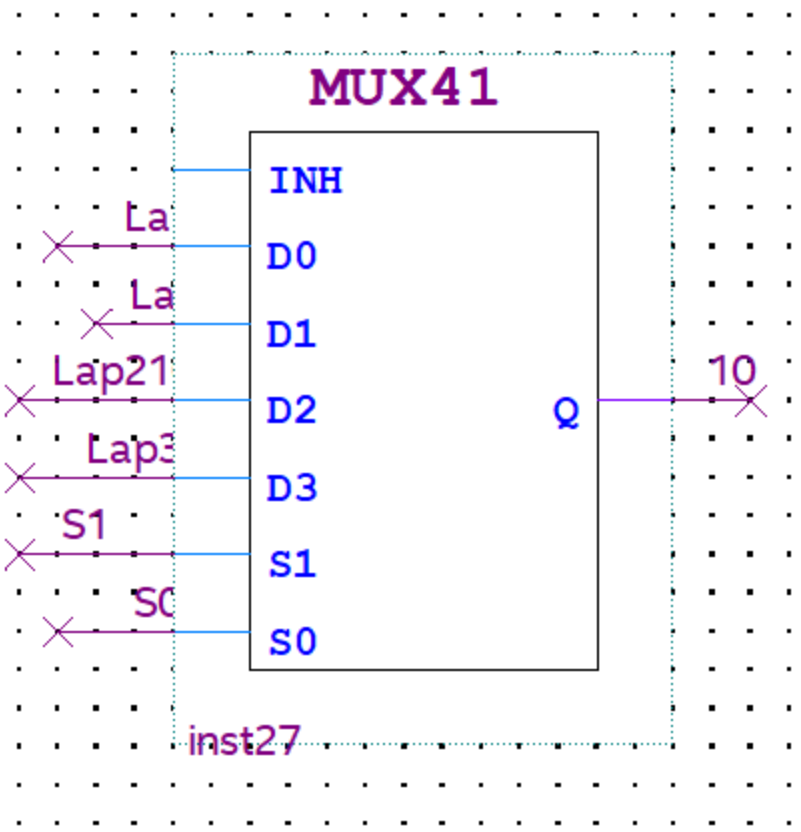



Fig 17 (above) : in-built MUX 41

```
module seven_deg_decoder(W, X, Y, Z, A, B, C, D, E, F, G);
    input Z, Y, X, W;
    output reg A, B, C, D, E, F, G;

    always @(W or X or Y or Z)
    begin
        case({W, X, Y, Z})
            //truth tabling
            4'b0000: {A, B, C, D, E, F, G} = 7'b0000001;
            4'b0001: {A, B, C, D, E, F, G} = 7'b1001111;
            4'b0010: {A, B, C, D, E, F, G} = 7'b0010010;
            4'b0011: {A, B, C, D, E, F, G} = 7'b0000110;
            4'b0100: {A, B, C, D, E, F, G} = 7'b1001100;
            4'b0101: {A, B, C, D, E, F, G} = 7'b0100100;
            4'b0110: {A, B, C, D, E, F, G} = 7'b0100000;
            4'b0111: {A, B, C, D, E, F, G} = 7'b0001111;
            4'b1000: {A, B, C, D, E, F, G} = 7'b0000000;
            4'b1001: {A, B, C, D, E, F, G} = 7'b0000100;
            4'b1010: {A, B, C, D, E, F, G} = 7'b0001000;
            4'b1011: {A, B, C, D, E, F, G} = 7'b1100000;
            4'b1100: {A, B, C, D, E, F, G} = 7'b0110001;
            4'b1101: {A, B, C, D, E, F, G} = 7'b1000010;
            4'b1110: {A, B, C, D, E, F, G} = 7'b0110000;
            4'b1111: {A, B, C, D, E, F, G} = 7'b0111000;
        endcase
    end

endmodule
```

Fig 18 (above) : seven_seg_decoder, verilog logic file

**CLOCK OF OUR STOPWATCH:**

The main clock we use for our stopwatch is named the clock_generator which we took from LAB_11.

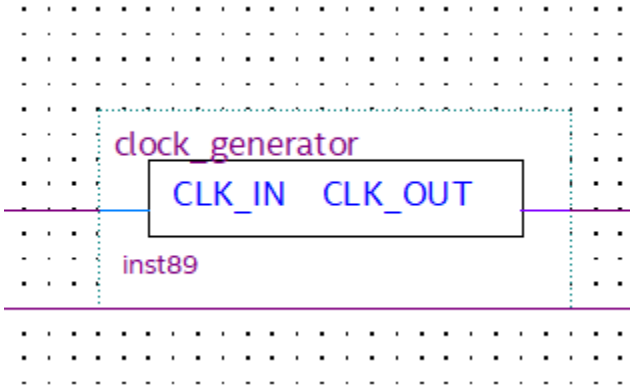



Fig (above): shows symbol for the clock_generator

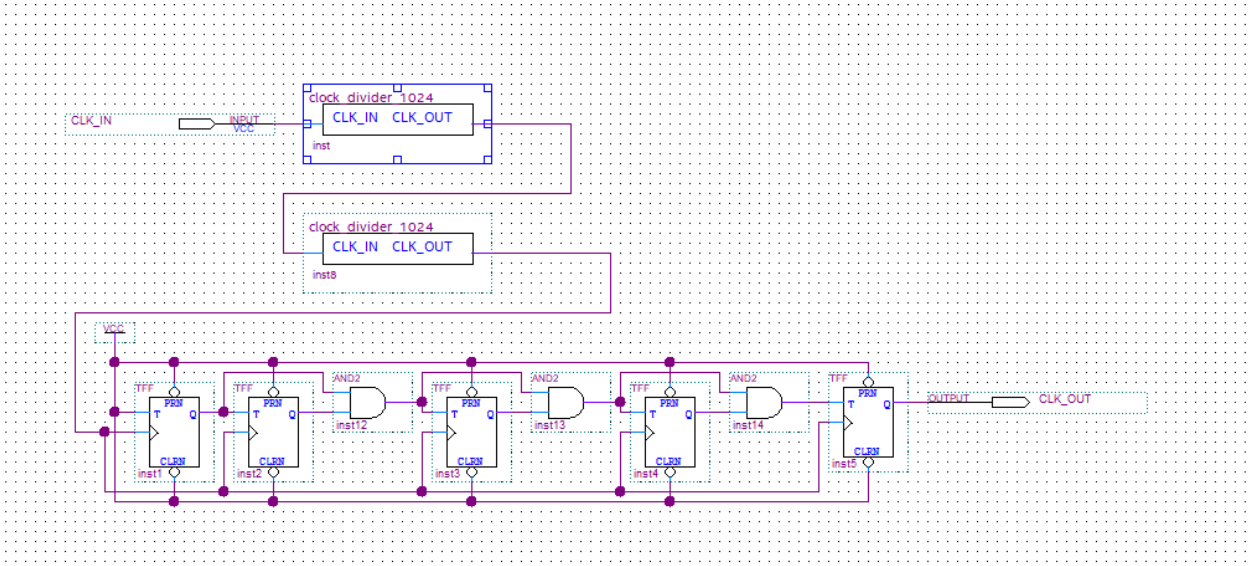



Fig (above) : shows the construction of the clock_generator

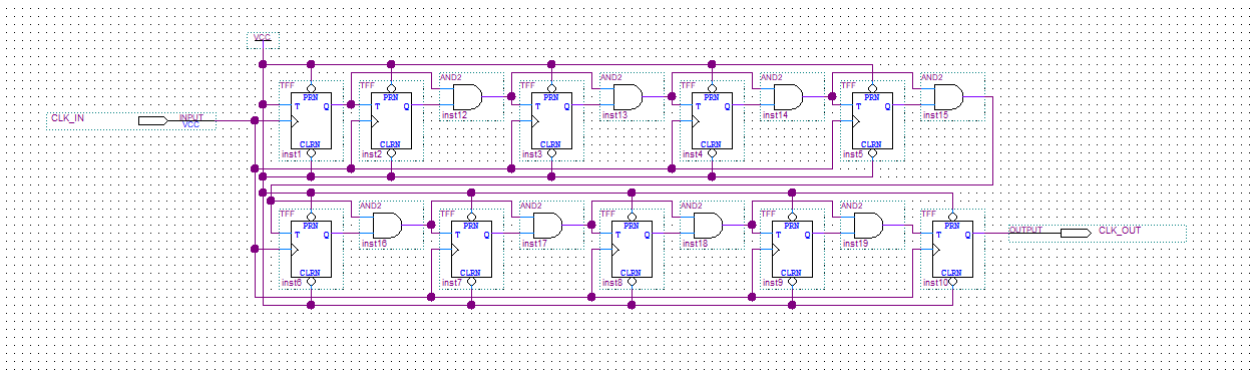



Fig (above) : shows the construction of the clock_divider_1024 used in clock_generator

The reason we used the clock generator as the clock for our stopwatch is because:

- The frequency of the clock on the ALTERA board is 50MHz which is too fast, thus we need to slow the frequency of the clock on the board.
- To do this we use a clock_generator which takes in the input of the clock of frequency 50MHz and gives out a clock of frequency nearly 1 Hz ( Time Period of our Clock = 0.7s)
- The clock_generator is made out of 2 clock_divider_1024 and 5 down counters. As we see in the images above a clock_divider_1024 block is just 10 down counters.
- Thus, all together our clock_generator has 10 + 10 + 5 down counters = 25 counters.
- Our main goal was to make the stopwatch have a time period as close to 1s. Each down counter helped slow the frequency by 2. Thus if we assume we need x counters.
- We thus use the expression 50000000 / 2^x = 1 or x = $log_2 50000000$ = 25.575 to get required counters. ( x= 25.575)
- We could either use 25 or 26 counters .
- We decided to use 25 counters making our stopwatch frequency approximately 1.49 Hz.

## FINITE STATE MACHINE - DIAGRAM , STATES and INPUTS

STATE A : STANDARD STOPWATCH RUNNING

STATE B : STOPWATCH GETS RESET

STATE D : STANDARD STOPWATCH STOPPED

STATE C1 : LAP1 TIME DISPLAYED

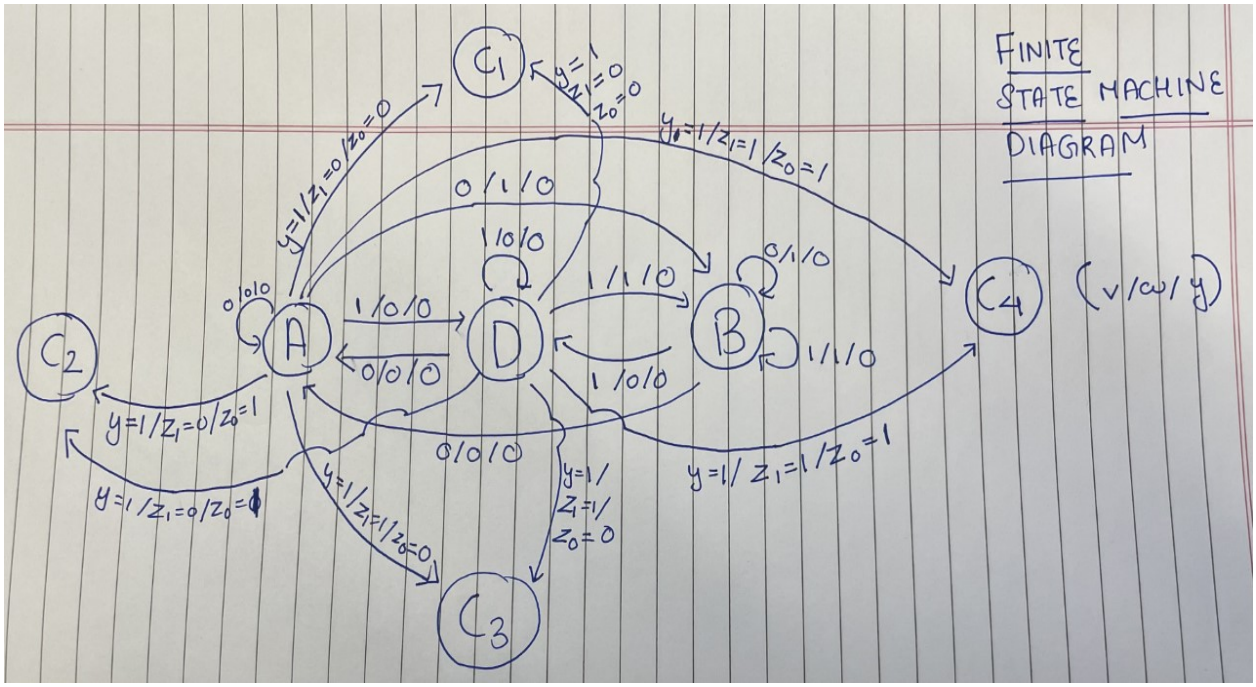
STATE C2 : LAP2 TIME DISPLAYED

STATE C3:  LAP3 TIME DISPLAYED

STATE C4:  LAP4 TIME DISPLAYED

INPUTS IN STATE DIAGRAM : v , w , y , z1 , z0

**FSM DIAGRAM PICTURE :**




**FSM Diagram**

## Sample Output of the Project:

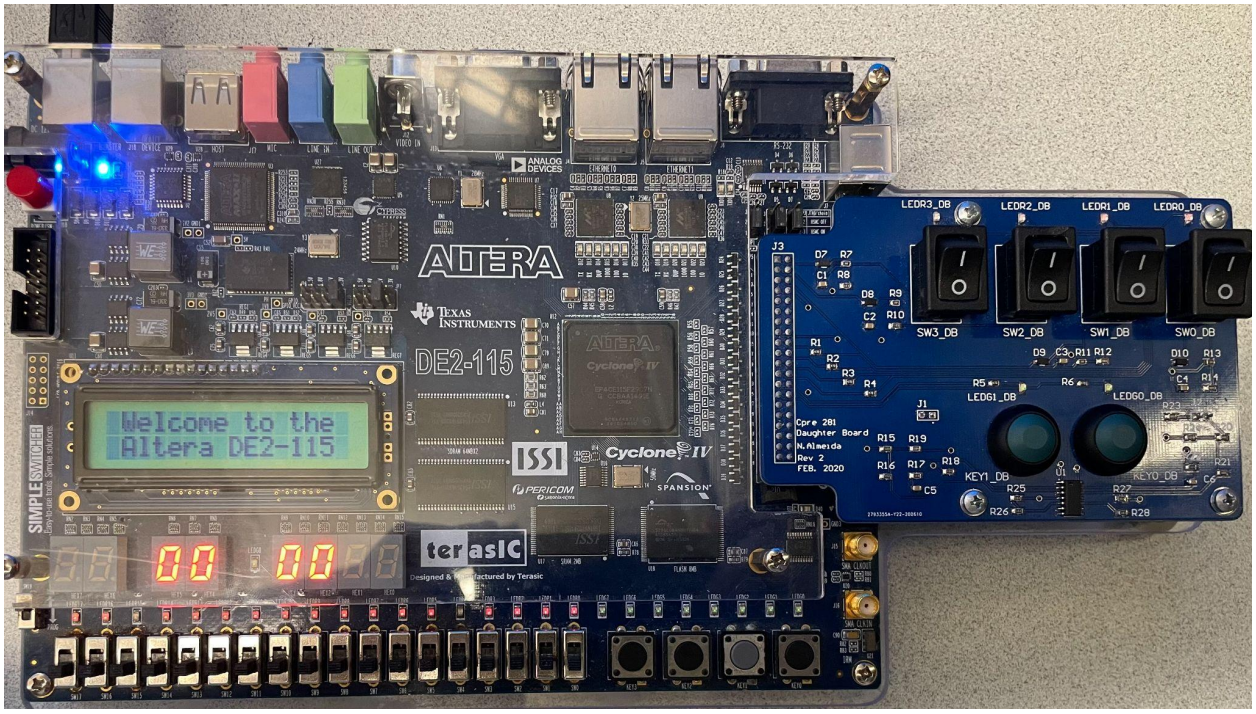
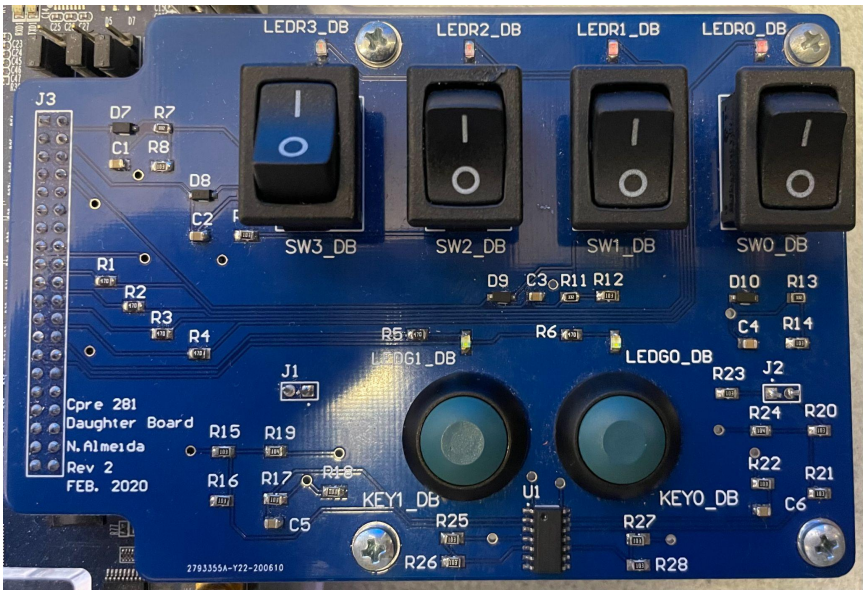



Fig 19 (above) : FPGA Board in its initial state(stopwatch has not started yet)

**RUNNING/ START :**



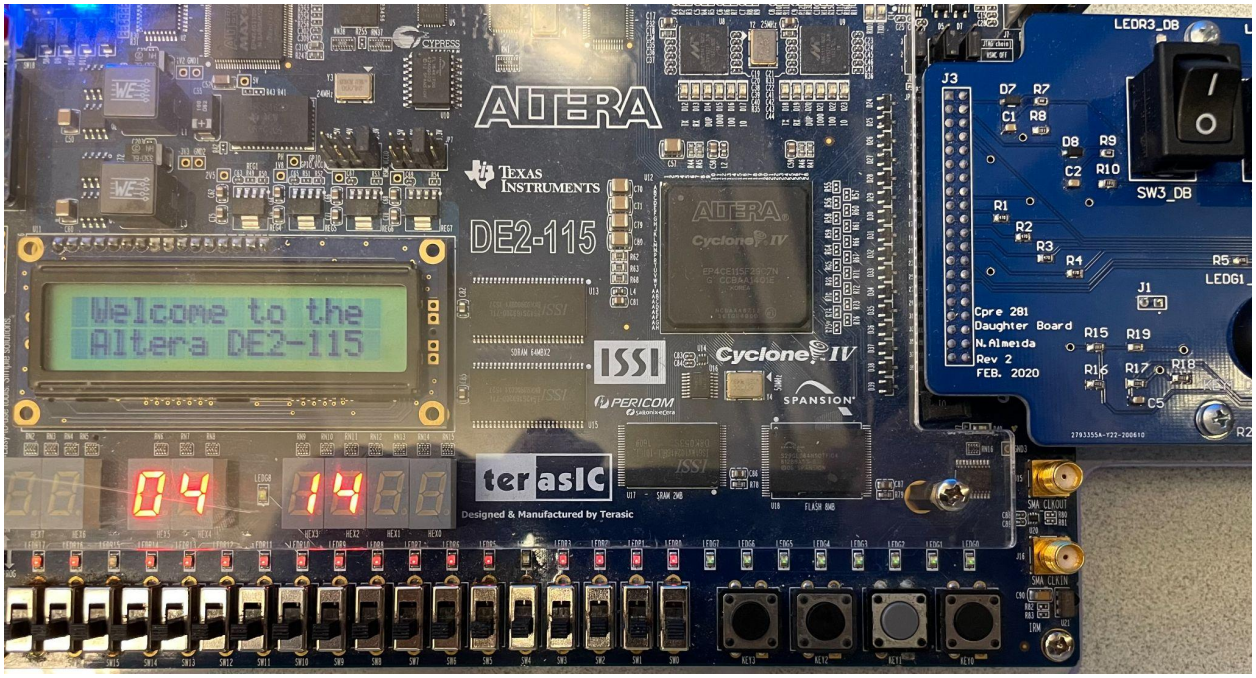



Fig 20 & 21 (above) : Represents the stopwatch running

In order to start the stopwatch, we need to start/turn on the SW3_DB switch on the daughter board of the FPGA. This switch acts as a start/stop or play/pause button for our stopwatch. The Output shown here shows that the stopwatch is counting 4 minutes and 14 seconds. Which proves its ability to display time in minutes with an upper limit of (59 minutes and 59 seconds). The running state is displayed using STATE A in the FSM diagram.

**STOP/ PAUSED STATE :**

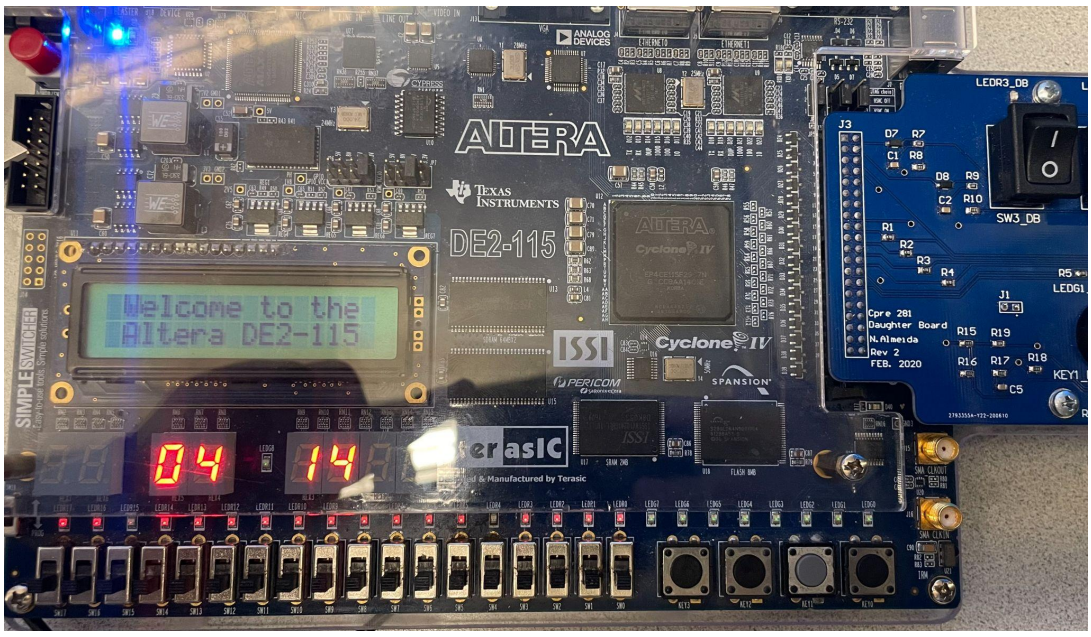



Fig 22 (above) : Shows the stopwatch in paused state

In order to stop/pause the stopwatch, we need to turn off the SW3_DB switch on the daughter board of the FPGA. The paused state is displayed using STATE D in the FSM diagram.

**RESET STATE :**

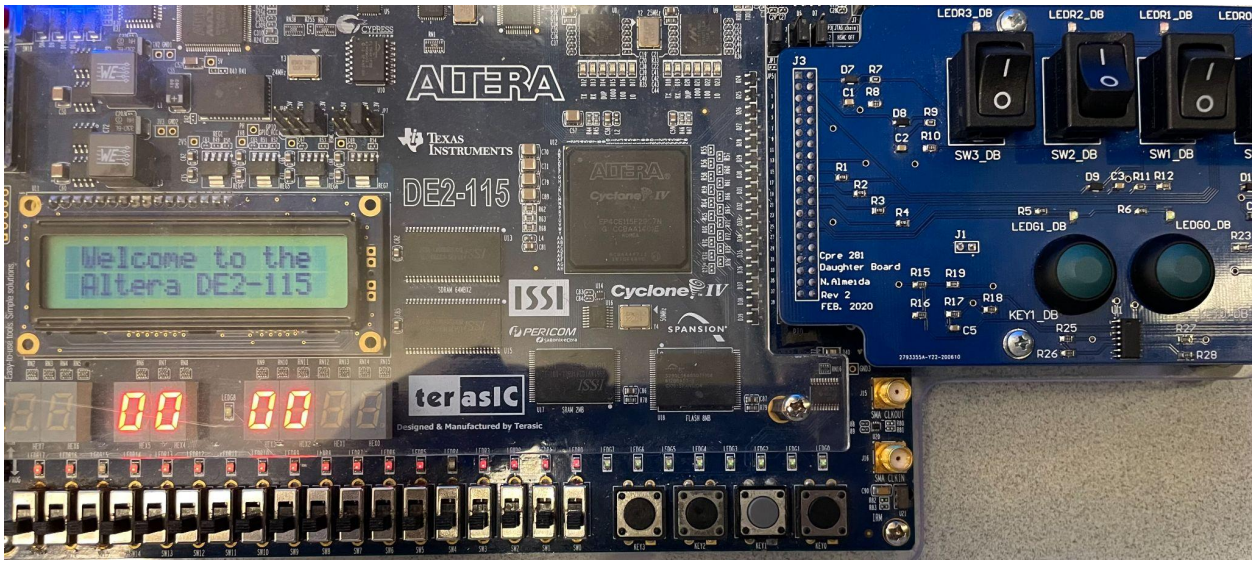



Fig 23 (above) :Shows the reset state of out machine where the stopwatch is rest to 00:00

The next state our clock can go to is the reset state where our minutes and hours on the stopwatch are set to 00:00 as shown in the picture above. To reach the following state we need to toggle/switch on the SW2_DB rocker switch on the FPGA board. Switching it on will immediately set the stopwatch to 00:00 and the stopwatch will stop counting until the SW2_DB button is switched off/toggled back to its initial state.This state is represented as state B in the FSM diagram.

**LAP STATE:**
**How to lap time using the FPGA Board in our stopwatch?**

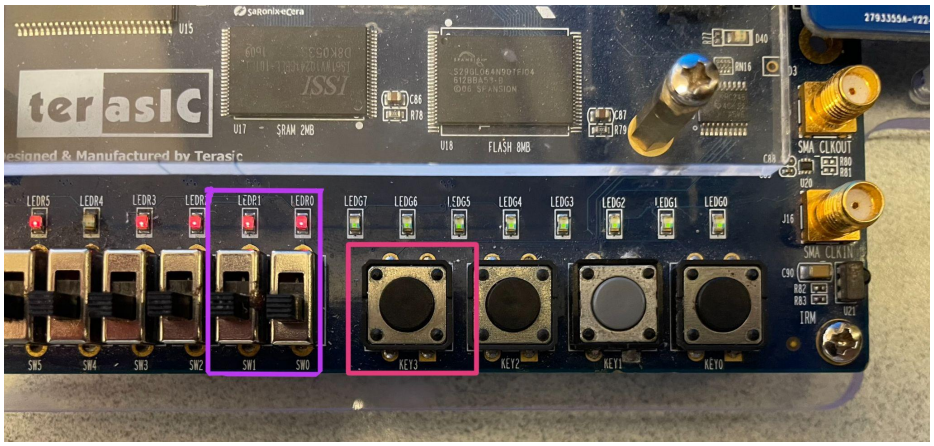



Fig 24 (above): Shows two slide switches SW1, SW0, and one push button Key3

- We have incorporated the feature of multiple laps on our stopwatch using a register file. To lap according to our need we make use of the two slide switches SW1 and SW0, and also the push button Key3 shown in the diagram above.
- We first need to choose the lap we want to store our time into. This is done using the switch SW1 and SW0.
- To store time into lap 1 we set SW1 to 0/ set it the off side and set SW2 to 0/set it also to the offside. ( SW1 = 0 and SW2 = 0). Once we do this we press the Key3 push button exactly at the time we want to lap and release it before it switches to the next second.
- Similarly to store time into lap2 we set SW1 to 0 , SW1 to 1 , and press the Key3 push button at the exact time we want to store in our lap 2.
- Similarly to store into lap3 we set SW1 to 1, SW0 to 0, and press the Key3 push button at the exact time we want to store in our lap 3.
- Finally to store lap4 we set SW4 to 1, SWO to 1 , and press the Key4 push button at the exact time we want to store in our lap 4.
- Note we can lap time when the stopwatch is running and can also lap time when the stopwatch is in the paused/stop state.

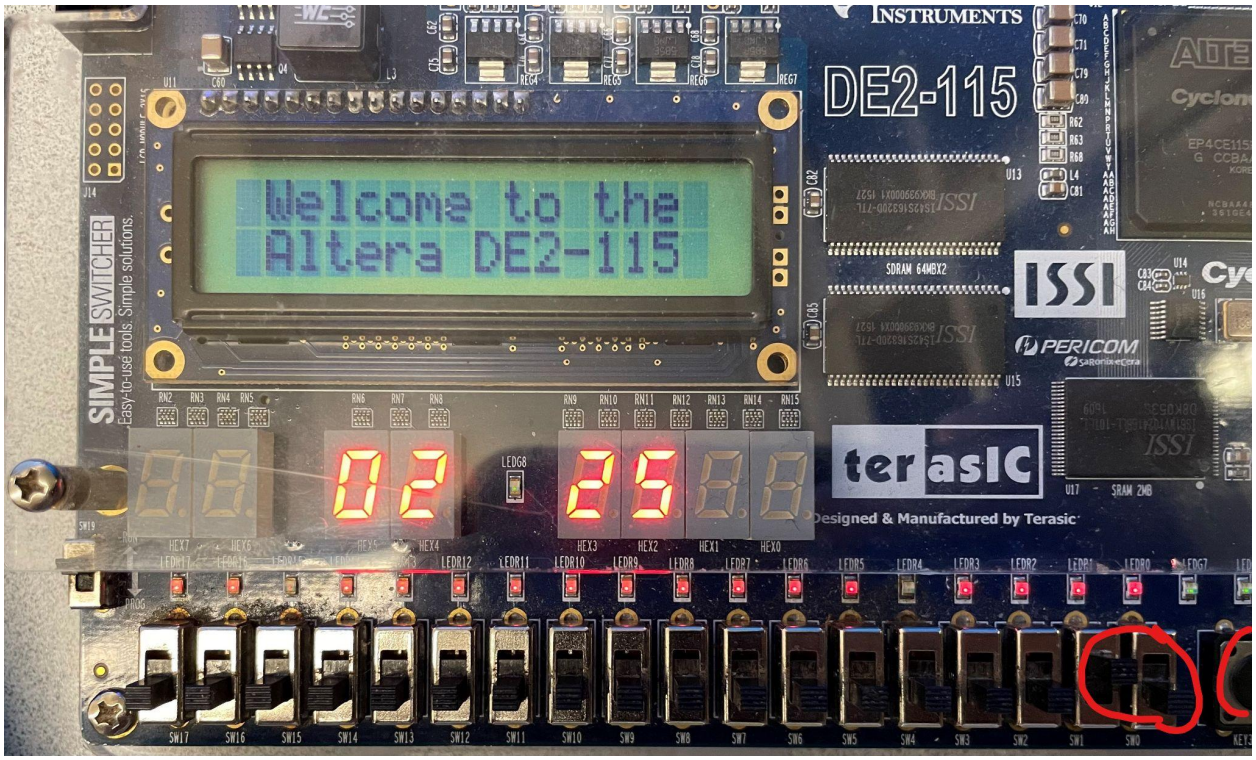



Fig 25 (above): We are storing the time 2:25 into lap 3 as SW1 =1 and SW0=0 in the figure above as we press Key3

| SW1 | SW0 | LAP STORED INTO |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 2 |
| 1 | 0 | 3 |
| 1 | 1 | 4 |

Table above shows which lap is used as we change SW1 and SW0 values

**How to display the different lap times on our FPGA board?**

- To display the lap time We have used the Rocker Switch :SW0-DB on the FPGA board.
- We can display the time stored by each lap one at a time using the same slide switches SW1 and SW0 which we used to store the lap time.
- This means the minute our SW0_DB (Rocker Switch) is switched on our display will stop showing the stopwatch counter and will show the lap time stored according to what SW0 and SW1 is set at. For example if we set SW0_DB to 1 , with SW1 =1 and SW0 =0 the FPGA board will display the lap3 time on the board.
- Note if we leave the SW3_DB at 1 as we set SW0_DB to 1 the stopwatch will keep counting even though the display will show the lap stored.

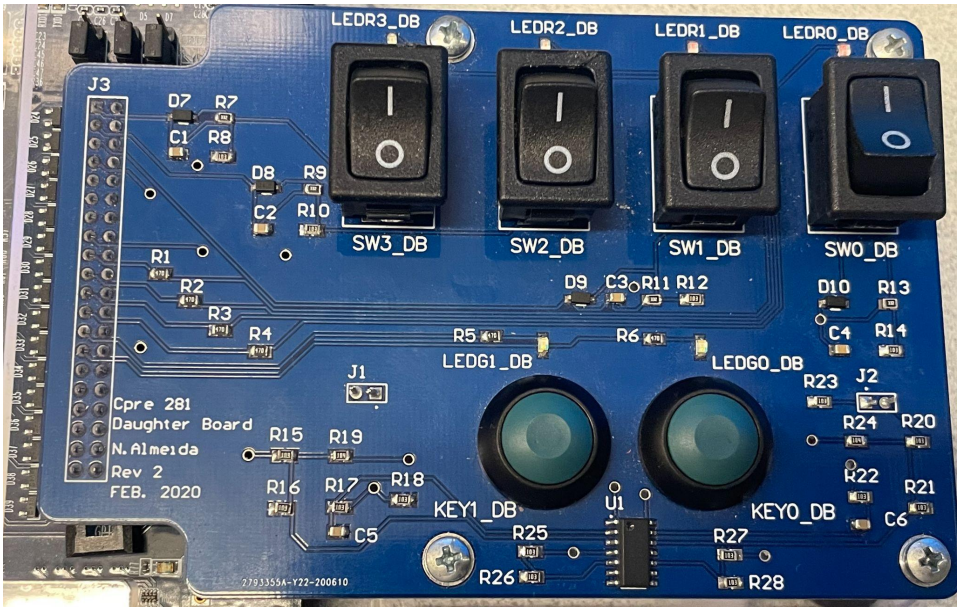



Fig 26 (above) : Shows us displaying a lap time as the stopwatch is stopped

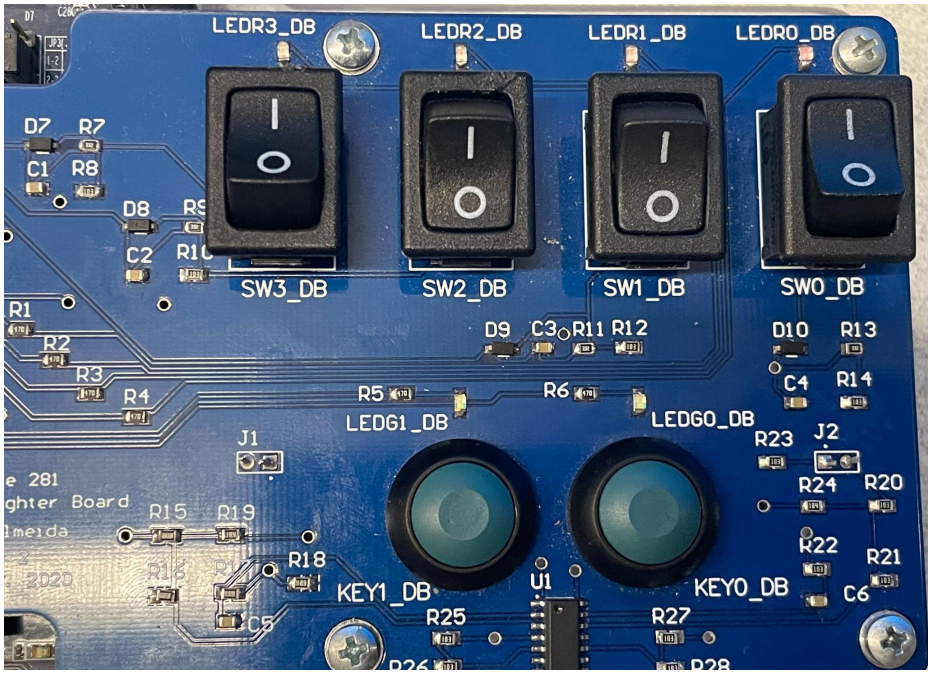



Fig 27 (above) : Shows us displaying a lap time as the stopwatch is running