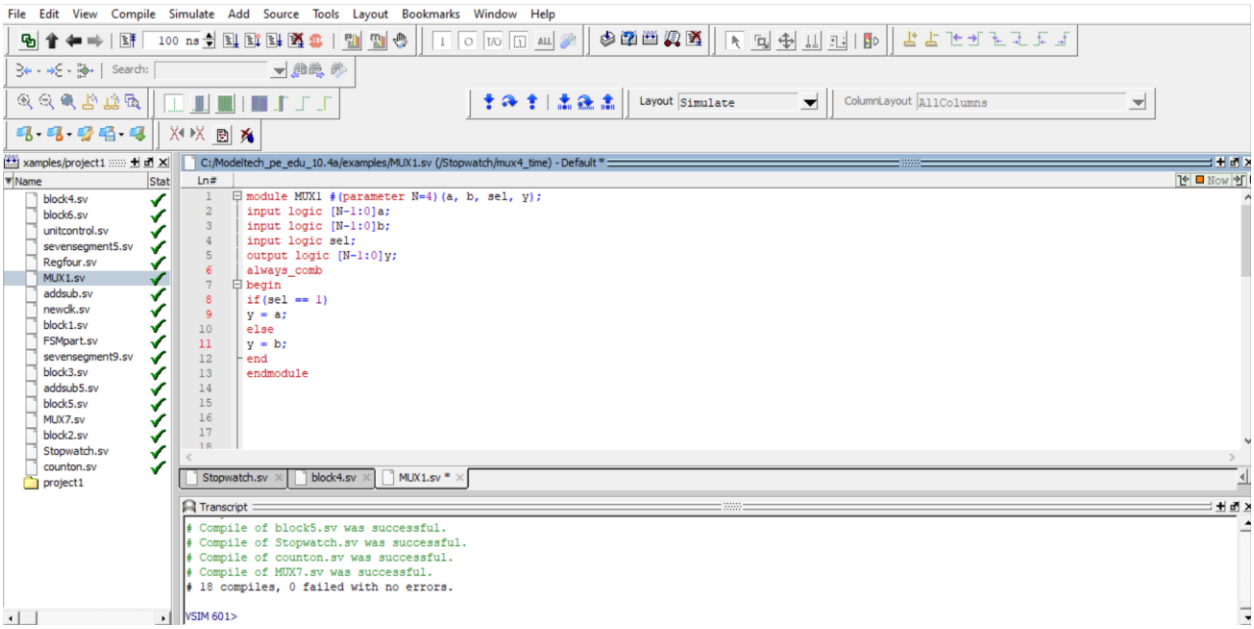
Stopwatch -Watch Project

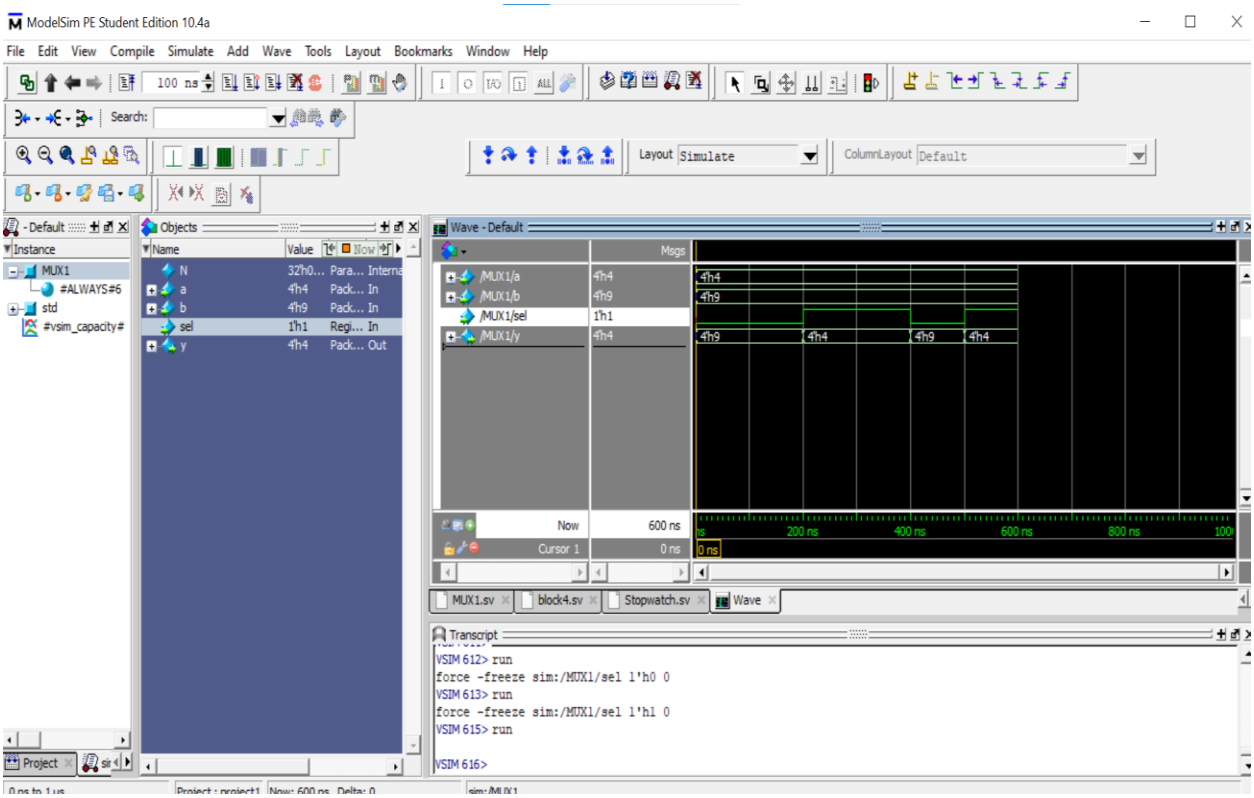
This device is mainly Stopwatch and watch. And the main idea of the both watch and stopwatch is counting the number of seconds and display (minutes and seconds in form of “mm: ss” form 00:00 to 59:59 in case of stopwatch), or (minutes and hours in form of “hh: mm” from 00:00 to 23:59 in case of watch) in the seven segment displays. For example, if the seven segment displays are going to print the clock, which is 6 and half PM, they would print 18:30. on the other hand, if the user is doing on the stopwatch part and the stopwatch counted just 3 minutes and 48 seconds, the seven segment displays are going to print 03:48. So let’s First start with the main component used in both watch and stopwatch. Then, we are to mention the main blocks in the device and its function, code, and simulation. Finally, we will combine all of these block to make the whole project.

Firstly, the basic components used in both the watch and the stop watch:

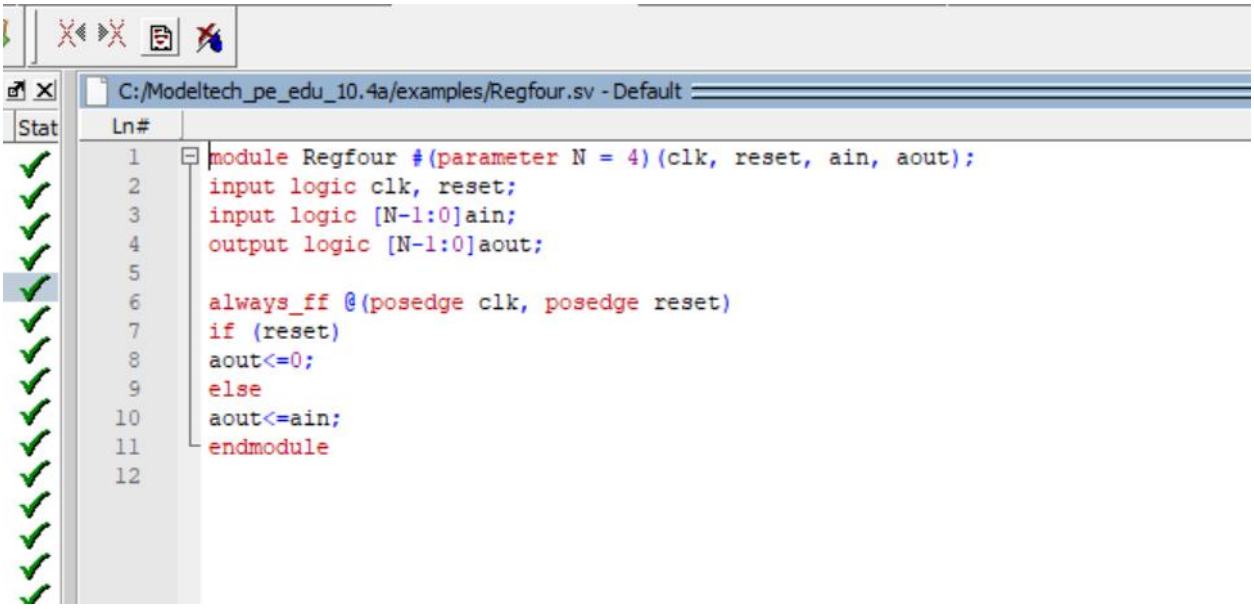
1) MUX1 is a multiplexer that takes two inputs, if condition, and select which one of the two inputs to present on the output. And in the code it is parameterized to give it however the number of the inputs and the output.



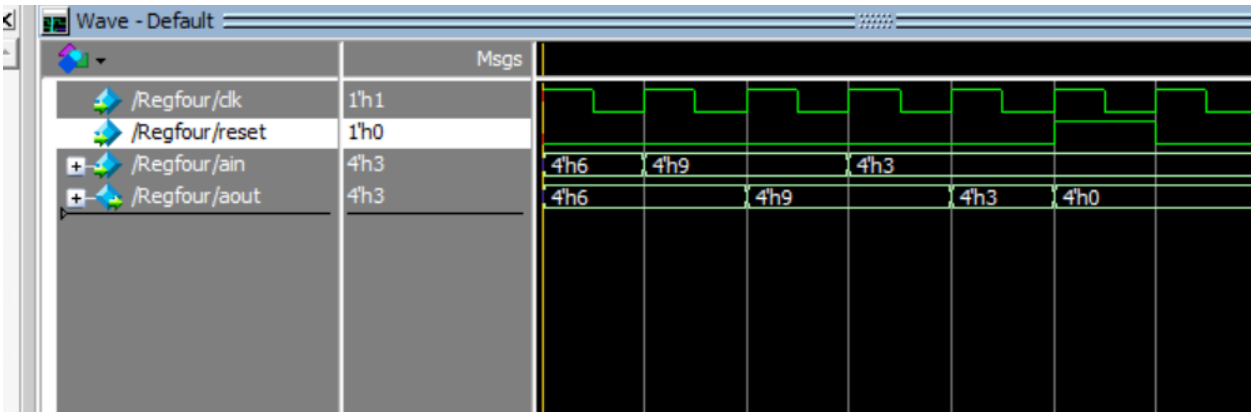
It is clear that the output follows the first input in case of the if condition (which is called “sel” in the code) equal to one, but follows the second input in case of the if condition is equal to zero.



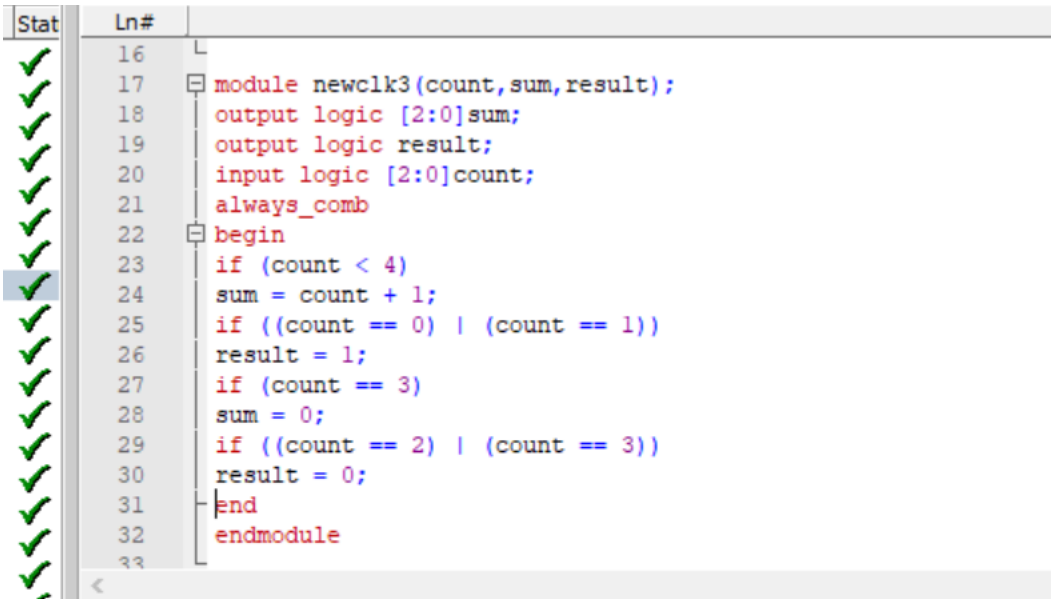
2) Regfour is a register that transfer the data once per one period of the clock (which is clk in the code). So, if the clock is 50MHz. So, it makes this operation 50 million times per a second. However, in our case we will assume that the clock is 1Hz (the next part is about how to transfer the 50MHz clock into the 1Hz clock), so this operation is to be done once per a second. It is parameterized too.



Also, Regfour has the option to reset the value to zero. And to make the reset whenever I want, the “always_ff” is sensitive to the positive edge of reset and definitely to the clock to respond once per second as It mentioned. In the simulation is responding each after each positive edge of the clock as expected. Also, when we reset the output takes zero value.

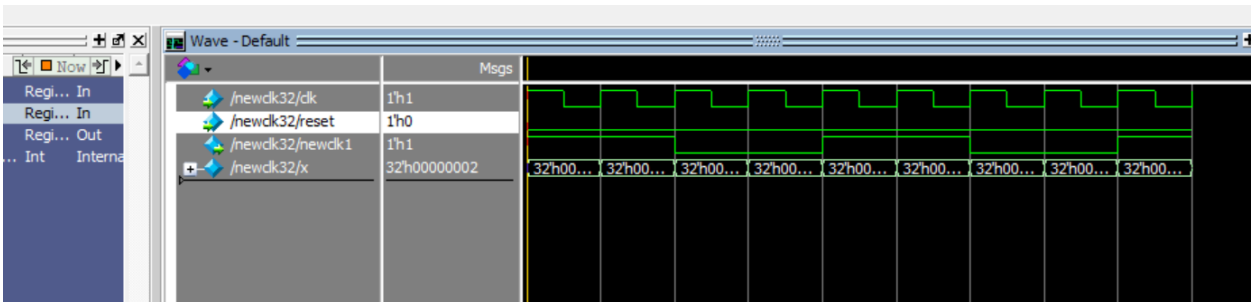


3) This part to show how to convert the 50MHz into the 1Hz, but to test, or simulate as we will clock about 50 million times to simulate just one clock from (1Hz). So, we will convert form 4Hz into 1Hz. The idea is almost a counter that count 2, then makes the output is equal to one and count 2, then makes the output is equal to zero.



Also, there is another method for this converting which is to use flip flop to count the number of the old clock and so make the new clock, as in the next code

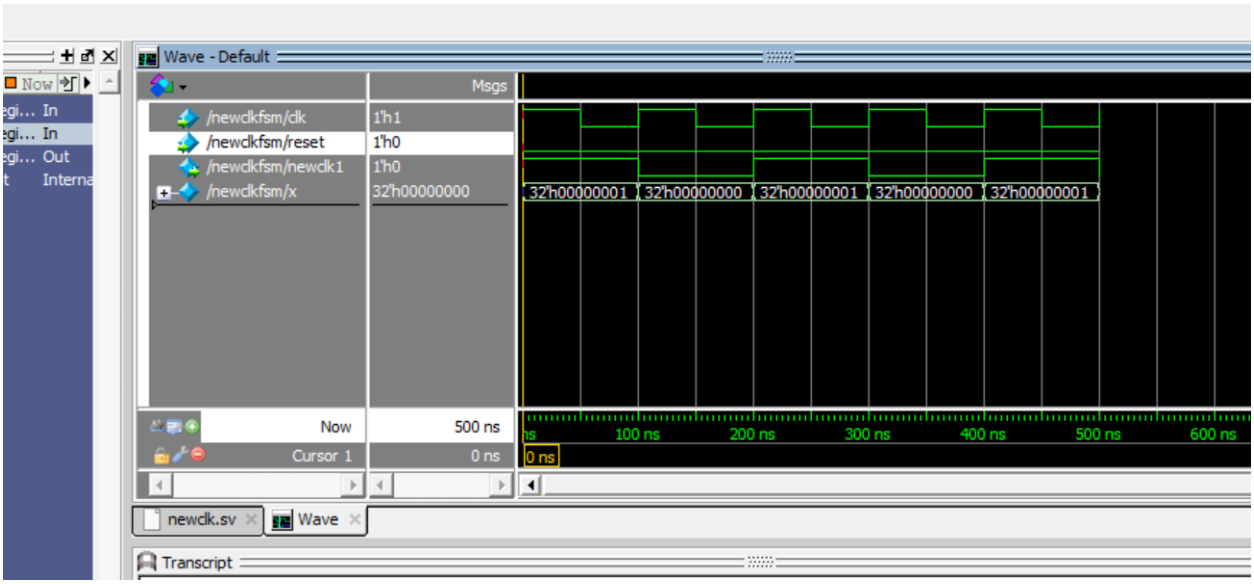
```
15
16  module newclk32(clk,reset,newclk1);
17  input logic clk,reset;
18  output logic newclk1;
19  int x = 0;
20  always_ff@(posedge clk, posedge reset)
21  begin
22  if (x <= 4)
23  x = x + 1;
24  if(x<=2)
25  newclk1 = 1;
26  else
27  newclk1 = 0;
28  if (x == 4 || reset == 1)
29  x = 0;
30  end
31  endmodule
32
```



4) This part is mainly for making the periodic moving of the power saver mood and to make the two red leds in between the seven segment displays turn on in a half a second, and turn off in the other half of the second. This part is to make another clock that is neither 1Hz nor the entered clock. So, this clock is something in between (half of the entered clock). And The main reason for this new clock is not to make the moving period so fast like the entered clock(50MHz) or so slow like the clock (1Hz), but something in between. And the two leds to turn on, and turn off each half a second. Since we used the (4Hz)

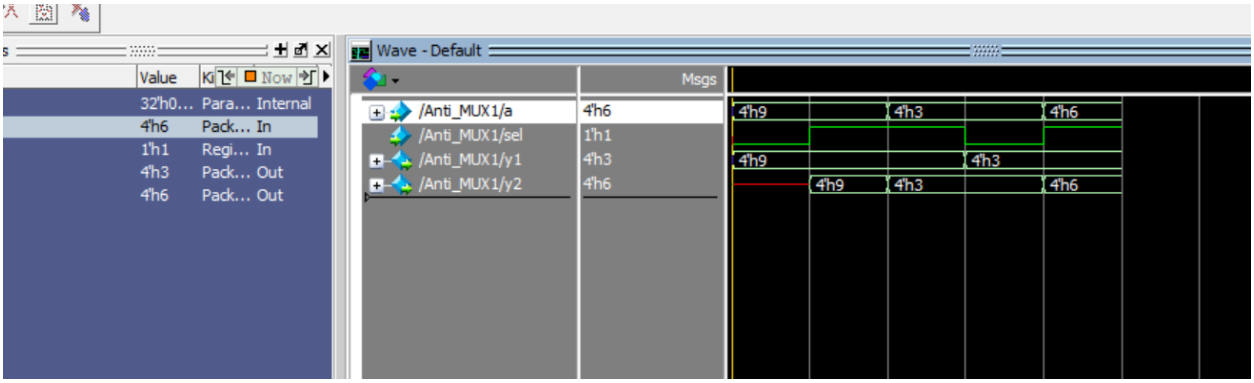
instead of (50 MHz), then this part will use (2Hz which is “half of the entered clock(4Hz)” and double of the clock of the circuit (1Hz)).

```
54  module newclkfsm(clk,reset,newclk1);
55  input logic clk,reset;
56  output logic newclk1;
57  int x = 0;
58  always_ff@(posedge clk, posedge reset)
59  begin
60  if (x <= 2)
61  x = x + 1;
62  if(x<=1)
63  newclk1 = 1;
64  else
65  newclk1 = 0;
66  if (x == 2 || reset == 1)
67  x = 0;
68  end
69
```

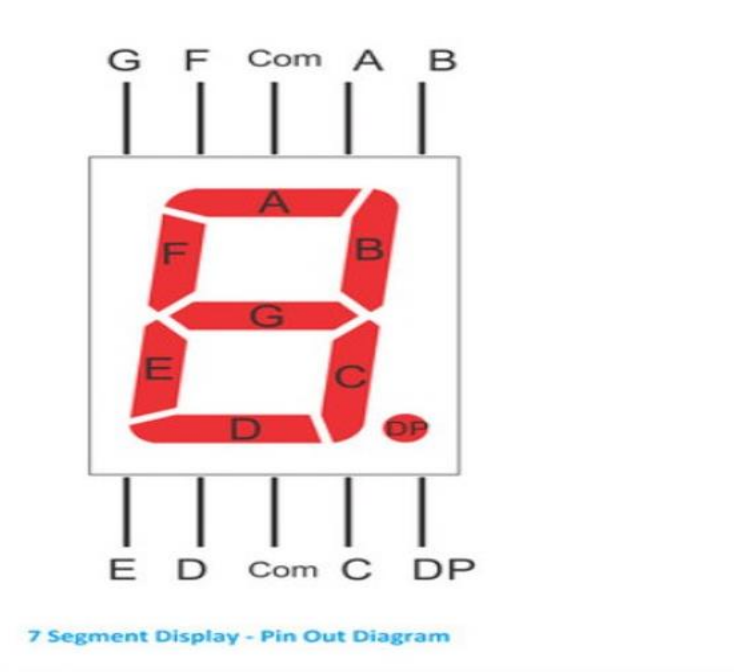


5) Anti_MUX1 is somehow similar to the MUX1. Anti_MUX1 takes just one input, and if condition and gives two inputs. According to the if condition makes one input is equal to the input.

```
module Anti_MUX1 #(parameter N=4)(a, sel, y1,y2);
input logic [N-1:0]a;
input logic sel;
output logic [N-1:0]y1;
output logic [N-1:0]y2;
always_comb
begin
if(sel == 1)
y2 = a;
else
y1 = a;
end
endmodule
```



6) seven segment displays that is mainly seven leds, each led will turn on in case of it take a value of one or take the voltage of the circuit. (1 - on, 0 - off). and this the picture of it to study its own functionality.



7 Segment Display - Pin Out Diagram

https://www.mynewsdesk.com/cn/kynix/blog_posts/seven-segment-display-operation-by-using-atmega32-and-cd4511b-57730

So, if com is connected with the ground, and for example, the led number B, and C are equal to 1. Meanwhile, the other leds is equal to 0, then the seven segment will print one. Also, if A, F, G, C, and B leds are equal to one, but the other leds are equal to zero, then the seven segment will print 5, and so one. In our project, the DP led will not be used so is always connected to the ground. Now, the value will come from four wires to be presented into the seven segment, so we are in need to make a decoder that convert the value coming through the four wires $\{Q_0, Q_1, Q_2, Q_3\}$ into seven wires $\{y_0, y_1, y_2, y_3, y_4, y_5, y_6\}$ to present the number. y_0 will directly connect with B in the seven segment display, y_1 will directly connect with C in the seven segment display, y_2 will directly connect with D in the seven segment display, y_3 will directly connect with E in the seven segment display, y_4 will directly connect with G in the seven segment display, y_5 will directly connect with F in the seven segment display, y_6 will directly connect with A in the seven segment display.

There are two decoders. One of them is to print from zero to nine on the seven segment. The second one is to print from zero to nine.

The First decoder that will print maximum 9:

number	Q0	Q1	Q3	Q4	y0 = B	y1= C	y2= D	y3= E	y4= G	y5= F	y6= A
0	0	0	0	0	1	1	1	1	0	1	1
1	0	0	0	1	1	1	0	0	0	0	0
2	0	0	1	0	1	0	1	1	1	0	1
3	0	0	1	1	1	1	1	0	1	0	1
4	0	1	0	0	1	1	0	0	1	1	0
5	0	1	0	1	0	1	1	0	1	1	1
6	0	1	1	0	0	1	1	1	1	1	1
7	0	1	1	1	1	1	0	0	0	0	1
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	1	1	1

and with doing k-map for this truth table, we can reach the following equations

$$y_0 = \overline{Q2} + \overline{Q0}\,\overline{Q1} + Q1\,Q0$$

$$y_1 = \overline{Q1} + Q3 + Q2 + Q0$$

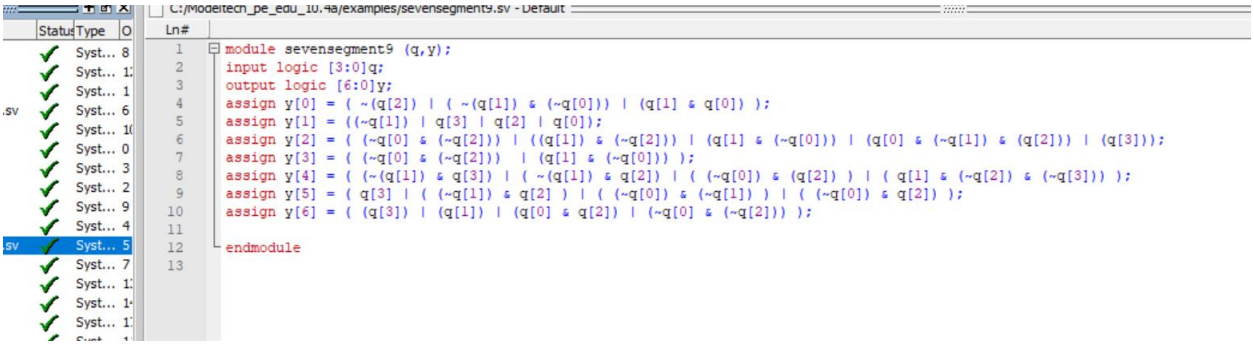
$$y_2 = \overline{Q0}\,\overline{Q2} + Q1\,\overline{Q2} + \overline{Q0}\,Q1 + Q0\,\overline{Q1}\,Q2 + Q3$$

$$y_3 = \overline{Q0}(\overline{Q2} + Q1)$$

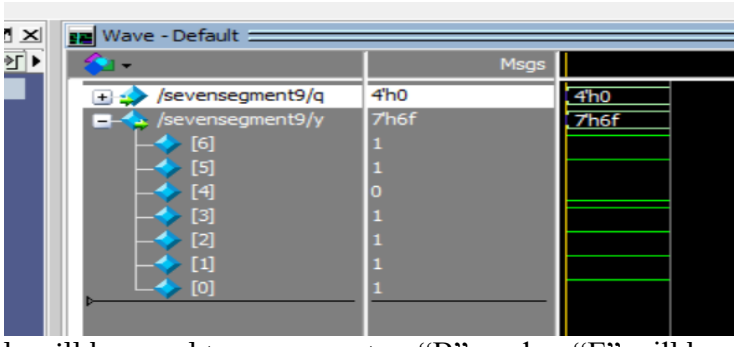
$$y_4 = \overline{Q1}\,Q2 + \overline{Q1}\,Q3 + \overline{Q0}\,Q2 + Q1\,\overline{Q2}\,\overline{Q3}$$

$$y_5 = Q3 + \overline{Q1}\,Q2 + \overline{Q0}\,\overline{Q1} + \overline{Q0}\,Q2$$

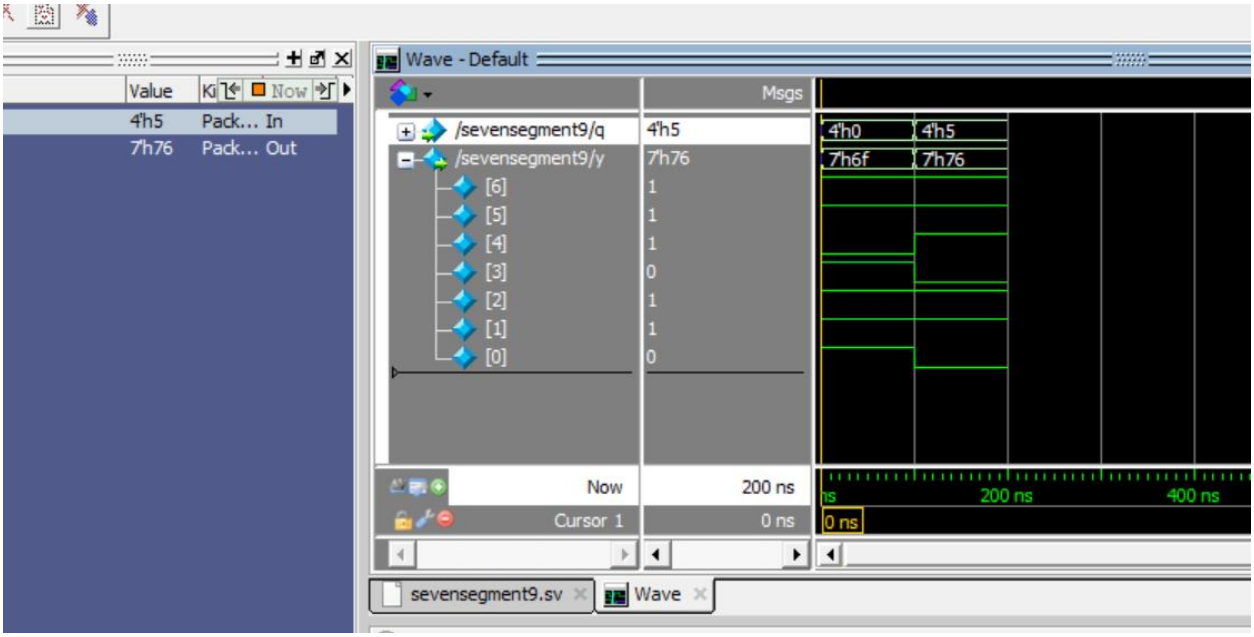
$$y_6 = Q1 + Q3 + \overline{Q0}\,\overline{Q2} + Q0\,Q2$$



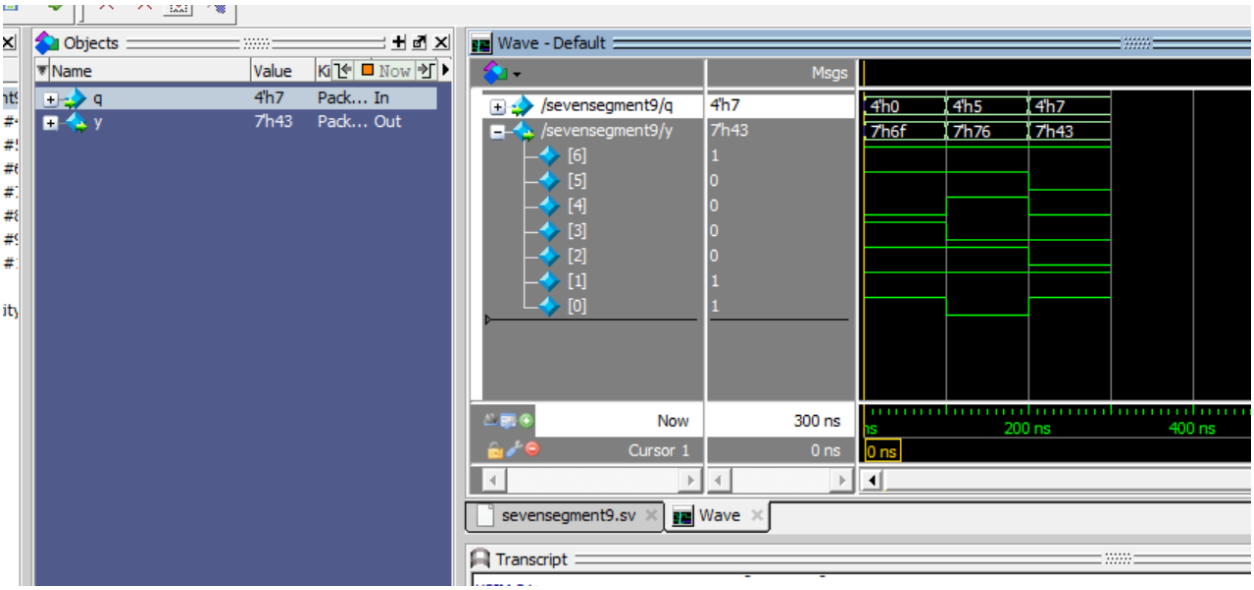
So, to print zero leds will be equal to one except y_4 “G” will be equal to zero.



and to print 5, all leds will be equal to one except y_0 “B”, and y_3 “E” will be equal to zero.



also to print 7, the led number y_0 “B”, and y_1 “C”, y_6 “A” will be equal to one. Meanwhile, the other leds will be equal to zero.



and for the other decoder that prints from 0 to 5 there is another decoder. The truth table of this decoder is

number	Q0	Q1	Q3	Q4	y0 = B	y1= C	y2= D	y3= E	y4= G	y5= F	y6= A
0	0	0	0	0	1	1	1	1	0	1	1
1	0	0	0	1	1	1	0	0	0	0	0
2	0	0	1	0	1	0	1	1	1	0	1
3	0	0	1	1	1	1	1	0	1	0	1
4	0	1	0	0	1	1	0	0	1	1	0
5	0	1	0	1	0	1	1	0	1	1	1

and with doing k-map with this truth table, we can get the next equations:

$$y_0 = \overline{Q2} + \overline{Q0}\,\overline{Q1}$$

$$y_1 = \overline{Q1} + Q2 + Q0$$

$$y_2 = \overline{Q0}\,\overline{Q2} + Q1 + Q0\,Q2$$

$$y_3 = \overline{Q0}(\overline{Q2} + Q1)$$

$$y_4 = Q2 + Q1$$

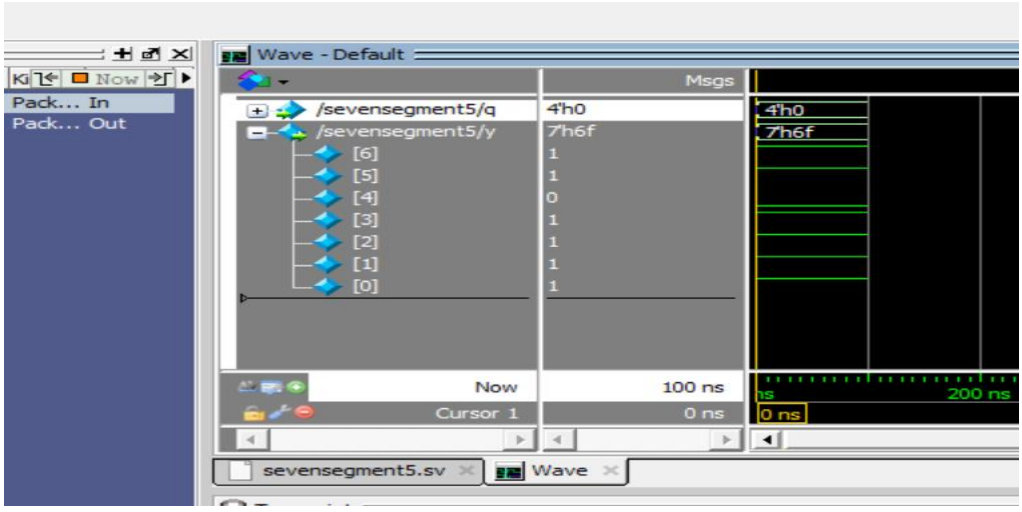
$$y_5 = Q2\,\overline{Q3} + \overline{Q0}\,\overline{Q1}$$

$$y_6 = \overline{Q0}\,\overline{Q2} + Q1 + Q0\,Q2$$

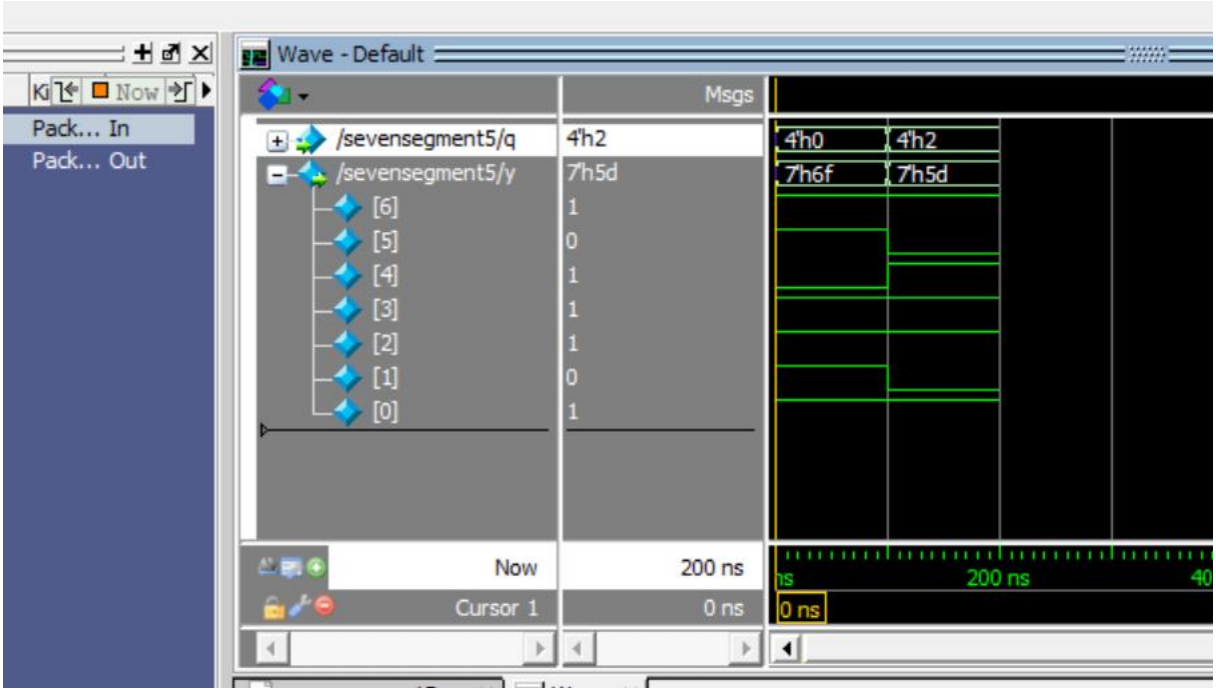
C:/Modeltech_pe_edu_10.4a/examples/sevensegment5.sv - Default *

```
Ln#
1   module sevensegment5(q,y);
2   input logic [3:0]q;
3   output logic [6:0]y;
4   assign y[0] = (((~q[0]) & (~q[1])) |(~q[2]));
5   assign y[1] = (q[0] |(~q[1]) | (q[2]));
6   assign y[2] = (q[1]|((~q[0])&(~q[2]))|(q[0]&q[2]));
7   assign y[3] = ((~q[0])&(q[1]|(~q[2])));
8   assign y[4] = (q[1]|q[2]);
9   assign y[5] = (((~q[0])&(~q[1]))|(q[2]&(~q[3])));
10  assign y[6] = (q[1] |(~q[0]&(~q[2])) | (q[0]&q[2]) );
11  endmodule
12
13
14
15
```

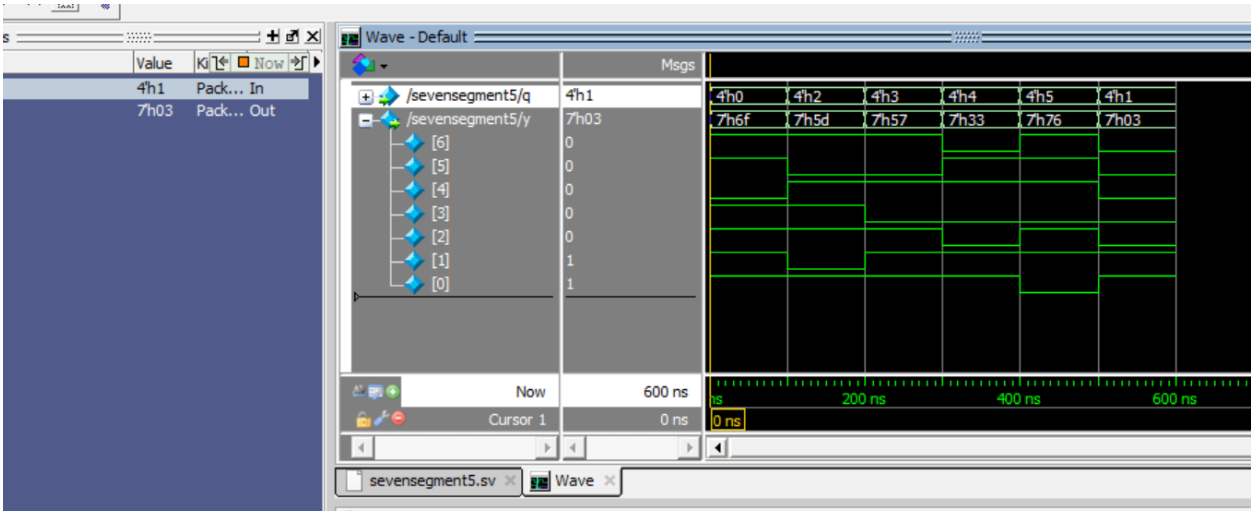
So, to print zero leds will be equal to one except y_4 “G” will equal to zero.



and to print 2, all leds will equal to one except y_1 “C”, and y_5 “F” will equal to zero.



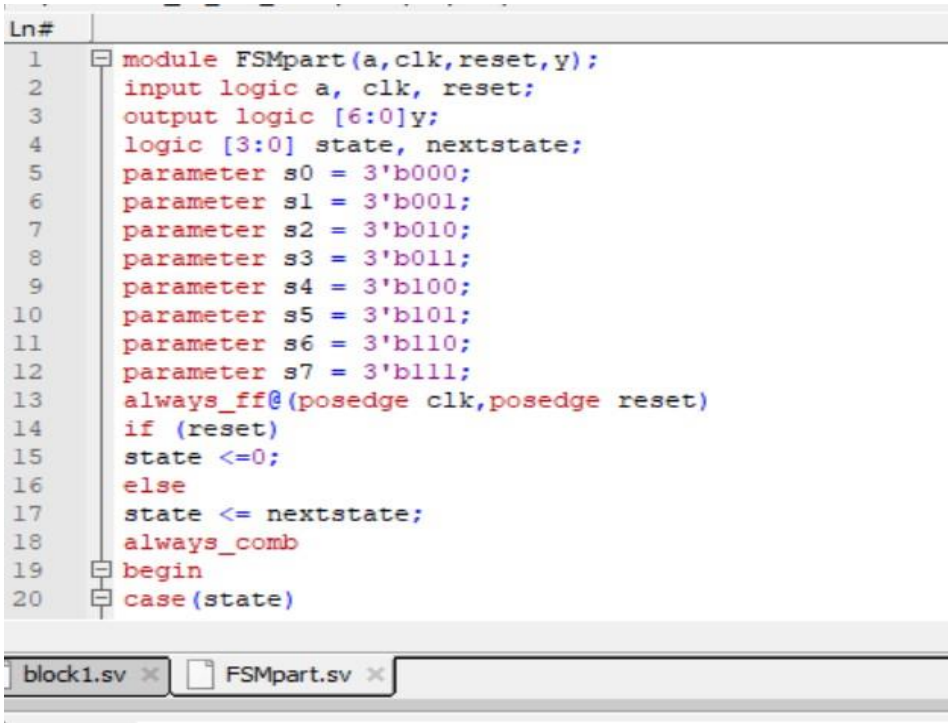
, and so on with all numbers.

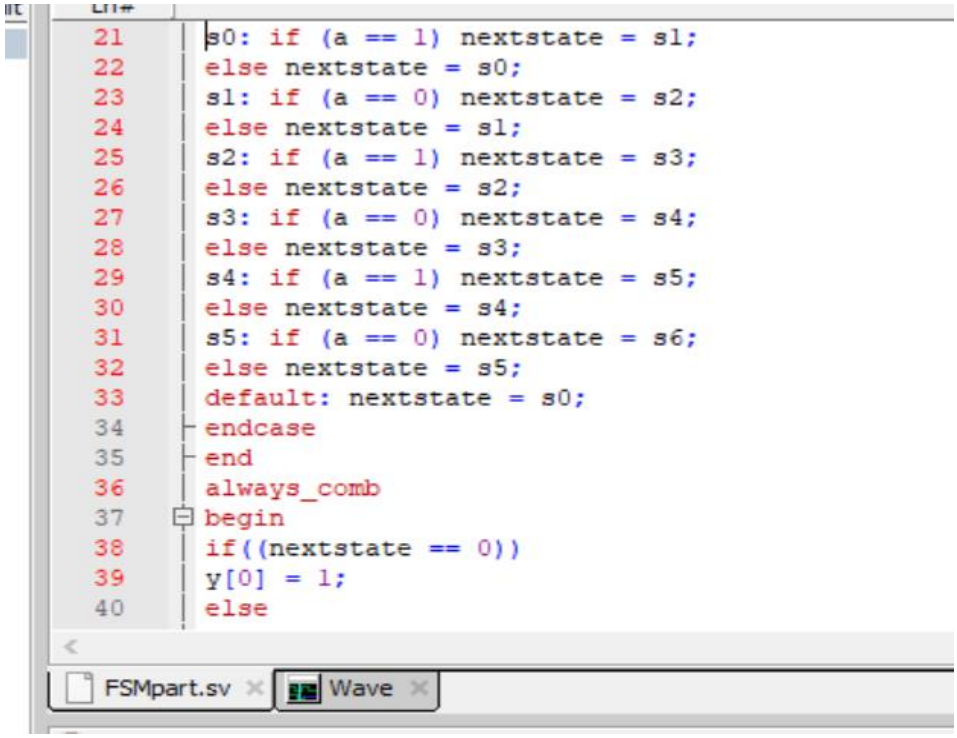


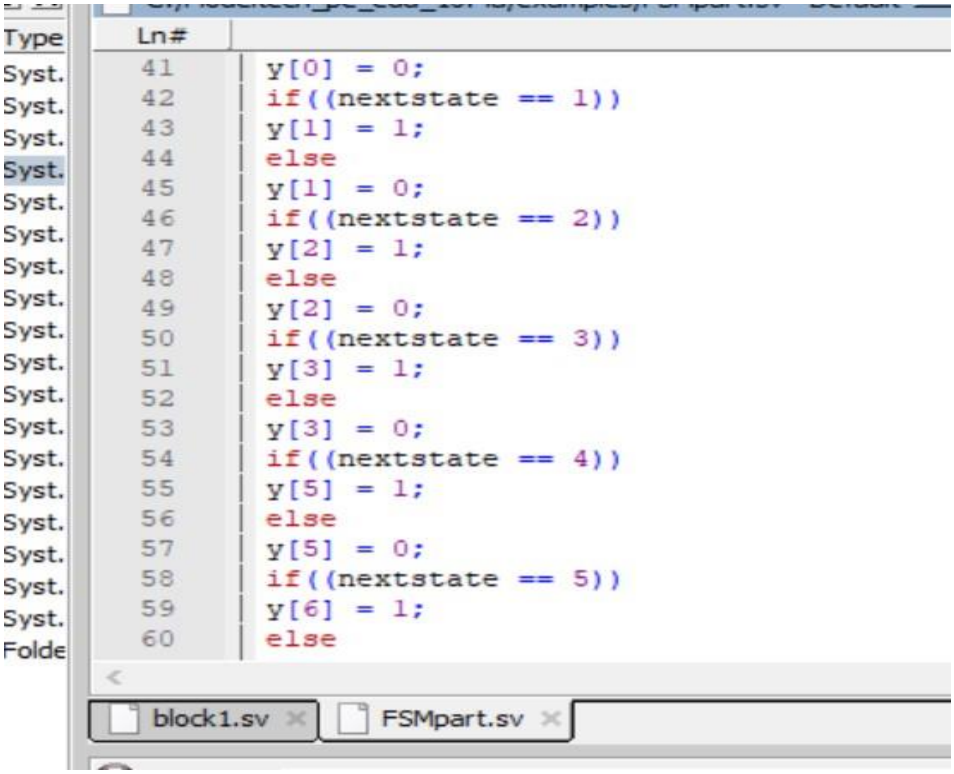
Neither the first decoder will neither get a number out of range (0: 9), nor the second decoder will get a number out of range (0: 5) because our counters count only within the range.

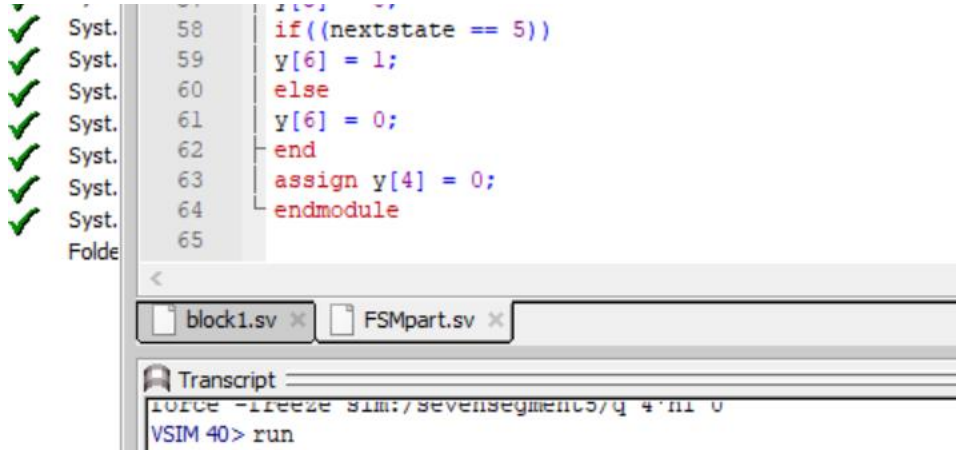
7) FSMpart is responsible for making the periodic motion on power saver mode as we can see in the simulation below that there is a periodic change between the whole output except the y_4 as this periodic motion will be in the outer leds only. the default state in the FSM is S_0 and if the input is one, then the next state will be S_1 but if the input (which is called “a” in code) is zero, the next state will be S_0, and vice versa with S_1 if the input is one, the next state will be S_1 too. But, if the input is zero it will be S_2. and vice versa with S_3 and so on. this because to give an enough period of time to recognize the led motion.

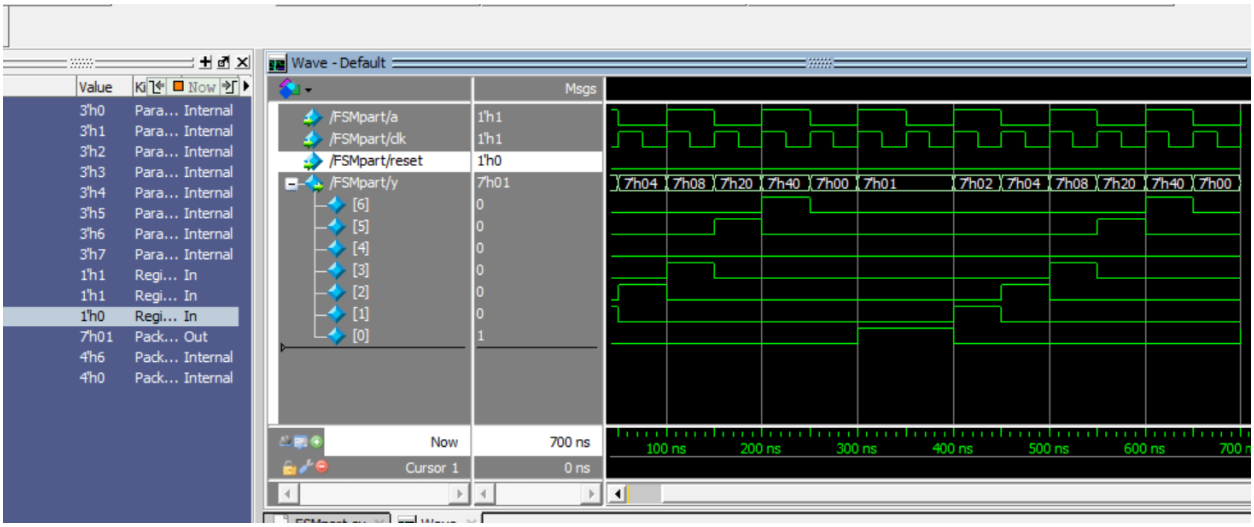
Also, the motion should be not slow. For example, if the input is always equal to one, there will be just one led turned on per second (if the clock is 1Hz). So, the input a is double the clock used in the circuit. Finally, each led will turn on for a half second expect second expect the led number 4 will not be turned on. and the led number 0 will turn on for a complete one second.






Type
Ln#
Syst.
Folde
41 y[0] = 0;
42 if((nextstate == 1))
43 y[1] = 1;
44 else
45 y[1] = 0;
46 if((nextstate == 2))
47 y[2] = 1;
48 else
49 y[2] = 0;
50 if((nextstate == 3))
51 y[3] = 1;
52 else
53 y[3] = 0;
54 if((nextstate == 4))
55 y[5] = 1;
56 else
57 y[5] = 0;
58 if((nextstate == 5))
59 y[6] = 1;
60 else
block1.sv
FSMpart.sv

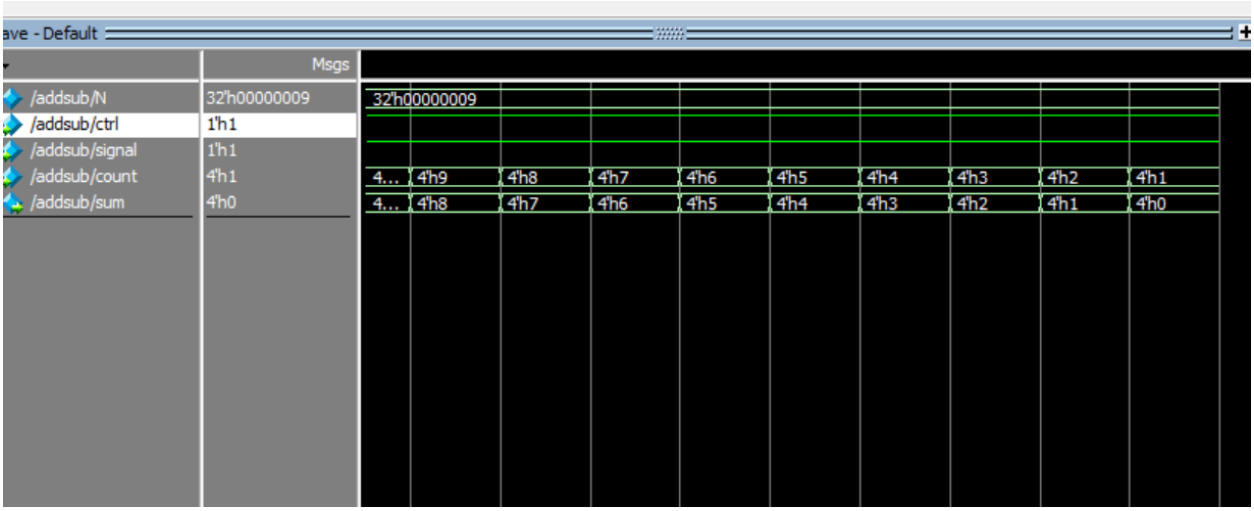

Syst.
Folde
58 if((nextstate == 5))
59 y[6] = 1;
60 else
61 y[6] = 0;
62 end
63 assign y[4] = 0;
64 endmodule
65
block1.sv
FSMpart.sv
Transcript
VSIM 40> run



Now almost the whole main components we will use in this project are explained in details. So,

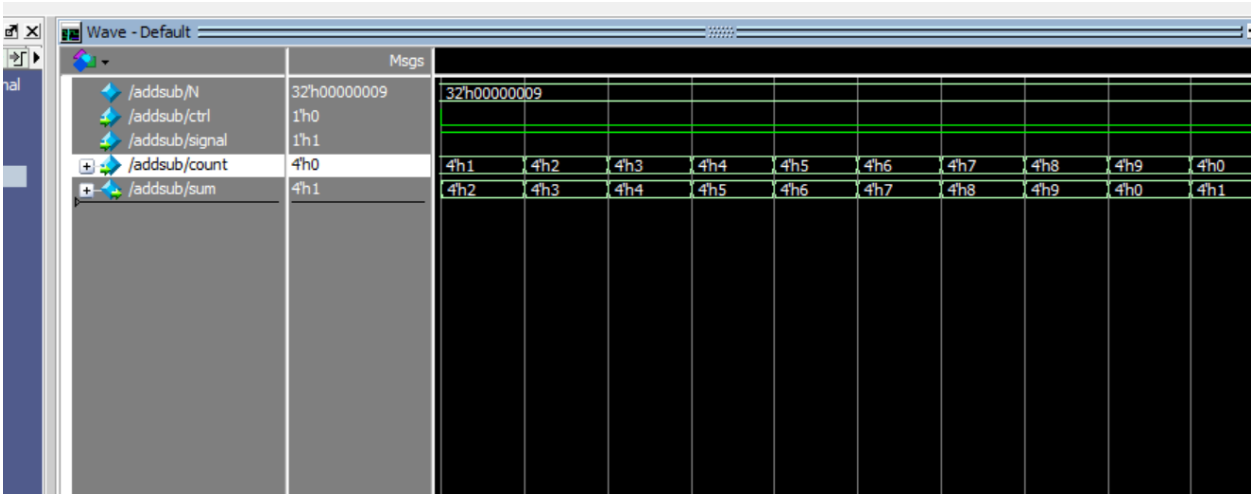
8) addsub: is the counter that will count up or down according to the ctrl. If ctrl is equal to zero, it would count up. On the other hand, if ctrl is equal to one, it would count down. This block is parameterized. As this parameter is to define the edge to count to, or from. For example, if the parameter is 9. So, it would count from 0 to 9, or from 9 to 0 according to the ctrl. Note that signal defined in the code is to define continue on counting, or not. If signal equal to one, the counter will work well. Otherwise, it would stop.

```
Ln#
 1  module addsub# (parameter N = 9)(count,sum,ctrl,signal);
 2  output logic [3:0]sum;
 3  input logic ctrl,signal;
 4  input logic [3:0]count;
 5  always_comb
 6  begin
 7  if ((~ctrl) & signal)
 8  begin
 9  if (~(count == N))
10  sum = count + 1;
11  if (count == N)
12  sum = 0;
13  end
14  if(ctrl & signal)
15  begin
16  if (~(count == 0))
17  sum = count - 1;
18  if (count == 0)
19  sum = N;
20  end
21  end
22  endmodule
23
```



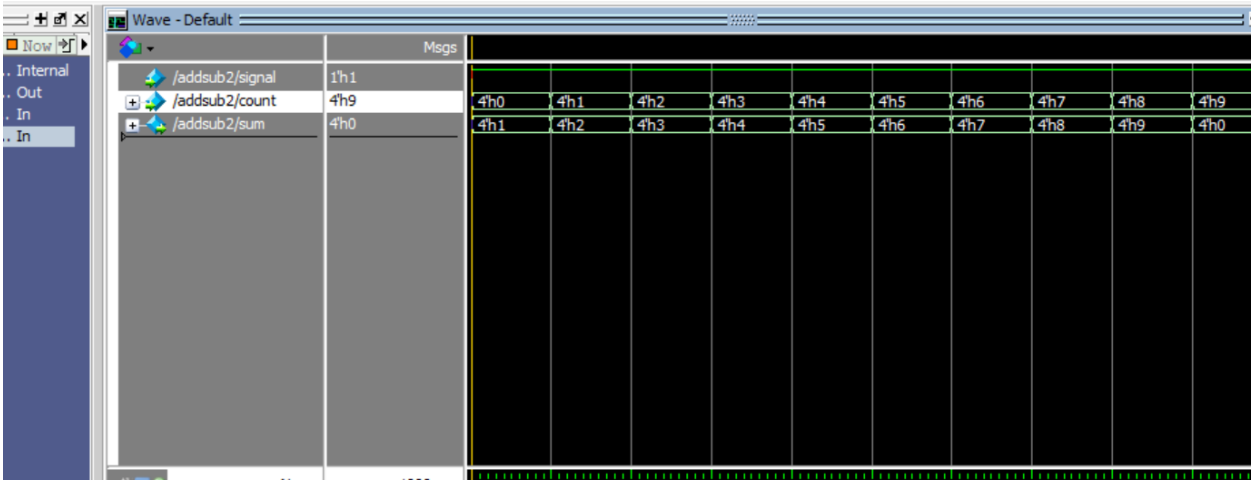
Note that sum is the result and count is the input.

and to test counting up:



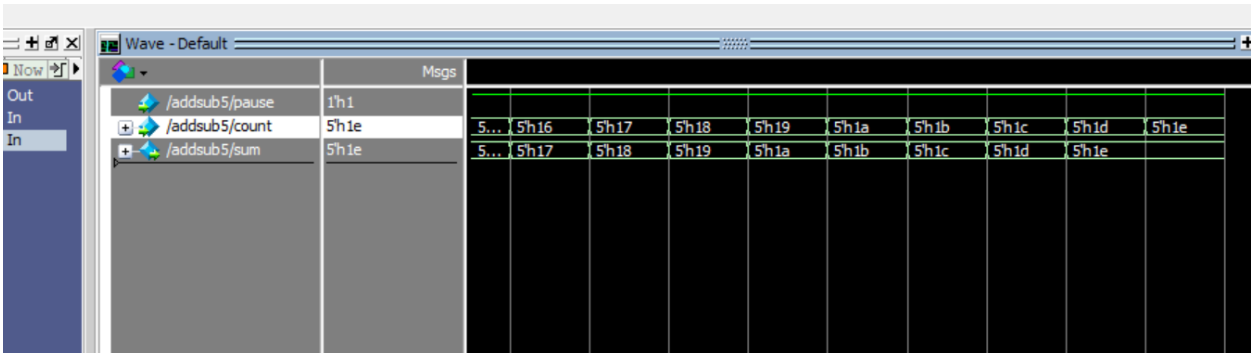
9) addsub2: is typical to addsub, but there is no option to count down

```
26  module addsub2 #(parameter N = 9)(count,sum,signal);
27  output logic [3:0]sum;
28  input logic signal;
29  input logic [3:0]count;
30  always_comb
31  begin
32  if (signal)
33  begin
34  if (~(count == N))
35  sum = count + 1;
36  if (count == N)
37  sum = 0;
38  end
39  end
40  endmodule
41
```



10) addsub5: is typical to addsub2, but counting from zero to 30. And the reason for not using addsub2 and giving it a parameter to count till 30 is that the number of wires differ in both cases. So, it would be better and easier to use another counter like addsub5

```
Ln#
 1  module addsub5(count,sum,pause);
 2  output logic [4:0]sum;
 3  input logic pause;
 4  input logic [4:0]count;
 5  always_comb
 6  begin
 7  if (pause)
 8  begin
 9  if (count < 30)
10  sum = count + 1;
11  end
12  if (~pause & (count == 30))
13  sum = 0;
14  end
15  endmodule
16
```



11) block1 is the heart of the device. block1 is the main component that count the number of seconds and then send the whole data to another block which access the seven segment displays to present it. **This block is only for the stopwatch.**

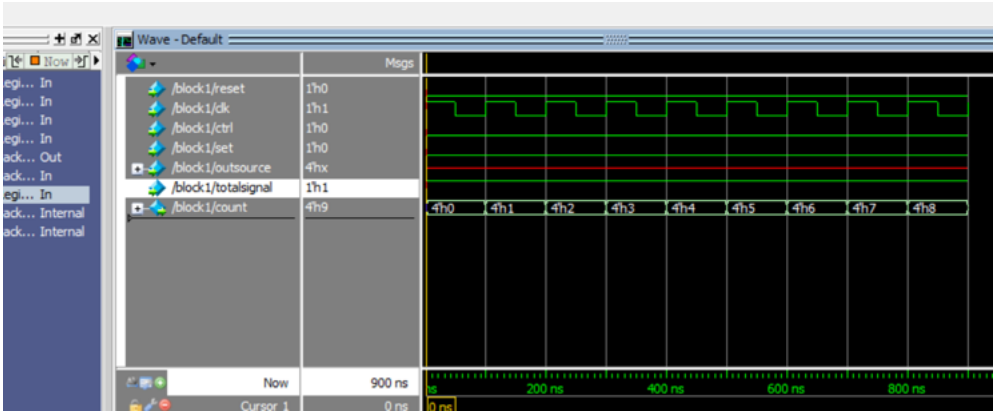
C:/Modeltech_pe_edu_10.4a/examples/block1.sv (/block1) - Default *

```
Ln#
 1  module block1# (parameter N = 9)(clk,reset,ctrl,set,outsource,count,totalsignal);//first block
 2  input logic reset,clk,set,totalsignal,ctrl; // internal for updating after reaching the end.
 3  output logic [3:0]count;
 4  input logic [3:0]outsource;
 5  logic [3:0] sum1,sum2,sum0;
 6  assign count = 0;
 7  addsub #(N) A1(count,sum0,ctrl,totalsignal);
 8  MUX1 #(4) muxcheck(count,sum0,~totalsignal,sum1);
 9  MUX1 #(4) mux1(outsource,sum1,set,sum2);
10  Regfour #(4) R1(clk,reset,sum2, count);
11  endmodule
12
13
```

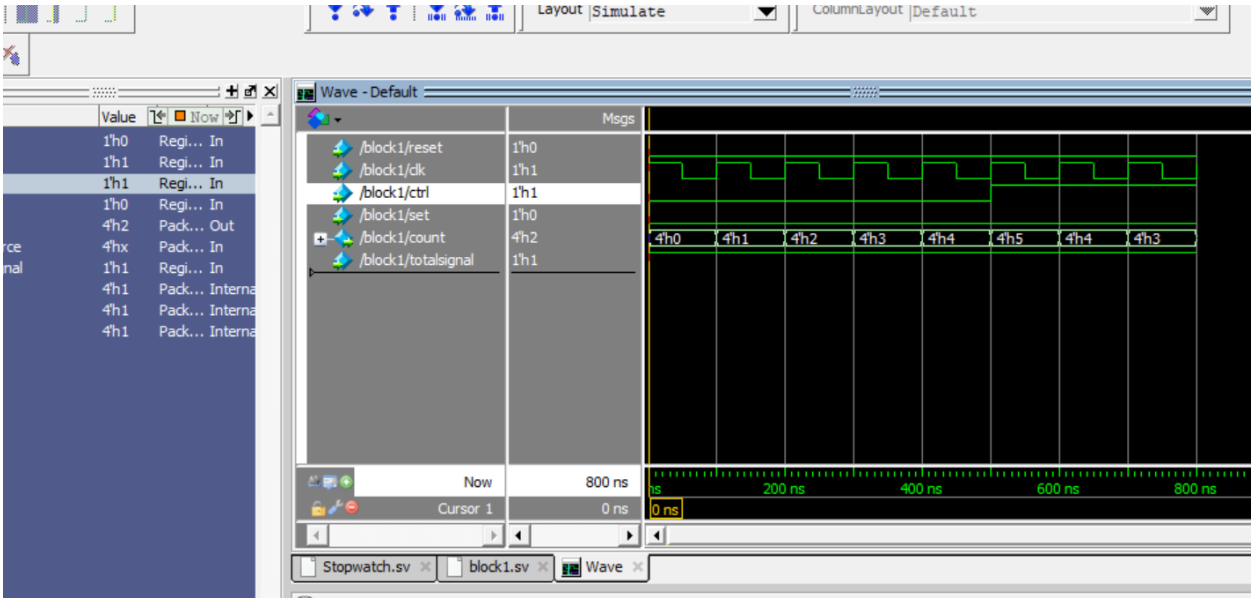
This block takes as inputs:

1) clock “clk” which is supposed to be 1 Hz.
2) reset and this is an option to reset the block and start counting form zero again.
3) ctrl is mainly to select count up, or count down.
4) set is an option if the user wants to enter the value in which the counter starts counting from.
5) outsource is to receive the number that the user entered.
6) totalsignal is the signal that state that the counter should count or stop.
7) sum0, sum1, and sum2 are internal wires to pass the values.

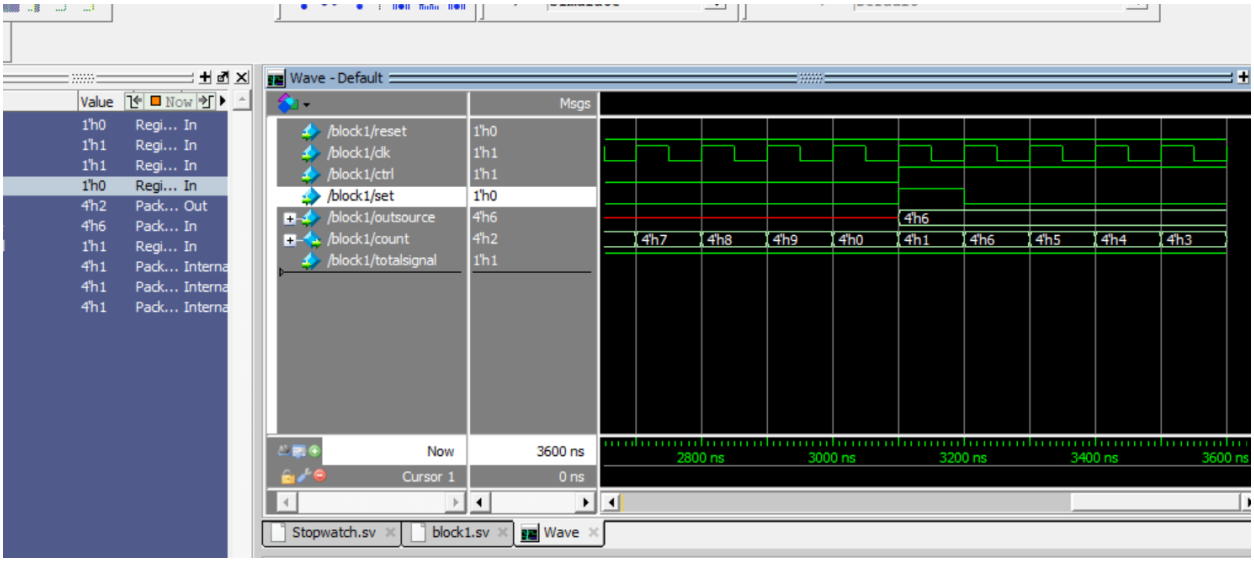
It’s all start with the counter(addsub) takes the value of the count variable, and add, or subtract one to it and put the result on sum0. then sum0 entered a multiplexer MUX1(called “muxcheck”) and to know its importance it is in case of stopping, the counter stops once the user pressed stop and in the end of this part we will make the simulation with this multiplexer and without it. Next, the output of this multiplexer is sum1 that enters another multiplexer MUX1 (called “mux1” in the code) to choose between the sum1, and the entered value if the user wanted to set the stopwatch and give the new value into sum2. After that this operation repeated with each positive edge of the clock (thanks to Regfour) and present the value finally on count wires, then count will add, or subtract one according to the ctrl and so on. we can reset whenever we want and it is a part of Regfour mission. Finally, this block is parameterized to count from zero to the number of the parameter. For example, if the parameter is 9, the block will count from 0 to 9 if the ctrl is zero. In case of, the ctrl is one it will count from 9 to 0.



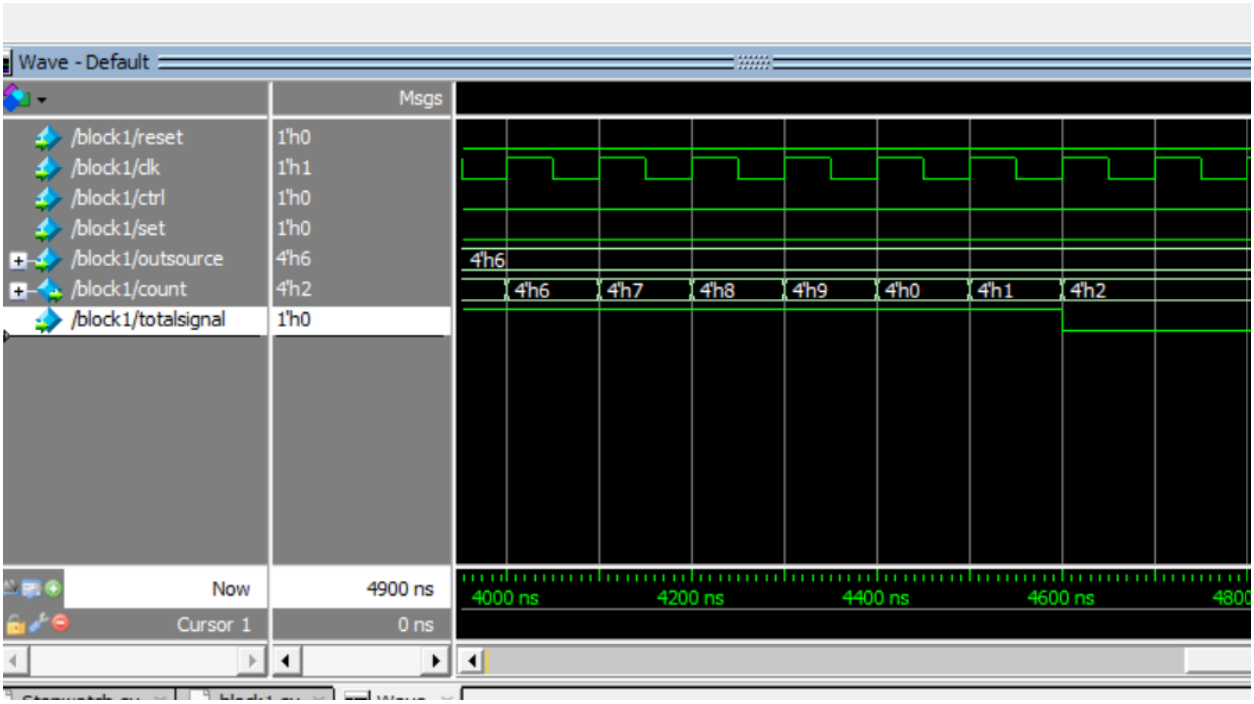
and when we change the mood from counting up to counting down, the result would be.



and to test the set option



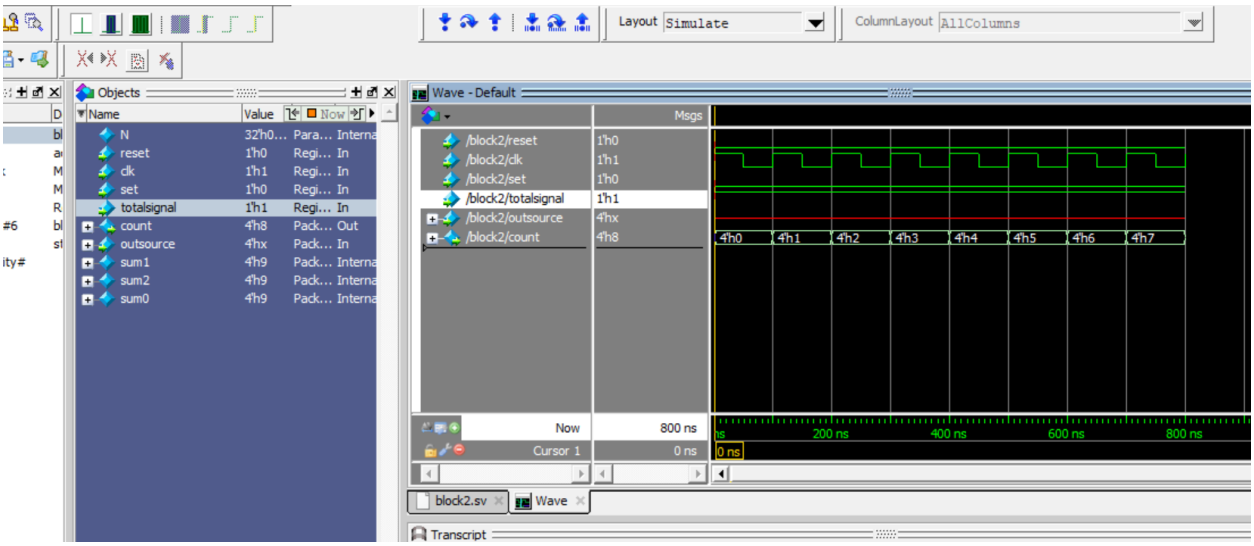
We can check the result with muxcheck in the following simulation just after stopping the counting by changing the value of totalsignal from one to zero.



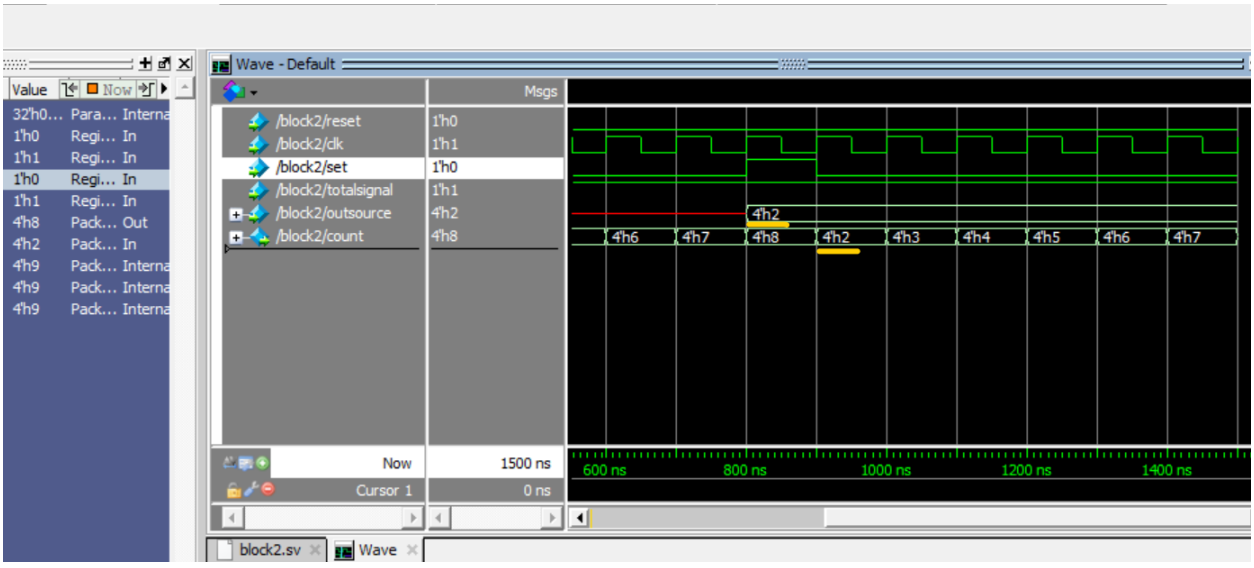
and the next simulation without this multiplexer (muxcheck). the counter stops after a second and it is not wanted.

12) block2 is typical to block1 but for the watch so there is no option of ctrl (which is responsible to count up, or count down). Also, this block is parameterized. This parameter determines that the number that the count will count to. For example, if the parameter is 4 then this count will count from zero to four.

```
C:/Modeltech_pe_edu_10.4a/examples/block2.sv - Default
Ln#
1  module block2# (parameter N = 9)(clk,reset,set,outsource,count,totalsignal);//first block
2  input logic reset,clk,set,totalsignal; // internal for updating after reaching the end.
3  output logic [3:0]count;
4  input logic [3:0]outsource;
5  logic [3:0] sum1,sum2,sum0;
6  assign count = 0;
7  addsub2 #(N) A1(count,sum0,totalsignal);
8  MUX1 #(4) muxcheck(count,sum0,~totalsignal,sum1);
9  MUX1 #(4) mux1(outsource,sum1,set,sum2);
10 Regfour #(4) R1(clk,reset,sum2, count);
11 endmodule
12
```



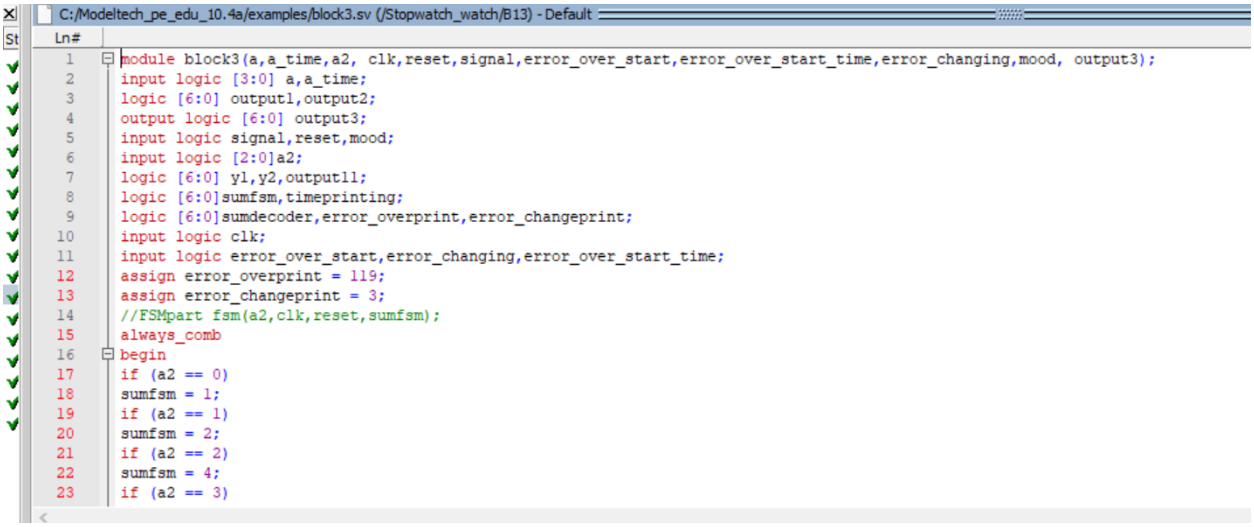
and to test the set option



and to test stopping the program.

now we finished the blocks which is so necessary for counting.

13) block3 is the block which is responsible for printing information on the first seven segment display that represent the seconds in case of stopwatch, and minutes in case of watch.




```
1  module block3(a,a_time,a2, clk,reset,signal,error_over_start,error_over_start_time,error_changing,mood, output3);
2  input logic [3:0] a,a_time;
3  logic [6:0] output1,output2;
4  output logic [6:0] output3;
5  input logic signal,reset,mood;
6  input logic [2:0]a2;
7  logic [6:0] y1,y2,output11;
8  logic [6:0]sumfsm,timeprinting;
9  logic [6:0]sumdecoder,error_overprint,error_changeprint;
10 input logic clk;
11 input logic error_over_start,error_changing,error_over_start_time;
12 assign error_overprint = 119;
13 assign error_changeprint = 3;
14 //FSMpart fsm(a2,clk,reset,sumfsm);
15 always_comb
16 begin
17 if (a2 == 0)
18 sumfsm = 1;
19 if (a2 == 1)
20 sumfsm = 2;
21 if (a2 == 2)
22 sumfsm = 4;
23 if (a2 == 3)
```

```
24    sumfsm = 8;
25    if (a2 == 4)
26    sumfsm = 16;
27    if (a2 == 5)
28    sumfsm = 32;
29    if (a2 == 6)
30    sumfsm = 64;
31  end
32  sevensegment9 D1(a,sumdecoder);
33  sevensegment9 D2(a_time,timeprinting);
34  MUX1 #(7) mux1(sumfsm,sumdecoder,signal,y1);
35  MUX1 #(7) mux2(error_changeprint,y1,error_changing, output1);
36  MUX1 #(7) mux3(error_overprint,output1,error_over_start, output11);
37
38  MUX1 #(7) mux5(error_overprint,timeprinting,error_over_start_time, output2);
39
40  MUX1 #(7) mux6(output2,output11,mood, output3);
41  endmodule
42
43
44
```

This block takes as inputs:

1) a: is the number which is related to the stopwatch to be presented on the seven segments display.
2) a_time: is the number which is related to the watch to be presented on the seven segments display.
3) clk: is the entered clock which is supposed to be 1Hz.
4) a2 is the double of the entered clock to make the periodic motion on the seven segments display.
5) signal: is a signal comes from block5 to ensure that the stopwatch entered in the power saver mode or not.
6) signal_time: is similar to the last signal but detect entering power saver mode of the watch instead of the stopwatch.
7) error_over_start: is a signal to show that the user entered an invalid value to count down from in the stopwatch mood. For example, if the user entered numbers higher than 59:59, this signal will be equal to one. Else it would be equal to zero.
8) error_over_start_time: is a signal to show that the user initialized the clock mood with an invalid value. For example, if the user entered numbers higher than 23:59, this signal will equal to one. Else it would be equal to zero.
9) error changing: this error happens when the user changing from counting up to counting down or from counting down to counting up without stopping the program.
10) mood: is to define the mood (watch- stopwatch).
11) output3: is the final result that would be connected to the seven segment display.

This block contains small blocks:

1) FSMpart: to make the periodic motion on the seven segment display in case of the program (watch, or stopwatch) entered in the power saver mood.
2) sevensegment9 (D1): is the **decoder** to represent the number reached on the counter (in the stopwatch).

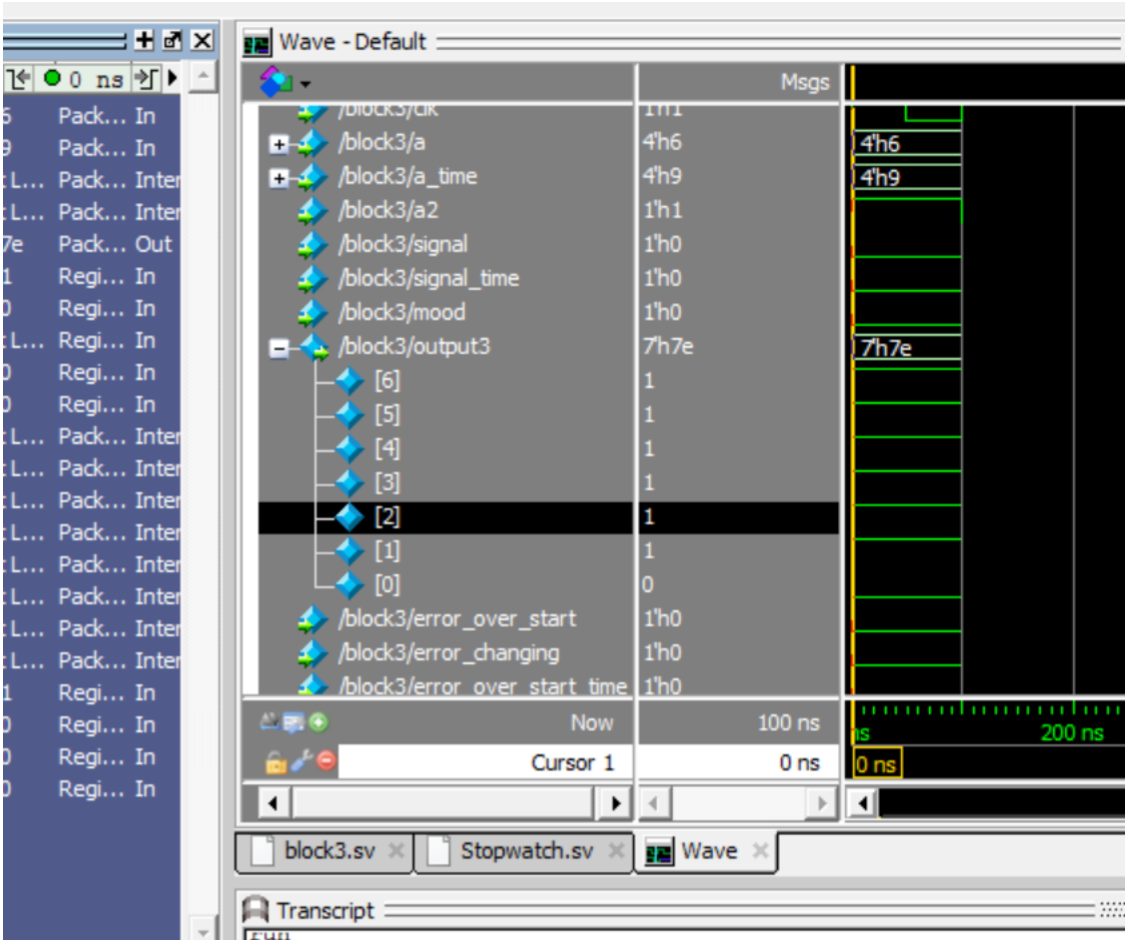
3) sevensegment9 (D2): is the **decoder** to represent the number reached on the counter (in the clock mood).
4) MUX1 (mux1): is to select between printing the number reached by the counter of the stopwatch, or the periodic motion on the seven segment display. After that putting the output on the "**y1**".
5) MUX1 (mux2): is to select between printing the number reached by mux1 "**y1**", or what to be printed on the seven segment display in case of error changing mood without stopping the program in this block is "**1**". After that putting the output on the "**output1**".
6) MUX1 (mux3): is to select between printing the number reached by mux2 "**output1**", or what to be printed on the seven segment display in case of error surpassing the maximum value (59:59) in this block is "**9**". After that putting the output on the "**output11**".
7) MUX1 (mux4): is to select between printing the number reached by the counter of the watch, or the periodic motion on the seven segment display. After that putting the output on the "**y2**".
8) MUX1 (mux5): is to select between printing the number reached by mux1 "**y2**", or what to be printed on the seven segment display in case of error changing mood without stopping the program in this block is "**1**". After that putting the output on the "**output2**".
9) MUX1 (mux6): is to select printing the data on "**output11**", if mood was equal to 0 or the data on "**output2**", if mood was equal to 1.

Finally, what has a direct access on the seven segment display is mux6.

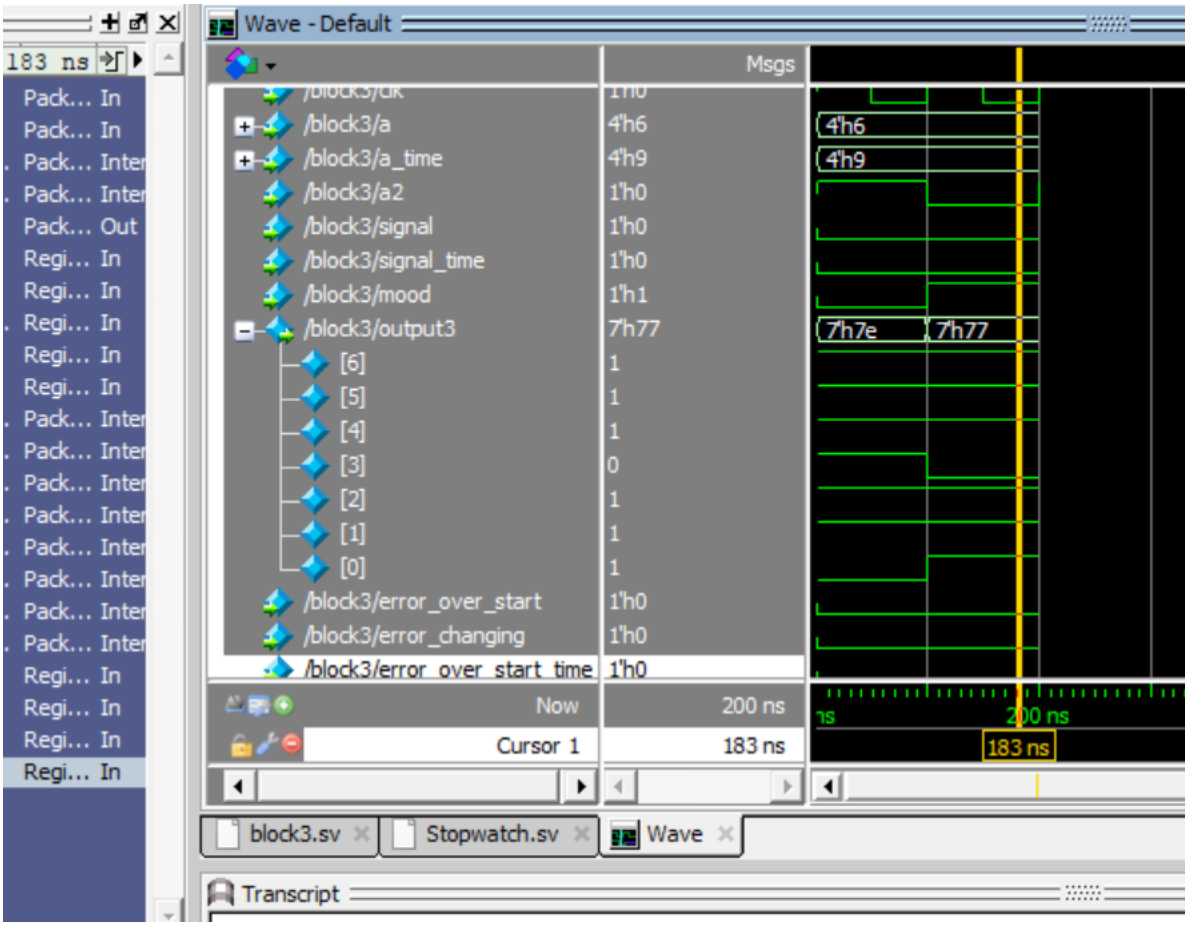
The reason of assigning error_overprint to be equal to 119 is that would print 9 on the seven segment display. Because all leds will be equal to one except y_3 will equal to zero, so it would be equal to $((1*2^0) + (1*2^1) + (1*2^2) + (0*2^3) + (1*2^4) + (1*2^5) + (1*2^6)) = 119$.

Also, for the same reason error_changeprint equal 3. As it would print 1 on the seven segment display.

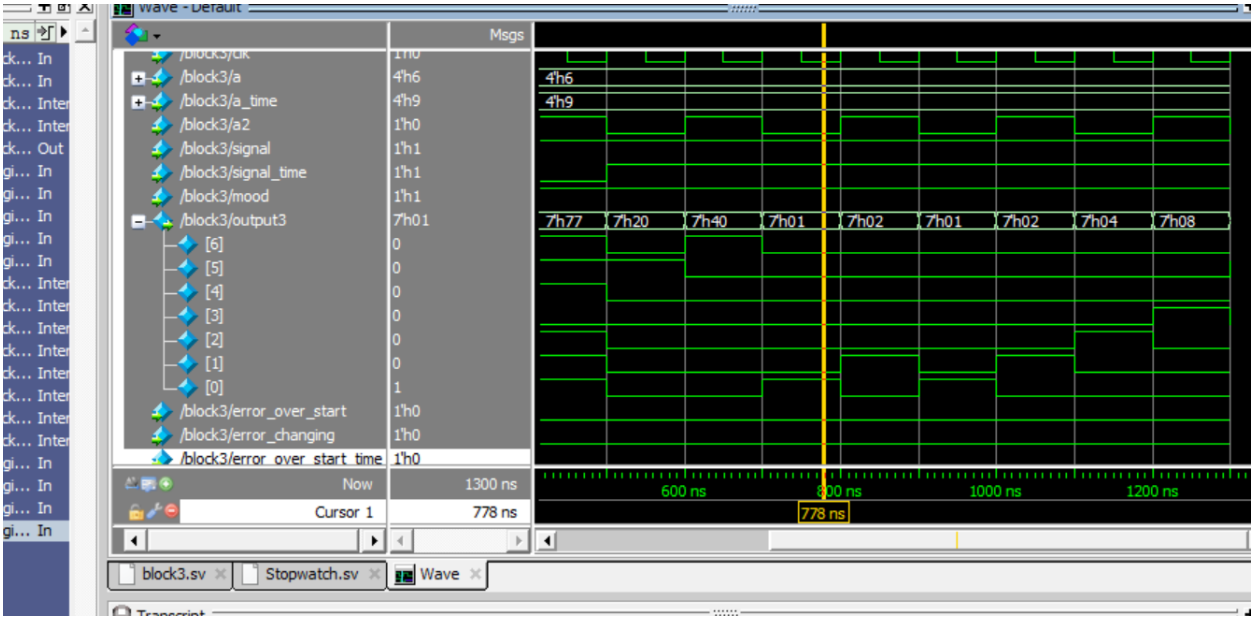
to test printing the number of the counter on the screen, let's initialize the number ("a" in the code) to be equal to 6 and we can see that all wires are equal to one except y_0, so it would print six in the screen.



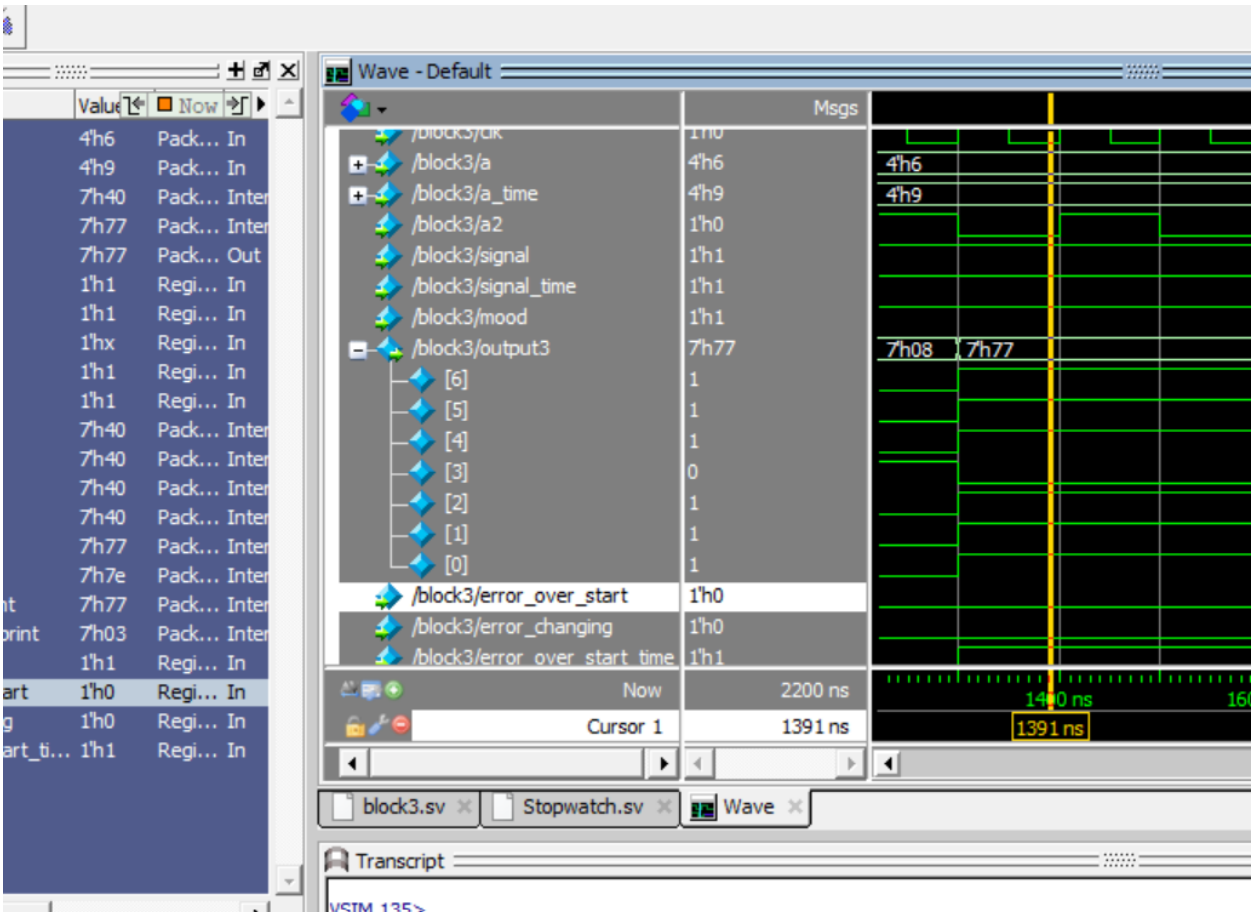
and now changing mood to be equal to 1 and we would see that the output3 will print the number of (a_time which is 9) on the seven segment display. So, all wires will be equal to one except y_3 will be equal to zero.



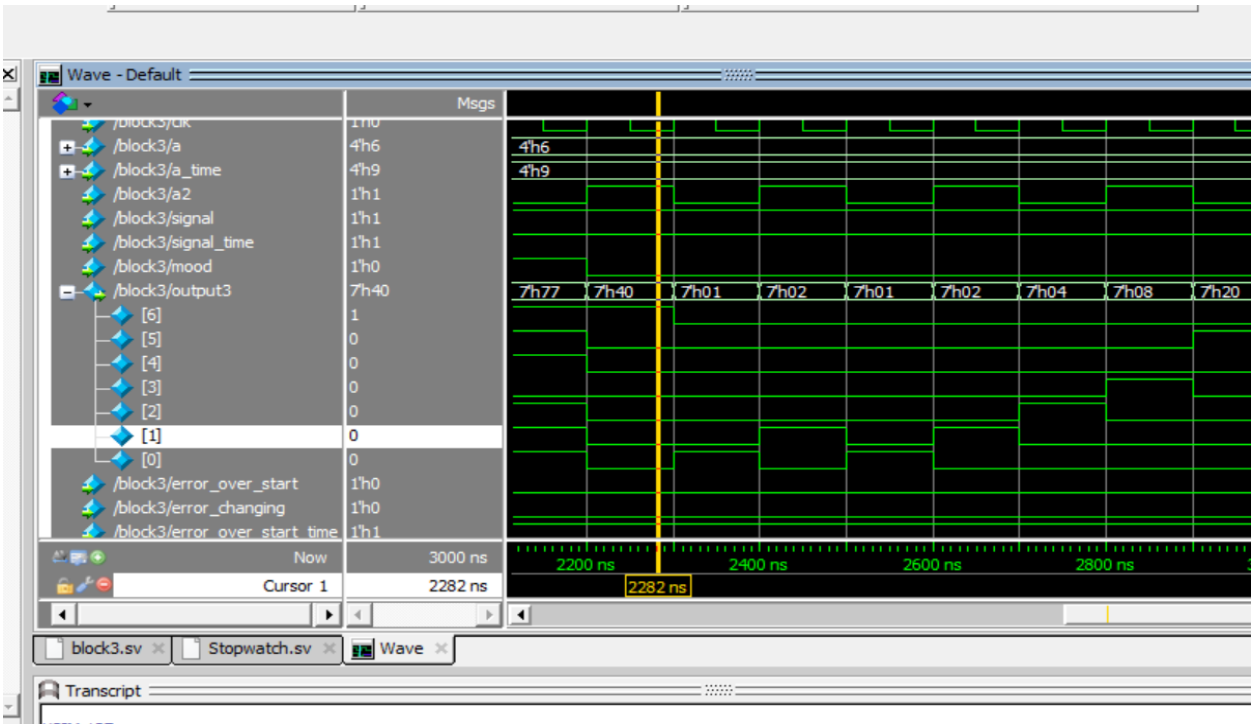
and for testing FSMpart in the watch part



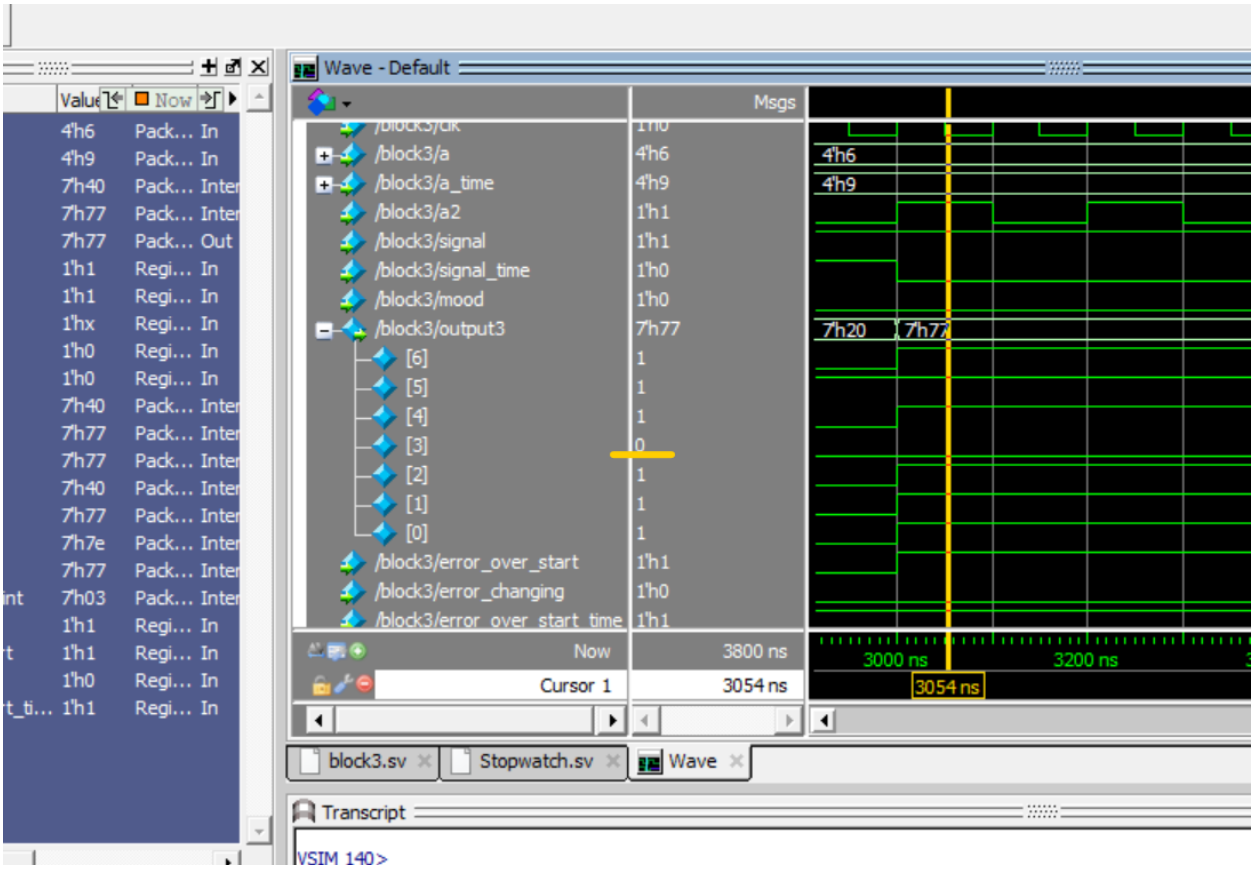
to test error_over_start_time. It would print 9 as reported above. So, it would all wires would equal to one except y_3 will be equal to zero to print 9. (watch part)



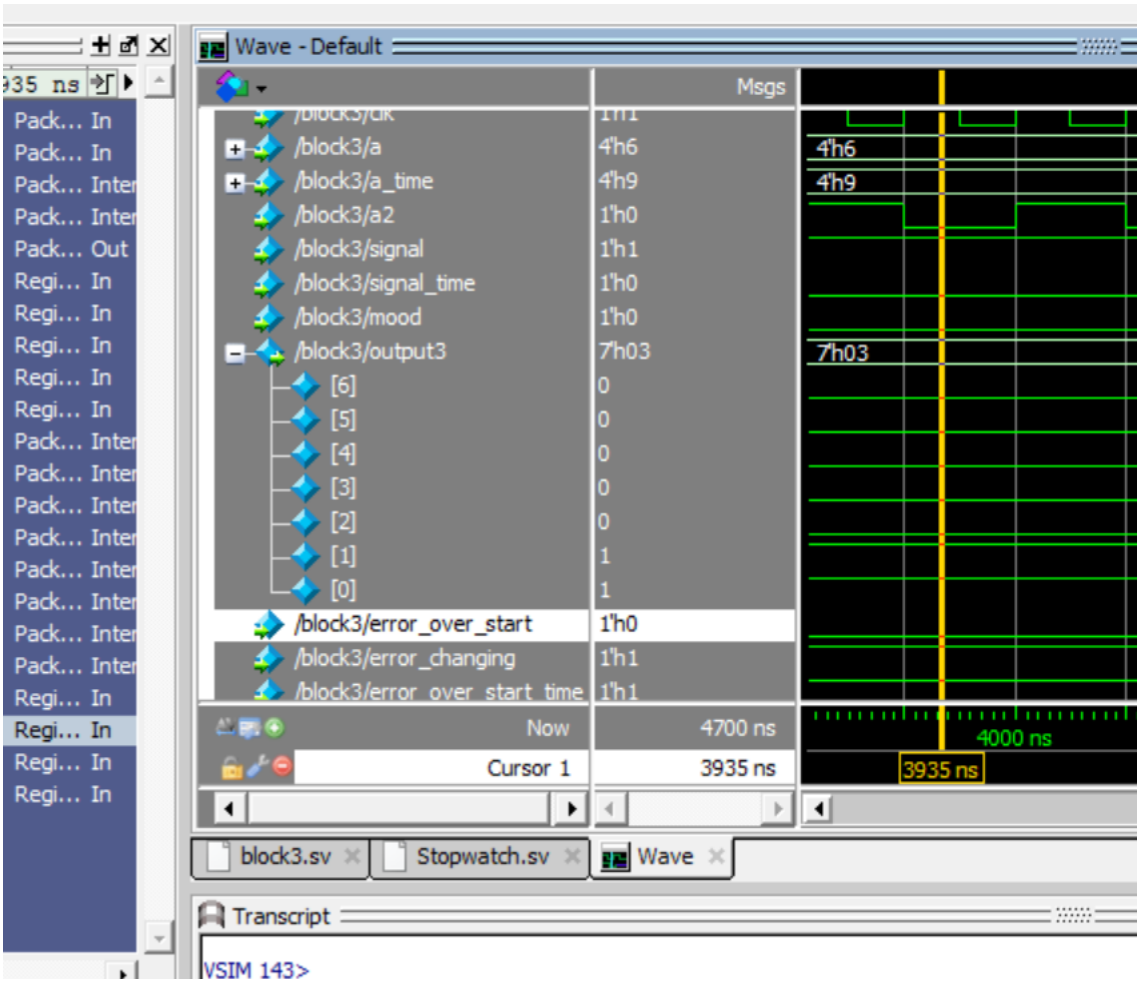
to test the FSMpart in stopwatch by changing the mood to be equal to zero.



to test the error_over_start which is on the stopwatch part, it would print 9 also



Finally, to test the error_changing, it would print one on the seven segment display so only y_0 and y_1 will be equal to one.



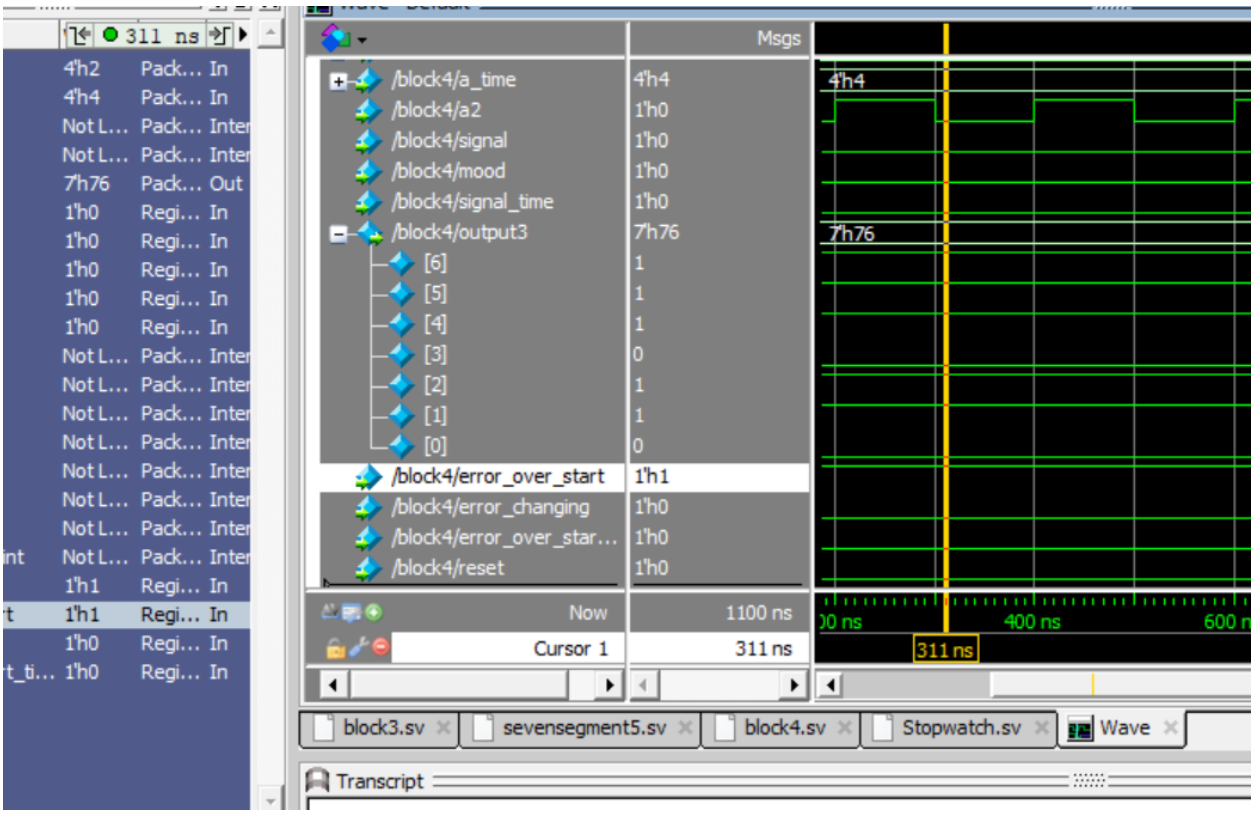
14) block4: is typical to block3 except what to be presented in case there is an error, and the decoder used in it is to present from zero to five.

```
module block4(a,a_time,a2, clk,reset,signal,error_over_start,error_over_start_time,error_changing,mood, output3);
input logic [3:0] a,a_time;
logic [6:0] output1,output2;
output logic [6:0] output3;
input logic signal,reset,mood;
input logic [2:0]a2;
logic [6:0] y1,y2,output11;
logic [6:0]sumfsm,timeprinting;
logic [6:0]sumdecoder,error_overprint,error_changeprint;
input logic clk;
input logic error_over_start,error_changing,error_over_start_time;
assign error_overprint = 118;
assign error_changeprint = 111;
//FSMpart fsm(a2,clk,reset,sumfsm);
always_comb
begin
if (a2 == 0)
sumfsm = 1;
if (a2 == 1)
sumfsm = 2;
if (a2 == 2)
sumfsm = 4;
if (a2 == 3)
```

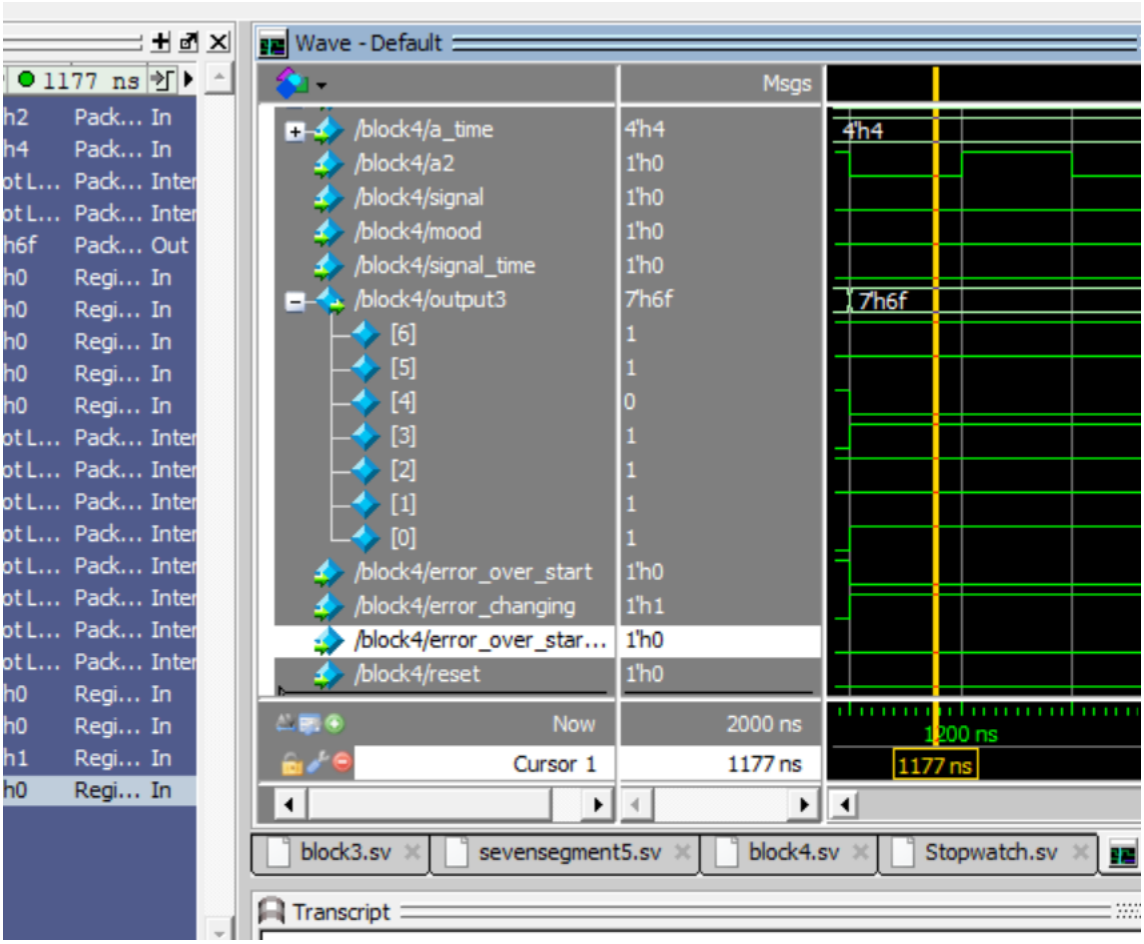
```
24      sumfsm = 8;
25      if (a2 == 4)
26      sumfsm = 16;
27      if (a2 == 5)
28      sumfsm = 32;
29      if (a2 == 6)
30      sumfsm = 64;
31      end
32      sevensegment9 D1(a,sumdecoder);
33      sevensegment9 D2(a_time,timeprinting);
34      MUX1 #(7) mux1(sumfsm,sumdecoder,signal,y1);
35      MUX1 #(7) mux2(error_changeprint,y1,error_changing, output1);
36      MUX1 #(7) mux3(error_overprint,output1,error_over_start, output11);
37
38      MUX1 #(7) mux5(error_overprint,timeprinting,error_over_start_time, output2);
39
40      MUX1 #(7) mux6(output2,output11,mood, output3);
41      endmodule
42
```

error_overprint is equal to 118 to print 5 on the seven segment display, and error_changeprint is equal to 111 to print 0 on the seven segment display.

in case of error the user entered a number which is not valid, it would print 5 on the seven segment display. So, y_0 and y_4 will be equal to zero. Meanwhile, others will be equal to one.



and to test error_changing, it would print 0. all wires will be equal to one except y_4 will be equal to zero.

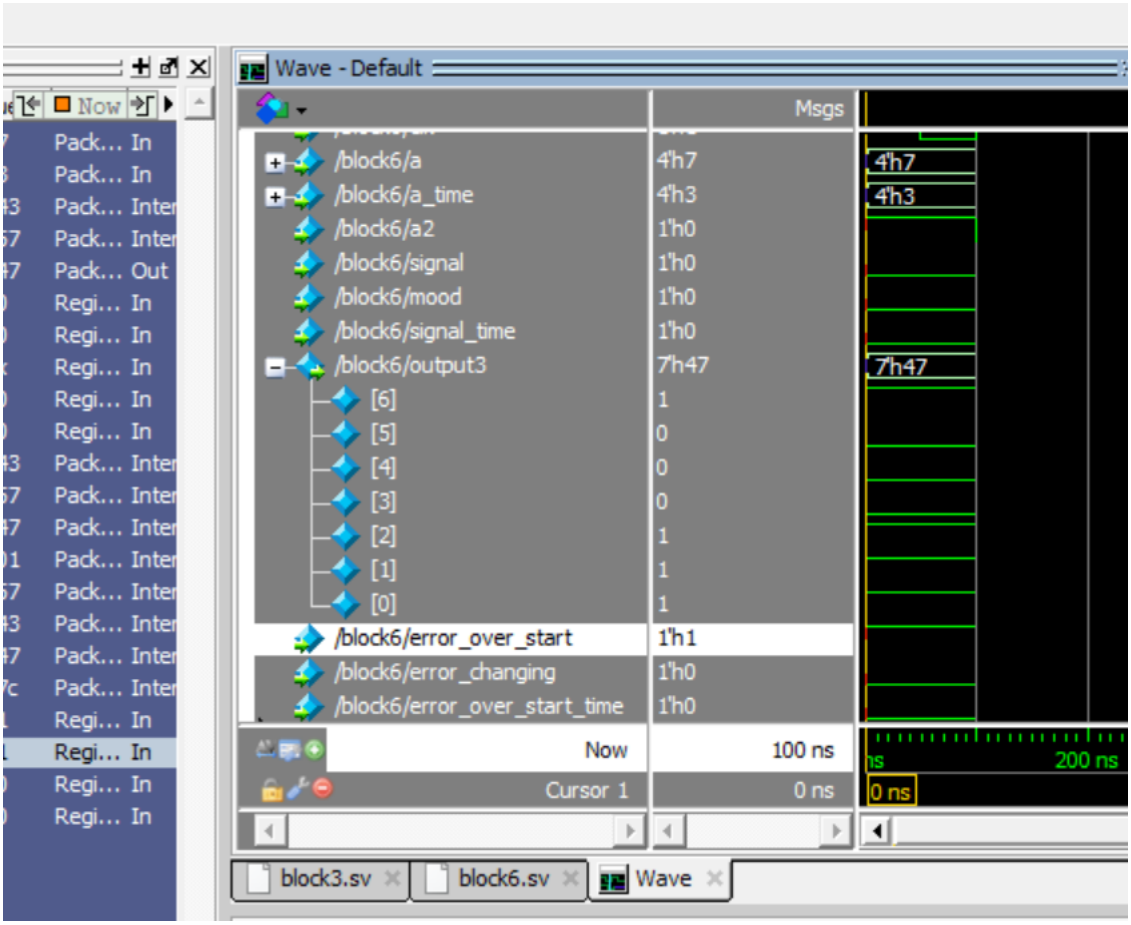


15) block6: this block is typical to block3, the only difference is what to present in case there is an error. For example, in case of error of entering invalid number, the seven segment display will print "**]**". and for the error of changing mode while counting, it would print "E" on the seven segment display.

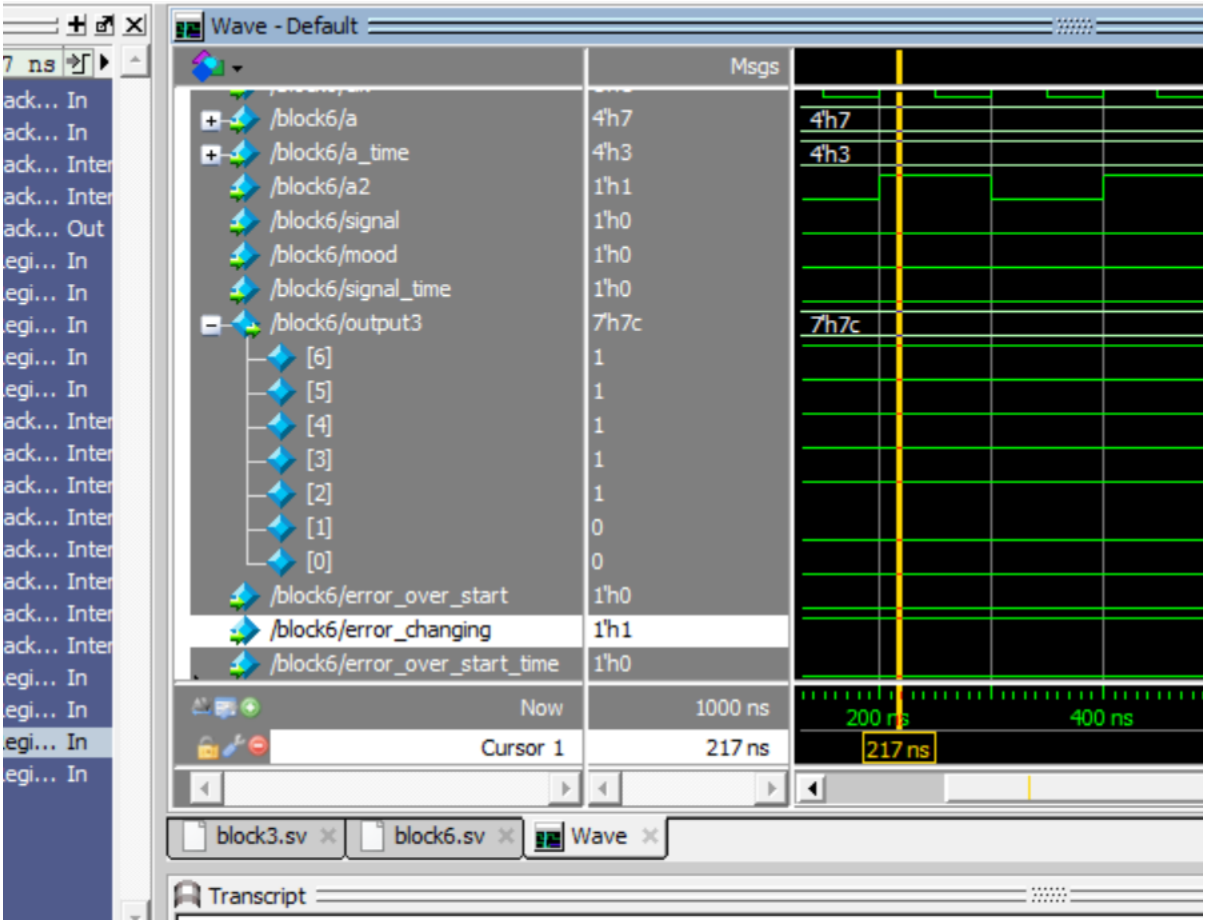
```
Ln#
 1  module block6(a,a_time,a2, clk,reset,signal,error_over_start,error_over_start_time,error_changing,mood, output3);
 2  input logic [3:0] a,a_time;
 3  logic [6:0] output1,output2;
 4  output logic [6:0] output3;
 5  input logic signal,reset,mood;
 6  input logic [2:0]a2;
 7  logic [6:0] y1,y2,output11;
 8  logic [6:0]sumfsm,timeprinting;
 9  logic [6:0]sumdecoder,error_overprint,error_changeprint;
10  input logic clk;
11  input logic error_over_start,error_changing,error_over_start_time;
12  assign error_overprint = 119;
13  assign error_changeprint = 3;
14  //FSMpart fsm(a2,clk,reset,sumfsm);
15  always_comb
16  begin
17  if (a2 == 0)
18  sumfsm = 1;
19  if (a2 == 1)
20  sumfsm = 2;
21  if (a2 == 2)
22  sumfsm = 4;
23  if (a2 == 3)
```

```
24    sumfsm = 8;
25    if (a2 == 4)
26    sumfsm = 16;
27    if (a2 == 5)
28    sumfsm = 32;
29    if (a2 == 6)
30    sumfsm = 64;
31  end
32  sevensegment9 D1(a,sumdecoder);
33  sevensegment9 D2(a_time,timeprinting);
34  MUX1 #(7) mux1(sumfsm,sumdecoder,signal,y1);
35  MUX1 #(7) mux2(error_changeprint,y1,error_changing, output1);
36  MUX1 #(7) mux3(error_overprint,output1,error_over_start, output11);
37
38  MUX1 #(7) mux5(error_overprint,timeprinting,error_over_start_time, output2);
39
40  MUX1 #(7) mux6(output2,output11,mood, output3);
41  endmodule
42
```

in case there is an error of entering a number which is not valid, the seven segment display will print "**]**". y_0, y_1, y_2, and y_6 will equal to one. Meanwhile, others will equal to zero.



in case there is an error of changing mode while counting, the seven segment display will print "**E**". y_2, y_3, y_4, y_5 and y_6 will equal to one. Meanwhile, others will equal to zero.

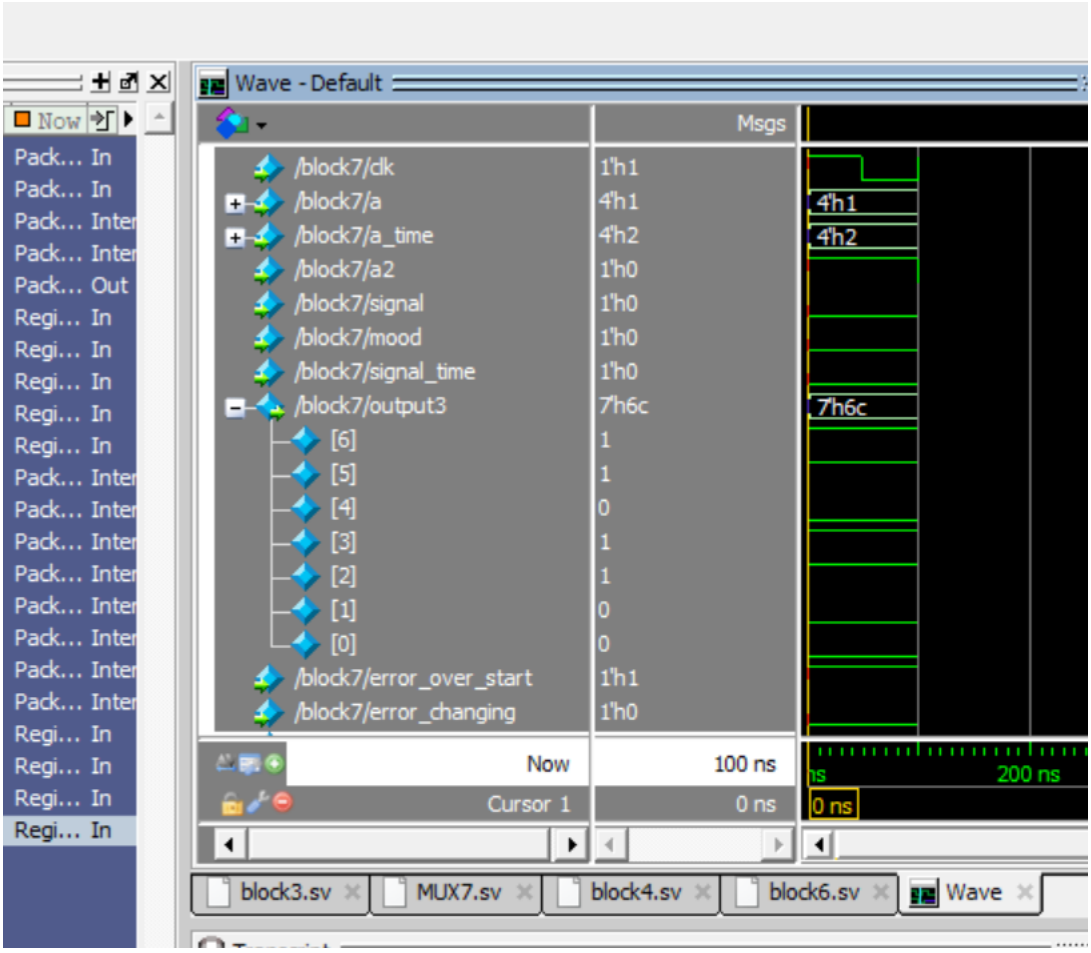


16) block7: this block is typical to block3, the only difference is what to present in case there is an error. For example, in case of error of entering invalid number, the seven segment display will print "**[**". and for the error of changing mode while counting, it would print "**-**" on the seven segment display.

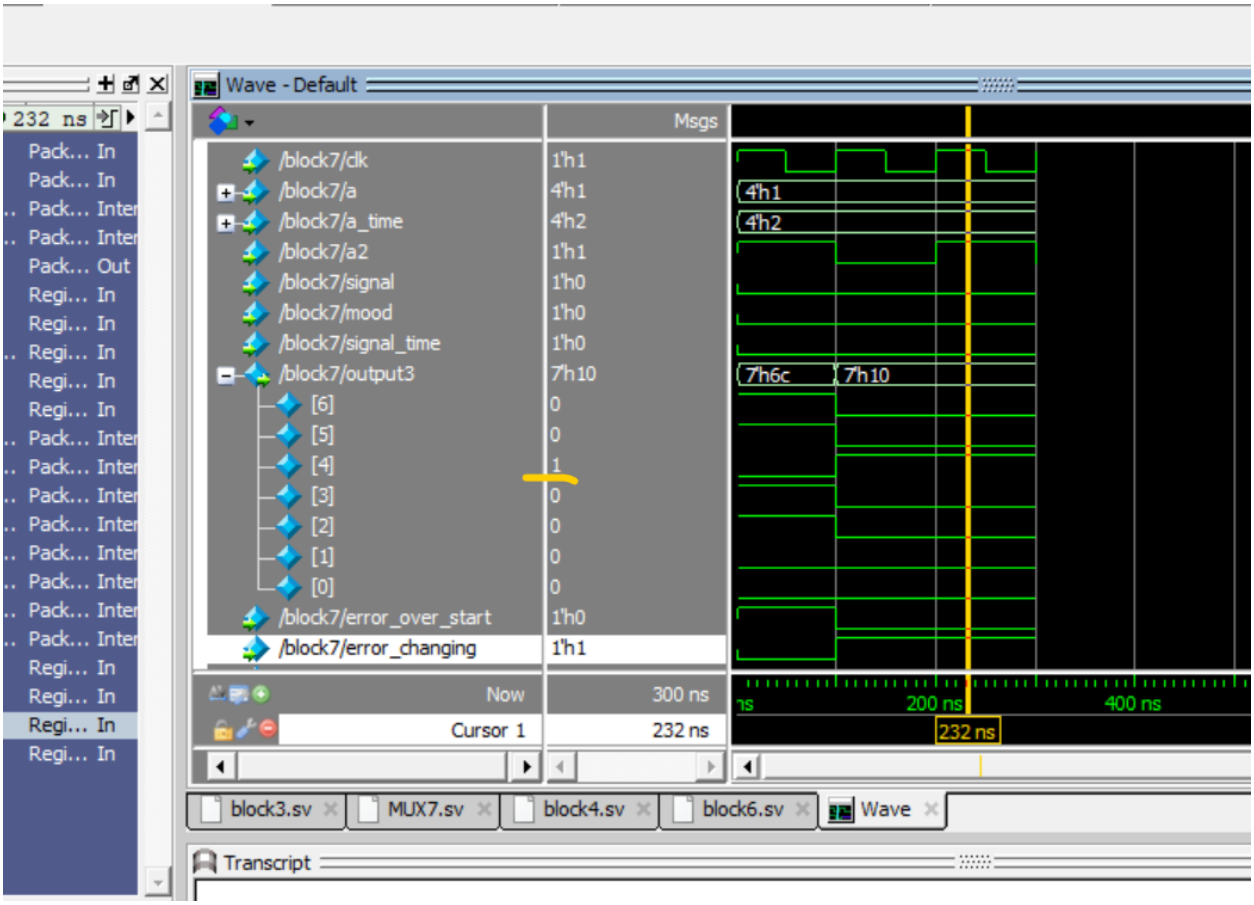
```
n#
3   module block7(a,a_time,a2, clk,reset,signal,error_over_start,error_over_start_time,error_changing,mood, output3);
4   input logic [3:0] a,a_time;
5   logic [6:0] output1,output2;
6   output logic [6:0] output3;
7   input logic signal,reset,mood;
8   input logic [2:0]a2;
9   logic [6:0] y1,y2,output11;
0   logic [6:0]sumfsm,timeprinting;
1   logic [6:0]sumdecoder,error_overprint,error_changeprint;
2   input logic clk;
3   input logic error_over_start,error_changing,error_over_start_time;
4   assign error_overprint = 108;
5   assign error_changeprint = 16;
6   //FSMpart fsm(a2,clk,reset,sumfsm);
7   always_comb
8   begin
9   if (a2 == 0)
0   sumfsm = 1;
1   if (a2 == 1)
2   sumfsm = 2;
3   if (a2 == 2)
4   sumfsm = 4;
5   if (a2 == 3)
```

```
63    if (a2 == 2)
64    sumfsm = 4;
65    if (a2 == 3)
66    sumfsm = 8;
67    if (a2 == 4)
68    sumfsm = 16;
69    if (a2 == 5)
70    sumfsm = 32;
71    if (a2 == 6)
72    sumfsm = 64;
73    end
74    sevensegment9 D1(a,sumdecoder);
75    sevensegment9 D2(a_time,timeprinting);
76    MUX1 #(7) mux1(sumfsm,sumdecoder,signal,y1);
77    MUX1 #(7) mux2(error_changeprint,y1,error_changing, output1);
78    MUX1 #(7) mux3(error_overprint,output1,error_over_start, output11);
79
80    MUX1 #(7) mux5(error_overprint,timeprinting,error_over_start_time, output2);
81
82    MUX1 #(7) mux6(output2,output11,mood, output3);
83    endmodule
84
85
```

in case there is an error of entering a number which is not valid, the seven segment display will print "**[**". y_1, y_2, y_3, y_5, and y_6 will equal to one. Meanwhile, others will equal to zero.



in case there is an error of changing the mood while counting, the seven segment display will print "-". y_4 will equal to one. Meanwhile, others will equal to zero.

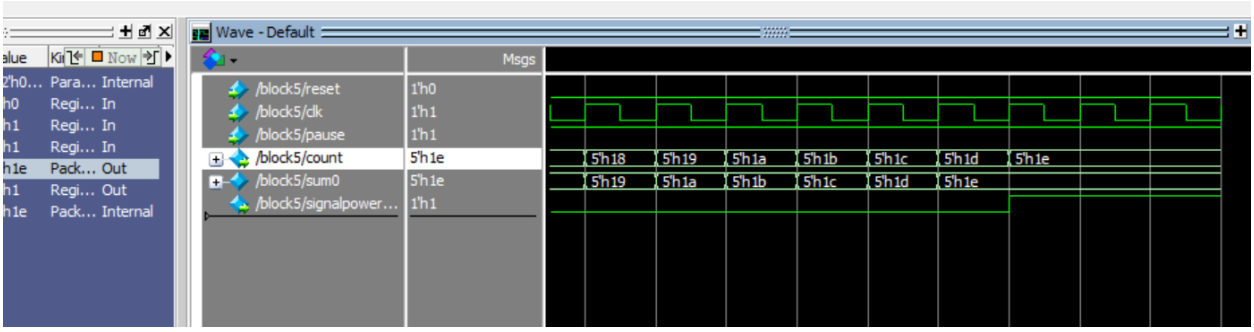


17) block5: is to count from zero to thirty seconds. In case the user doesn't give an order for the stopwatch, or the watch for thirty seconds.

```
Ln#
 1  module block5# (parameter N = 5)(clk,reset,count,pause,signalpowersaver);//first block
 2  input logic reset,clk,pause;
 3  output logic [N-1:0]count;
 4  output logic signalpowersaver;
 5  logic [N-1:0] sum0;
 6  assign count = 0;
 7  addsub5 A1(count,sum0,pause);
 8  Regfour #(5) R1(clk,reset,sum0,count);
 9  always_comb
10  begin
11  if((~count[0]) & (count[1]) & (count[2]) & (count[3]) & (count[4]))
12  signalpowersaver = 1;
13  else
14  signalpowersaver = 0;
15  end
16  endmodule
```

addsub5 count from zero to thirty, and this operation happened each a complete second using Regfour. and once the counter reach thirty the wire signalpowersaver will be equal to one. Otherwise, it would be equal to zero.

and here we can see that the block count till "**1e**" which in decimal system is equal to 30, then singalpowersaver will be equal to one.

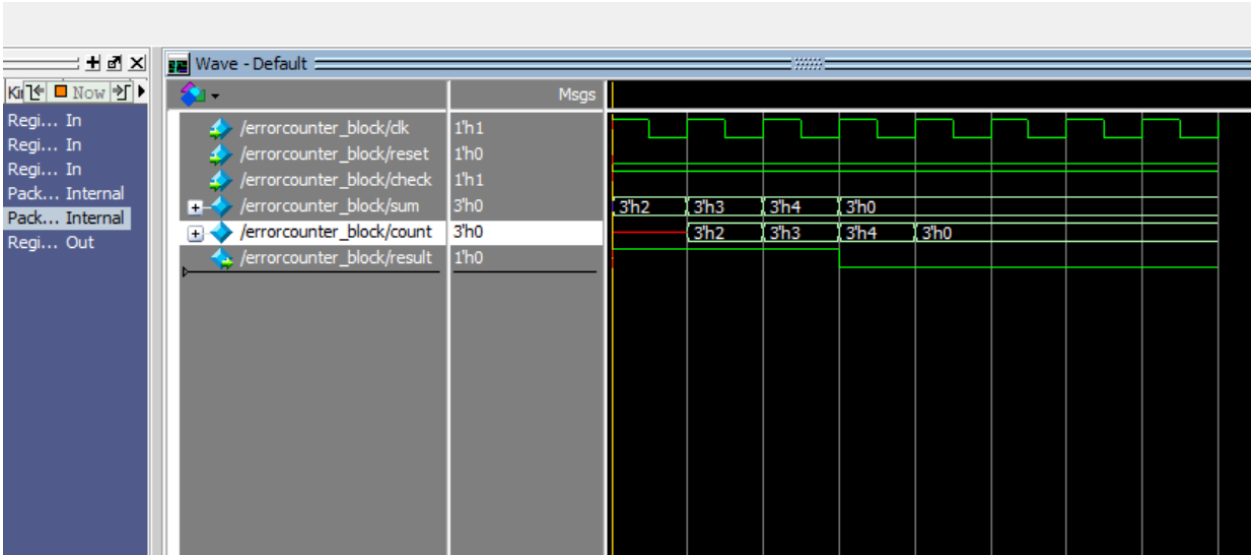


Finally, to manage errors and make it like full options, the error of setting invalid number should print on the seven segment display an error message three times. So, I have to stop the counter for three seconds for not losing data while printing. but why is supposed not printing while stopping the program and to answer this question is that it is mainly made just in case the user entered the run the program that may count on the previous mood. For example, if the user counts up, then he stopped the program, next set in valid number, finally press unstop without this program it would print the error message, and miss the data that counter got after stopping. I hope this becomes obvious now.

18) errorcounter_block: this block is working with another block called errorcounter1 that count 1,2,3 after each call and this is the counter that stops the counter that count seconds “block1” for three times and print the error message on the screen three times. result which is in errorcounter_block is the signal that enters to block3, 4, 6, and 7. to print the error message. Also, result has the authority to stop the program, if it is counting.

```
module errorcounter1(count,sum,result);
input logic [2:0] count;
output logic [2:0] sum;
output logic result;
always_comb
begin
case(count)
1: sum = 2;
2: sum = 3;
3: sum = 4;
4: sum = 0;
endcase
if ( (sum == 2) | (sum == 3) | (sum == 4) )
result = 1;
else
result = 0;
end
endmodule
```

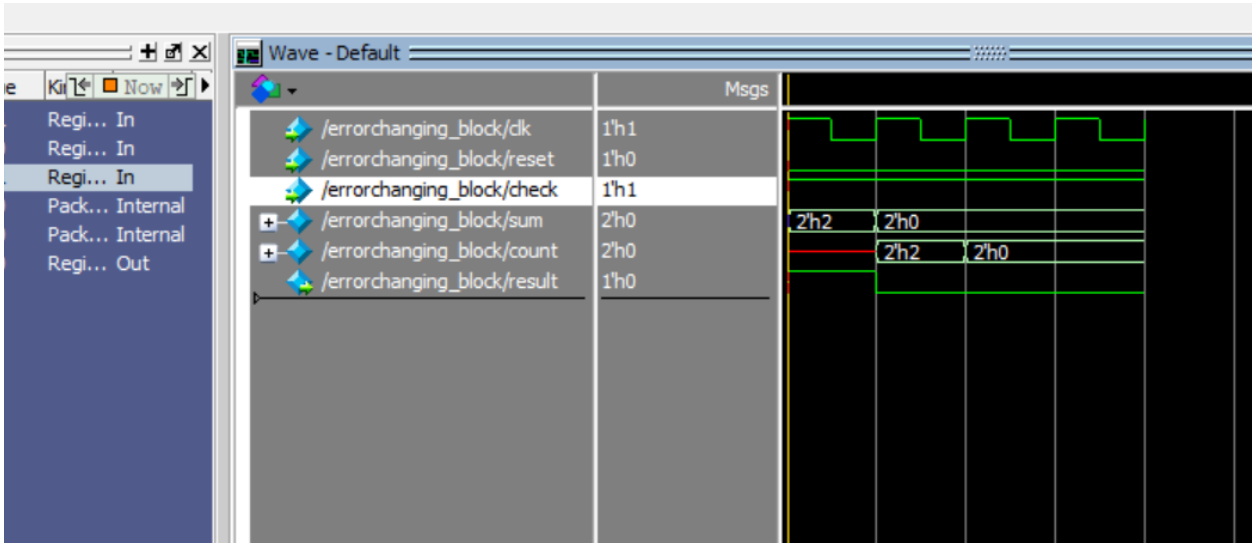
```
module errorcounter_block(clk,reset,check,result);
input logic clk,reset,check;
logic [2:0]sum;
logic [2:0]count;
output logic result;
always_comb
begin
if (check) // It is always equal to one
count = 1;
end
errorcounter1 newclk11(count,sum,result);
Regfour #(4) R1(clk,reset,sum, count);
endmodule
```



19) errorchanging_block: this block is typical to errorcounter_block but counting just one (Not 1, 2, 3)

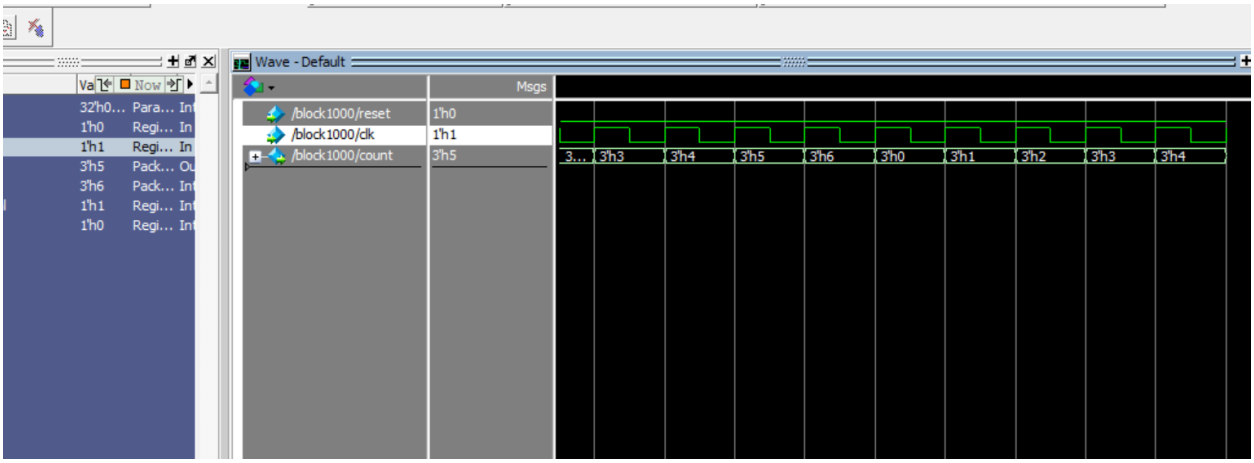
```
41  module errorcounter2(count,sum,result);
42  input logic [1:0] count;
43  output logic [1:0] sum;
44  output logic result;
45  always_comb
46  begin
47  case(count)
48  1: sum = 2;
49  2: sum = 0;
50  endcase
51  if ( (sum == 2) )
52  result = 1;
53  else
54  result = 0;
55  end
56  endmodule
```

```
module errorchanging_block(clk,reset,check,result);
input logic clk,reset,check;
logic [1:0]sum;
logic [1:0]count;
output logic result;
always_comb
begin
if (check) // It is always equal to one
count = 1;
end
errorcounter2 newclk11(count,sum,result);

Regfour #(4) R1(clk,reset,sum, count);
endmodule
```



20) block1000: counts from 0 to 6 to make value to the fsm part in block3, 4, 6, and 7. to make the periodic motion. (this part is made recently to make the periodic motion)

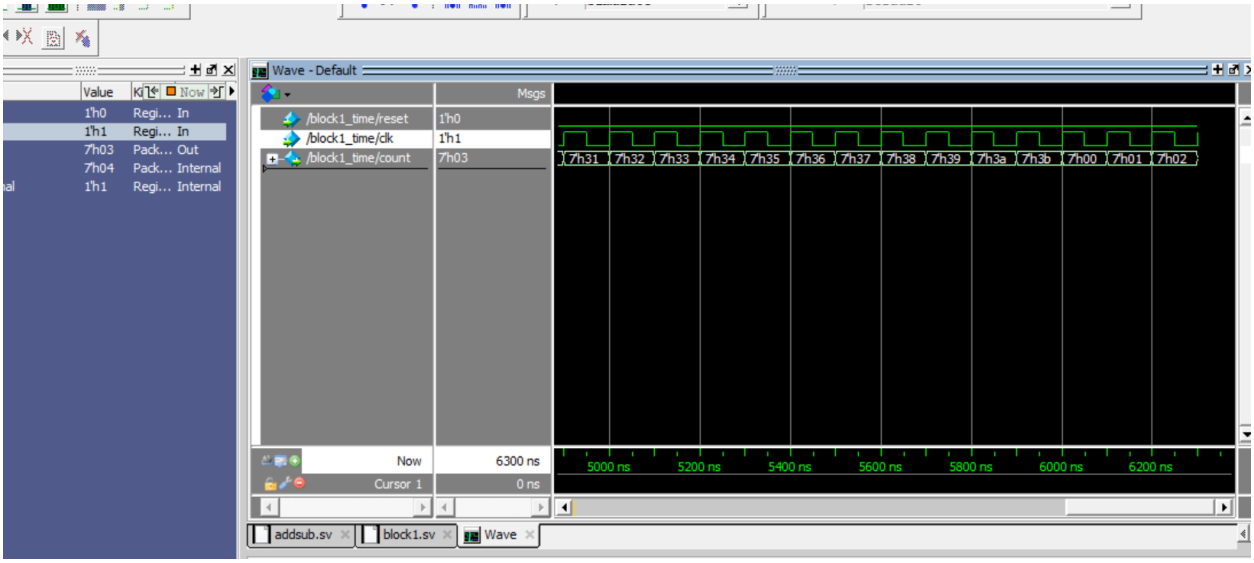
```
14
15  module block1000# (parameter N = 6)(clk,reset,count);//first block
16  input logic reset,clk; // internal for updating after reaching the end.
17  output logic [2:0]count;
18  logic [2:0] sum0;
19  logic totalsignal,ctrl;
20  assign count = 0;
21  assign ctrl = 0;
22  assign totalsignal = 1;
23  addsub #(N) A1(count,sum0,ctrl,totalsignal);
24  Regfour #(4) R1(clk,reset,sum0, count);
25  endmodule
```



21) block1_time: the last block that count seconds (59 seconds) for the clock (**bonus objective one**).

```
module addsub59 #(parameter N = 59)(count,sum,signal);
output logic [6:0]sum;
input logic signal;
input logic [6:0]count;
always_comb
begin
if (signal)
begin
if (~(count == N))
sum = count + 1;
if (count == N)
sum = 0;
end
end
endmodule
```

```
module block1_time(clk,reset,count);//first block
input logic reset,clk; // internal for updating after reaching the end.
output logic [6:0]count;
logic [6:0] sum0;
logic totalsignal;
assign count = 0;SS
assign totalsignal = 1;
addsub59 #(59) A1(count,sum0,totalsignal);
Regfour #(7) R1(clk,reset,sum0, count);
endmodule
```



Now hopefully, we can derive the whole code, and simulate it.

The code for the whole project:

```
Ln#
1  module Stopwatch_watch(reset0,clk,ctrl0,set,pause,mood,outsource1,outsource2,outsource3,outsource4,display0,display1,display2,display3);//
2  input logic reset0,clk,ctrl0,set,pause,mood; // mood here to choose between time or stopwatch
3  output logic [6:0] display0,display1,display2,display3;
4  input logic [3:0]outsource1,outsource2,outsource3,outsource4;
5  logic [3:0]outsource11,outsource12,outsource13,outsource14;
6  logic [3:0]outsource21,outsource22,outsource23,outsource24;
7  logic fsmchange,signalpowersaver,check,ctrl;
8  logic [3:0]count0,count1,count2,count3;
9  logic [3:0]count0_time,count1_time,count2_time,count3_time;
10 logic [4:0]count4;
11 logic enter0,enter1,enter2,enter3;
12 logic enterfin0,enterfin1,enterfin2,enterfin3;
13 logic enterfin0_time,enterfin1_time,enterfin2_time,enterfin3_time;
14 logic reset,reset1,reset2,set_time;
15 logic error_surpass,sel,pause1,reseting,stopreseting;
16 logic error_surpass_printing, error2,errorchanging_printing;
17 logic error_surpass_time,error_surpass_printing_time,notzero;
18 logic resetblock51,resetingblock51;
19
```

This is the first part in the code of the whole project, and it is mainly defining the all wires used in the program, so let's define the inputs and outputs of the project.

1) reset0 (input): is to reset stopwatch, or watch according to the mood.
2) clk (input): is the clock of the Regfour will count on (1Hz).
3) ctrl0 (input): is to define counting up, or counting down in case of stopwatch.
4) set (input): is to set in case of watch, or in case of stopwatch counting down. This can be defined according to mood.
5) pause (input): is to pause watch, or stopwatch according to mood.
6) mood (input): is to define the mood watch, or stopwatch (1-watch / 0-stopwatch).
7) outsource1 (input): it would receive the value of setting of the first seven segment. For example, in case the user wants to set the program at 12:34. outsource1 will receive 4. And outsource2 will receive 3. outsource3 will receive 2, and finally outsource4 will receive 1.

8) display0: is the first seven segment display (the first right one).
9) display1: is the second seven segment display (the second right one).
10) display2: is the third seven segment display (the third right one).
11) display3: is the forth seven segment display (the first left one).

```
19
20  assign check = 1;
21  assign reset = 0;
22  assign pause = 1;
23  assign set = 0;
24  assign mood = 0;
25  assign ctrl0 = 0;
26  assign reset0 = 0;
27
28  assign outsource1 = 0;
29  assign outsource2 = 0;
30  assign outsource3 = 0;
31  assign outsource4 = 0;
```

this piece of code (starts from 20: 31) is to initialize the whole program at first. As it is the default values for the program. When the user run the program, the reset should be equal to zero, the program is stopped at first till the user press pause again to change it from 1 to 0, set, mood, ctrl0, are all equal to zero. And the outsources that takes the number from the user is initialized to be zero.

```
always@(posedge reset1)
begin
pause1 = 1;
end
always@( negedge pause, posedge pause)
begin
if ( (pause1 == 0) & (~(mood)) )
pause1 = 1;
if ( (pause1 == 1) & (~(mood)) )
pause1 = 0;
else
begin
if ( (~(mood)) )
pause1 = 0;
end
end
```

the previous piece of code is to organize pausing the stopwatch. if the stopwatch finished counting, it would be reset “reset1 will be equal to one”. So, the stopwatch will be on the pause mood “pause1”. And the counter will start over, if the user presses the pause button “toggling”. So, the second always statement if the user press pause to be zero, or one, pause1 will be changed. And note that else statement is to initialize pause1. In case the user press on the pause button, but pause1 does not equal to zero, or one, and the mood is in the stopwatch then make pause1 to equal to zero. (pause is equal to one at first as the one before this piece of code says)

```
51  always_comb
52  begin
53  if ( (outsource11 > 9) | (outsource12 > 5) | (outsource13 > 9) | (outsource14 > 5) )
54  error_surpass = 1;
55  else
56  error_surpass = 0;
57  assign notzero = ( (~(outsource11 == 0)) | (~(outsource12 == 0)) | (~(outsource13 == 0)) | (~(outsource14 == 0)) );
58  if (( (~(error_surpass)) & notzero & (set & (pause1) & (ctrl0) ) ) | (reset1 & ctrl)) //
59  sel = 1;
60  else
61  sel = 0;
62  if (~(ctrl == ctrl0))
63  error2 = 1;
64  else
65  error2 = 0;
66  if ( (outsource21 > 9) | (outsource22 > 5) | (outsource23 > 3) | (outsource24 > 2) )
67  error_surpass_time = 1;
68  else
69  error_surpass_time = 0;
70  if( (~(error_surpass_time)) & (set & mood & pause) )//I added the pause recently
71  set_time = 1;
72  else
73  set_time = 0;
74  end
75
```

and this piece of code (starts from line 51: 75 on the code) is all for organizing the checks. For example, if the user entered the value to count down in the stopwatch mood to be more than 59: 59, there should be an error signal to say that there is an error on entering the value, and this signal is to print the error message on the seven segment display and to stop the counter while printing the error (just in case the user press unpause after entering the invalid value). this signal called in the code “error_surpass”. Note that outsource11, outsource12, outsource13, outsource14 are the carriers of the values that enters the stopwatch. Meanwhile, outsource21, outsource22, outsource23, outsource24 are the carriers of the values that enters the watch. The maximum for the watch is 23:59. “error_surpass_time” detect if that happens, or not. And almost, has the same behavior like error_surpass.

Also, sel which is defined in the code is the if condition that says there should be set for counting down or not. sel is equal to one if the user entered valid value, the program is stopped, the counter is on the count down option, and finally, the user press set. Also, sel have to be equal to one, if the user reset the program while counting down to enter the value

and if the user wants to change from count up to count down, or from count down to count up. without stopping the counter, so there will be an error. this error signal can be detected by comparing the current count mood (up, or down) of the user “ctrl0” with the wire that actually control the program “ctrl”. Note that ctrl0 is the mood of counting that is under control of the user. Meanwhile, ctrl is the direct owner of counting up or down on the counter.

```
errorchanging_block stopreset1(clk,reset1,reseting,stopreseting);

always@(negedge pausel)
begin
count4 = 0;
end
```

I used the “errorchanging_block” here to stop a serious issue is that while changing mood from counting down to counting up, I have to reset the value in the block, not to counting up on the stored value. So, I added on the reset of the stopwatch that if the user changing the mood from counting down to counting up the program should be reset, but it enters in a while loop. So, after adding the block in the line 58 reseting is equal to one and after a second stopreseting would be equal to one and I added an if condition if stopreseting is equal to one, so make reseting is equal to zero. reseting has access on reset1 that reset the stopwatch. Also, this trick is made with block5 that count from zero to thirty.

In line 62, the always statement the makes the mood of counting up, or counting down cannot be changed without pausing the program, and not to count up unless setting the counters with valid values. This always statement is sensitive on both the positive, and negative edge of the changing mood the controlled by the user, and pause of the stopwatch.

line 77 states that if the user changing the mood from 1 to zero (from counting down to count up) make reseting equal to 1 which would reset the stopwatch.

```
always@(negedge pausel)
begin
count4 = 0;
end
```

this for reseting the value of block5. This because the counter of the block5 save the previous number, so if the user pause for 15, and make start, then pause for 15 seconds again, the block5 will reach 30, and the seven segment display would enter in the power saver mood. So, in each pause block5 would be reset.

```
103     always_comb
104   begin
105     if (mood&set)
106   begin
107     outsource21 = outsource1;
108     outsource22 = outsource2;
109     outsource23 = outsource3;
110     outsource24 = outsource4;
111   end
112     if ( (~(mood)) & set & ctrl)
113   begin
114     outsource11 = outsource1;
115     outsource12 = outsource2;
116     outsource13 = outsource3;
117     outsource14 = outsource4;
118   end
119   end
```

In the code from line 103 to 119:

From line 103 to 119, this always statement is to make the values entered by the user according to the mood, and the user wants to set or not. So, the program enters the value to the setting of the stopwatch, or to the setting of the watch.

```
120   errorcounter_block mala(clk,reset1,error_surpass,error_surpass_printing);//added
121   errorchanging_block mala2(clk,reset1,error2,errorchanging_printing);
122
123   newclkfsm changeon(clk,reset1,fsmchange);
124
125   block5 #(5) B5(clk,resetingblock51,count4,pause1,signalpowersaver); // Giving the signalpowersaver
126
127   assign enter0 = (ctrl & ((~(count0 == 0)) | (~(count1 == 0)) | (~(count2 == 0)) | (~(count3 == 0))) );
128   assign enterfin0 = ( (((~ctrl) & (~pause1)) | ((~pause1) & enter0)) & (~(error_surpass_printing)) & (~(errorchanging_printing)) );
129
130   block1 #(9) Block1(clk,reset1,ctrl,sel,outsource11,count0,enterfin0);
131   block3 B13(count0,count0_time,fsmchange,clk,reset1,signalpowersaver,error_surpass_printing,
132   error_surpass_printing_time,errorchanging_printing,mood,display0);
133
134   assign enter1 = ( (count0 == 9) & (~ctrl) );
135   assign enterfin1 = (enter1 | ( (count0 == 0) & ( (~(count3 == 0)) | (~(count2 == 0 )) | (~(count1 == 0))) & ctrl ));
136
137   block1 #(5)Block2(clk,reset1,ctrl,sel,outsource12,count1,enterfin1);
138   block4 B24(count1,count1_time,fsmchange,clk,reset1,signalpowersaver,error_surpass_printing,
139   error_surpass_printing_time,errorchanging_printing,mood,display1);
140
141   assign enter2 = (enter1 & (count1 == 5) );
142   assign enterfin2 = (enter2 | ( (count0 == 0) & (count1 == 0) & ( (~(count3 == 0)) | (~(count2 == 0)) ) & ctrl ));
143
```

line 120 is the block that gives a signal for 3 seconds to stop counting. Meanwhile, printing the error message on the seven segment display for 3 seconds. line 121 is the same but for the error of changing mood while counting, so it gives just one second.

line 123 is to make the periodic motion on the seven segment display, and it would be presented only in case that block5 gives the signal after stopping the program for 30 seconds.

line 125 is block5 that count from zero to thirty if the stopwatch is stopped, and the signalpowersaver is to be one if block5 reaches 30 in the counter.

line 127, and 128 is to make the counter on according to some conditions that the user does not press pause, the 4 counters don't reach 59:59 while counting up, nor reaching 00:00 while counting down. In case that the counter reach one of those two values. the program would reset. and these two lines is responsible for making the first block count on.

line 130 is the first counter that will count from 0 to 9 and vice versa. line131, and 132 is that printing the count of the first block of the stopwatch, and watch on the seven segment display, or other errors can be printed.

line 134, and 135 is to make the checks like in line 127, and 128 but for the second block. Also, note that this signal will be equal to one, if the last block "counter" reach 9. So, the second block would print 1 and the first one would print 0 (this would happen only if all other checks like the user does not stop the program, the counters reach 00:00, nor the counters reach 59:59).

line 137, 138, and 139 is the same like line 130, 131, and 132, but for the second seven segment display.

```
141   assign enter2 = (enter1 & (count1 == 5) );
142   assign enterfin2 = (enter2 | ( (count0 == 0) & (count1 == 0) & ( (~(count3 == 0)) | (~(count2 == 0)) ) & ctrl ));
143
144   block1 #(9)Block6(clk,reset1,ctrl,sel,outsource13,count2,enterfin2);
145   block6 B33(count2,count2_time,fsmchange,clk,reset1,signalpowersaver,error_surpass_printing,
146   error_surpass_printing_time,errorchanging_printing,mood,display2);
147
148
149
150   assign enter3 = (enter2 & (count2 == 9));
151   assign enterfin3 = (enter3 | ( ((count0 == 0) & (count1 == 0) & (count2 == 0) & (~(count3 == 0)) & ctrl) ) );
152
153   block1 #(6)Block7(clk,reset1,ctrl,sel,outsource14,count3,enterfin3);
154   block7 B44(count3,count3_time,fsmchange,clk,reset1,signalpowersaver,error_surpass_printing,
155   error_surpass_printing_time,errorchanging_printing,mood,display3);
156
157   assign reset1 = ((enterfin0 & ( (count3 == 6) & (count2 == 0) & (count1 == 0) & (count0 == 0) )) | (reset0 & (~(mood)) ) | reseting);
158
159
```

Also, line 141, and 142 is the checks like 134, and 135. but for the third counter. line 144, 145, and line 146 is like130, 131, and 132 but for the third seven segment display. line 150, and 151 is like 141, but for the forth counter. line 153, 154, and 155 is like 130, 131, 132 but for the forth seven segment display.

line 157 is the reset for the stopwatch that reset if the counter reach 60:00 but (which means that once it reaches 59:59, then would be reset).

```
161
162  block2 #(9) Block10(clk,reset2,set_time,outsource21,count0_time,enterfin0_time);
163  block2 #(5) Block11(clk,reset2,set_time,outsource22,count1_time,enterfin1_time);
164  block2 #(9) Block12(clk,reset2,set_time,outsource23,count2_time,enterfin2_time);
165  block2 #(2) Block13(clk,reset2,set_time,outsource24,count3_time,enterfin3_time);
166
167  assign enterfin0_time = 1;
168  assign enterfin1_time = (count0_time == 9);
169  assign enterfin2_time = (enterfin1_time & (count1_time == 5) );
170  assign enterfin3_time = (enterfin2_time & (count2_time == 9) );
171
172  assign reset2 = ( (enterfin0_time & ( (count3_time == 2)&(count2_time == 4)&(count1_time == 0)&(count0_time == 0) )) | (reset0 & (mood) ));
173  errorcounter_block mala3(clk,reset2,error_surpass_time,error_surpass_printing_time);//added
174
175  endmodule
176
```

Finally,

(there is an assumption here that the watch will treat the minutes like the seconds and the hours like the minutes. So for example, 23:59, 59 will be treated like seconds and 23 is treated like minutes that is because simulate it easily after proving that it is simulated as it could be I will add a block that counts from 0 to 59 and put it before the first block, so the first two blocks will count minutes and the second two blocks will count hours).

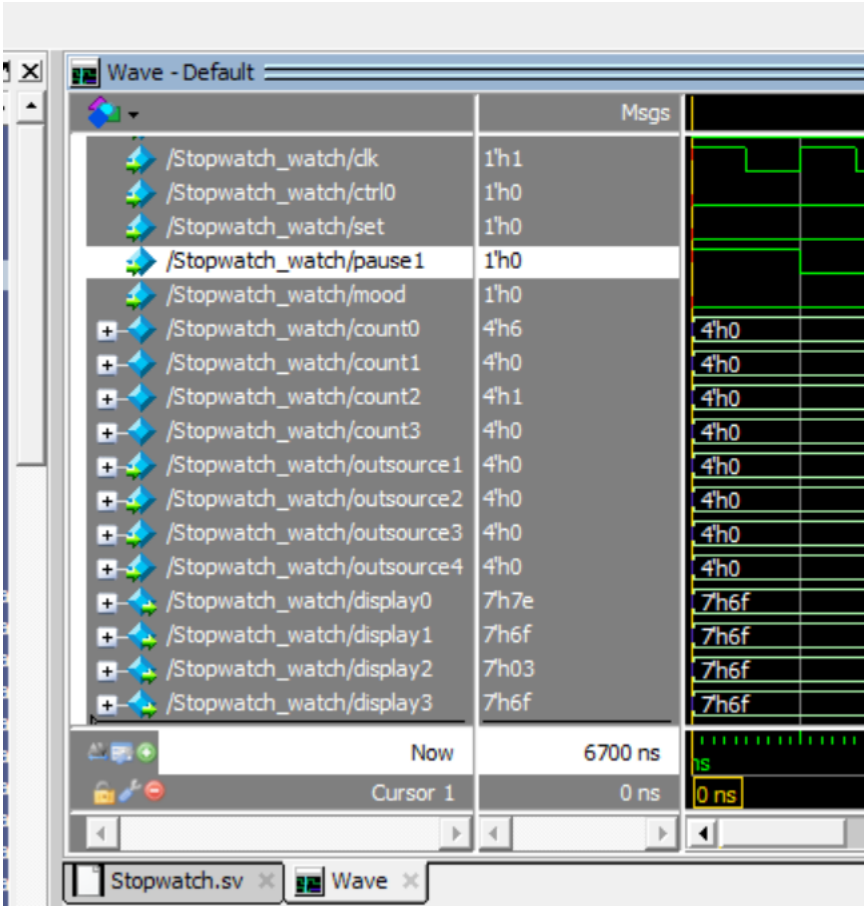
from line 162 to 165 the four block that count from 00:00 to 23:59. And from line 167 to 170 will makes all checks, and organize counting.

line 172 is to reset the watch if the watch reach 24:00 (So, after 23:59, it would be 00:00)

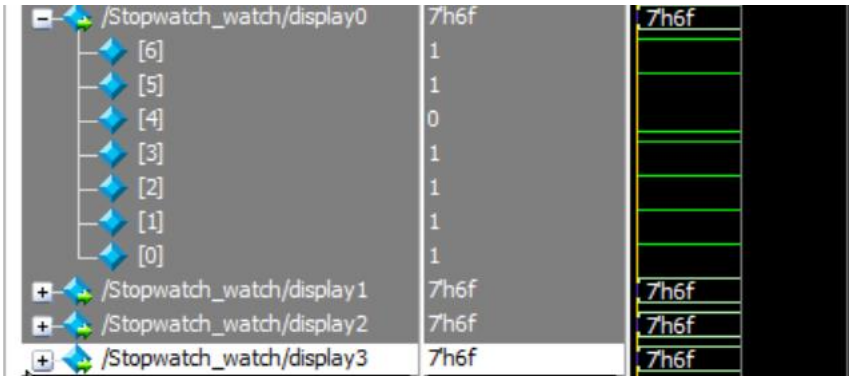
line 173 is to give a signal for printing an error message on the seven segment displays, and to stop counting, if the user entered an invalid value while setting.

line 165 is to assign the pause of the watch that happens when the user presses the pause button while the mood is on the watch part. line 166 is block5 that counts from 0 to 30 to enter the power saver mood.

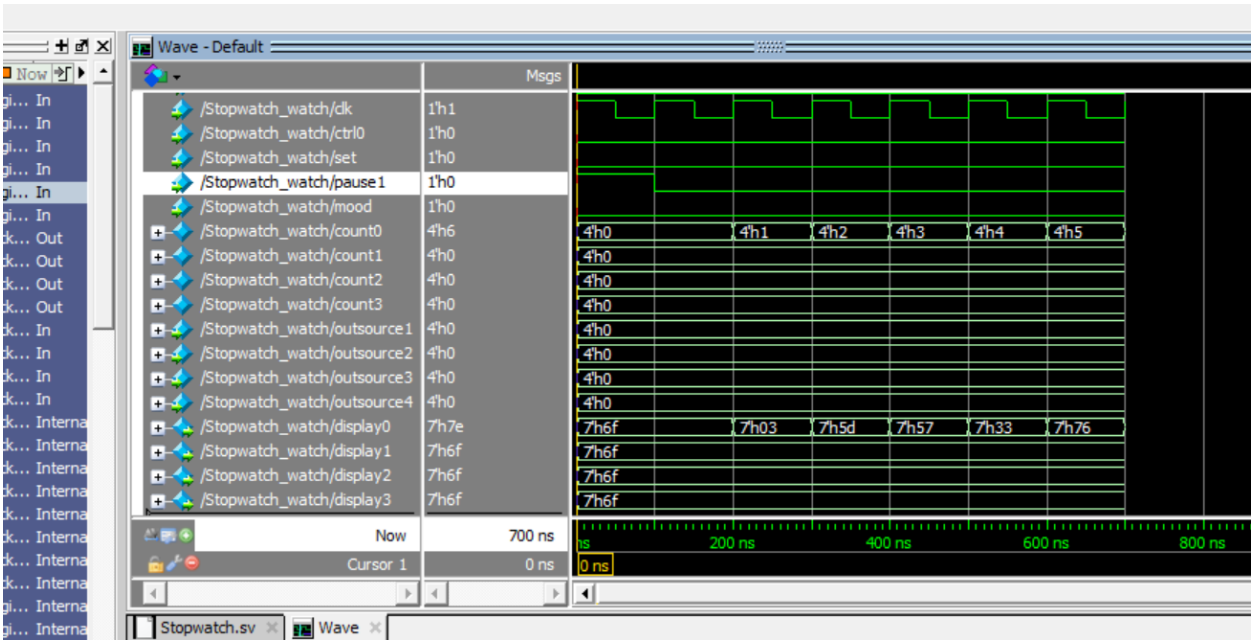
simulation Part for the stopwatch:



This last simulation is for the opening the device, I initialized only the clock “clk” the pause manually. others are self-initialization that is because the user is not in deep need to initialize all parameters while turning on the stopwatch, or the watch. $display_0$ is the first seven segment display, $display_1$ is the second seven segment display, $display_2$ is the third seven segment display, and $display_3$ is the forth seven segment display. So, at first the screen should display 00:00 so all wires should be one, but the $display_0$ [4], $display_1$ [4], $display_2$ [4], and $display_3$ [4].

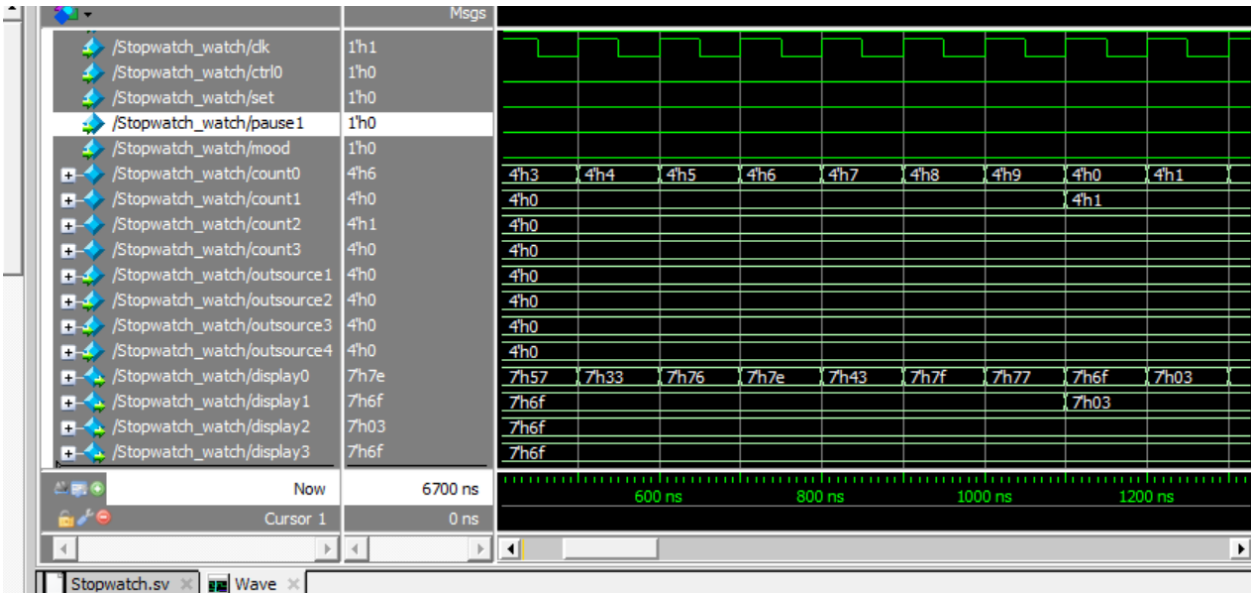


so, let’s press pause now to run the stop watch



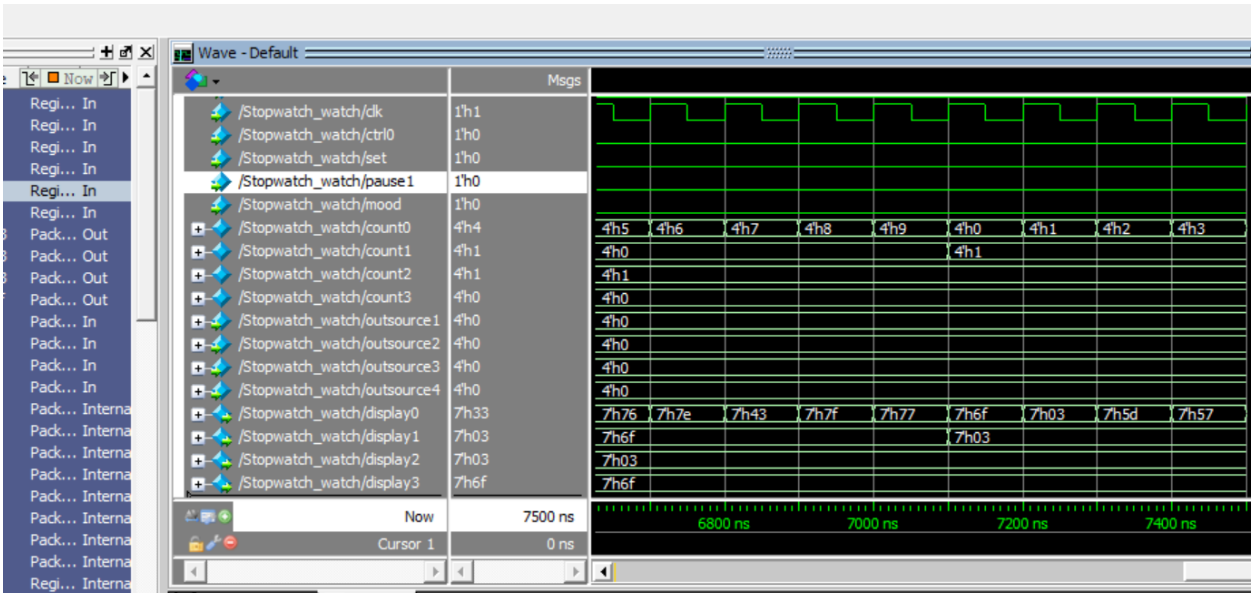
So, the first block is counting up and the seven segment printing those numbers. The following table is to translate numbers represented on the seven segment into hexadecimal (these hexadecimal numbers are not the translating of the numbers its self, it is the translating of the printed numbers on the screen. For example, 0 in decimal system is equal to 0 in hexadecimal system. but in the presentation on the screen all wires are equal to one, but the wire number 4 is equal to zero, so this is the translation of these numbers into the hexadecimal number which is equal to “6f”). Simply the number represented on the seven segment by the following numbers.

the hexadecimal that represent the number on the seven segmnet	number
" 6f "	0
" 03"	1
" 5d "	2
"57"	3
"33"	4
"76"	5
"7e"	6
"43"	7
"7f"	8
"77"	9



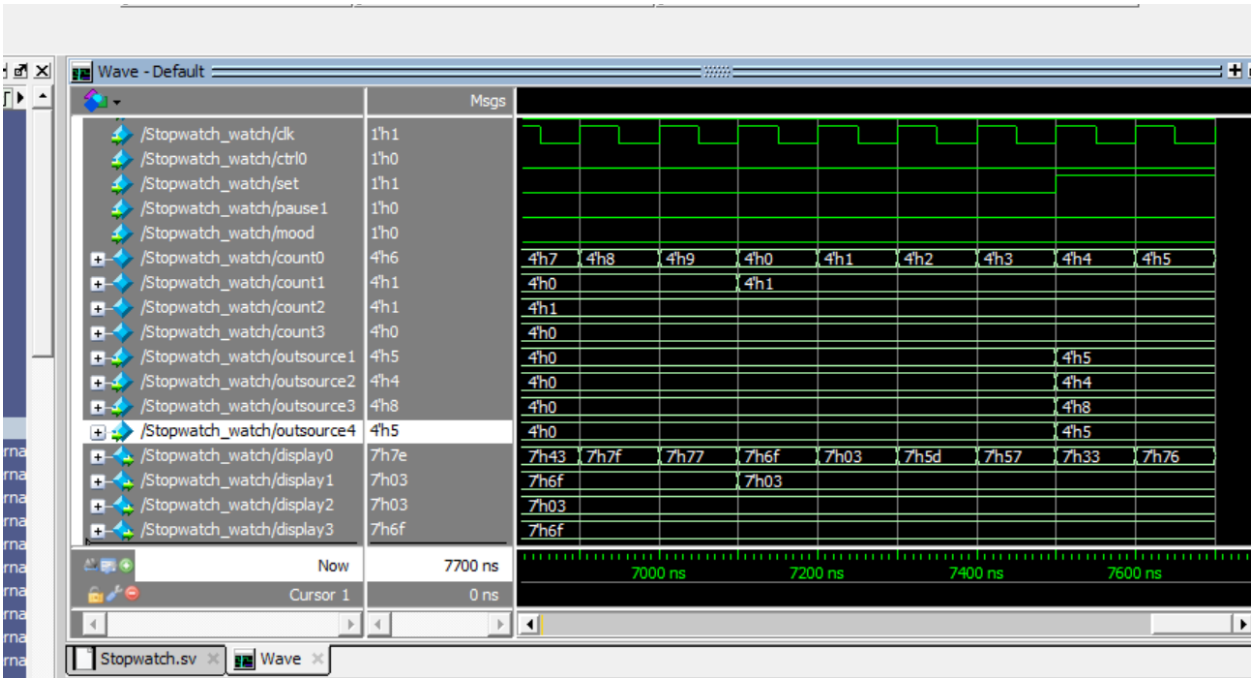
here in the last simulation the second block starts counting once the first one reach 9 as expected. the second block counts only if the first one reach 9.

also here the third counter count only if the first count reach 9, and the second one reach 5, and so on. what to be printed on the seven segment now (01:00).

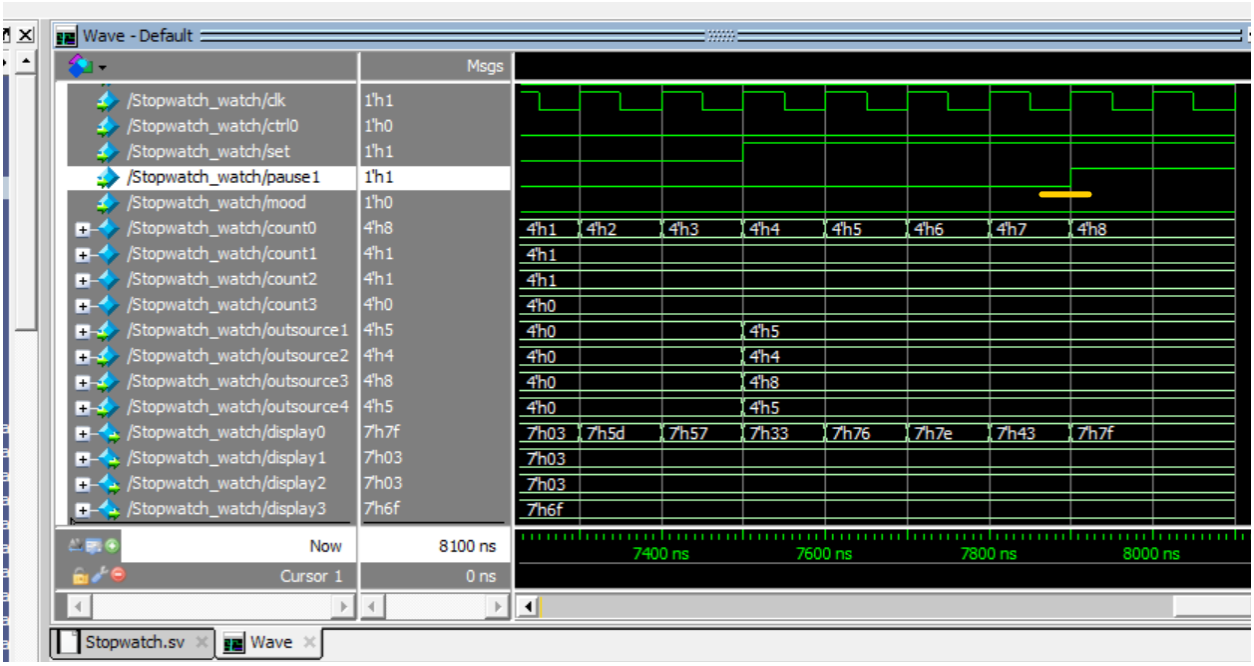


here the counter seems to count smoothly.

but now let's try in the next simulation a false setting by now stopping the program, nor the count mood to be down, but giving a start value 58:45.

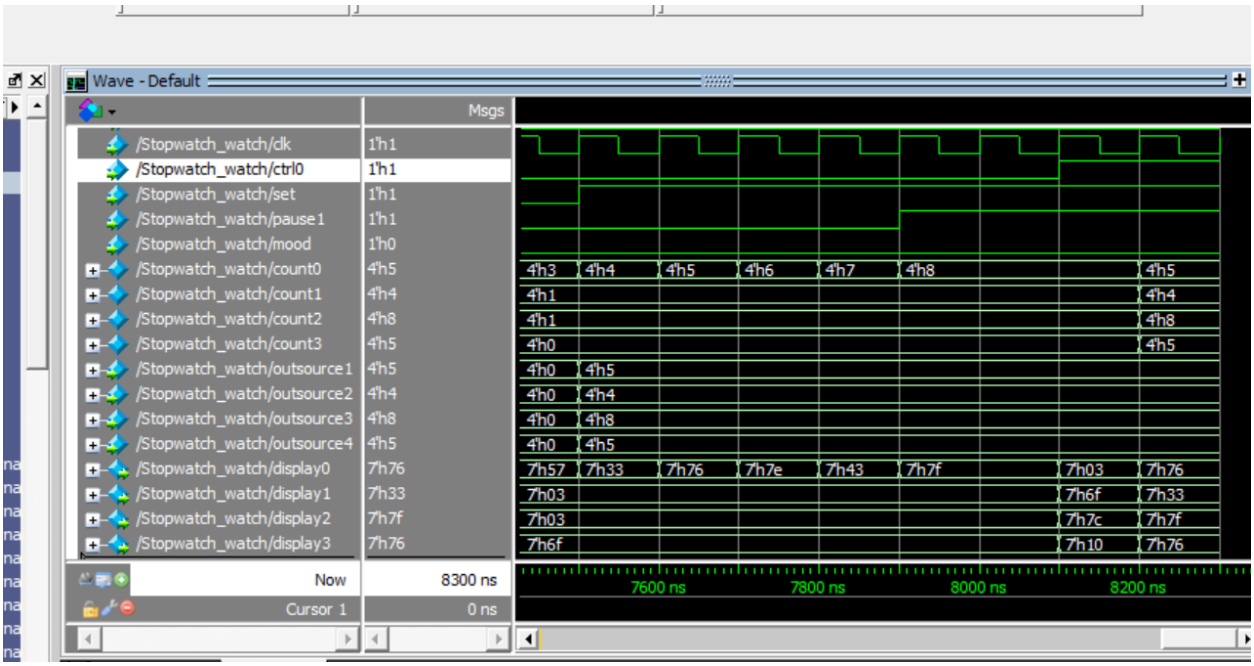


there is nothing happens to be done in the counters that is because we did not make the counter to count down, nor stopping the program. So, let's try pause the stopwatch and letting the counter in counting up to see what would happen.



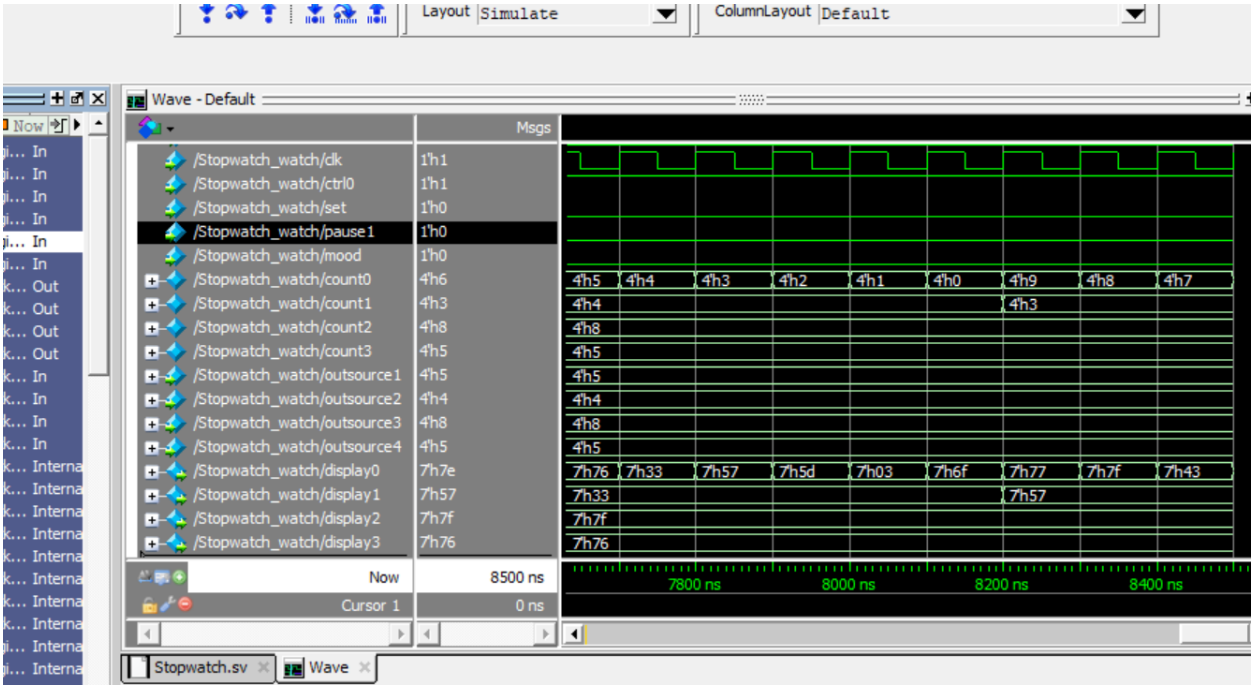
here the counter stops counting.

So, let's changing the counter mood to count down by changing ctr0 to be one.

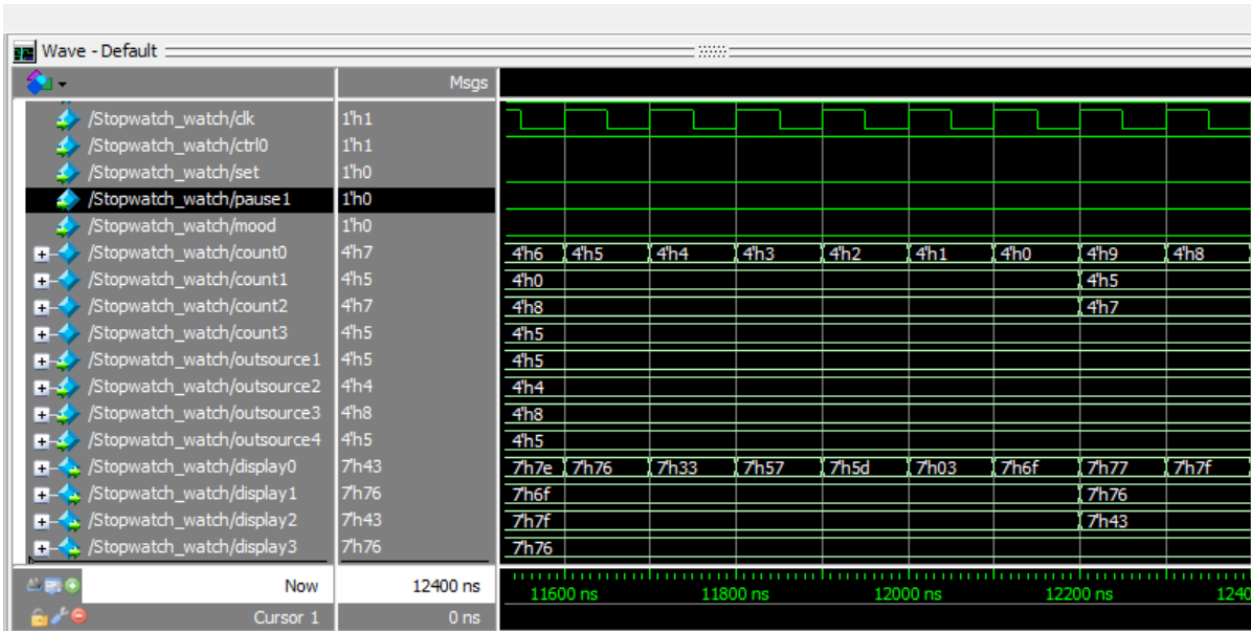


, and now the counter got the number entered by the outsources ($outsource_0$, $outsource_1$, $outsource_2$, $outsource_3$).

now, let’s make pause, and set to be equal to zero.

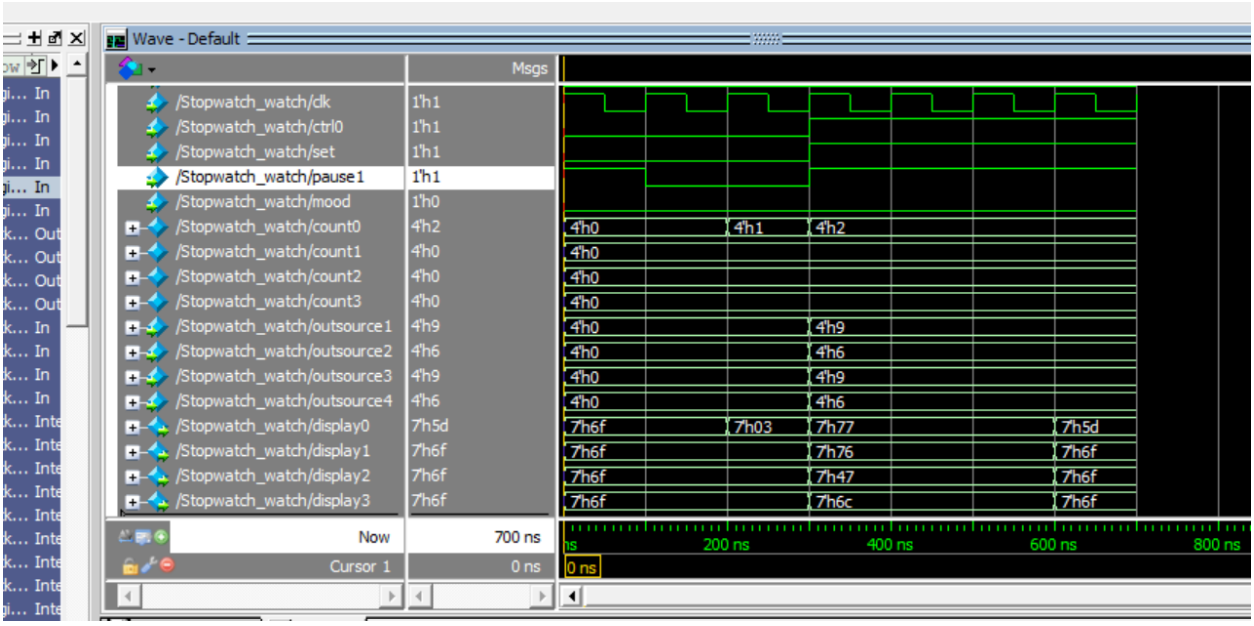


it seems like it counts down smoothly too. but to make sure that it counts down smoothly. let’s run till reaching 07:59

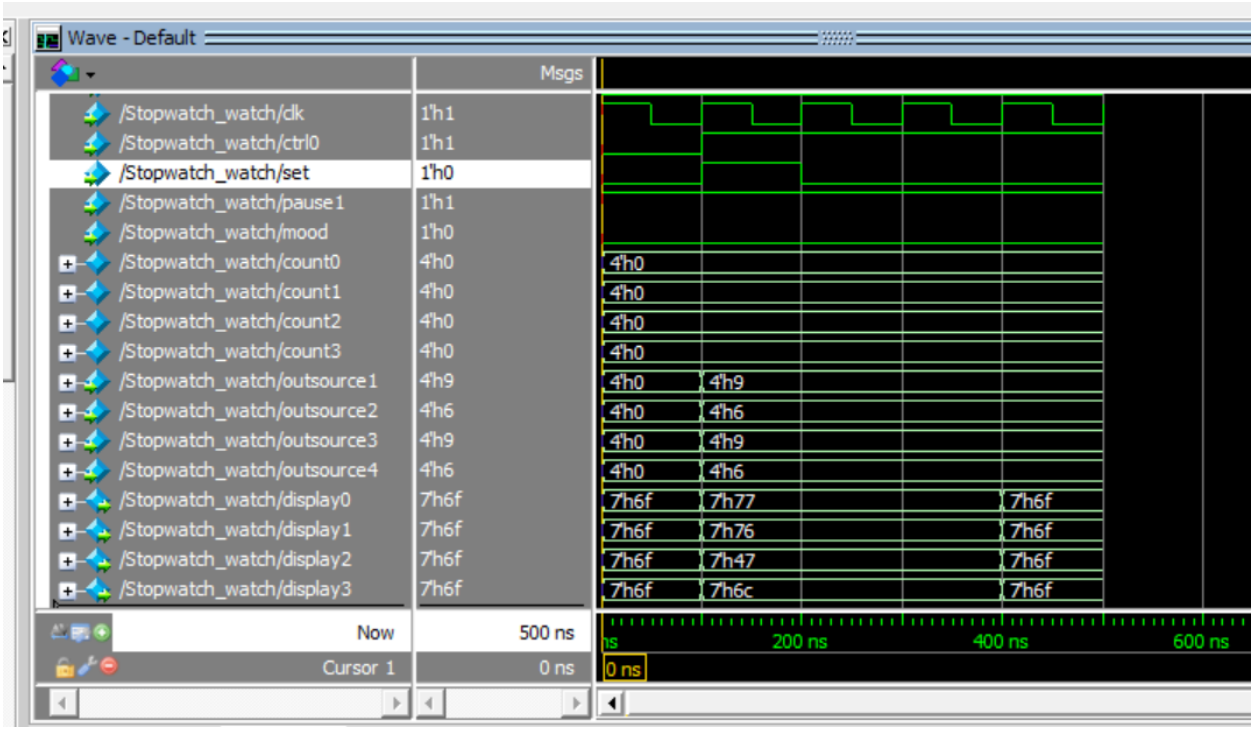


So, the counters are doing well.

now let’s enter an invalid number like 69:69 to set the stopwatch.

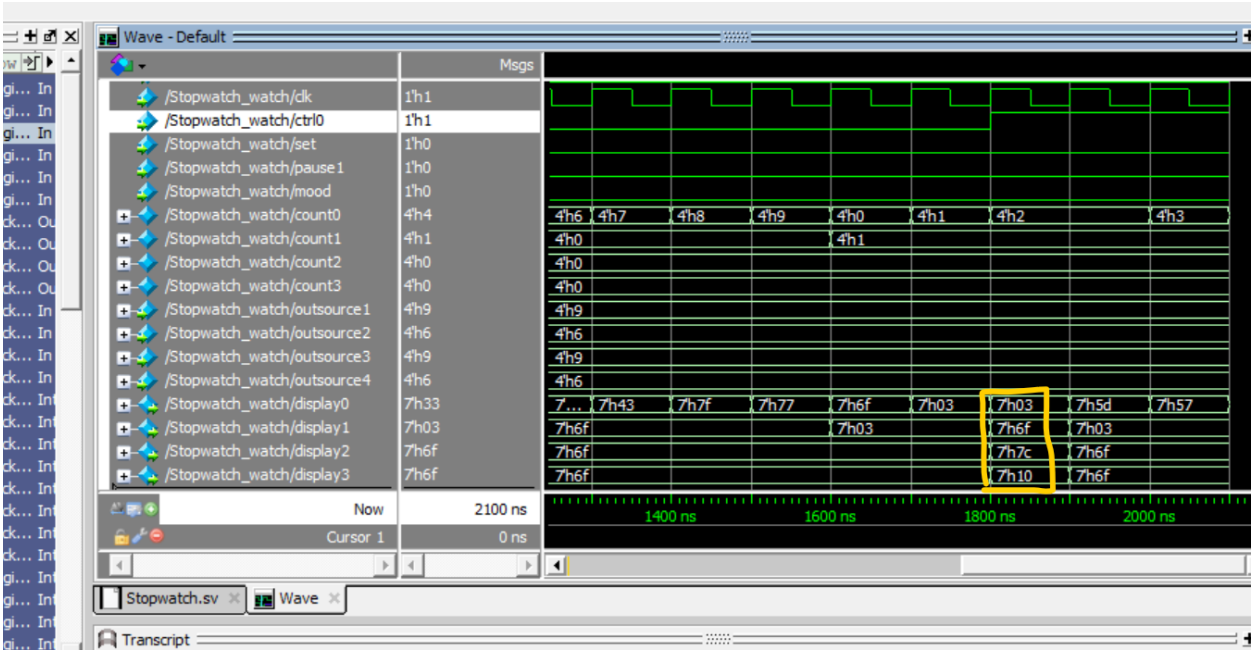


we can see here $display_0$, $display_1$, $display_2$, $display_3$ print something for three times. Actually, they print []:59, then they print what was in the counter 00:00.

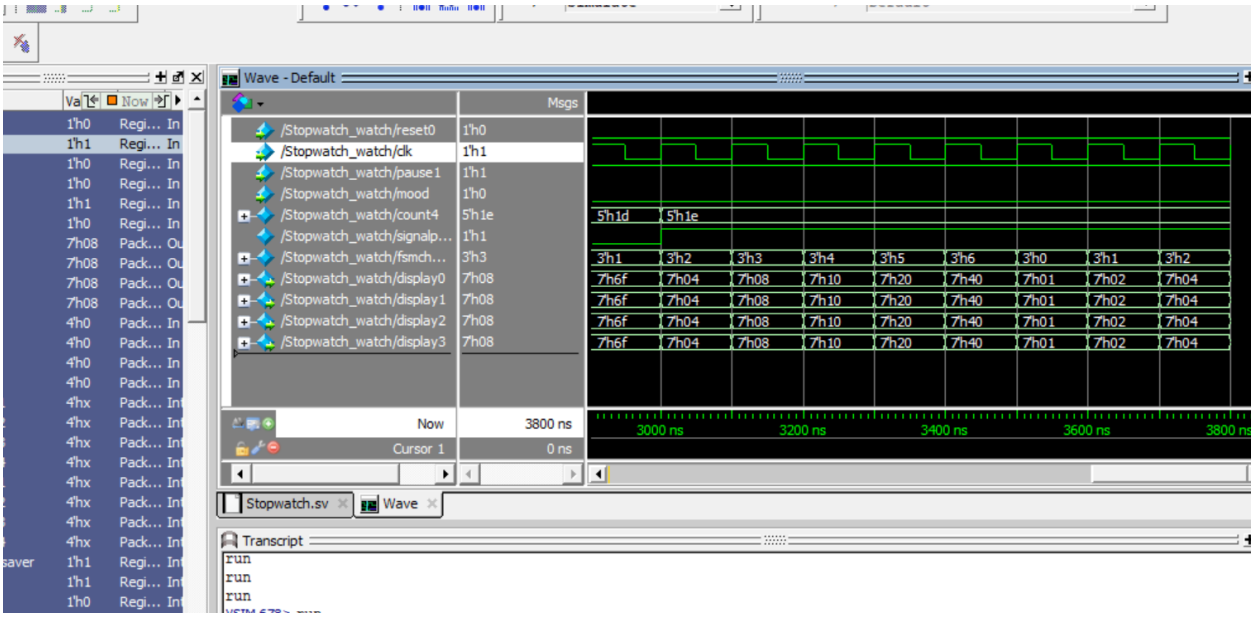


the error of changing mode while counting we will test it in the next simulation.

and we can see the seven segment displays print –E:01. for a second then printing according to the previous number

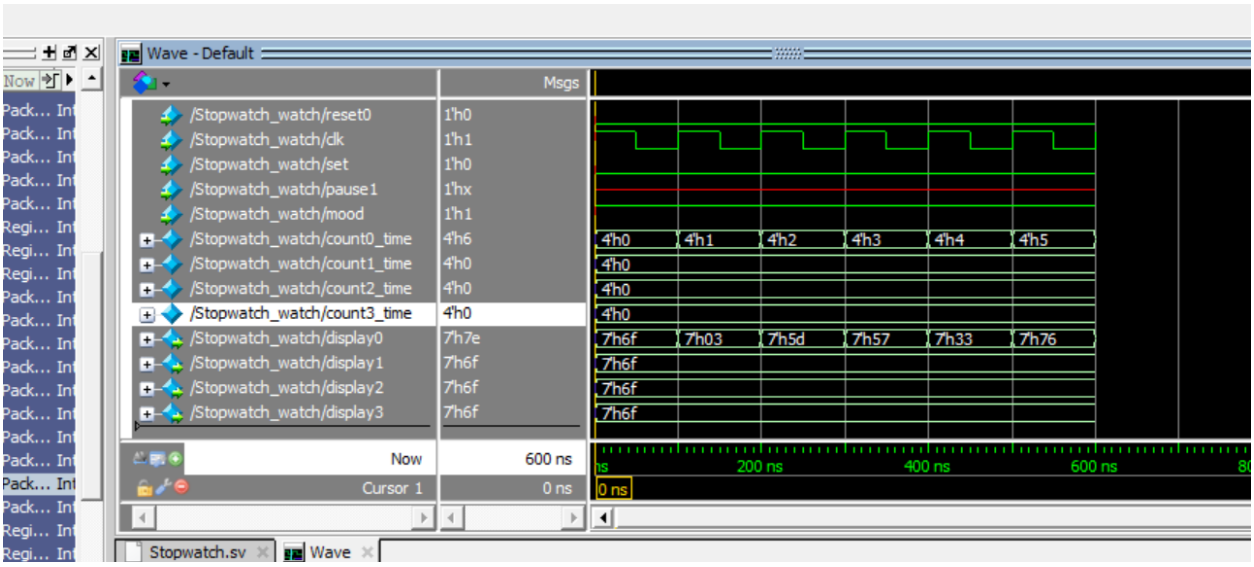


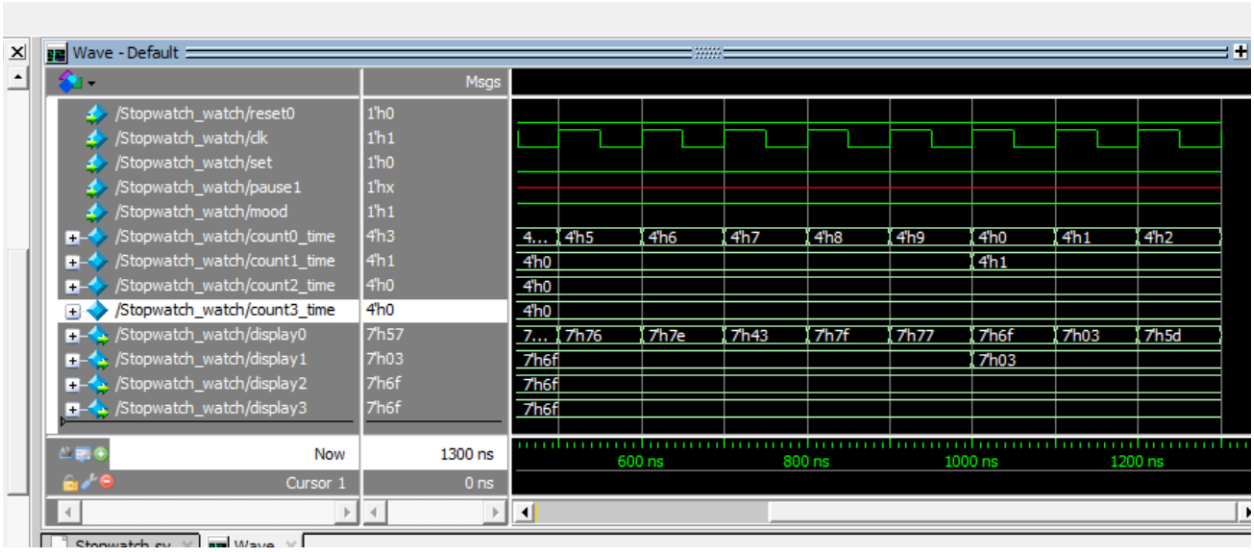
now, let’s check the power saver mood



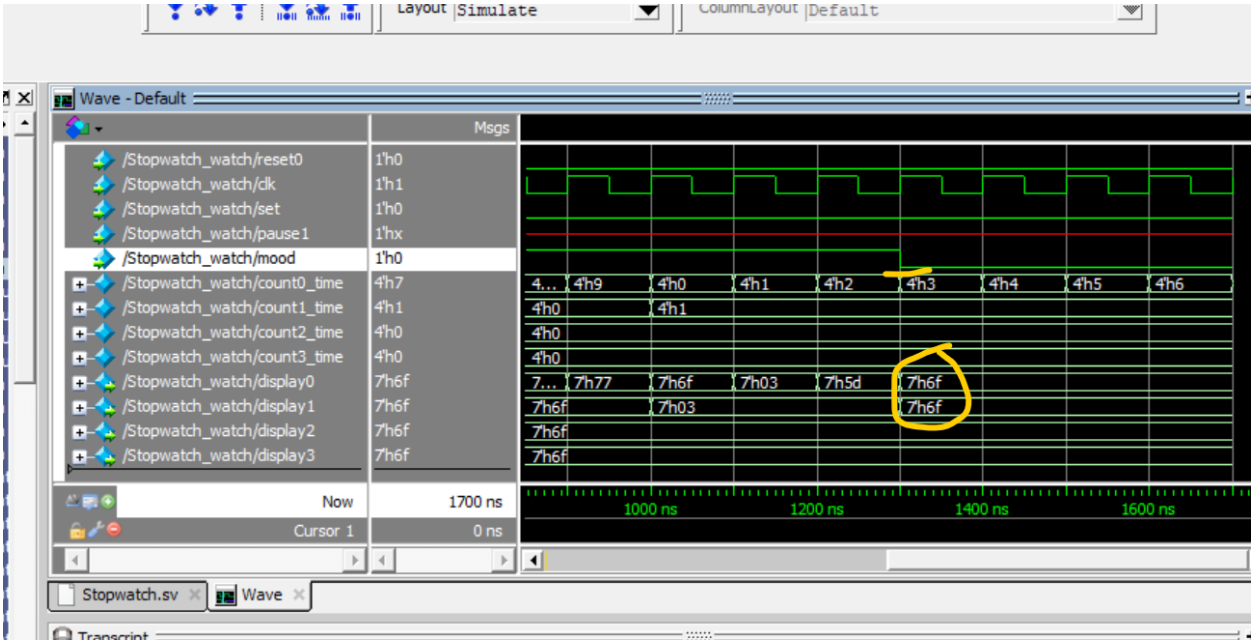
here, once the counter “count4” reaches 30 starts to represent the periodic motion

now, let’s test the watch mood. and I am here makes pause is not defined to show that watch is not dependent on it



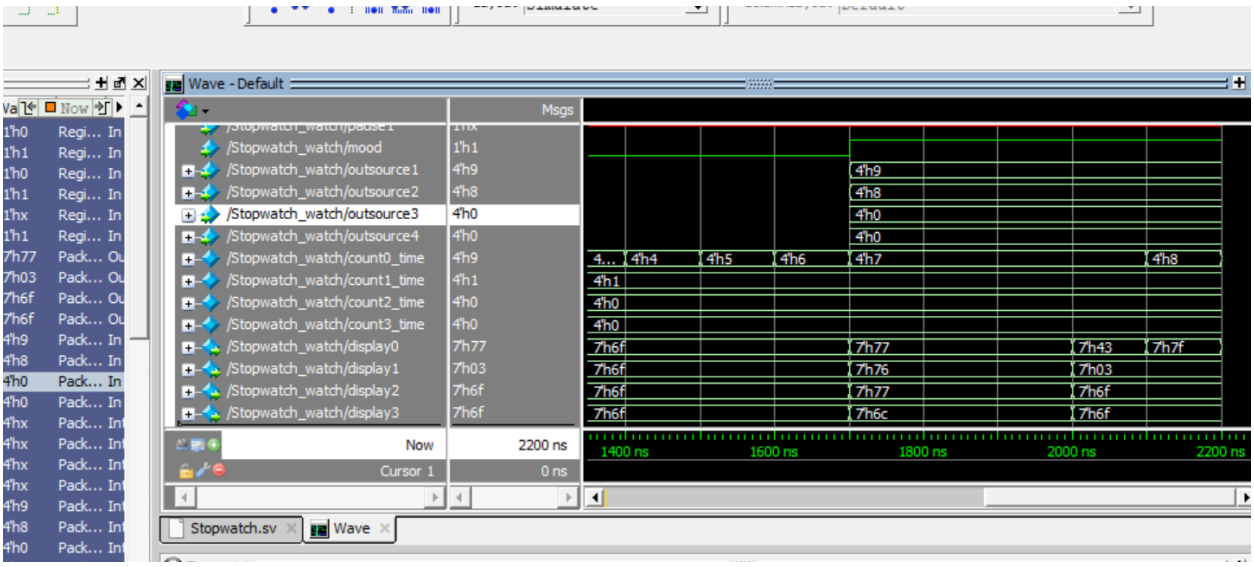


and here, as the same as the stopwatch mood. the second block count only if the last one reach 9.

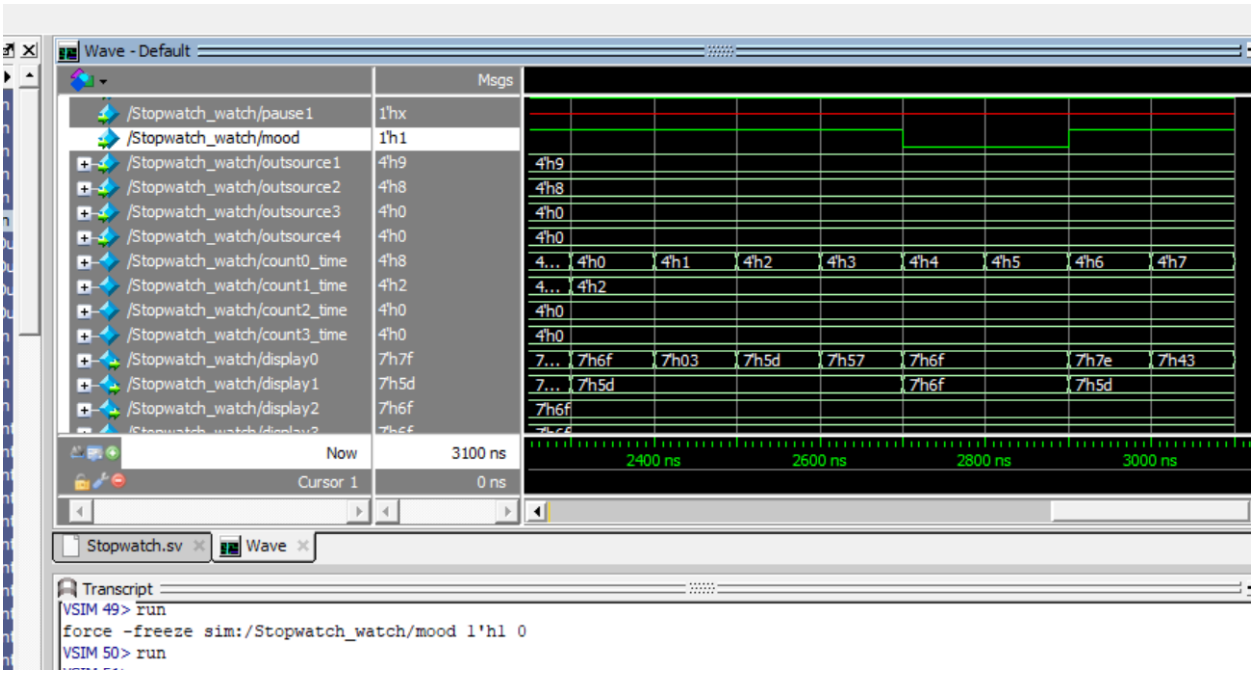


here changing mood to be in the stopwatch does not affect the counters (count0_time, counter1_time, counter2_time, and counter3_time), but what is mainly affected is that the seven segment display. to print what is on the stopwatch now which is 00:00

and now what about setting a value to the clock, and not for stopwatch (the user here should set a valid number or the program print []:59 for three seconds like in the stopwatch), then printing the last number, and continuing on counting again.

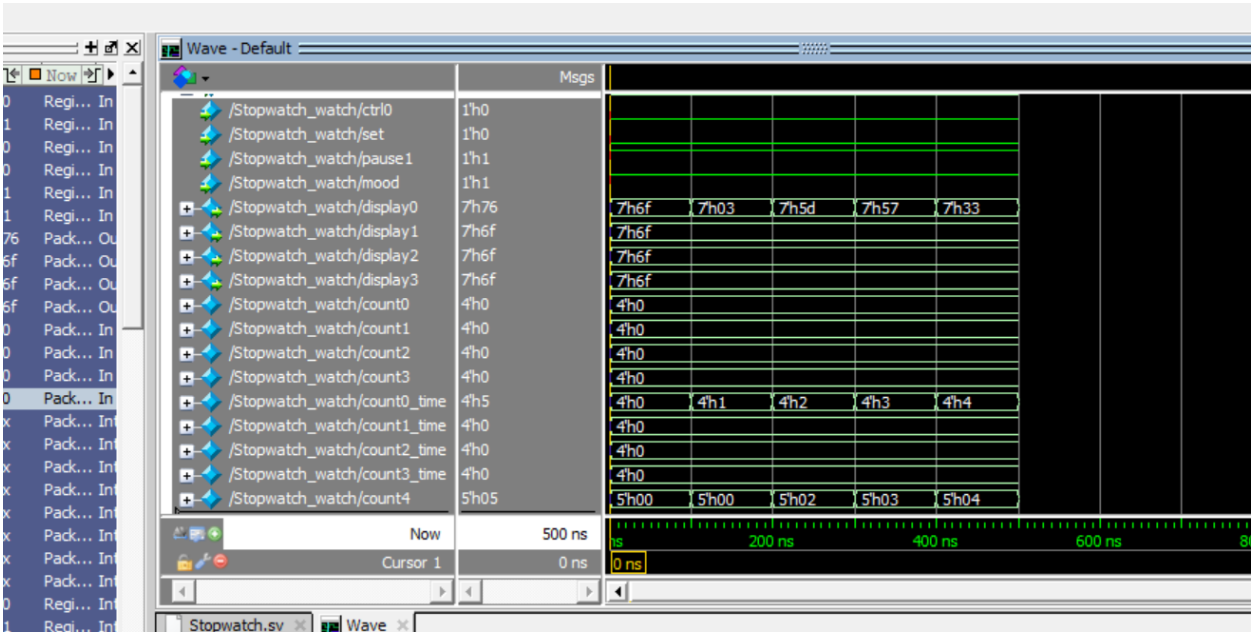


now changing the mood again on- off to test what is the effect.



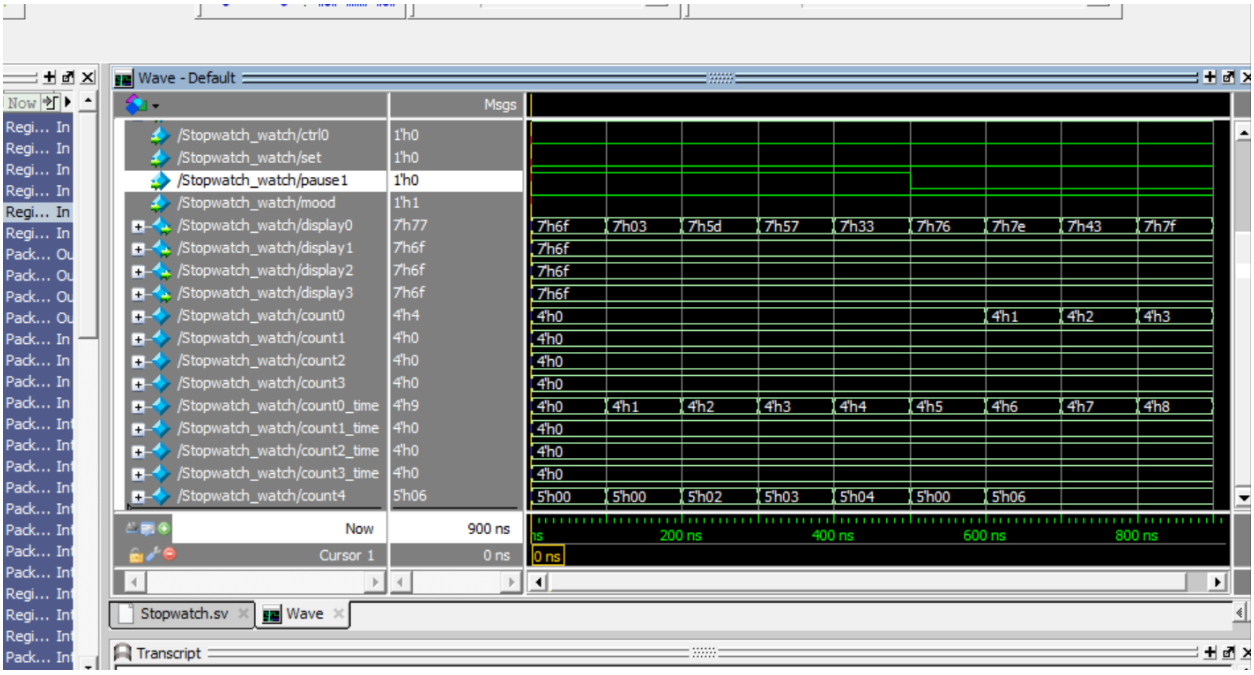
again there is no effect of changing the mood on counting on the counters. The only change is in what to be printed on the screen.

now, let's simulate the whole program:



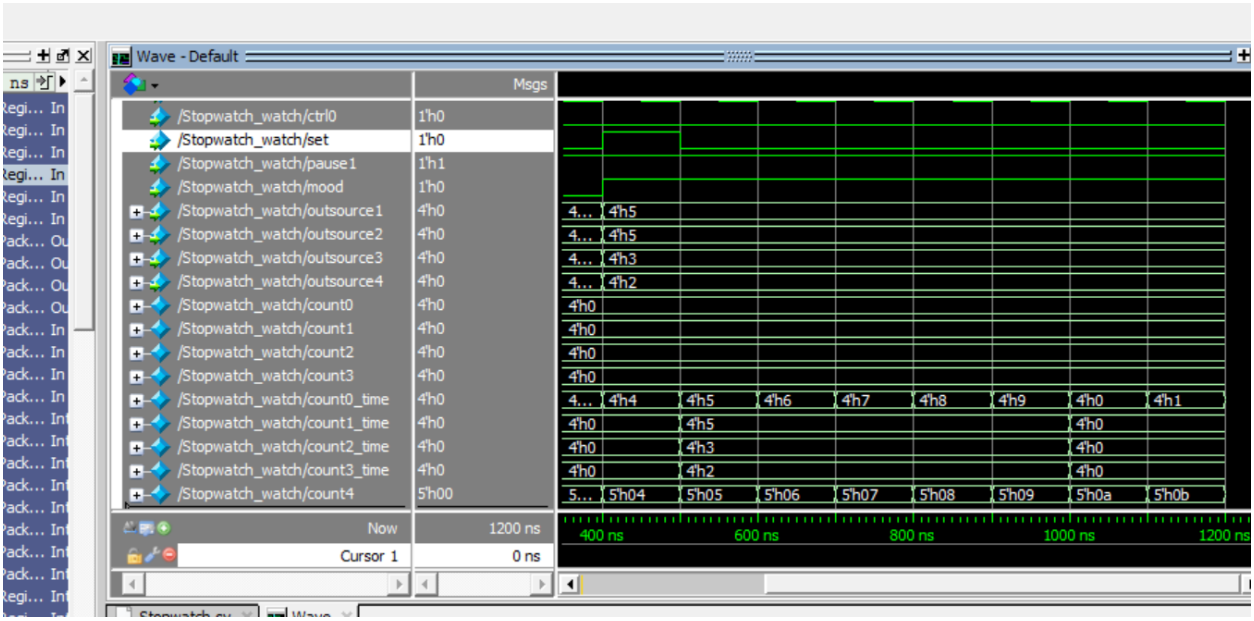
now, the clock start counting but the stopwatch is not counting that is because we press pause. and the seven segment display print what in the watch as mood is equal to one

so, let's make pause equal to zero to see what will happen.



here we can see that the counter of the stopwatch starts to count in parallel with the watch. and note that block 4 stops as we made pause equal to 0.

Let's try reset the clock to a number near to 23:59 to see it in this very moment.



So, the clock is running well, and as expected.

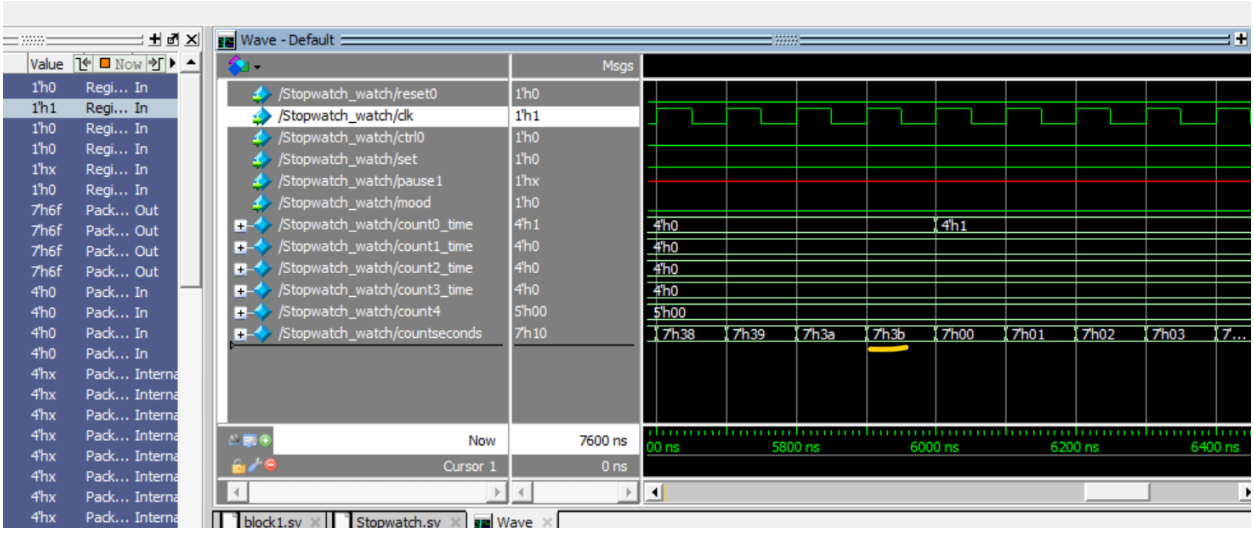
So, let's add the block that count 59 seconds before the counters that count 23:59.

So, I added block1_time before the first block of the watch.

The last piece of code should be like

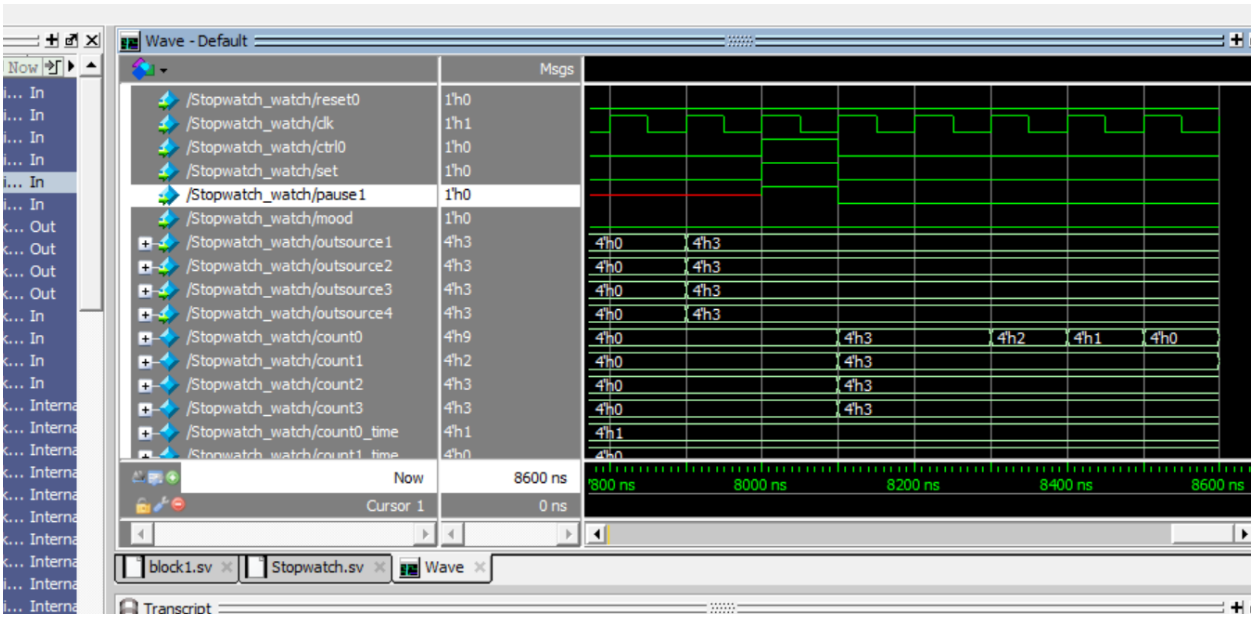
```
block1_time blockseconds(clk,reset2,countseconds);
block2 #(9) Block10(clk,reset2,set_time,outsource21,count0_time,enterfin0_time);
block2 #(5) Block11(clk,reset2,set_time,outsource22,count1_time,enterfin1_time);
block2 #(9) Block12(clk,reset2,set_time,outsource23,count2_time,enterfin2_time);
block2 #(2) Block13(clk,reset2,set_time,outsource24,count3_time,enterfin3_time);

assign enterfin0_time = (countseconds == 59);
assign enterfin1_time = (enterfin0_time & count0_time == 9);
assign enterfin2_time = (enterfin1_time & (count1_time == 5) );
assign enterfin3_time = (enterfin2_time & (count2_time == 9) );

assign reset2 = ((enterfin0_time&((count3_time == 2)&(count2_time == 4)&(count1_time == 0)&(count0_time == 0)&(countseconds == 0))))|(reset0&(mood
errorcounter_block mala3(clk,reset2,error_surpass_time,error_surpass_printing_time);//added

endmodule
```

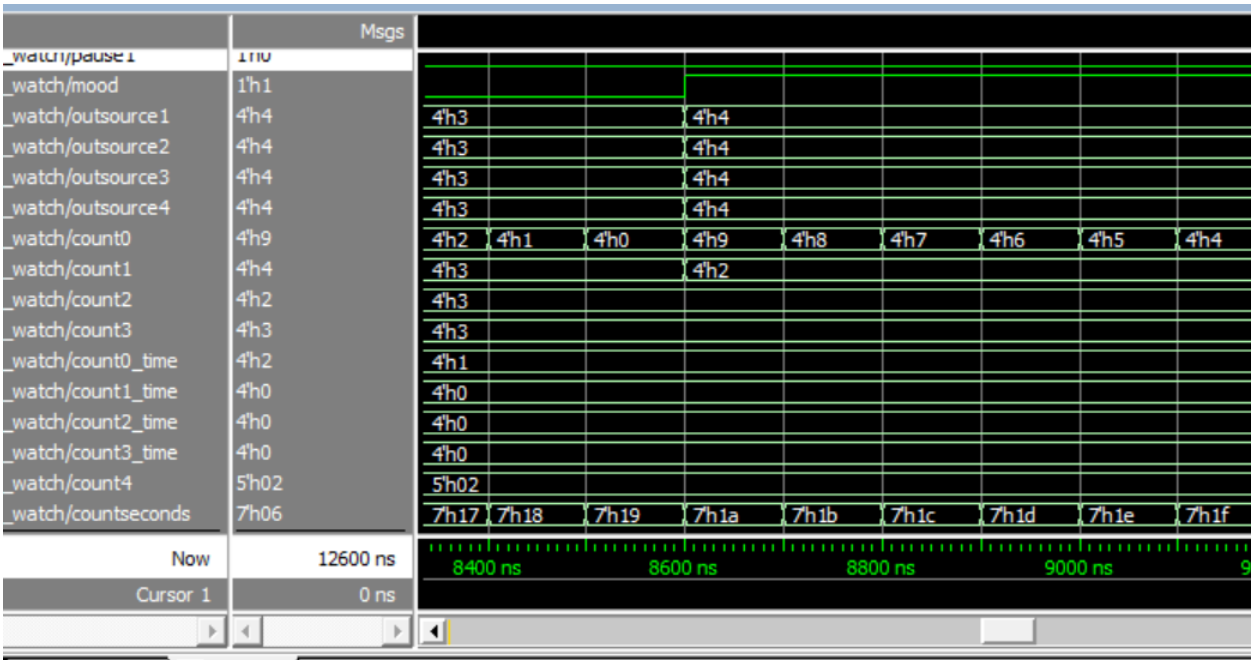
with simulating this part



3b in hexadecimal means 59 in decimal. So it works well.

in the next simulation, we are going to set both clock, stopwatch. the first image is setting the stopwatch with 33:33. The second one is setting the watch with 44:44. Note that I setting the stopwatch by making (set, pause and ctrl0) are equal to one, but mood is equal to zero. the second one by making set, and mood are equal to one.

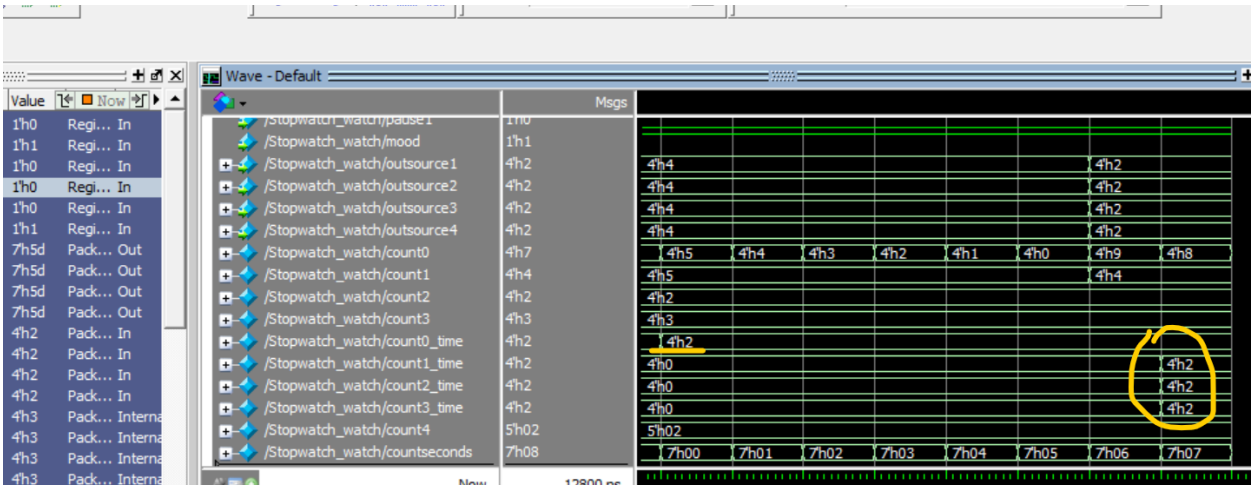




but there is no result. why?

this because the maximum is 23:59.

So, now let’s make it with 22:22 instead of 44:44



Finally, about the code of phase two, I edited it as it has about two components that can be replaced with if conditions. For example, if the counter reaches 9, I made a multiplexer in block1 to reset the counter if the counter reaches 9. I replaced it with an if condition in the addsub (adder, and subtracter) if count ==9, then count = 0. Also, I added some more details in block3,4,6, and 7 to organize the printing between the clock, errors and the stopwatch, but the previous piece of code is almost the same idea.