

Lab 6 Report

Name: Frank Le

UT EID: fpl227

Section #: 16100

a) Lab Report

When starting on the lab I chose to reuse all of my code from lab 4. By using previous code, all I needed to do to finish this lab was to adjust my previous code and to add another module called stopwatch.

My main approach to implementing the stopwatch functionality was using if-else statements. All 4 of the modes I needed to implement were placed in an always block with if statements that would direct which mode the stopwatch was in currently. I had an input “mode” that would select which mode the stopwatch was in. I also created variables such as startstop and finish to indicate whether the stop button was pushed and to check if the stopwatch finished counting.

I implemented the first mode (counting from 00.00 to 99.99) by first checking to see if both the stopwatch is stopped and reset. By doing so I would load in the starting values such as 0 into the registers to begin counting from 0. Next, I checked to see if the stopwatch was not stopped and if finish was not true. If these conditions were met, then the stopwatch would begin counting from 00.00 to 99.99. This was implemented through nested if-else statements which would check from left most to right most digit to see if the value 9 was reached. If 9 wasn't reached then the register would increment, if it was reached it would

rollover to zero until finally all registers had 9 as their values, signifying the stopwatch reached 99.99. At any point during counting the user could press the stop button to stop counting and press it again to resume. I implemented this by checking to see if stop was pressed and reset was not pressed. Once these conditions were met, I loaded in the current value into the register to ensure no incrementing was occurring.

Once I finished the first mode's functionality the remaining modes to implement were basically the same with minor adjustments. For mode 2, instead of initially loading 0 for all the digits, I loaded the tens and ones digit with the values from the switches. For mode 3, I loaded 9 into all the digits initially and decremented the registers instead of incrementing. The if statements also checked to see if the register reached 0 instead of 9 like the previous modes. Finally, mode 4 was exactly like mode 3 but I loaded the values from the switches into the tens and one place just like mode 2.

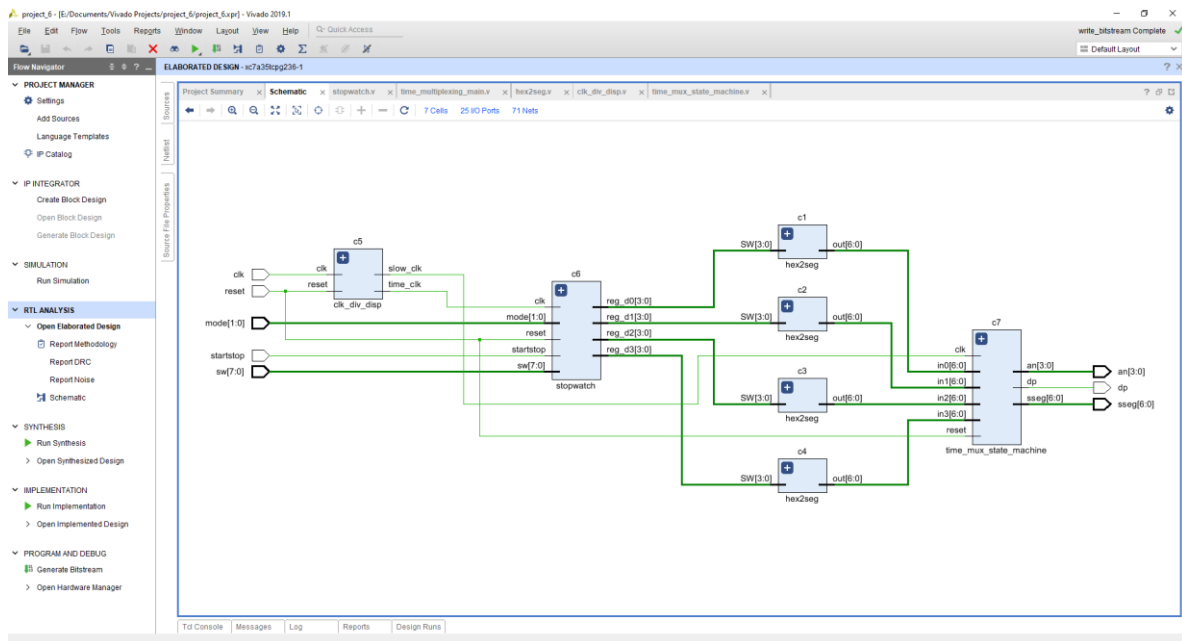
One of the major challenges with this lab was getting mode 2 and mode 4 to function correctly with an input greater than 9. Initially my design caused the stopwatch to rollover incorrectly because it didn't account for this corner case. However, this issue was resolved through if statements to check if the input was greater than 9.

b) Summary of system

This lab is a design for a stopwatch using Verilog and a FPGA board. The stopwatch has 4 different modes depending on the user's input on SW[1:0]: a mode to increment from 00.00 to 99.99, a mode that will increment from an externally loaded value from SW[15:8], a mode to decrement from 99.99 to 00.00, and a mode that will decrement from an

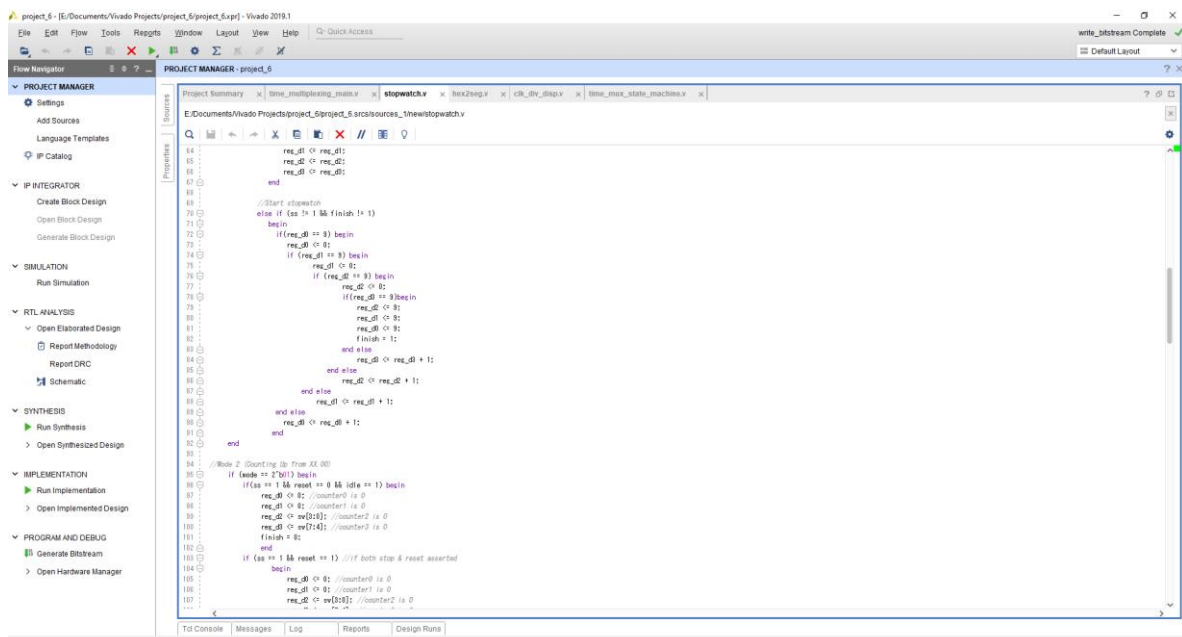
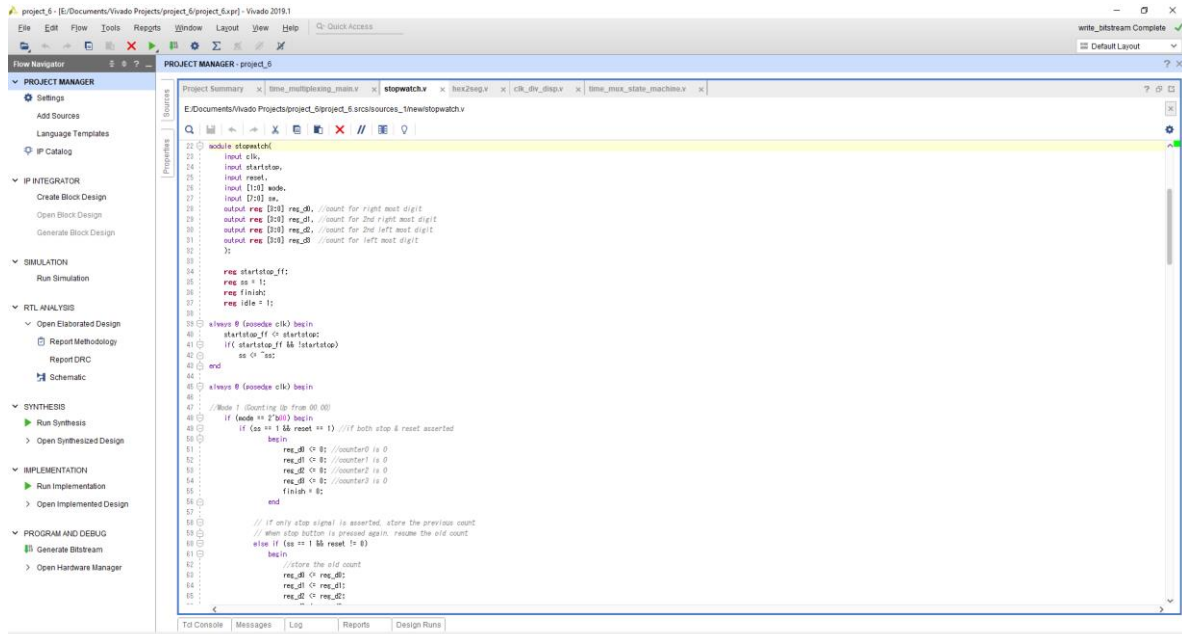
externally loaded value from SW[15:8]. The user can stop and resume the stopwatch by pressing the center button and can also reset the stopwatch by pressing the up button.

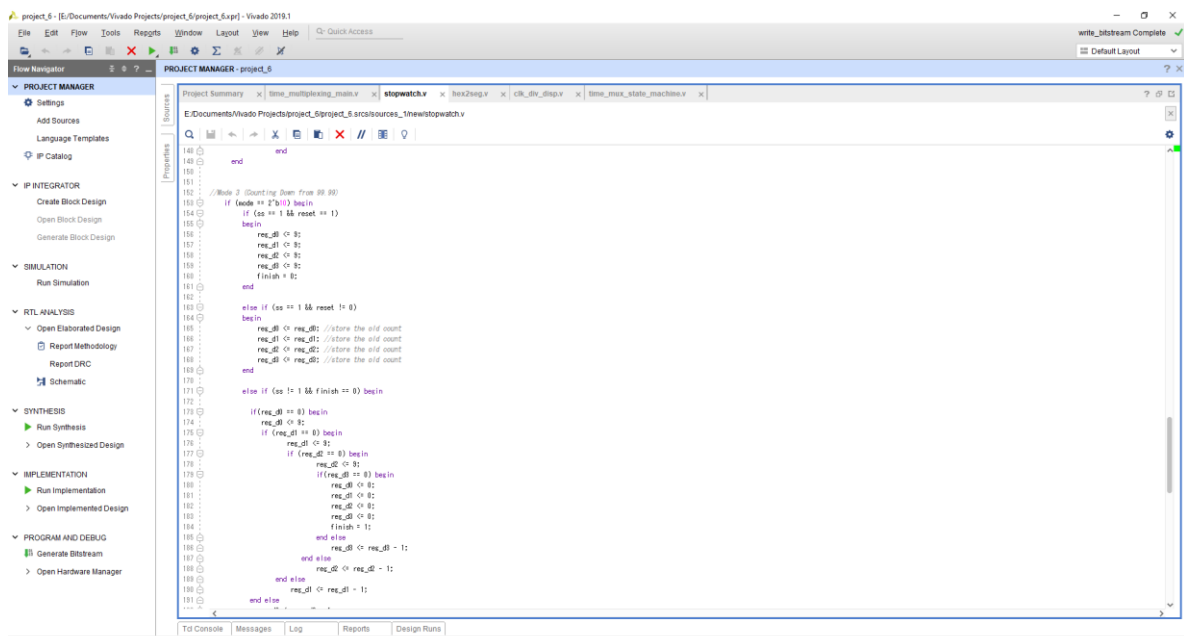
c) High Level Block Diagram

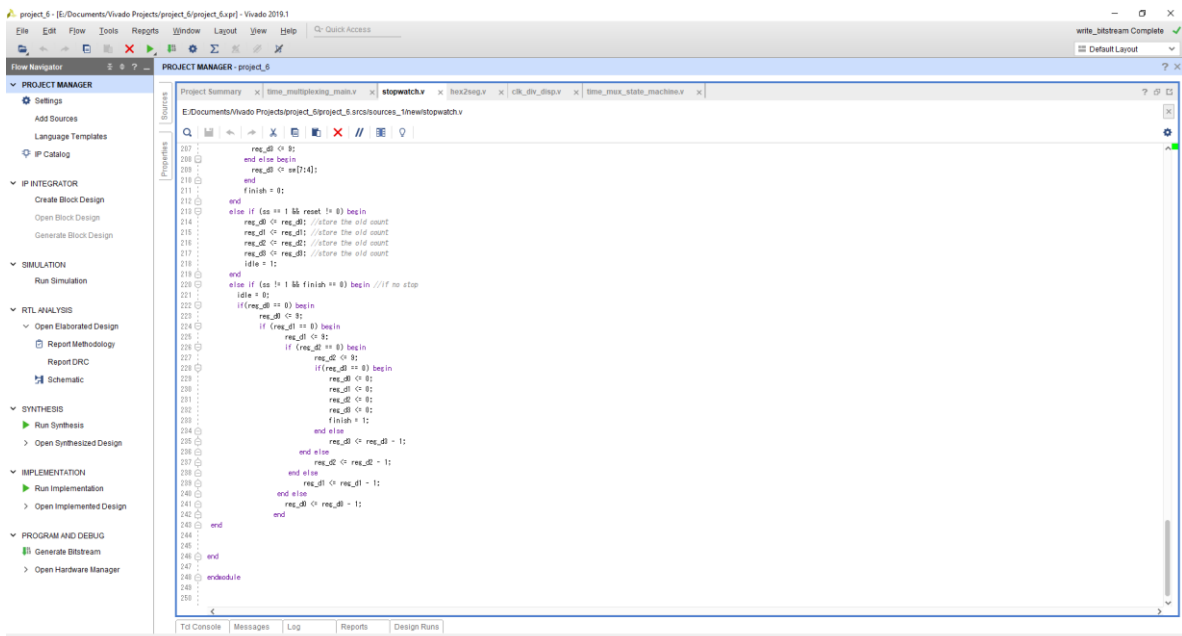
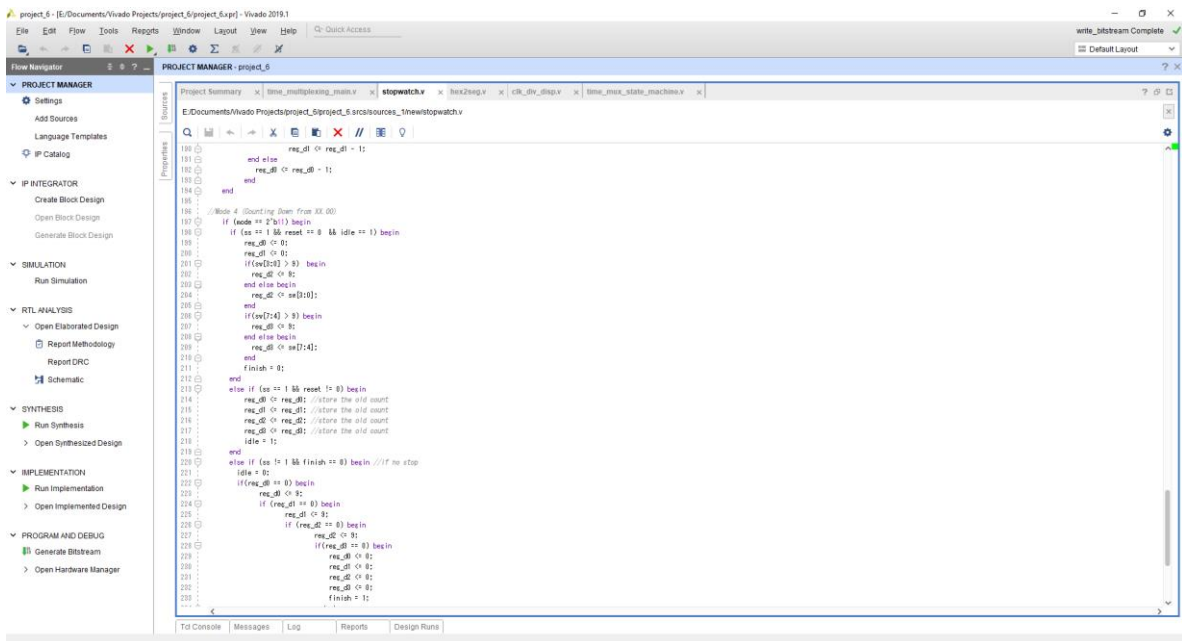


.v Files:

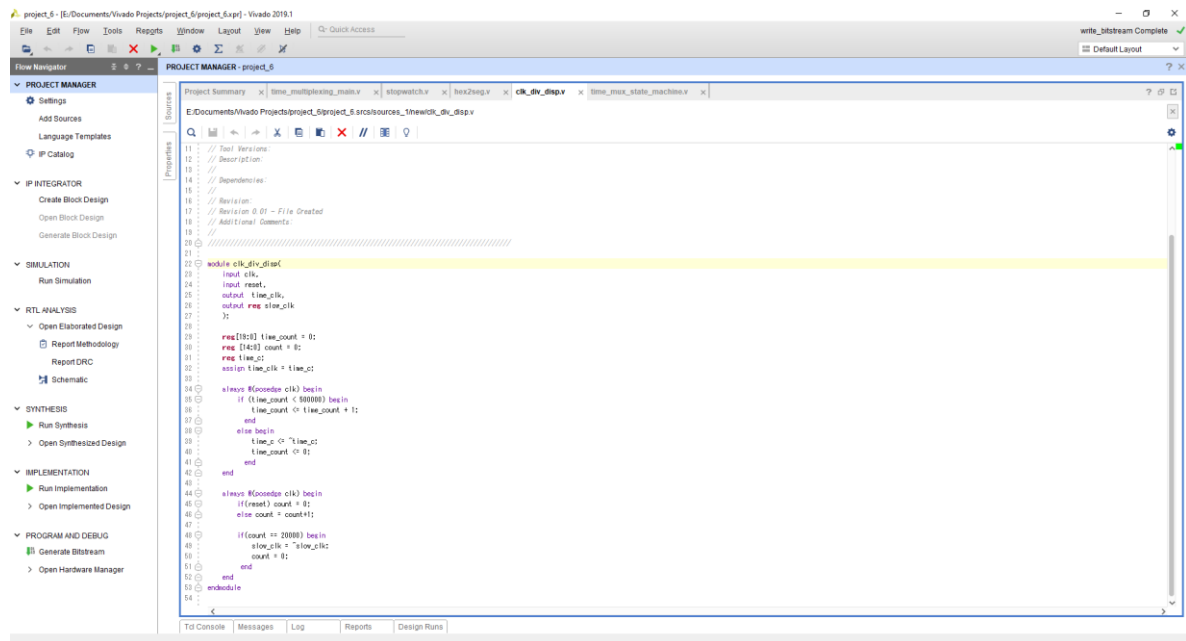
Stopwatch:



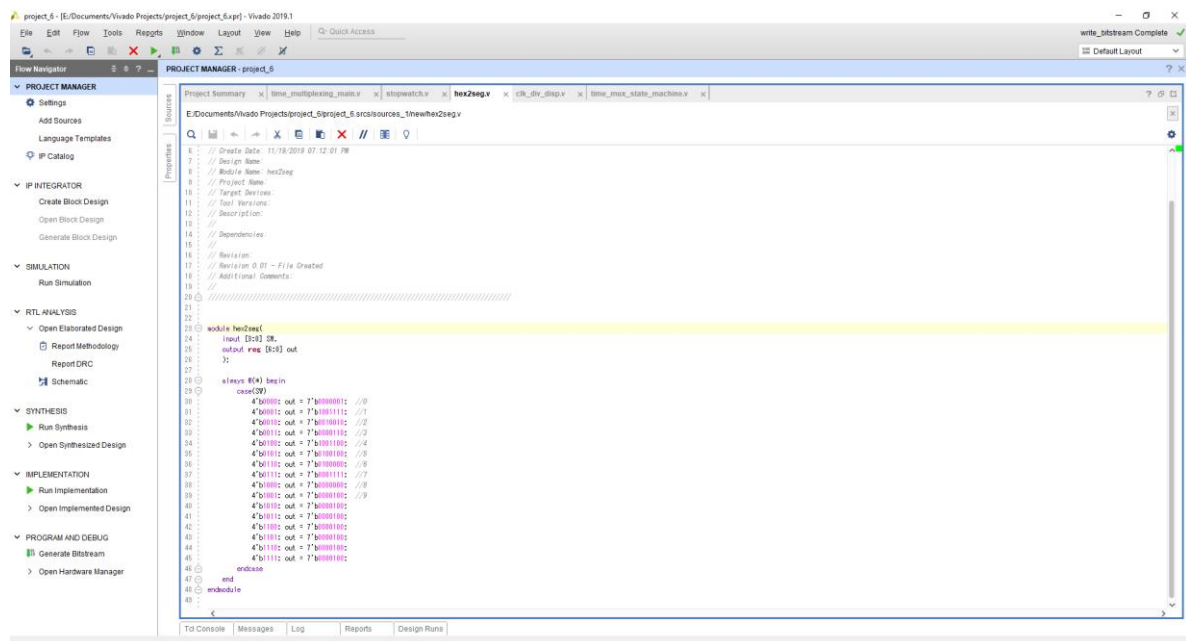




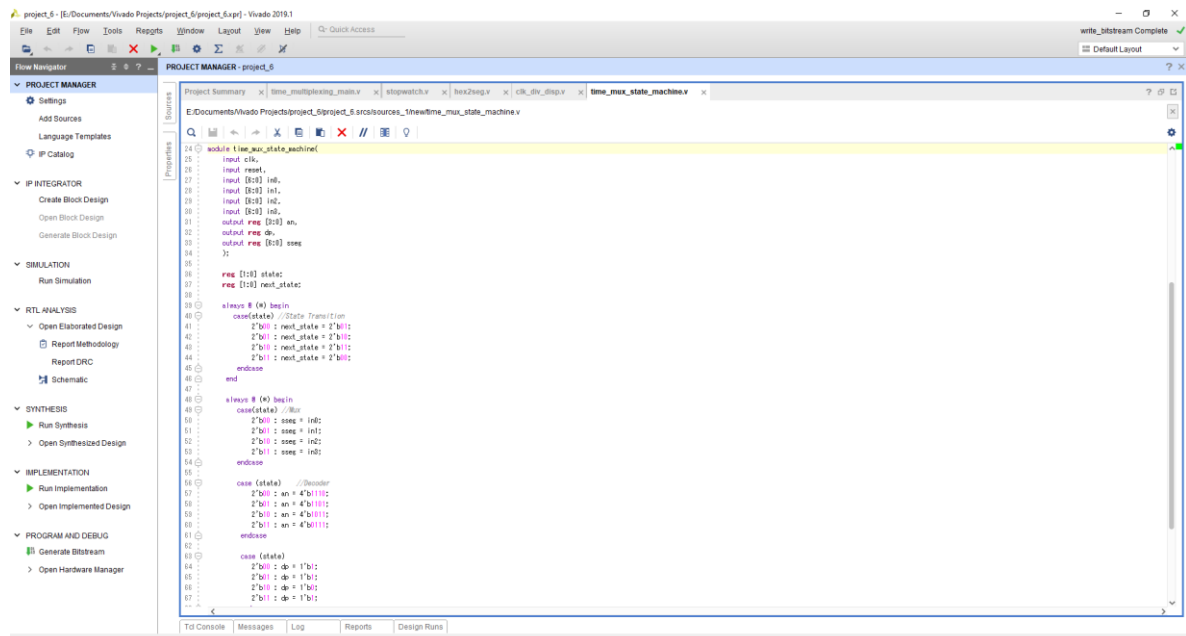
clk div disp:



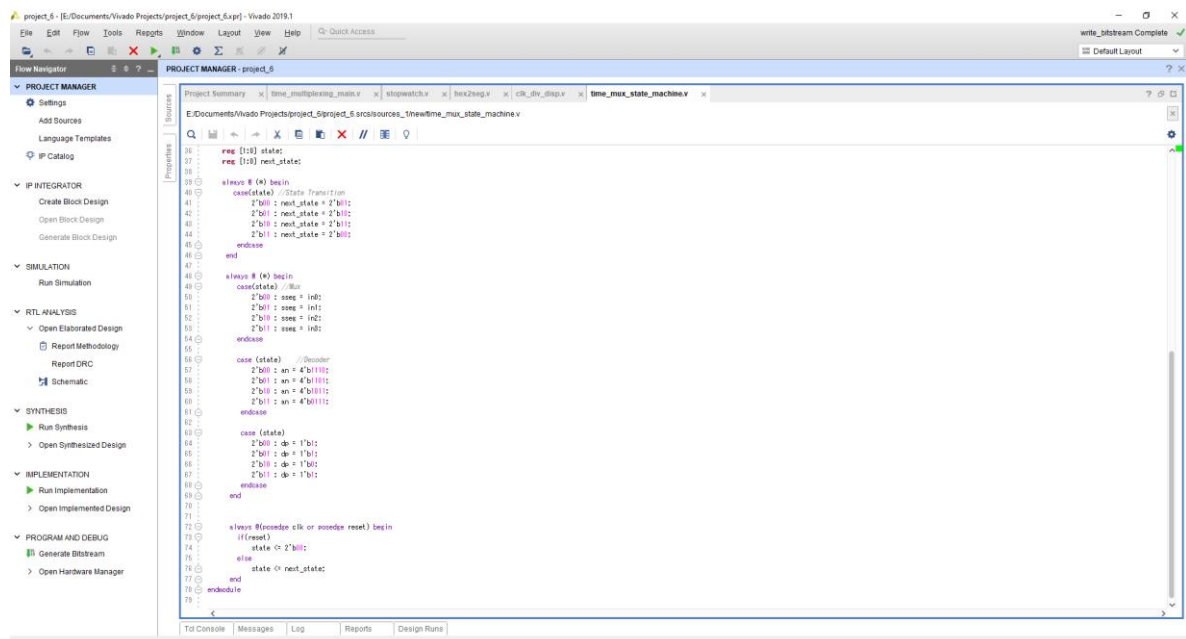
hex2seg:



time mux state machine:



```
24 module time_mux_state_machine(
25     input clk,
26     input reset,
27     input [8:0] ind,
28     input [8:0] ind2,
29     input [8:0] ind3,
30     output reg [3:0] an,
31     output reg [8:0] sreg,
32     output reg [8:0] sreg2,
33 );
34
35 reg [1:0] state;
36 reg [1:0] next_state;
37
38
39 always @(clk) begin
40     case(state) //State Transition
41         2'b01 : next_state = 2'b11;
42         2'b11 : next_state = 2'b01;
43         2'b10 : next_state = 2'b11;
44         2'b11 : next_state = 2'b11;
45     endcase
46 end
47
48 always @(clk) begin
49     case(state) //Mux
50         2'b01 : sreg = ind;
51         2'b11 : sreg = ind2;
52         2'b10 : sreg = ind2;
53         2'b11 : sreg = ind;
54     endcase
55
56     case (state) //Decoder
57         2'b01 : an = 4'b1101;
58         2'b11 : an = 4'b1010;
59         2'b10 : an = 4'b0110;
60         2'b11 : an = 4'b0111;
61     endcase
62
63     case (state)
64         2'b01 : dp = 1'b1;
65         2'b11 : dp = 1'b0;
66         2'b10 : dp = 1'b0;
67         2'b11 : dp = 1'b0;
68     endcase
69 end
```



```
36 reg [1:0] state;
37 reg [1:0] next_state;
38
39 always @(clk) begin
40     case(state) //State Transition
41         2'b01 : next_state = 2'b11;
42         2'b11 : next_state = 2'b01;
43         2'b10 : next_state = 2'b11;
44         2'b11 : next_state = 2'b11;
45     endcase
46 end
47
48 always @(clk) begin
49     case(state) //Mux
50         2'b01 : sreg = ind;
51         2'b11 : sreg = ind2;
52         2'b10 : sreg = ind2;
53         2'b11 : sreg = ind;
54     endcase
55
56     case (state) //Decoder
57         2'b01 : an = 4'b1101;
58         2'b11 : an = 4'b1010;
59         2'b10 : an = 4'b0110;
60         2'b11 : an = 4'b0111;
61     endcase
62
63     case (state)
64         2'b01 : dp = 1'b1;
65         2'b11 : dp = 1'b0;
66         2'b10 : dp = 1'b0;
67         2'b11 : dp = 1'b0;
68     endcase
69 end
70
71
72 always @(posedge clk or posedge reset) begin
73     if(reset)
74         state <= 2'b01;
75     else
76         state <= next_state;
77 end
78 endmodule
79
```


time multiplexing main:

The screenshot shows the Vivado IDE interface. The top menu bar includes File, Edit, Flow, Tools, Reports, Window, Layout, View, and Help. The top toolbar contains icons for opening files, saving, undo, redo, and other standard editing functions. The left sidebar shows the Project Navigator with the Project Manager and Flow Navigator tabs. The main workspace displays the source code for 't1wime_multiplexing_main.v'. The code is a Verilog module for a 4-to-1 multiplexer. It includes input registers for 'clk', 'start', 'reset', 'in0', 'in1', 'in2', and 'in3'. It also has output registers for 'out0', 'out1', 'out2', and 'out3'. The module uses a 'mux4to1' component from the 'xilinx_ip' library. The code is organized into sections: initialization, input/output logic, and a main loop. Comments indicate that the code is a simplified version of a more complex one, and it includes a note about the 'mux4to1' component.

```

1 // Module instantiation of the multiplexer
2 // Replace mux4to1 with clk for simulation and vice versa
3
4 module t1wime_multiplexing_main
5
6     input clk,
7     input start,
8     input reset,
9     input [3:0] in,
10    output [3:0] out,
11    output [3:0] reg;
12
13    wire [3:0] in0, in1, in2, in3;
14    wire [3:0] reg;
15    wire [3:0] reg_d;
16    wire [3:0] reg_d1;
17    wire [3:0] reg_d2;
18    wire [3:0] reg_d3;
19    wire [3:0] reg_d4;
20    wire [3:0] reg_d5;
21    wire [3:0] reg_d6;
22    wire [3:0] reg_d7;
23    wire [3:0] reg_d8;
24    wire [3:0] reg_d9;
25    wire [3:0] reg_d10;
26    wire [3:0] reg_d11;
27    wire [3:0] reg_d12;
28    wire [3:0] reg_d13;
29    wire [3:0] reg_d14;
30    wire [3:0] reg_d15;
31    wire [3:0] reg_d16;
32    wire [3:0] reg_d17;
33    wire [3:0] reg_d18;
34    wire [3:0] reg_d19;
35    wire [3:0] reg_d20;
36    wire [3:0] reg_d21;
37    wire [3:0] reg_d22;
38    wire [3:0] reg_d23;
39    wire [3:0] reg_d24;
40    wire [3:0] reg_d25;
41    wire [3:0] reg_d26;
42    wire [3:0] reg_d27;
43    wire [3:0] reg_d28;
44    wire [3:0] reg_d29;
45    wire [3:0] reg_d30;
46    wire [3:0] reg_d31;
47    wire [3:0] reg_d32;
48    wire [3:0] reg_d33;
49    wire [3:0] reg_d34;
50    wire [3:0] reg_d35;
51    wire [3:0] reg_d36;
52    wire [3:0] reg_d37;
53    wire [3:0] reg_d38;
54    wire [3:0] reg_d39;
55    wire [3:0] reg_d40;
56    wire [3:0] reg_d41;
57    wire [3:0] reg_d42;
58    wire [3:0] reg_d43;
59    wire [3:0] reg_d44;
60    wire [3:0] reg_d45;
61    wire [3:0] reg_d46;
62    wire [3:0] reg_d47;
63    wire [3:0] reg_d48;
64    wire [3:0] reg_d49;
65    wire [3:0] reg_d50;
66    wire [3:0] reg_d51;
67    wire [3:0] reg_d52;
68    wire [3:0] reg_d53;
69    wire [3:0] reg_d54;
70    wire [3:0] reg_d55;
71    wire [3:0] reg_d56;
72    wire [3:0] reg_d57;
73    wire [3:0] reg_d58;
74    wire [3:0] reg_d59;
75    wire [3:0] reg_d60;
76    wire [3:0] reg_d61;
77    wire [3:0] reg_d62;
78    wire [3:0] reg_d63;
79    wire [3:0] reg_d64;
80    wire [3:0] reg_d65;
81    wire [3:0] reg_d66;
82    wire [3:0] reg_d67;
83    wire [3:0] reg_d68;
84    wire [3:0] reg_d69;
85    wire [3:0] reg_d70;
86    wire [3:0] reg_d71;
87    wire [3:0] reg_d72;
88    wire [3:0] reg_d73;
89    wire [3:0] reg_d74;
90    wire [3:0] reg_d75;
91    wire [3:0] reg_d76;
92    wire [3:0] reg_d77;
93    wire [3:0] reg_d78;
94    wire [3:0] reg_d79;
95    wire [3:0] reg_d80;
96    wire [3:0] reg_d81;
97    wire [3:0] reg_d82;
98    wire [3:0] reg_d83;
99    wire [3:0] reg_d84;
100   wire [3:0] reg_d85;
101   wire [3:0] reg_d86;
102   wire [3:0] reg_d87;
103   wire [3:0] reg_d88;
104   wire [3:0] reg_d89;
105   wire [3:0] reg_d90;
106   wire [3:0] reg_d91;
107   wire [3:0] reg_d92;
108   wire [3:0] reg_d93;
109   wire [3:0] reg_d94;
110   wire [3:0] reg_d95;
111   wire [3:0] reg_d96;
112   wire [3:0] reg_d97;
113   wire [3:0] reg_d98;
114   wire [3:0] reg_d99;
115   wire [3:0] reg_d100;
116   wire [3:0] reg_d101;
117   wire [3:0] reg_d102;
118   wire [3:0] reg_d103;
119   wire [3:0] reg_d104;
120   wire [3:0] reg_d105;
121   wire [3:0] reg_d106;
122   wire [3:0] reg_d107;
123   wire [3:0] reg_d108;
124   wire [3:0] reg_d109;
125   wire [3:0] reg_d110;
126   wire [3:0] reg_d111;
127   wire [3:0] reg_d112;
128   wire [3:0] reg_d113;
129   wire [3:0] reg_d114;
130   wire [3:0] reg_d115;
131   wire [3:0] reg_d116;
132   wire [3:0] reg_d117;
133   wire [3:0] reg_d118;
134   wire [3:0] reg_d119;
135   wire [3:0] reg_d120;
136   wire [3:0] reg_d121;
137   wire [3:0] reg_d122;
138   wire [3:0] reg_d123;
139   wire [3:0] reg_d124;
140   wire [3:0] reg_d125;
141   wire [3:0] reg_d126;
142   wire [3:0] reg_d127;
143   wire [3:0] reg_d128;
144   wire [3:0] reg_d129;
145   wire [3:0] reg_d130;
146   wire [3:0] reg_d131;
147   wire [3:0] reg_d132;
148   wire [3:0] reg_d133;
149   wire [3:0] reg_d134;
150   wire [3:0] reg_d135;
151   wire [3:0] reg_d136;
152   wire [3:0] reg_d137;
153   wire [3:0] reg_d138;
154   wire [3:0] reg_d139;
155   wire [3:0] reg_d140;
156   wire [3:0] reg_d141;
157   wire [3:0] reg_d142;
158   wire [3:0] reg_d143;
159   wire [3:0] reg_d144;
160   wire [3:0] reg_d145;
161   wire [3:0] reg_d146;
162   wire [3:0] reg_d147;
163   wire [3:0] reg_d148;
164   wire [3:0] reg_d149;
165   wire [3:0] reg_d150;
166   wire [3:0] reg_d151;
167   wire [3:0] reg_d152;
168   wire [3:0] reg_d153;
169   wire [3:0] reg_d154;
170   wire [3:0] reg_d155;
171   wire [3:0] reg_d156;
172   wire [3:0] reg_d157;
173   wire [3:0] reg_d158;
174   wire [3:0] reg_d159;
175   wire [3:0] reg_d160;
176   wire [3:0] reg_d161;
177   wire [3:0] reg_d162;
178   wire [3:0] reg_d163;
179   wire [3:0] reg_d164;
180   wire [3:0] reg_d165;
181   wire [3:0] reg_d166;
182   wire [3:0] reg_d167;
183   wire [3:0] reg_d168;
184   wire [3:0] reg_d169;
185   wire [3:0] reg_d170;
186   wire [3:0] reg_d171;
187   wire [3:0] reg_d172;
188   wire [3:0] reg_d173;
189   wire [3:0] reg_d174;
190   wire [3:0] reg_d175;
191   wire [3:0] reg_d176;
192   wire [3:0] reg_d177;
193   wire [3:0] reg_d178;
194   wire [3:0] reg_d179;
195   wire [3:0] reg_d180;
196   wire [3:0] reg_d181;
197   wire [3:0] reg_d182;
198   wire [3:0] reg_d183;
199   wire [3:0] reg_d184;
200   wire [3:0] reg_d185;
201   wire [3:0] reg_d186;
202   wire [3:0] reg_d187;
203   wire [3:0] reg_d188;
204   wire [3:0] reg_d189;
205   wire [3:0] reg_d190;
206   wire [3:0] reg_d191;
207   wire [3:0] reg_d192;
208   wire [3:0] reg_d193;
209   wire [3:0] reg_d194;
210   wire [3:0] reg_d195;
211   wire [3:0] reg_d196;
212   wire [3:0] reg_d197;
213   wire [3:0] reg_d198;
214   wire [3:0] reg_d199;
215   wire [3:0] reg_d200;
216   wire [3:0] reg_d201;
217   wire [3:0] reg_d202;
218   wire [3:0] reg_d203;
219   wire [3:0] reg_d204;
220   wire [3:0] reg_d205;
221   wire [3:0] reg_d206;
222   wire [3:0] reg_d207;
223   wire [3:0] reg_d208;
224   wire [3:0] reg_d209;
225   wire [3:0] reg_d210;
226   wire [3:0] reg_d211;
227   wire [3:0] reg_d212;
228   wire [3:0] reg_d213;
229   wire [3:0] reg_d214;
230   wire [3:0] reg_d215;
231   wire [3:0] reg_d216;
232   wire [3:0] reg_d217;
233   wire [3:0] reg_d218;
234   wire [3:0] reg_d219;
235   wire [3:0] reg_d220;
236   wire
```

The screenshot shows the Vivado IDE interface. On the left is the Project Manager pane with a tree view of project components. The main editor window displays the Verilog code for 'time_muxing_main.v'. The code is a module for a 4-to-1 multiplexer, including module instantiations for a decoder, a clock divider, and the multiplexer itself. Comments indicate how to use the module for simulation and implementation.

```

1  wire mux_out;
2  wire time_clk;
3  wire [3:0] reg_d0; //count for right most digit
4  wire [3:0] reg_d1; //count for second right most digit
5  wire [3:0] reg_d2; //count for second left most digit
6  wire [3:0] reg_d3; //count for left most digit
7
8  // Module instantiation of hex2to4mux decoder
9  hex2seq c1 (.2B(reg_d0), .out(m0));
10 hex2seq c2 (.2B(reg_d1), .out(m1));
11 hex2seq c3 (.2B(reg_d2), .out(m2));
12 hex2seq c4 (.2B(reg_d3), .out(m3));
13
14 // Module instantiation of clock divider
15 // same functionality as the clk_div before, but may have a different width requirement
16 clk_div_4m d3 (.clk(clk), .reset(reset), .mux_clk(mux_clk), .time_clk(time_clk));
17
18 stimulus s1 {
19     clk(time_clk),
20     start_top(start_top),
21     reset(reset),
22     mode(mode[1:0]),
23     en(en[7:0]),
24     reg_d0(reg_d0),
25     reg_d1(reg_d1),
26     reg_d2(reg_d2),
27     reg_d3(reg_d3);
28 }
29
30 // Module instantiation of the multiplexer
31 //replace mux_4k with clk for simulation, and vice versa
32 time_mux_state_machine d1 (
33     .clk(time_clk),
34     .reset(reset),
35     .m0(m0),
36     .m1(m1),
37     .m2(m2),
38     .m3(m3),
39     .en(en),
40     .mux_out(mux_out));
41
42 module
43
44
45

```

Testbench:

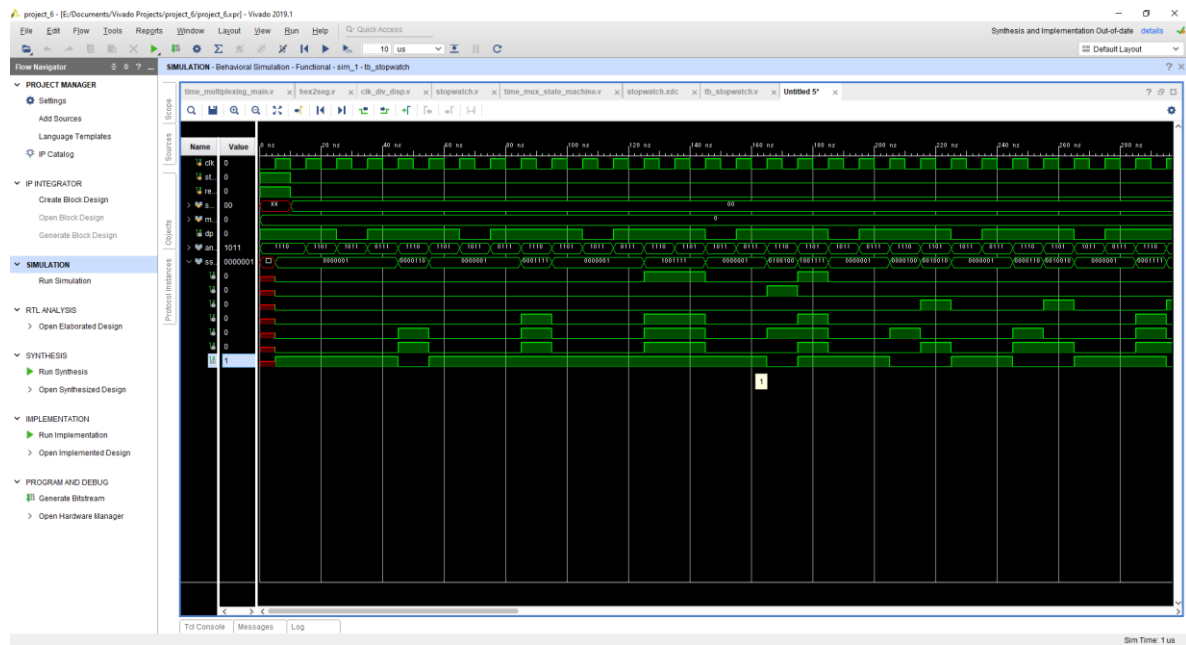
```
tb_stopwatch.v
E:\Documents\Hado Projects\project_6\src\sim_tnewtb_stopwatch.v

23 module tb_stopwatch;
24 reg clk;
25 reg startstop;
26 reg reset;
27 reg [7:0] cnt;
28 reg [1:0] mode;
29 wire do;
30 wire [2:0] enc;
31 wire [1:0] sseq;
32
33 time multileveling_main ut (
34     .clk(clk),
35     .startstop(startstop),
36     .reset(reset),
37     .en(en),
38     .mode(mode),
39     .do(do),
40     .enc(enc),
41     .sseq(sseq)
42 );
43
44 initial begin
45
46     //Mode 0:
47     //clk = 0;
48     //mode = 0;
49     //startstop = 1;
50     //reset = 1;
51     //RST
52     //startstop = 0;
53     //reset = 0;
54     //sw = $ 000000000;
55
56     //Mode 1:
57     //clk = 0;
58     //sw = $ 00001000;
59     //mode = 1;
60     //startstop = 1;
61     //reset = 1;
62     //RST
63     //startstop = 0;
64     //reset = 0;
65
66     //Mode 2:
67     //clk = 0;
68     //mode = 2;
69     //startstop = 1;
70     //reset = 1;
71     //RST
72     //startstop = 0;
73     //reset = 0;
74
75     //Mode 3:
76
77 end
```

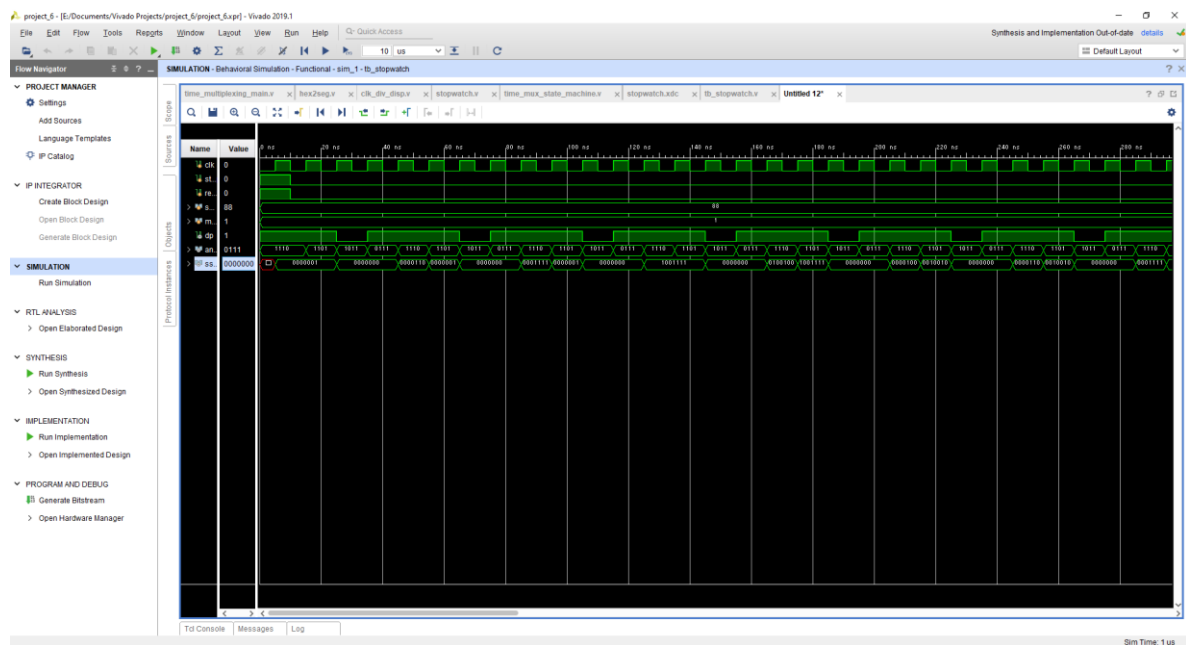
```
tb_stopwatch.v
E:\Documents\Hado Projects\project_6\src\sim_tnewtb_stopwatch.v

40 .en(en),
41 .sseq(sseq)
42 );
43
44 initial begin
45
46     //Mode 0:
47     //clk = 0;
48     //mode = 0;
49     //startstop = 1;
50     //reset = 1;
51     //RST
52     //startstop = 0;
53     //reset = 0;
54     //sw = $ 000000000;
55
56     //Mode 1:
57     //clk = 0;
58     //sw = $ 00001000;
59     //mode = 1;
60     //startstop = 1;
61     //reset = 1;
62     //RST
63     //startstop = 0;
64     //reset = 0;
65
66     //Mode 2:
67     //clk = 0;
68     //mode = 2;
69     //startstop = 1;
70     //reset = 1;
71     //RST
72     //startstop = 0;
73     //reset = 0;
74
75     //Mode 3:
76     //clk = 0;
77     //sw = $ 000000001;
78     //mode = 3;
79     //startstop = 1;
80     //reset = 1;
81     //RST
82     //startstop = 0;
83     //reset = 0;
84
85 end
86
87 always
88 @ 55 clk = ~clk;
89
90 endmodule
```

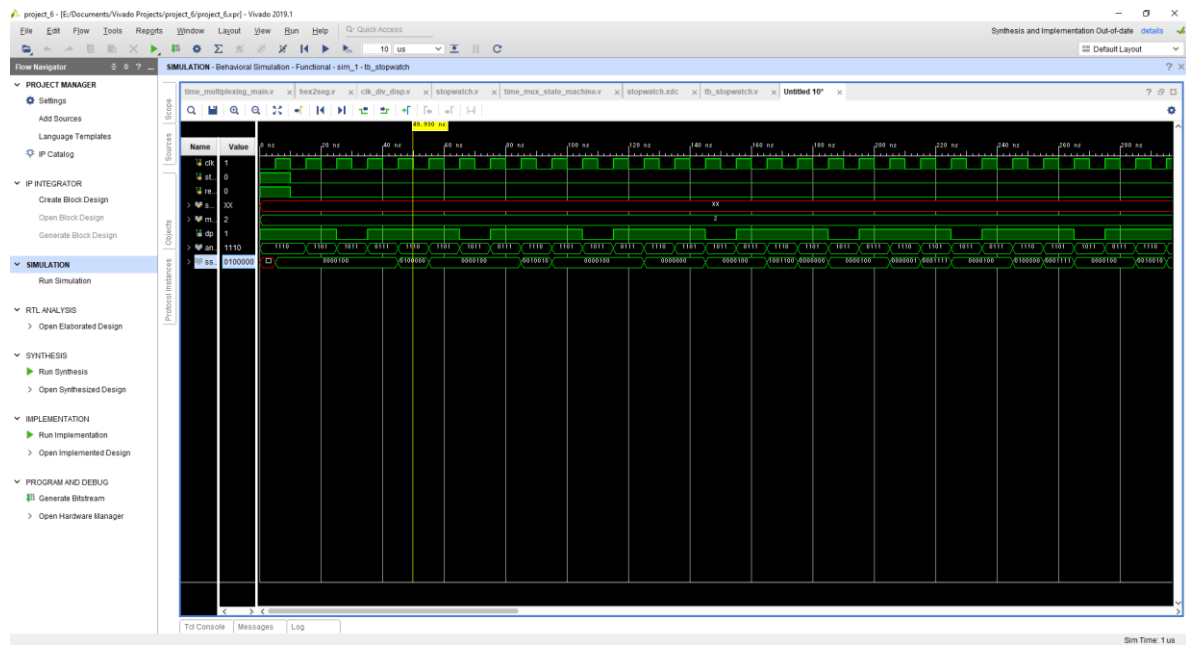
Mode 1:



Mode 2:



Mode 3:



Mode 4:

