TOP VHDL Code

```
1
2   LIBRARY ieee;
3   USE ieee.std_logic_1164.ALL;
4   USE ieee.numeric_std.ALL;
5
6   ENTITY LogicalStep_Lab5_top IS
7      PORT
8      (
9      clkin_50    : in  std_logic;                     -- The 50 MHz FPGA Clockinput
10     rst_n       : in  std_logic;                     -- The RESET input (ACTIVE LOW)
11     pb          : in  std_logic_vector(3 downto 0); -- The push-button inputs (ACTIVE LOW)
12     sw          : in  std_logic_vector(7 downto 0); -- The switch inputs
13     leds        : out std_logic_vector(7 downto 0); -- for displaying the switch content
14     seg7_data   : out std_logic_vector(6 downto 0); -- 7-bit outputs to a 7-segment
15     seg7_char1  : out std_logic;                     -- seg7 digi selectors
16     seg7_char2  : out std_logic                      -- seg7 digi selectors
17     );
18  END LogicalStep_Lab5_top;
19
20  ARCHITECTURE SimpleCircuit OF LogicalStep_Lab5_top IS
21
22
23     component cycle_generator port (
24           clkin            : in  std_logic;
25           rst_n            : in  std_logic;
26           modulo           : in  integer;
27           strobe_out       : out std_logic;
28           full_cycle_out   : out std_logic
29     );
30     end component;
31
32     component segment7_mux port (
33           clk         : in  std_logic := '0';
34           DIN2        : in  std_logic_vector(6 downto 0);
35           DIN1        : in  std_logic_vector(6 downto 0);
36           DOUT        : out std_logic_vector(6 downto 0);
37           DIG2        : out std_logic;
38           DIG1        : out std_logic
39     );
40     end component;
41
42     component Moore_SM IS Port
43     (
44        enable,rst_n, clk    : IN std_logic;
45        night,reduced        : IN std_logic;--night mode and reduced systems mode
46        EW_button, NS_button :  IN std_logic;
47        EW_clear, NS_clear   : out std_logic;
```

```
48          cur_val     : out std_logic_vector(3 downto 0)
49      );
50      end component;
51
52      component Decoder is port (
53          state       : in std_logic_vector(3 downto 0);
54          clken5        : in std_logic;
55          clken1      : in std_logic;
56          NS_Decoded  : out std_logic_vector(6 downto 0);
57          EW_Decoded  : out std_logic_vector(6 downto 0)
58      );
59      end component;
60
61      component Sync is port (
62      data_in         : in std_logic;
63      clk             : in std_logic;
64      rst_n       : in std_logic;
65      data_out    : out std_logic
66      );
67      end component;
68
69      component InputLatch is port (
70      data_in         : in std_logic;
71      clear       : in std_logic;
72      clk             : in std_logic;
73      enable          : in std_logic;
74      rst_n       : in std_logic;
75      data_out    : out std_logic
76      );
77      end component;
78
79  ------------------------------------------------------------------------------------------------
80      CONSTANT SIM                    :  boolean :=FALSE;
81
82      CONSTANT CNTR1_modulo           :  integer := 25000000;       -- modulo count for 1Hz cycle generator 1 with 50Mhz clocking input
83      CONSTANT CNTR2_modulo           :  integer := 5000000;     -- modulo count for 5Hz cycle generator 2 with 50Mhz clocking input
84      CONSTANT CNTR1_modulo_sim       :  integer := 199;            -- modulo count for cycle generator 1 during simulation
85      CONSTANT CNTR2_modulo_sim       :  integer :=  39;            -- modulo count for cycle generator 2 during simulation
86
87      SIGNAL CNTR1_modulo_value       :  integer ;                  -- modulo count for cycle generator 1
88      SIGNAL CNTR2_modulo_value       :  integer ;                  -- modulo count for cycle generator 2
89
90      SIGNAL clken1,clken2            :  STD_LOGIC;                 -- clock enables 1 & 2
91
92      SIGNAL strobe1, strobe2         :  std_logic;                 -- strobes 1 & 2 with each one being 50% Duty Cycle
```

```
    SIGNAL NS_clear, EW_clear        : std_logic;
    SIGNAL syncInput1, syncInput2    :  std_logic;
    SIGNAL pbOut1, pbOut2            :std_logic;
    SIGNAL cur_state                 :std_logic_vector(3 downto 0);--Holds current state output from Moore_SM
    SIGNAL seg7_A, seg7_B            :  STD_LOGIC_VECTOR(6 downto 0); -- signals for inputs into seg7_mux.
    SIGNAL pb_bar0, pb_bar1          :std_logic;
    SIGNAL night, reduced            :std_logic;


BEGIN
-------------------------------------------------------------------------------------------------


MODULO_1_SELECTION:  CnTR1_modulo_value <= CNTR1_modulo when SIM = FALSE else CNTR1_modulo_sim;

MODULO_2_SELECTION:  CNTR2_modulo_value <= CNTR2_modulo when SIM = FALSE else CNTR2_modulo_sim;



-------------------------------------------------------------------------------------------------
-- Component Hook-up:
pb_bar0 <= NOT(pb(0));
pb_bar1 <=NOT(pb(1));
--1 Hz and 5 Hz cycle generators
GEN1:    cycle_generator port map(clkin_50, rst_n, CNTR1_modulo_value, strobe1, clken1);
GEN2:    cycle_generator port map(clkin_50, rst_n, CNTR2_modulo_value, strobe2, clken2);

--push button inputs
INST1: Sync port map(pb_bar0,clkin_50,rst_n,syncInput1);
--saves the last pb input until cleared
INST2: InputLatch port map(syncInput1, EW_clear, clkin_50, clken2,rst_n,pbOut1);--ADD CLEAR FROM MOORE

--same but for pb[1]
INST3: Sync port map(pb_bar1,clkin_50,rst_n,syncInput2);
INST4: InputLatch port map(syncInput2, NS_clear, clkin_50, clken2,rst_n,pbOut2);--ADD CLEAR FROM MOORE

--synchronizes the switch input with the clock
INST5: Sync port map(sw(0),clkin_50,rst_n,night );
INST6: Sync port map(sw(1),clkin_50,rst_n, reduced);

--instantiate the moore_sm, the decoder, and the seven segment display
INST7: Moore_SM port map(clken1,rst_n, clkin_50, night, reduced, pbOut1,pbOut2,EW_clear, NS_clear, cur_state);--Takes input full-cycle 1 Hz clock so the Moore_SM updates every 1 second
INST8: Decoder port map (cur_state,strobe2,strobe1, seg7_A,seg7_B);--Takes the current state from the Moore_SM to decode into the appropriate light signal
INST9: segment7_mux port map (clkin_50, seg7_B,seg7_A,seg7_data, seg7_char2, seg7_char1);

leds(1 downto 0) <= Strobe1 & Strobe2;
--leds(3 downto 2) <= clken1 & clken2;
leds(7) <= pbOut1 OR pbOut2;
leds(6) <= night OR reduced;
leds(5 downto 2) <= cur_state;



END SimpleCircuit;

```

MOORE_SM VHDL

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

Entity Moore_SM IS Port
(
    enable,rst_n, clk    : IN std_logic;--Enable from the 1hz clock, and clk from 50MHz clock
    night,reduced        : IN std_logic;--night mode and reduced systems mode
    EW_button, NS_button : IN std_logic;
    EW_clear, NS_clear   : out std_logic;
    cur_val     : out std_logic_vector(3 downto 0)
);
end Moore_SM;

Architecture MSM of Moore_SM is



 TYPE STATE_NAMES IS (S0, S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11, S12, S13, S14, S15, S16, S17); -- list all the STATES


 SIGNAL current_state, next_state   :  STATE_NAMES;   -- signals of type STATE_NAMES
 Signal compareRes:     std_logic_vector (2 downto 0);--Groups the output of the comparator (A>B,A=B,A<B)
 Signal decoded:     std_logic_vector (3 downto 0);

BEGIN

 -------------------------------------------------------------------------------
 --State Machine:
 -------------------------------------------------------------------------------

-- REGISTER LOGIC PROCESS
 -- add clock and any related inputs for state machine register section into Sensitivity List

Register_Section: PROCESS ( rst_n, clk,enable, next_state)  -- this process synchronizes the activity to a clock
 BEGIN
    IF (rst_n = '0') THEN
        current_state <= S0;
    ELSIF( rising_edge(clk) and enable = '1') THEN
        current_state <= next_State;
    ELSE
        current_state <= current_state;
    END IF;
 END PROCESS;


-- TRANSTION LOGIC PROCESS (to be combinational only)
 -- add all transition inputs for state machine into Transition section Sensitivity List
 -- make sure that all conditional statement options are complete otherwise VHDL will infer LATCHES.
 --INST1: compx4 port map(target(3 downto 0),decoded(3 downto 0),compareRes(2 downto 0)); --target>decoded, target=decoded, target< decoded: determine if we need to count down, up, or hold


Transition_Section: PROCESS (current_state,enable)
 BEGIN
```

```
61  -- increments everystate when the clken1 is high
62  -- when pb[0] is pressed states 0 to 4 will skip forward to state 6, or  if pb[1] is pressed, states 8 to 12 skip forward to state 14
63  --when switch 0 or 1 are on, it either state 7 or 15 will skip to their respective hold states 16 and 17 until both switches are off
64      CASE current_state IS
65          WHEN S0 =>
66          NS_clear <= '0';
67             IF( enable = '1') THEN
68                IF(EW_button = '1') THEN
69                   next_state <= S6;
70                ELSE
71                   next_state <= S1;
72                END IF;
73             ELSE
74                next_state <= S0;
75             END IF;
76          WHEN S1 =>
77             IF( enable = '1') THEN
78                IF(EW_button = '1') THEN
79                   next_state <= S6;
80                ELSE
81                   next_state <= S2;
82                END IF;
83             ELSE
84                next_state <= S1;
85             END IF;
86          WHEN S2 =>
87             IF( enable = '1') THEN
88                IF(EW_button = '1') THEN
89                   next_state <= S6;
90                ELSE
91                   next_state <= S3;
92                END IF;
93             ELSE
94                next_state <= S2;
95             END IF;
96          WHEN S3 =>
97             IF( enable = '1') THEN
98                IF(EW_button = '1') THEN
99                   next_state <= S6;
100               ELSE
101                  next_state <= S4;
102               END IF;
103            ELSE
104               next_state <= S3;
105            END IF;
106         WHEN S4 =>
107            IF( enable = '1') THEN
108               IF(EW_button = '1') THEN
109                  next_state <= S6;
110               ELSE
111                  next_state <= S5;
112               END IF;
113            ELSE
114               next_state <= S4;
115            END IF;
116         WHEN S5 =>
117            IF( enable = '1') THEN
118               next_state <= S6;
119            ELSE
120               next_state <= S5;
```

```
121             END IF;
122          WHEN S6 =>
123             IF( enable = '1') THEN
124                IF(EW_button = '1') THEN
125                   EW_clear <= '1';
126                END IF;
127                next_state <= S7;
128             ELSE
129                next_state <= S6;
130             END IF;
131          WHEN S7 =>
132             EW_clear <= '0';
133             IF( enable = '1') THEN
134                IF(reduced = '1') THEN
135                   next_state <= S17;
136                ELSIF (night = '1') THEN
137                   next_state <= S16;
138                ELSE
139                   next_state <= S8;
140                END IF;
141             ELSE
142                next_state <= S7;
143             END IF;
144          WHEN S8 =>
145             IF( enable = '1') THEN
146                IF(NS_button = '1') THEN
147                   next_state <= S14;
148                ELSE
149                   next_state <= S9;
150                END IF;
151             ELSE
152                next_state <= S8;
153             END IF;
154          WHEN S9 =>
155             IF( enable = '1') THEN
156                IF(NS_button = '1') THEN
157                   next_state <= S14;
158                ELSE
159                   next_state <= S10;
160                END IF;
161             ELSE
162                next_state <= S9;
163             END IF;
164          WHEN S10 =>
165             IF( enable = '1') THEN
166                IF(NS_button = '1') THEN
167                   next_state <= S14;
168                ELSE
169                   next_state <= S11;
170                END IF;
171             ELSE
172                next_state <= S10;
173             END IF;
174          WHEN S11 =>
175             IF( enable = '1') THEN
176                IF(NS_button = '1') THEN
177                   next_state <= S14;
178                ELSE
179                   next_state <= S12;
180                END IF;
```

```
181             ELSE
182                 next_state <= S11;
183             END IF;
184         WHEN S12 =>
185             IF( enable = '1') THEN
186                 IF(NS_button = '1') THEN
187                     next_state <= S14;
188                 ELSE
189                     next_state <= S13;
190                 END IF;
191             ELSE
192                 next_state <= S12;
193             END IF;
194         WHEN S13 =>
195             IF( enable = '1') THEN
196                 next_state <= S14;
197             ELSE
198                 next_state <= S13;
199             END IF;
200         WHEN S14 =>
201             IF( enable = '1') THEN
202                 next_state <= S15;
203             ELSE
204                 next_state <= S14;
205             END IF;
206         WHEN S15 =>
207             IF( enable = '1') THEN
208                 IF(NS_button = '1') THEN
209                     NS_clear <= '1';
210                 END IF;
211                 IF(reduced = '1') THEN
212                     next_state <= S17;
213                 ELSIF (night = '1') THEN
214                     next_state <= S16;
215                 ELSE
216                     next_state <= S0;
217                 END IF;
218             ELSE
219                 next_state <= S15;
220             END IF;
221         WHEN S16 =>
222             IF(reduced = '1') THEN
223                 next_state <= S17;
224             ELSIF(night = '0') THEN
225                 next_state <= S6;
226             ELSE
227                 next_state<=S16;
228             END IF;
229         WHEN S17 =>
230             IF(reduced = '0') THEN
231                 IF(night = '1') THEN
232                     next_state <= S16;
233                 ELSE
234                     next_state <= S6;
235                 END IF;
236             ELSE
237                 next_state <= S17;
238             END IF;
239         WHEN others =>
240             next_state <= S0;
```

```
241         END CASE;
242     END PROCESS;
243
244
245
246     --Decoder to convert the decimal state into binary
247     Decoder_Section: with Current_State select
248       decoded <= "0000" when S0,
249       "0001" when S1,
250       "0010" when S2,
251       "0011" when S3,
252       "0100" when S4,
253       "0101" when S5,
254       "0110" when S6,
255       "0111" when S7,
256       "1000" when S8,
257       "1001" when S9,
258       "1010" when S10,
259       "1011" when S11,
260       "1011" when S12,--state 12 and 13 are replaced with states 16 and 17 since the state's out put is the same.
261       "1011" when S13,
262       "1110" when S14,
263       "1111" when S15,
264       "1100" when S16,--replaces state 12
265       "1101" when S17,--replaces state 13
266       "0000" when others;
267
268   cur_val <= decoded;--Output the current state to be displayed on the 7 segment
269
270   END ARCHITECTURE MSM;
271
```

DECODER VHDL

```
 1  library ieee;
 2  use ieee.std_logic_1164.all;
 3  use ieee.numeric_std.all;
 4
 5  ----------------------------------------------------------------------------
 6
 7
 8  entity Decoder is port (
 9     state       : in std_logic_vector(3 downto 0);
10     clken5      : in std_logic;
11     clken1      : in std_logic;
12     NS_Decoded  : out std_logic_vector(6 downto 0);
13     EW_Decoded  : out std_logic_vector(6 downto 0)
14  );
15  end Decoder;
16
17  architecture Behavioral of Decoder is
18  signal NS : std_logic_vector(6 downto 0);
19  signal NS_Strobe :std_logic_vector(6 downto 0);
20  signal EW : std_logic_vector(6 downto 0);
21  signal EW_Strobe :std_logic_vector(6 downto 0);
22  begin
23
24
25
26        NS_Decoded <= NS OR NS_Strobe;
27        EW_Decoded <= EW OR EW_Strobe;
28
29     --Based on the current state, set the 7 segment for NS and EW accordingly, except for the flashing state
30     with state select          --GFEDCBA
31        NS                   <="0001000" when "0010",                 -- [2]      +---- a -----+
32                               "0001000" when "0011",     -- [3]       |           |
33                               "0001000" when "0100",     -- [4]       |           |
34                               "0001000" when "0101",     -- [5]       f           b
35                               "1000000" when "0110",     -- [6]       |           |
36                               "1000000" when "0111",     -- [7]       |           |
37                               "0000001" when "1000",     -- [8]       +---- g -----+
38                               "0000001" when "1001",     -- [9]       |           |
39                               "0000001" when "1010",     -- [A]       |           |
40                               "0000001" when "1011",     -- [b]       e           c
41                               "0001000" when "1100",     -- [C]       |           |
42                               --                         -- [d]       |           |
43                               "0000001" when "1110",     -- [E]       +---- d -----+
44                               "0000001" when "1111",     -- [F]
45                               "0000000" when others;     -- [ ]
46
47     with state select          --GFEDCBA
48     EW                   <= "0000001" when "0000",           -- [0]
49                               "0000001" when "0001",     -- [1]
50                               "0000001" when "0010",     -- [2]       +---- a -----+
51                               "0000001" when "0011",     -- [3]       |           |
52                               "0000001" when "0100",     -- [4]       |           |
53                               "0000001" when "0101",     -- [5]       f           b
54                               "0000001" when "0110",     -- [6]       |           |
55                               "0000001" when "0111",     -- [7]       |           |
56                               --                               +---- g -----+
57                               --                                 |           |
58                               "0001000" when "1010",     -- [A]       |           |
59                               "0001000" when "1011",     -- [b]       e           c
60                               "0000001" when "1100",     -- [C]       |           |
```

```
                                    --                      -- [d]      |             |
                                    "1000000" when "1110",  -- [E]      +----- d -----+
                                    "1000000" when "1111",  -- [F]
                                    "0000000" when others;  -- [ ]
    --This process is for the flashing state
    strobe: process(clken5,clken1, state) IS
    begin
    --If the clock is high then check if it's in the strobe state
    --checks the state, if it's in a state that requires strobing based off either the 5hz or the 1hz strobe depending on what the state requires

          IF(state = "0000" or state = "0001") THEN
             IF (clken5 = '1') THEN
                NS_Strobe <= "0001000";
             ELSE
                NS_Strobe <= "0000000";
             END IF;
          ELSIF (state = "1000" or state = "1001") THEN
             IF (clken5 = '1') THEN
                EW_Strobe<="0001000";
             ELSE
                EW_Strobe <= "0000000";
             END IF;
          ELSIF(state = "1101") THEN
             IF(clken1 = '1')THEN
                NS_Strobe <= "1000000";
                EW_Strobe<="0000001";
             ELSE
                NS_Strobe <= "0000000";
                EW_Strobe<="0000000";
             END IF;
          ELSE
             NS_Strobe <= "0000000";
             EW_Strobe<="0000000";
          END IF;

    end process;

 end architecture Behavioral;
```

SYNCHRONIZER VHDL

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Sync is port (
    data_in         : in std_logic;
    clk             : in std_logic;
    rst_n        : in std_logic;
    data_out     : out std_logic
);
end Sync;

architecture arch of Sync is
    component DFlipFlop is port (
        data_In         : in std_logic;
        clock        : in std_logic;
        rst_n        : in std_logic;
        data_Out     : out std_logic
    );
    end component;
signal DFF_Out :  std_logic;

begin
--prevents a metastable state from occuring by caputing the last recorded state from the first d-flipflop nd outputting that from the econd dflip flop
    INST1: DFlipFlop port map (data_In,clk,rst_n,DFF_Out);
    INST2: DFlipFlop port map( DFF_Out,clk,rst_n,data_Out);
end architecture arch;
```

CYCLE GENERATOR VHDL

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

Entity cycle_generator IS port (
          clkin            : in  std_logic;
          rst_n            : in  std_logic;
          modulo           : in  integer;
          strobe_out       : out std_logic;
          full_cycle_out   : out std_logic
   );
end entity;

ARCHITECTURE counter OF cycle_generator IS

   SIGNAL bin_counter               : UNSIGNED(31 DOWNTO 0);
   SIGNAL terminal_count            : std_logic;
   SIGNAL half_cycle, full_cycle    : std_logic;
   SIGNAL strobe                    : std_logic;

   BEGIN

   half_cycle <= terminal_count;
   full_cycle_out <= full_cycle;
   strobe_out <= strobe;


MODULO_COUNTING: PROCESS(clkin, rst_n) IS
   BEGIN

     IF (rst_n = '0') THEN
        bin_counter <= to_unsigned(modulo,32);
        terminal_count <= '0';

     ELSIF (rising_edge(clkin)) THEN               -- binary counter decrements on rising clock edge.

         IF(bin_counter = 0) THEN
                                                   -- when bin_counter reaches 0
            bin_counter <= to_unsigned(modulo,32); -- reload the (converted integer to 32 bit unsigned signal type) modulo value
            terminal_count <= '1';                 -- and output a terminal_count signal
         ELSE
            bin_counter <= bin_counter - 1;
            terminal_count <= '0';
         END IF;
     ELSE
         bin_counter <= bin_counter;
         terminal_count <= terminal_count;

     END IF;
   END PROCESS;


Strobe_gen: PROCESS(clkin, rst_n) IS         -- Strobe is with 50% duty cycle
   BEGIN
   --the strobe is toggled based on every half cycle
   IF(rst_n = '0') THEN
      strobe <= '0';
   ELSIF(rising_edge(clkin)) THEN
      IF(terminal_count = '1') THEN
         strobe <= not(strobe);
```

```
61      END IF;
62    ELSE
63      strobe <= strobe;
64    END IF;
65    END PROCESS;
66
67
68  CLKEN_GEN: PROCESS(clkin, rst_n) IS          -- full_cycle is one "clkin" cycle in duration and occure once for every two occurrences of half_cycle
69    BEGIN
70    --toggles full cycle when rising edge of the input clock and the terminal count being 1, on the next cycle the toggle is set back to 0
71    IF(rst_n = '0') THEN
72      full_cycle <= '0';
73    ELSIF(rising_edge(clkin)) THEN
74      IF(strobe = '0' and terminal_count = '1') THEN
75        full_cycle <= '1';
76      ELSE
77        full_cycle <= '0';
78      END IF;
79    ELSE
80      full_cycle<=full_cycle;
81    END IF;
82    END PROCESS;
83
84    END Architecture;
85
86
```

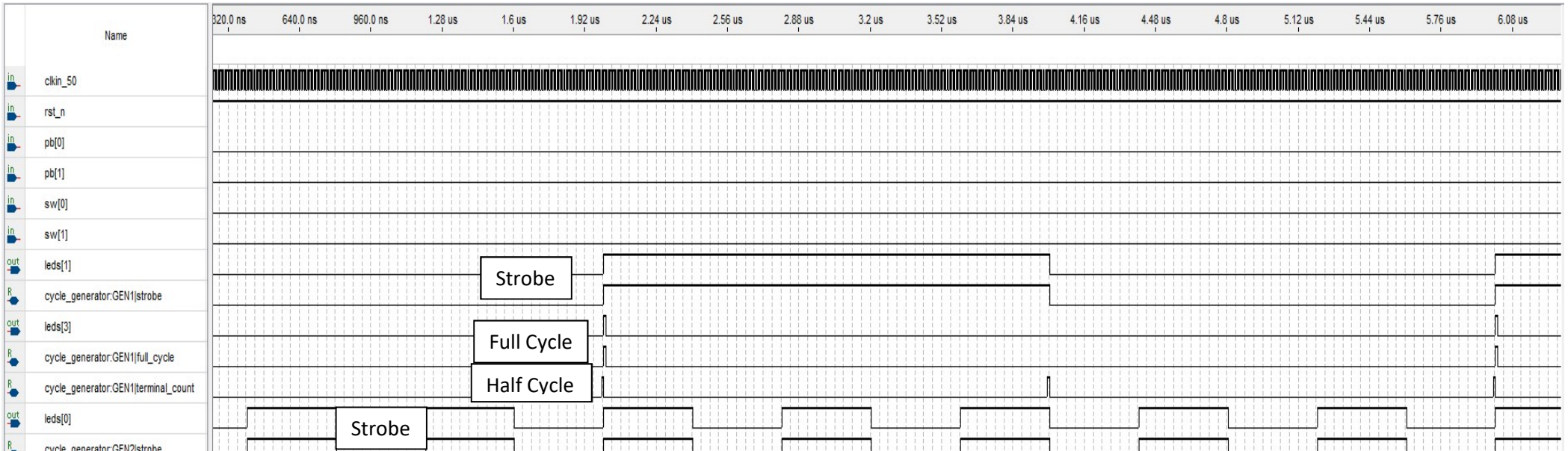
LATCH VHDL

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity InputLatch is port (
    data_in         : in std_logic;
    clear       : in std_logic;
    clk             : in std_logic;
    enable          : in std_logic;
    rst_n       : in std_logic;
    data_out    : out std_logic
);
end InputLatch;

architecture arch of InputLatch is
signal DFF_Out :  std_logic;
signal DFF_IN  :  std_logic;

begin
--holds the signal from either of the push buttons until the it recieves the clear signal from the statemachine
    DFF_IN <= NOT(clear) AND (data_in OR DFF_Out);

    DFF: Process (rst_n, clk,DFF_IN, enable)
    begin
       if(rst_n = '0' or clear = '1') THEN
          DFF_Out <= '0';
       elsif (rising_edge(clk) and enable = '1') THEN
          DFF_Out <= DFF_IN;
       else
          DFF_Out <= DFF_Out;
       end if;
    end process;
    data_out <= DFF_Out;

end architecture arch;
```

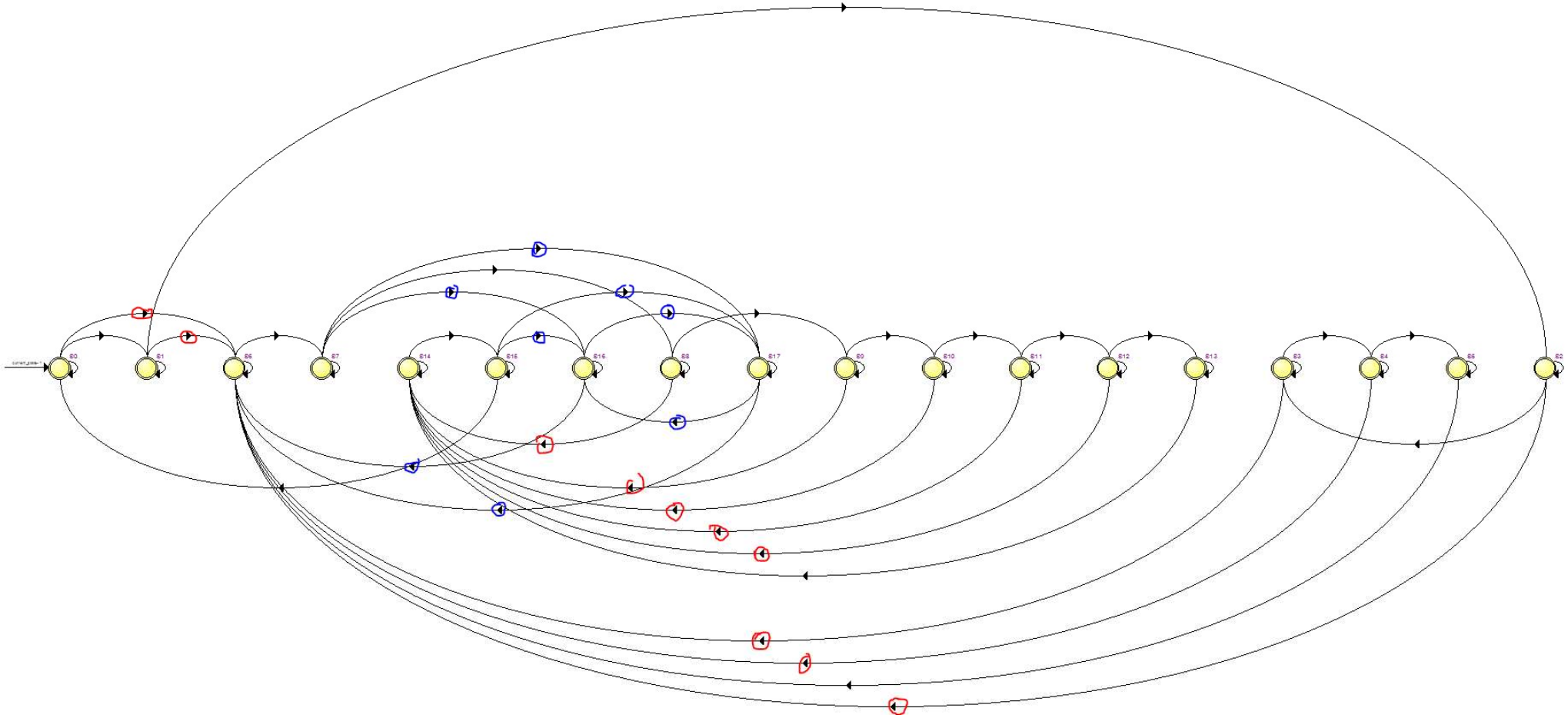
D-FLIPFLOP VHDL

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity DFlipFlop is port (
    data_In         : in std_logic;
    clock       : in std_logic;
    rst_n       : in std_logic;
    data_Out    : out std_logic
);
end DFlipFlop;

architecture arch of DFlipFlop is
    signal currData   :  std_logic;
begin
-- on the rising edge of the clock it outputs the input signal, or holds it otherwise
    logic: Process (rst_n, clock, data_In)
    begin
        if(rst_n = '0') THEN
            currData <= '0';
        elsif (rising_edge(clock)) THEN
            currData <= data_In;
        else
            currData <= currData;
        end if;
    end process;

    data_Out <= currData;

end architecture arch;
```

PART A SIMULATION



PART B,C,D STATE DIAGRAM



Any arrow circled in red is part C, blue is part D and anything uncircled is part B.