DIGITAL DESIGN LAB

2EC305

Special Assignment

-: Topic: -

STOP-WATCH & TIMER

Made By: -

Harsh Modi (21BEC068)

Prince Kanjiya (21BEC055)

Submitted To: - Prof.Manish Patel

Contents: -

Note:- codes are also added in additional files.

ABSTRACT

- *This stopwatch & Timer project is a software and hardware co-design. The time will be shown on the FPGA board and on the 7-segment display. The system will be modelled in Verilog HDL in Quartus Altera software implemented on the ALTERA CYCLONE II FPGA board.*

• ***INTRODUCTION***

- *The digital stopwatch & Timer designed is a time-keeping device that is meant to measure the time elapsed from the start to end of an event. The stopwatch & Timer has several different functions including START (which represents both begin and pause), RESET,4 seven segment displays on the FPGA board.*

These are used in many areas like Traffic signal, Temperature Indicator, Alarm, Indicator.

The Components used in the Project:

1. ***UP COUNTER & DOWN COUNTER***
2. ***4 X 1 MUX***
3. ***7 SEGMENT DISPLAY***

WORKING

The stop watch designed will be able to count till 10 min i.e. (9:59:9). It will be a 4digit stop watch which be displayed through seven segment displays on FPGA board counting from 0:00:0 to 9:59:9 seconds. The right most part will increment every 0.1 second, when it reaches to 9 it will increment the middle 2 digits of the stop watch which represents the seconds counter if stopwatch. When it reaches to 59 it will increment the minute display

The stopwatch will be displayed in the format as follow: M: SS:D

• ***CREATING A DELAY***

To create a stop watch more accurate we need a delay of 0.1 s for every cycle of the clock. Furthermore, here we use Altera CYCLONE II board which has the clock frequency of 50MHz.So, to create a 0.1 s delay, we do

*50MHz * 0.1 sec = 5000000*

So, the clock cycle is repeated 5M times in 0.1 second. The next step would be to calculate the size of the register which will hold the count. The calculation of this is give further below and an unknown parameter (z) is defined.

2exp(z)=5000000

log(2exp(z)) =log (5000000)

z log (2) =log (5000000)

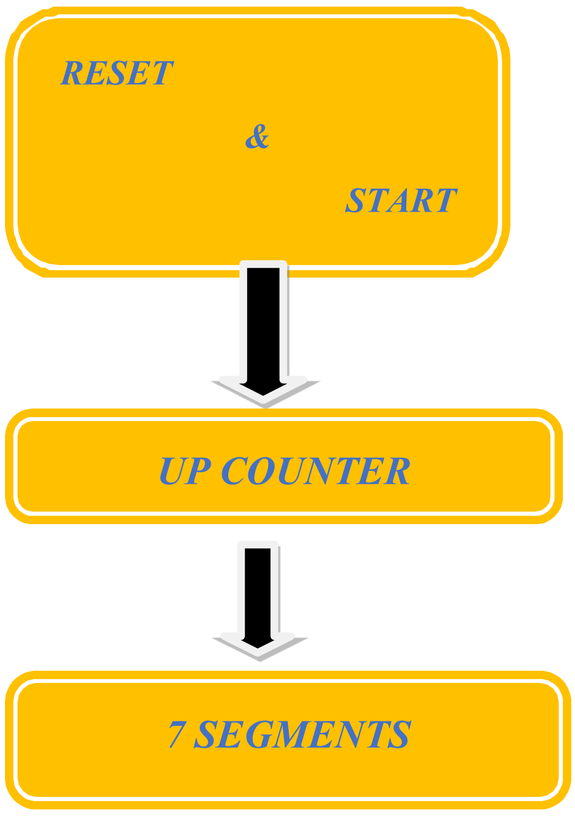
z=log(5000000)/log(2)

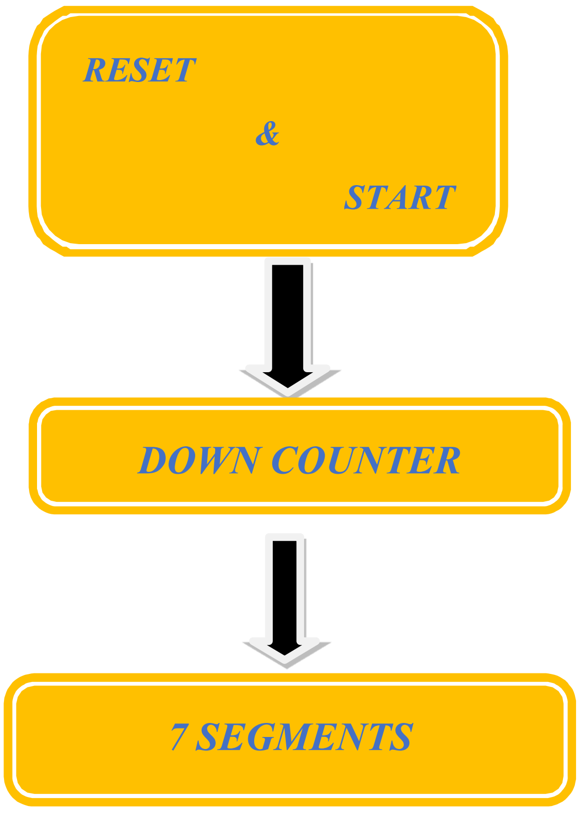
z=23(approx.)

So, the value of the register is nearly 23 bits.

• ***BLOCK DIAGRAM***

STOPWATCH: - ***TIMER:-***





- ***CODE***

***STOP-WATCH*:**

```
module STOP(
    input clock,
    input reset,
    input start,
    output
a0,b0,c0,d0,e0,f0,g0,a1,b1,c1,d1,e1,f1,g1,a2,b2,c2,d2,e2,f2,g2,a3,b3,c3,d3,e3,f3,g3);


reg [3:0]reg_d0, reg_d1, reg_d2, reg_d3; //registers that will hold the individual counts
reg [23:0] ticker; //23 bits needed to count up to 50M bits
wire click;
always @ (posedge clock or posedge reset)
begin
 if(reset)
  ticker <= 0;
 else if(ticker == 5000000) //if it reaches the desired max value reset it
  ticker <= 0;
 else if(start) //only start if the input is set high
  ticker <= ticker + 1;
end

assign click = ((ticker == 5000000)?1'b1:1'b0); //click to be assigned high every 0.1 second

always @ (posedge clock or posedge reset)
begin
 if (reset)
  begin
   reg_d0 <= 0;
   reg_d1 <= 0;
   reg_d2 <= 0;
   reg_d3 <= 0;
  end

 else if (click) //increment at every click
  begin
   if(reg_d0 == 9) //xxx9 - the 0.1 second digit
   begin  //if_1
    reg_d0 <= 0;

    if (reg_d1 == 9) //xx99
    begin  // if_2
     reg_d1 <= 0;
     if (reg_d2 == 5) //x599 - the two digit seconds digits
     begin //if_3
      reg_d2 <= 0;
      if(reg_d3 == 9) //9599 - The minute digit
       reg_d3 <= 0;
      else
       reg_d3 <= reg_d3 + 1;
     end
     else //else_3
      reg_d2 <= reg_d2 + 1;
    end

    else //else_2
     reg_d1 <= reg_d1 + 1;
   end

   else //else_1
```

```
    reg_d0 <= reg_d0 + 1;
  end
 end

 //7 segment encoding
//      0
//     ---
//  5 |   | 1
//     ---  <- 6
//  4 |   | 2
//     ---
//      3
 // through data flow modelling
 //d0

 assign a0 = ~((reg_d0[3]) | (reg_d0[1]) | (reg_d0[2] & reg_d0[0]) | (~reg_d0[2] &
~reg_d0[0]) );

assign b0 = ~((~reg_d0[2]) | (reg_d0[1] & reg_d0[0]) | (~reg_d0[1] & ~reg_d0[0]))  ;

assign c0 = ~((reg_d0[2]) | (~reg_d0[1]) | (reg_d0[0])) ;

assign d0 = ~((reg_d0[3]) | (~reg_d0[2] & ~reg_d0[0]) | (~reg_d0[2] & reg_d0[1]) |
(reg_d0[2] & ~reg_d0[1] & reg_d0[0]) | (reg_d0[1] & ~reg_d0[0])) ;

assign e0 = ~((~reg_d0[2] & ~reg_d0[0]) | (reg_d0[1] & ~reg_d0[0])) ;

assign f0 = ~((reg_d0[3]) | (reg_d0[2] & ~reg_d0[1]) | (reg_d0[2] & ~reg_d0[0]) |
(~reg_d0[1] & ~reg_d0[0])) ;

assign g0 =  ~((reg_d0[3]) | (~reg_d0[2] & reg_d0[1]) | (reg_d0[2] & ~reg_d0[1]) |
(reg_d0[2] & ~reg_d0[0])) ;
```

```
// d1

assign a1 = ~((reg_d1[3]) | (reg_d1[1]) | (reg_d1[2] & reg_d1[0]) | (~reg_d1[2] &
~reg_d1[0])) ;

assign b1 = ~((~reg_d1[2]) | (reg_d1[1] & reg_d1[0]) | (~reg_d1[1] & ~reg_d1[0]))  ;

assign c1 = ~((reg_d1[2]) | (~reg_d1[1]) | (reg_d1[0])) ;

assign d1 = ~((reg_d1[3]) | (~reg_d1[2] & ~reg_d1[0]) | (~reg_d1[2] & reg_d1[1]) |
(reg_d1[2] & ~reg_d1[1] & reg_d1[0]) | (reg_d1[1] & ~reg_d1[0])) ;

assign e1 = ~((~reg_d1[2] & ~reg_d1[0]) | (reg_d1[1] & ~reg_d1[0])) ;

assign f1 = ~((reg_d1[3]) | (reg_d1[2] & ~reg_d1[1]) | (reg_d1[2] & ~reg_d1[0]) |
(~reg_d1[1] & ~reg_d1[0])) ;

assign g1 =  ~((reg_d1[3]) | (~reg_d1[2] & reg_d1[1]) | (reg_d1[2] & ~reg_d1[1]) |
(reg_d1[2] & ~reg_d1[0])) ;

// d2

assign a2 = ~((reg_d2[3]) | (reg_d2[1]) | (reg_d2[2] & reg_d2[0]) | (~reg_d2[2] &
~reg_d2[0])) ;

assign b2 = ~((~reg_d2[2]) | (reg_d2[1] & reg_d2[0]) | (~reg_d2[1] & ~reg_d2[0]))  ;

assign c2 = ~((reg_d2[2]) | (~reg_d2[1]) | (reg_d2[0])) ;

assign d2 = ~((reg_d2[3]) | (~reg_d2[2] & ~reg_d2[0]) | (~reg_d2[2] & reg_d2[1]) |
(reg_d2[2] & ~reg_d2[1] & reg_d2[0]) | (reg_d2[1] & ~reg_d2[0])) ;
```

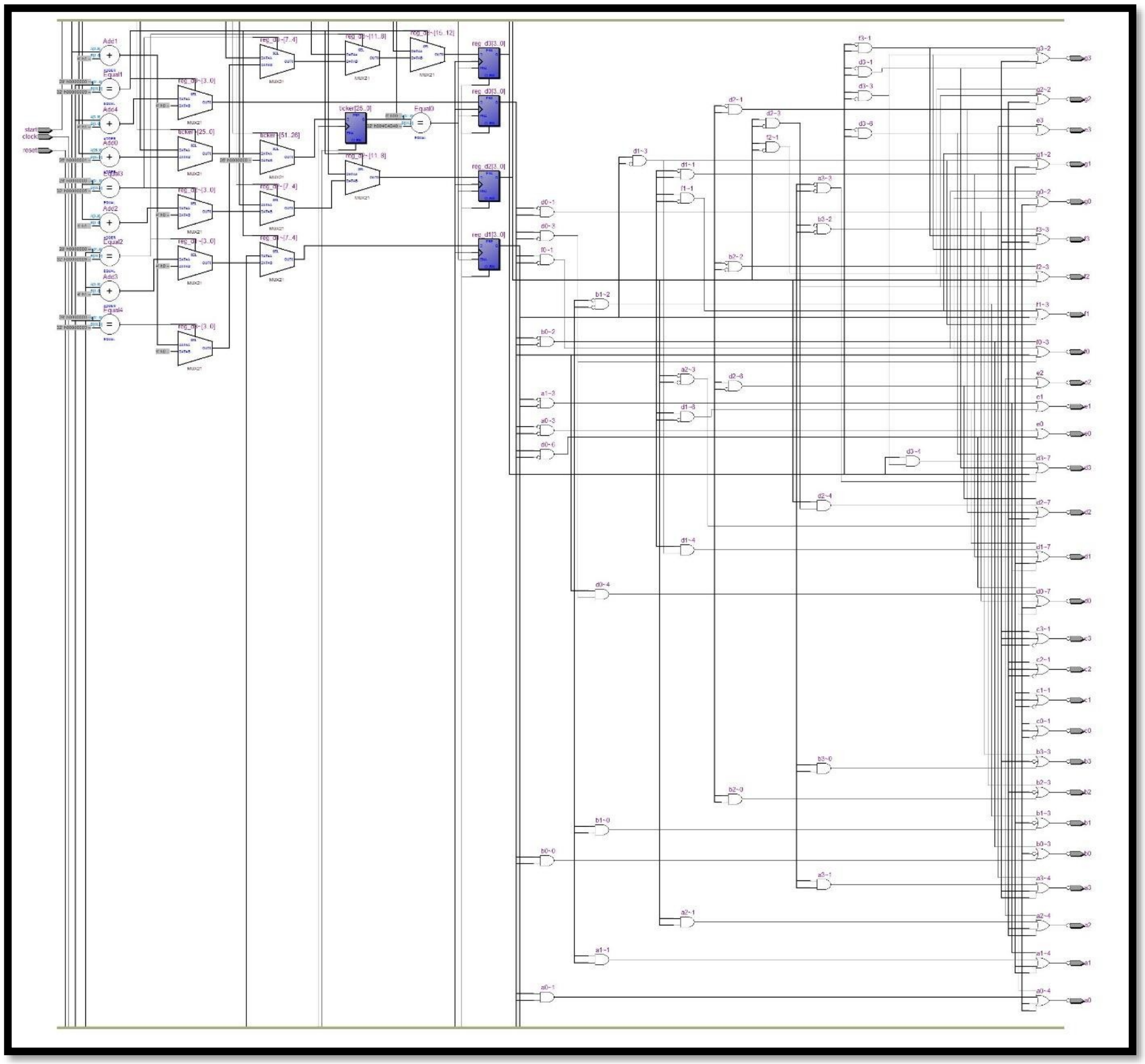
```
assign e2 = ~((~reg_d2[2] & ~reg_d2[0]) | (reg_d2[1] & ~reg_d2[0])) ;

assign f2 = ~((reg_d2[3]) | (reg_d2[2] & ~reg_d2[1]) | (reg_d2[2] & ~reg_d2[0]) |
(~reg_d2[1] & ~reg_d2[0])) ;

assign g2 =  ~((reg_d2[3]) | (~reg_d2[2] & reg_d2[1]) | (reg_d2[2] & ~reg_d2[1]) |
(reg_d2[2] & ~reg_d2[0])) ;

// d3

assign a3 = ~((reg_d3[3]) | (reg_d3[1]) | (reg_d3[2] & reg_d3[0]) | (~reg_d3[2] &
~reg_d3[0])) ;

assign b3 = ~((~reg_d3[2]) | (reg_d3[1] & reg_d3[0]) | (~reg_d3[1] & ~reg_d3[0]))  ;

assign c3 = ~((reg_d3[2]) | (~reg_d3[1]) | (reg_d3[0])) ;

assign d3 = ~((reg_d3[3]) | (~reg_d3[2] & ~reg_d3[0]) | (~reg_d3[2] & reg_d3[1]) |
(reg_d3[2] & ~reg_d3[1] & reg_d3[0]) | (reg_d3[1] & ~reg_d3[0])) ;

assign e3 = ~((~reg_d3[2] & ~reg_d3[0]) | (reg_d3[1] & ~reg_d3[0])) ;

assign f3 = ~((reg_d3[3]) | (reg_d3[2] & ~reg_d3[1]) | (reg_d3[2] & ~reg_d3[0]) |
(~reg_d3[1] & ~reg_d3[0])) ;

assign g3 = ~ ((reg_d3[3]) | (~reg_d3[2] & reg_d3[1]) | (reg_d3[2] & ~reg_d3[1]) |
(reg_d3[2] & ~reg_d3[0])) ;

endmodule
```

TIMER: -

```
module TIMER(
    input clock,
    input reset,
    input start,
    output
a0,b0,c0,d0,e0,f0,g0,a1,b1,c1,d1,e1,f1,g1,a2,b2,c2,d2,e2,f2,g2,a3,b3,c3,d3,e3,f3,g3);


reg [3:0] reg_d0, reg_d1, reg_d2, reg_d3; //registers that will hold the individual counts
//registers that will hold the individual counts
reg [23:0] ticker; //23 bits needed to count up to 50M bits
wire click;
always @ (posedge clock or posedge reset)
begin
 if(reset)


  ticker <= 0;


 else if(ticker == 50000000) //if it reaches the desired max value reset it
  ticker <= 0;
 else if(start) //only start if the input is set high
  ticker <= ticker - 1;
end
assign click = ((ticker == 50000000)?1'b1:1'b0); //click to be assigned high every 0.1 second

always @ (posedge clock or posedge reset)
begin
 if (reset)
  begin
   reg_d0 <= 9;
   reg_d1 <= 9;
   reg_d2 <= 9;
   reg_d3 <= 9;
  end

 else if (click) //decrement at every click
  begin
   if(reg_d0 == 0) //xxx9 - the 0.1 second digit
   begin  //if_1
    reg_d0 <= 9;

    if (reg_d1 == 0) //xx99
    begin  // if_2
     reg_d1 <= 9;
     if (reg_d2 == 0) //x599 - the two digit seconds digits
     begin //if_3
      reg_d2 <= 5;
      if(reg_d3 == 0) //9599 - The minute digit
       reg_d3 <= 9;
      else
       reg_d3 <= reg_d3 - 1;
     end
     else //else_3
      reg_d2 <= reg_d2 - 1;
    end

    else //else_2
     reg_d1 <= reg_d1 - 1;
   end

   else //else_1
    reg_d0 <= reg_d0 - 1;
  end
 end

 //7 segment encoding
 //      0
 //     ---
 //  5 |  | 1
 //     ---  <- 6
 //  4 |  | 2
 //     ---
 //      3

 // through data flow modelling

 //d0

 assign a0 = ~((reg_d0[3]) | (reg_d0[1]) | (reg_d0[2] & reg_d0[0]) | (~reg_d0[2] &
 ~reg_d0[0]) );

 assign b0 = ~((~reg_d0[2]) | (reg_d0[1] & reg_d0[0]) | (~reg_d0[1] & ~reg_d0[0]))  ;

 assign c0 = ~((reg_d0[2]) | (~reg_d0[1]) | (reg_d0[0])) ;

 assign d0 = ~((reg_d0[3]) | (~reg_d0[2] & ~reg_d0[0]) | (~reg_d0[2] & reg_d0[1]) |
 (reg_d0[2] & ~reg_d0[1] & reg_d0[0]) | (reg_d0[1] & ~reg_d0[0])) ;

 assign e0 = ~((~reg_d0[2] & ~reg_d0[0]) | (reg_d0[1] & ~reg_d0[0])) ;

 assign f0 = ~((reg_d0[3]) | (reg_d0[2] & ~reg_d0[1]) | (reg_d0[2] & ~reg_d0[0]) |
 (~reg_d0[1] & ~reg_d0[0])) ;
```

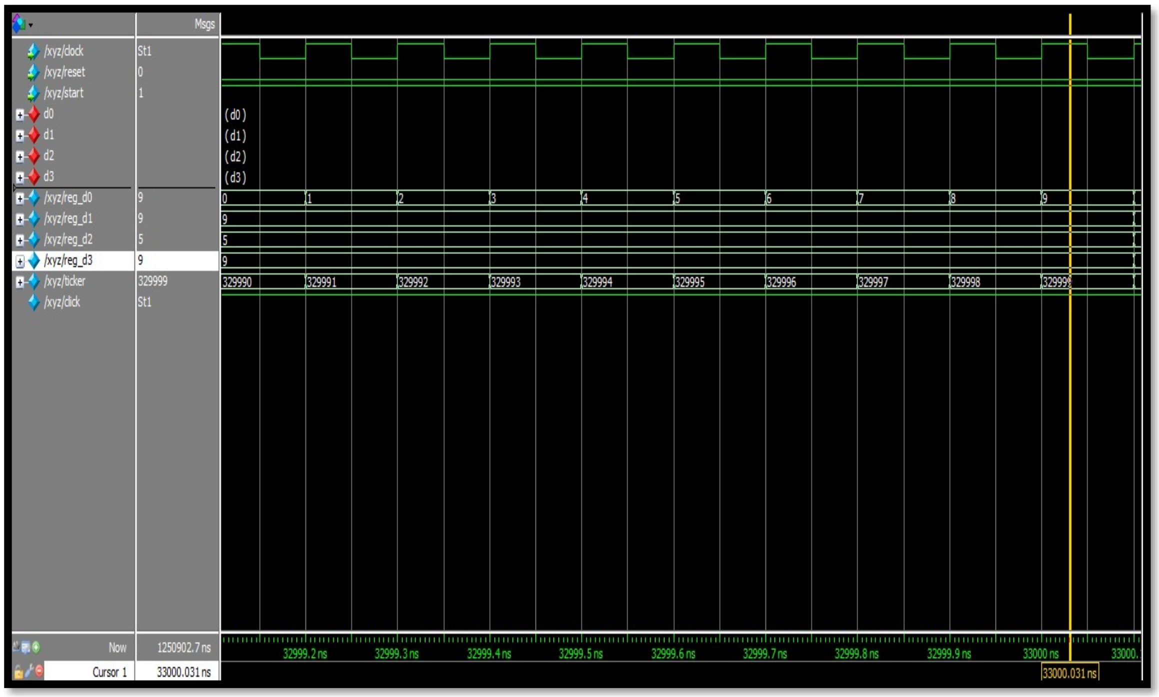
```
assign g0 =  ~((reg_d0[3]) | (~reg_d0[2] & reg_d0[1]) | (reg_d0[2] & ~reg_d0[1]) |
(reg_d0[2] & ~reg_d0[0])) ;


// d1


assign a1 = ~((reg_d1[3]) | (reg_d1[1]) | (reg_d1[2] & reg_d1[0]) | (~reg_d1[2] &
~reg_d1[0])) ;


assign b1 = ~((~reg_d1[2]) | (reg_d1[1] & reg_d1[0]) | (~reg_d1[1] & ~reg_d1[0]))  ;


assign c1 = ~((reg_d1[2]) | (~reg_d1[1]) | (reg_d1[0])) ;


assign d1 = ~((reg_d1[3]) | (~reg_d1[2] & ~reg_d1[0]) | (~reg_d1[2] & reg_d1[1]) |
(reg_d1[2] & ~reg_d1[1] & reg_d1[0]) | (reg_d1[1] & ~reg_d1[0])) ;


assign e1 = ~((~reg_d1[2] & ~reg_d1[0]) | (reg_d1[1] & ~reg_d1[0])) ;


assign f1 = ~((reg_d1[3]) | (reg_d1[2] & ~reg_d1[1]) | (reg_d1[2] & ~reg_d1[0]) |
(~reg_d1[1] & ~reg_d1[0])) ;


assign g1 =  ~((reg_d1[3]) | (~reg_d1[2] & reg_d1[1]) | (reg_d1[2] & ~reg_d1[1]) |
(reg_d1[2] & ~reg_d1[0])) ;


// d2


assign a2 = ~((reg_d2[3]) | (reg_d2[1]) | (reg_d2[2] & reg_d2[0]) | (~reg_d2[2] &
~reg_d2[0])) ;


assign b2 = ~((~reg_d2[2]) | (reg_d2[1] & reg_d2[0]) | (~reg_d2[1] & ~reg_d2[0]))  ;


assign c2 = ~((reg_d2[2]) | (~reg_d2[1]) | (reg_d2[0])) ;
assign d2 = ~((reg_d2[3]) | (~reg_d2[2] & ~reg_d2[0]) | (~reg_d2[2] & reg_d2[1]) |
(reg_d2[2] & ~reg_d2[1] & reg_d2[0]) | (reg_d2[1] & ~reg_d2[0])) ;


assign e2 = ~((~reg_d2[2] & ~reg_d2[0]) | (reg_d2[1] & ~reg_d2[0])) ;


assign f2 = ~((reg_d2[3]) | (reg_d2[2] & ~reg_d2[1]) | (reg_d2[2] & ~reg_d2[0]) |
(~reg_d2[1] & ~reg_d2[0])) ;


assign g2 =  ~((reg_d2[3]) | (~reg_d2[2] & reg_d2[1]) | (reg_d2[2] & ~reg_d2[1]) |
(reg_d2[2] & ~reg_d2[0])) ;


// d3
assign a3 = ~((reg_d3[3]) | (reg_d3[1]) | (reg_d3[2] & reg_d3[0]) | (~reg_d3[2] &
~reg_d3[0])) ;


assign b3 = ~((~reg_d3[2]) | (reg_d3[1] & reg_d3[0]) | (~reg_d3[1] & ~reg_d3[0]))  ;


assign c3 = ~((reg_d3[2]) | (~reg_d3[1]) | (reg_d3[0])) ;


assign d3 = ~((reg_d3[3]) | (~reg_d3[2] & ~reg_d3[0]) | (~reg_d3[2] & reg_d3[1]) |
(reg_d3[2] & ~reg_d3[1] & reg_d3[0]) | (reg_d3[1] & ~reg_d3[0])) ;


assign e3 = ~((~reg_d3[2] & ~reg_d3[0]) | (reg_d3[1] & ~reg_d3[0])) ;


assign f3 = ~((reg_d3[3]) | (reg_d3[2] & ~reg_d3[1]) | (reg_d3[2] & ~reg_d3[0]) |
(~reg_d3[1] & ~reg_d3[0])) ;


assign g3 = ~ ((reg_d3[3]) | (~reg_d3[2] & reg_d3[1]) | (reg_d3[2] & ~reg_d3[1]) |
(reg_d3[2] & ~reg_d3[0])) ;
 endmodule
```

• *RTL VIEW:*

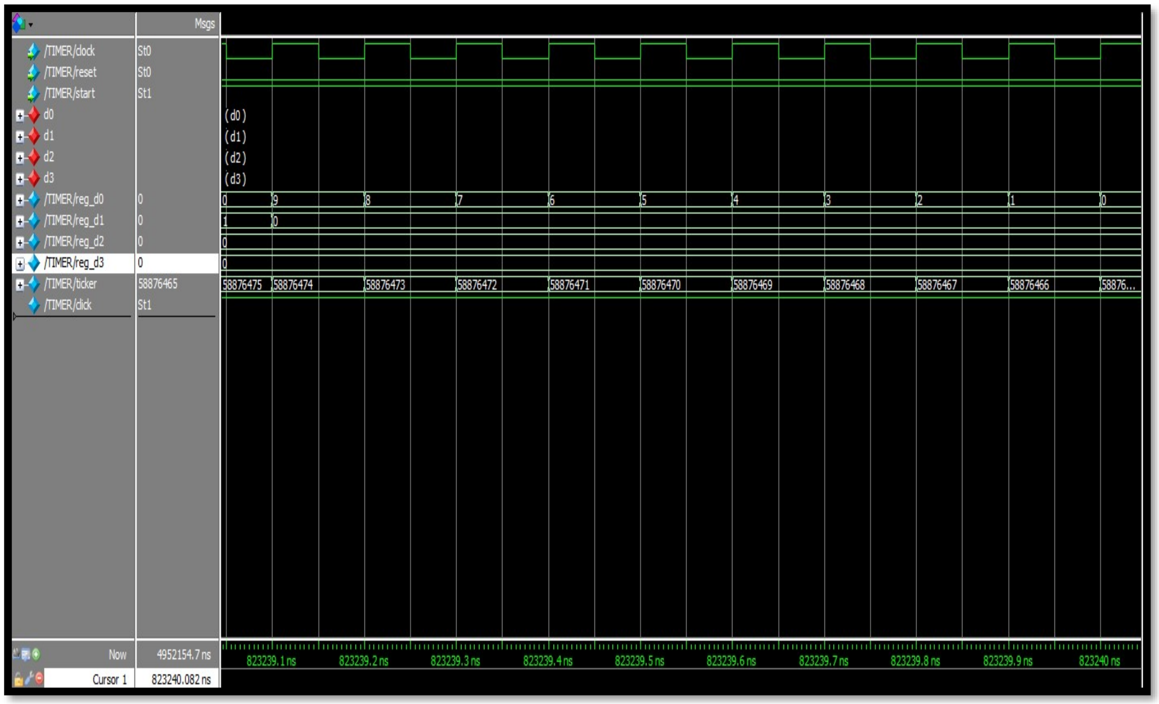


- ***SIMULATION:***

Stopwatch: -



TIMER: -



- *Conclusion: -*

  After successful compilation of Project, we learn the working and simulation of stopwatch. The stop watch designed would be able to count till 10 min i.e. (9:59:9) & Timer would be able to count backwards from 10 min i.e. (9:59:9 to 0:00:0) . We hereby conclude that the simulated design can be readily implemented on any FPGA board.

References: -

- *Verilog HDL A Guide to Digital Design and Systems by Samir Palnitkar*
- *Digital Logic and Computer Design by M. Morris Mano*