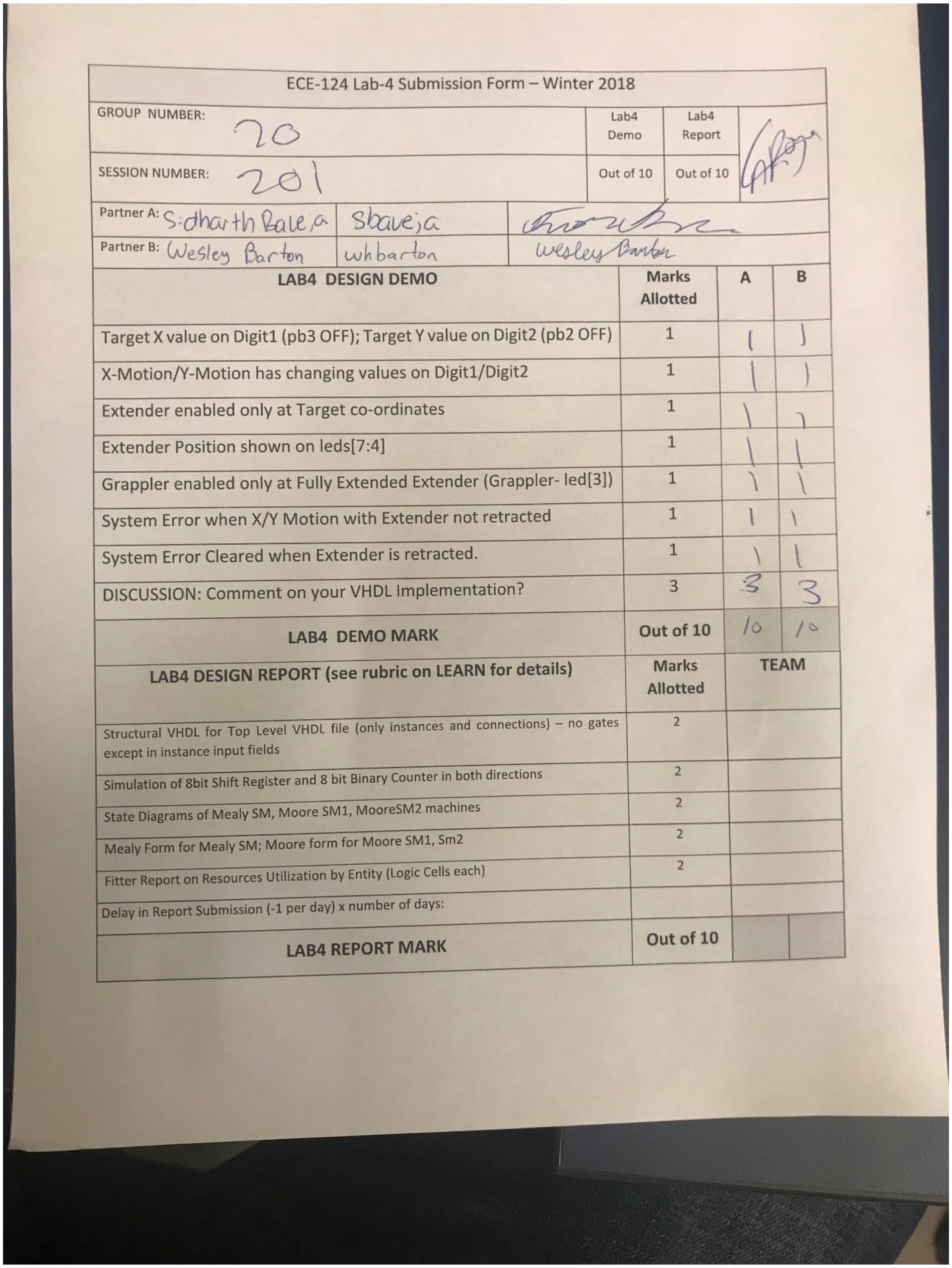


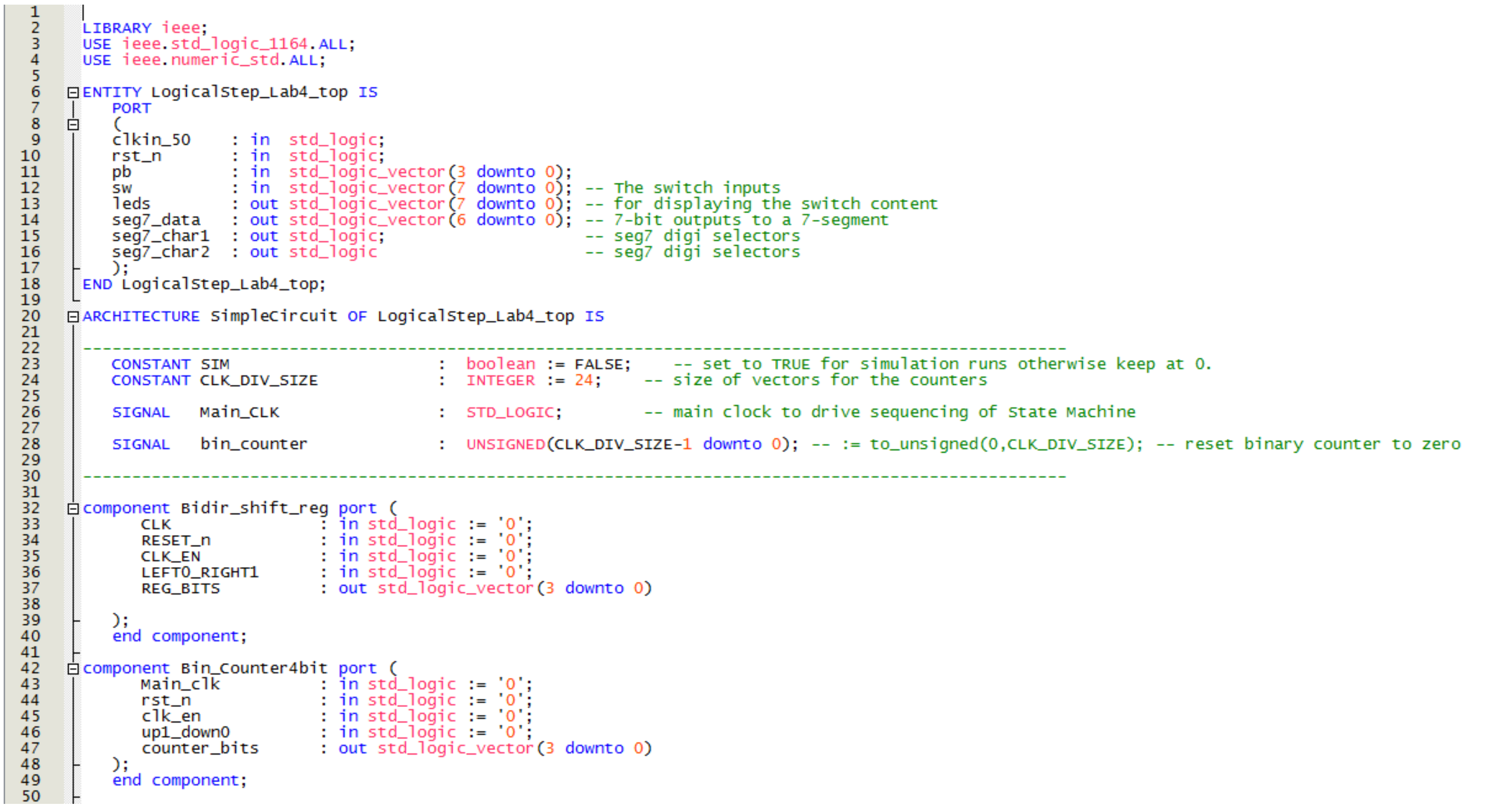
ECE-124 Lab-4 Submission Form – Winter 2018			
GROUP NUMBER: 20	Lab4 Demo	Lab4 Report	[illegible]
SESSION NUMBER: 201	Out of 10	Out of 10	

Partner A: Sidharth Baveja	sbaveja	[illegible]
Partner B: Wesley Barton	whbarton	Wesley Barton

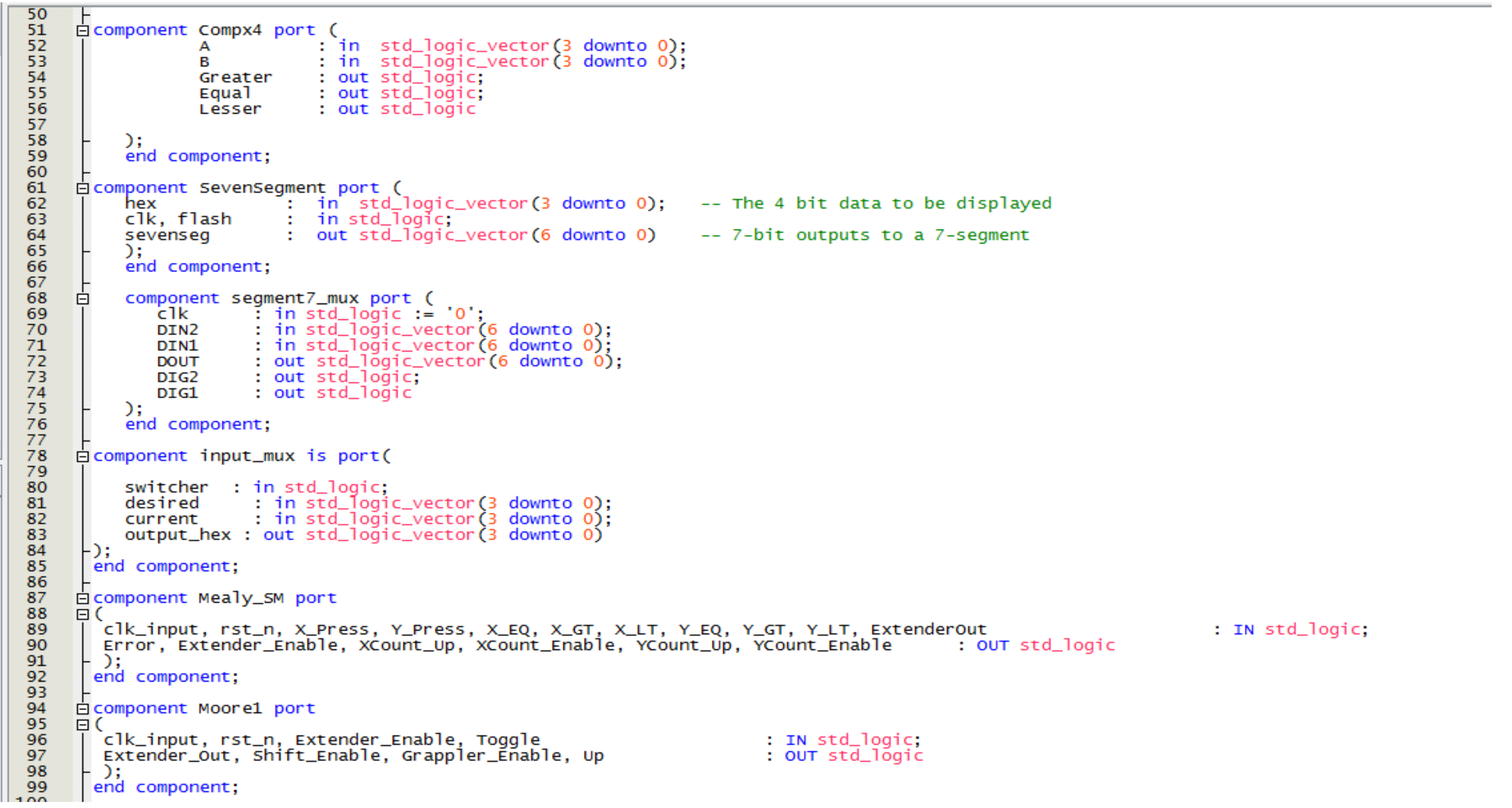
LAB4 DESIGN DEMO	Marks Allotted	A	B
Target X value on Digit1 (pb3 OFF); Target Y value on Digit2 (pb2 OFF)	1	1	1
X-Motion/Y-Motion has changing values on Digit1/Digit2	1	1	1
Extender enabled only at Target co-ordinates	1	1	1
Extender Position shown on leds[7:4]	1	1	1
Grappler enabled only at Fully Extended Extender (Grappler- led[3])	1	1	1
System Error when X/Y Motion with Extender not retracted	1	1	1
System Error Cleared when Extender is retracted.	1	1	1
DISCUSSION: Comment on your VHDL Implementation?	3	3	3
LAB4 DEMO MARK	**Out of 10**	10	10

LAB4 DESIGN REPORT (see rubric on LEARN for details)	Marks Allotted	TEAM	
Structural VHDL for Top Level VHDL file (only instances and connections) – no gates except in instance input fields	2		
Simulation of 8bit Shift Register and 8 bit Binary Counter in both directions	2		
State Diagrams of Mealy SM, Moore SM1, MooreSM2 machines	2		
Mealy Form for Mealy SM; Moore form for Moore SM1, Sm2	2		
Fitter Report on Resources Utilization by Entity (Logic Cells each)	2		
Delay in Report Submission (-1 per day) x number of days:			
LAB4 REPORT MARK	**Out of 10**		

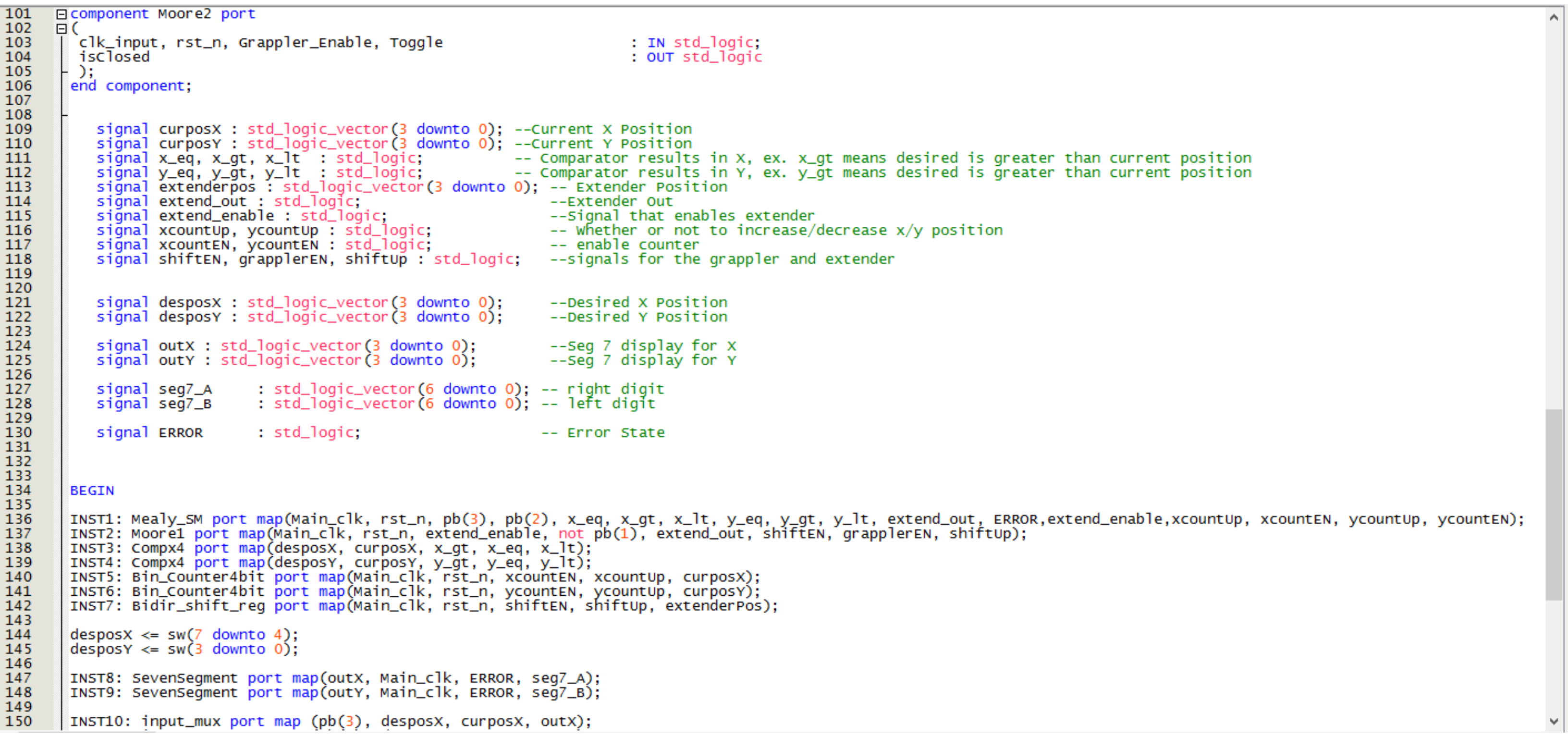
Top File:



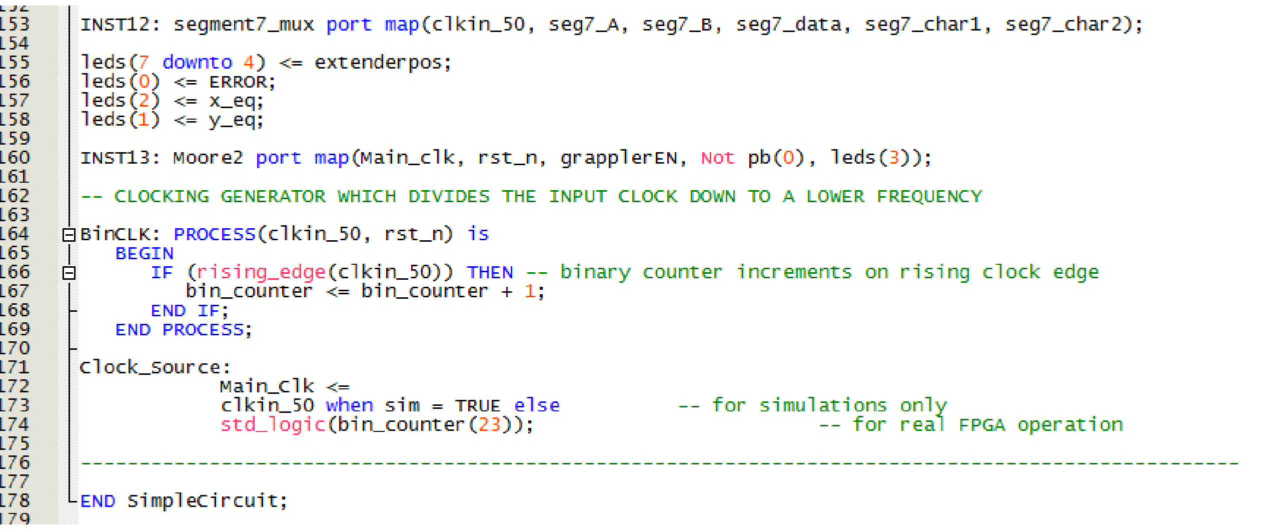
```vhdl

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

ENTITY LogicalStep_Lab4_top IS
   PORT
   (
   clkin_50     : in  std_logic;
   rst_n        : in  std_logic;
   pb           : in  std_logic_vector(3 downto 0);
   sw           : in  std_logic_vector(7 downto 0); -- The switch inputs
   leds         : out std_logic_vector(7 downto 0); -- for displaying the switch content
   seg7_data    : out std_logic_vector(6 downto 0); -- 7-bit outputs to a 7-segment
   seg7_char1   : out std_logic;                    -- seg7 digi selectors
   seg7_char2   : out std_logic                     -- seg7 digi selectors
   );
END LogicalStep_Lab4_top;

ARCHITECTURE SimpleCircuit OF LogicalStep_Lab4_top IS

----------------------------------------------------------------------------------------------------
   CONSTANT SIM                       :  boolean := FALSE;    -- set to TRUE for simulation runs otherwise keep at 0.
   CONSTANT CLK_DIV_SIZE              :  INTEGER := 24;    -- size of vectors for the counters

   SIGNAL   Main_CLK                  :  STD_LOGIC;        -- main clock to drive sequencing of State Machine

   SIGNAL   bin_counter               :  UNSIGNED(CLK_DIV_SIZE-1 downto 0); -- := to_unsigned(0,CLK_DIV_SIZE); -- reset binary counter to zero

----------------------------------------------------------------------------------------------------

component Bidir_shift_reg port (
      CLK                  : in std_logic := '0';
      RESET_n              : in std_logic := '0';
      CLK_EN               : in std_logic := '0';
      LEFT0_RIGHT1         : in std_logic := '0';
      REG_BITS             : out std_logic_vector(3 downto 0)

   );
   end component;

component Bin_Counter4bit port (
      Main_clk             : in std_logic := '0';
      rst_n                : in std_logic := '0';
      clk_en               : in std_logic := '0';
      up1_down0            : in std_logic := '0';
      counter_bits         : out std_logic_vector(3 downto 0)
   );
   end component;
```



```vhdl

component Compx4 port (
           A            : in  std_logic_vector(3 downto 0);
           B            : in  std_logic_vector(3 downto 0);
           Greater      : out std_logic;
           Equal        : out std_logic;
           Lesser       : out std_logic

   );
   end component;

component SevenSegment port (
   hex              :  in  std_logic_vector(3 downto 0);   -- The 4 bit data to be displayed
   clk, flash       :  in std_logic;
   sevenseg         :  out std_logic_vector(6 downto 0)    -- 7-bit outputs to a 7-segment
   );
   end component;

   component segment7_mux port (
      clk        : in std_logic := '0';
      DIN2       : in std_logic_vector(6 downto 0);
      DIN1       : in std_logic_vector(6 downto 0);
      DOUT       : out std_logic_vector(6 downto 0);
      DIG2       : out std_logic;
      DIG1       : out std_logic
   );
   end component;

component input_mux is port(

   switcher    : in std_logic;
   desired     : in std_logic_vector(3 downto 0);
   current     : in std_logic_vector(3 downto 0);
   output_hex  : out std_logic_vector(3 downto 0)
);
end component;

component Mealy_SM port
(
 clk_input, rst_n, X_Press, Y_Press, X_EQ, X_GT, X_LT, Y_EQ, Y_GT, Y_LT, ExtenderOut                : IN std_logic;
 Error, Extender_Enable, XCount_Up, XCount_Enable, YCount_Up, YCount_Enable       : OUT std_logic
 );
end component;

component Moore1 port
(
 clk_input, rst_n, Extender_Enable, Toggle                     : IN std_logic;
 Extender_Out, Shift_Enable, Grappler_Enable, Up               : OUT std_logic
 );
end component;
```



```vhdl
component Moore2 port
(
 clk_input, rst_n, Grappler_Enable, Toggle                    : IN std_logic;
 isClosed                                                     : OUT std_logic
 );
end component;


   signal curposX : std_logic_vector(3 downto 0); --Current X Position
   signal curposY : std_logic_vector(3 downto 0); --Current Y Position
   signal x_eq, x_gt, x_lt  : std_logic;          -- Comparator results in X, ex. x_gt means desired is greater than current position
   signal y_eq, y_gt, y_lt  : std_logic;          -- Comparator results in Y, ex. y_gt means desired is greater than current position
   signal extenderpos : std_logic_vector(3 downto 0); -- Extender Position
   signal extend_out : std_logic;                     --Extender Out
   signal extend_enable : std_logic;                  --Signal that enables extender
   signal xcountUp, ycountUp : std_logic;             -- Whether or not to increase/decrease x/y position
   signal xcountEN, ycountEN : std_logic;             -- enable counter
   signal shiftEN, grapplerEN, shiftUp : std_logic;   --signals for the grappler and extender


   signal desposX : std_logic_vector(3 downto 0);     --Desired X Position
   signal desposY : std_logic_vector(3 downto 0);     --Desired Y Position

   signal outX : std_logic_vector(3 downto 0);        --Seg 7 display for X
   signal outY : std_logic_vector(3 downto 0);        --Seg 7 display for Y

   signal seg7_A     : std_logic_vector(6 downto 0); -- right digit
   signal seg7_B     : std_logic_vector(6 downto 0); -- left digit

   signal ERROR      : std_logic;                    -- Error State



BEGIN

INST1: Mealy_SM port map(Main_clk, rst_n, pb(3), pb(2), x_eq, x_gt, x_lt, y_eq, y_gt, y_lt, extend_out, ERROR,extend_enable,xcountUp, xcountEN, ycountUp, ycountEN);
INST2: Moore1 port map(Main_clk, rst_n, extend_enable, not pb(1), extend_out, shiftEN, grapplerEN, shiftUp);
INST3: Compx4 port map(desposX, curposX, x_gt, x_eq, x_lt);
INST4: Compx4 port map(desposY, curposY, y_gt, y_eq, y_lt);
INST5: Bin_Counter4bit port map(Main_clk, rst_n, xcountEN, xcountUp, curposX);
INST6: Bin_Counter4bit port map(Main_clk, rst_n, ycountEN, ycountUp, curposY);
INST7: Bidir_shift_reg port map(Main_clk, rst_n, shiftEN, shiftUp, extenderPos);

desposX <= sw(7 downto 4);
desposY <= sw(3 downto 0);

INST8: SevenSegment port map(outX, Main_clk, ERROR, seg7_A);
INST9: SevenSegment port map(outY, Main_clk, ERROR, seg7_B);

INST10: input_mux port map (pb(3), desposX, curposX, outX);
```

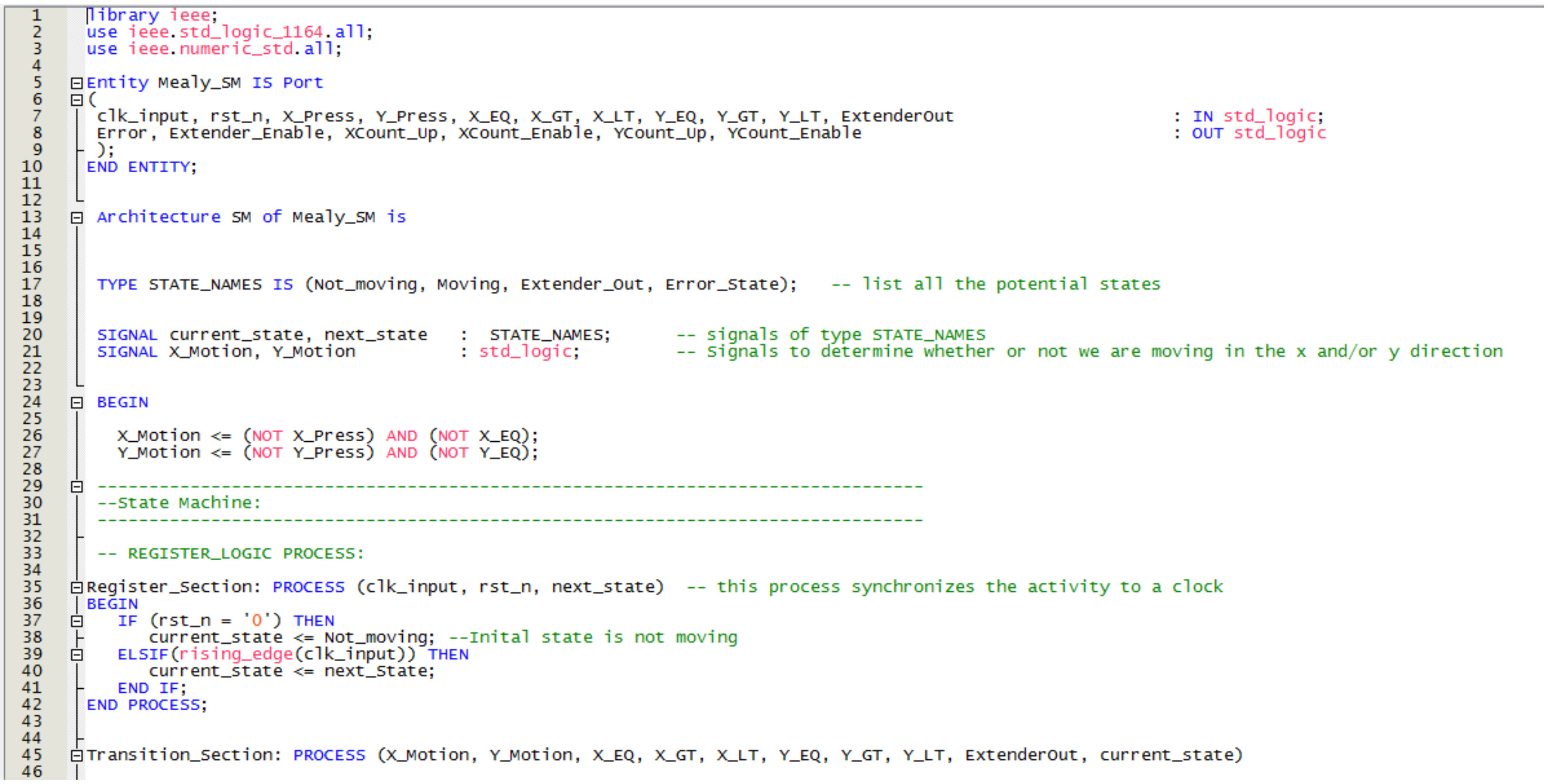


```vhdl
INST12: segment7_mux port map(clkin_50, seg7_A, seg7_B, seg7_data, seg7_char1, seg7_char2);

leds(7 downto 4) <= extenderpos;
leds(0) <= ERROR;
leds(2) <= x_eq;
leds(1) <= y_eq;

INST13: Moore2 port map(Main_clk, rst_n, grapplerEN, Not pb(0), leds(3));

-- CLOCKING GENERATOR WHICH DIVIDES THE INPUT CLOCK DOWN TO A LOWER FREQUENCY

BinCLK: PROCESS(clkin_50, rst_n) is
   BEGIN
      IF (rising_edge(clkin_50)) THEN -- binary counter increments on rising clock edge
         bin_counter <= bin_counter + 1;
      END IF;
   END PROCESS;

Clock_Source:
            Main_Clk <=
            clkin_50 when sim = TRUE else          -- for simulations only
            std_logic(bin_counter(23));                        -- for real FPGA operation

----------------------------------------------------------------------------------------------------

END SimpleCircuit;
```

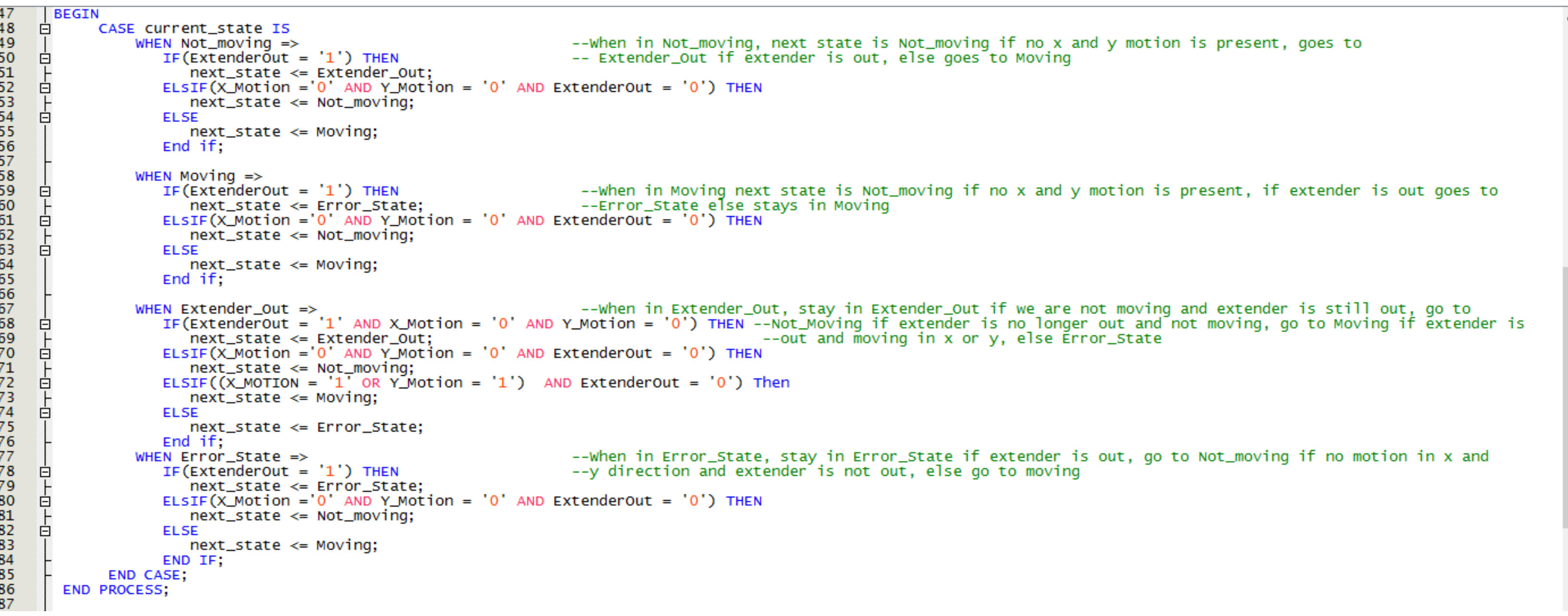
Difference Between Mealy and Moore State Machine:

In a Moore State Machine the outputs are only dependent on the current state, whereas in a Mealy State Machine the outputs are dependent on current state and the inputs. In a Moore the output equations consist of 1s and 0s only. Also Moore state machines are quite simple in logic, but usually contain more states, whereas a Mealy is complex in logic but contains less states. So in a Mealy State Machine the output equations consist of Boolean logic, 1s and 0s.

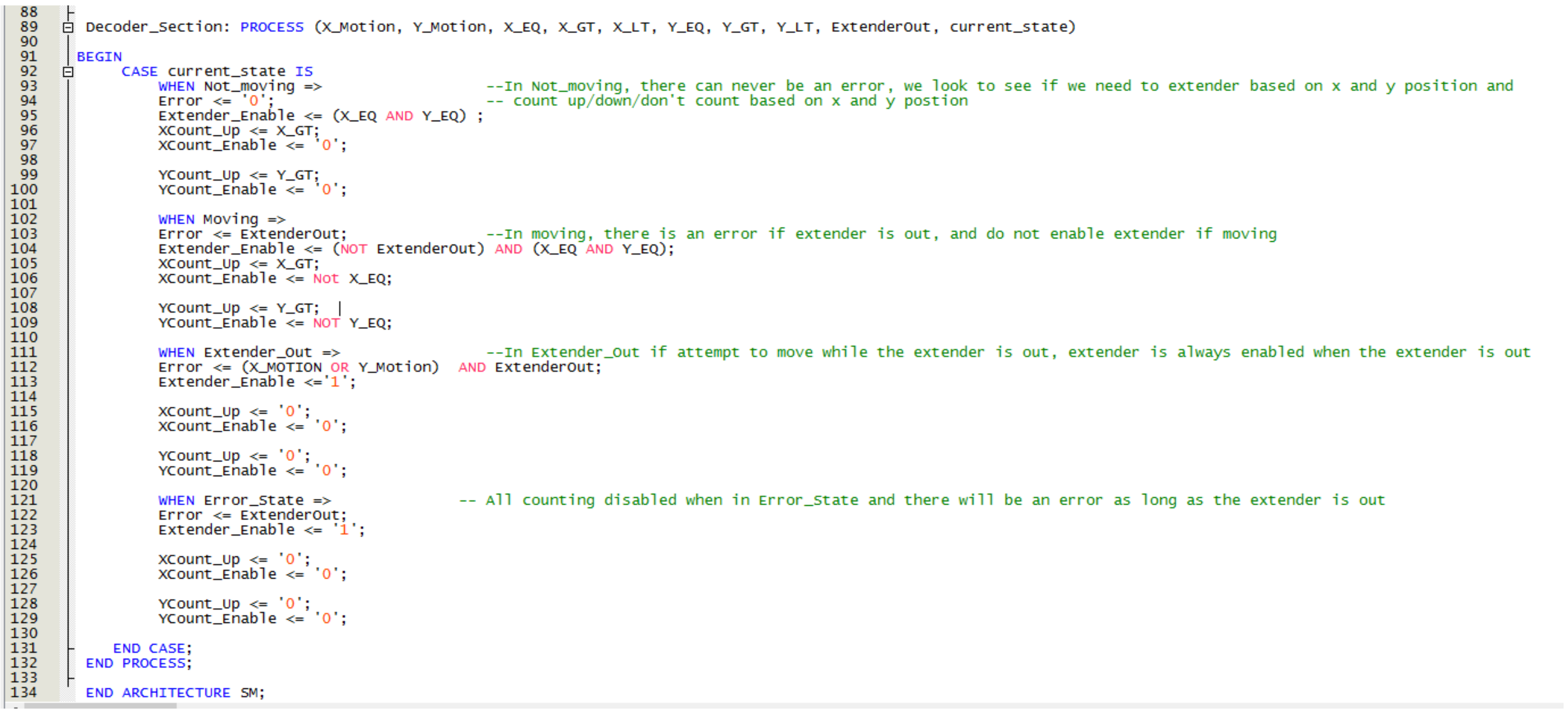
Mealy State Machine:



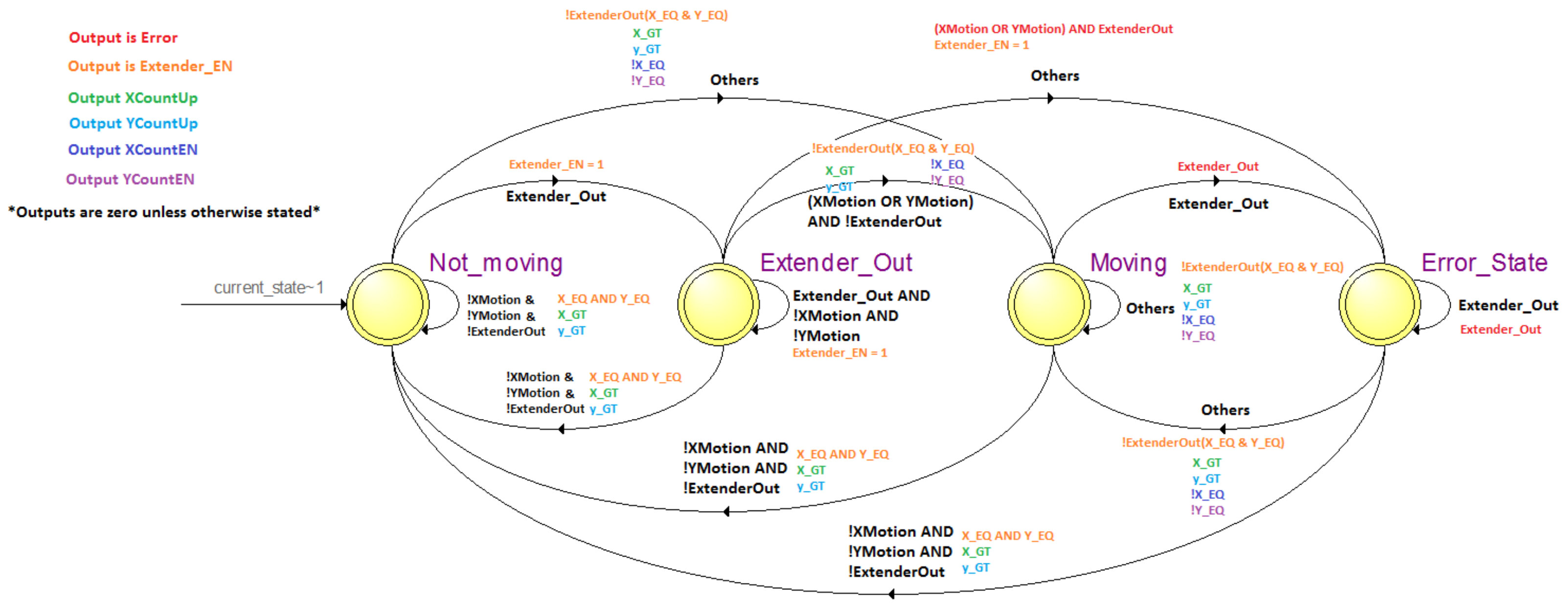
```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

Entity Mealy_SM IS Port
(
 clk_input, rst_n, X_Press, Y_Press, X_EQ, X_GT, X_LT, Y_EQ, Y_GT, Y_LT, ExtenderOut                    : IN std_logic;
 Error, Extender_Enable, XCount_Up, XCount_Enable, YCount_Up, YCount_Enable                             : OUT std_logic
 );
END ENTITY;


 Architecture SM of Mealy_SM is



 TYPE STATE_NAMES IS (Not_moving, Moving, Extender_Out, Error_State);   -- list all the potential states


 SIGNAL current_state, next_state   :  STATE_NAMES;      -- signals of type STATE_NAMES
 SIGNAL X_Motion, Y_Motion          : std_logic;         -- signals to determine whether or not we are moving in the x and/or y direction


 BEGIN

   X_Motion <= (NOT X_Press) AND (NOT X_EQ);
   Y_Motion <= (NOT Y_Press) AND (NOT Y_EQ);

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

 -- REGISTER_LOGIC PROCESS:

Register_Section: PROCESS (clk_input, rst_n, next_state)  -- this process synchronizes the activity to a clock
BEGIN
   IF (rst_n = '0') THEN
      current_state <= Not_moving; --Inital state is not moving
   ELSIF(rising_edge(clk_input)) THEN
      current_state <= next_State;
   END IF;
END PROCESS;


Transition_Section: PROCESS (X_Motion, Y_Motion, X_EQ, X_GT, X_LT, Y_EQ, Y_GT, Y_LT, ExtenderOut, current_state)

```



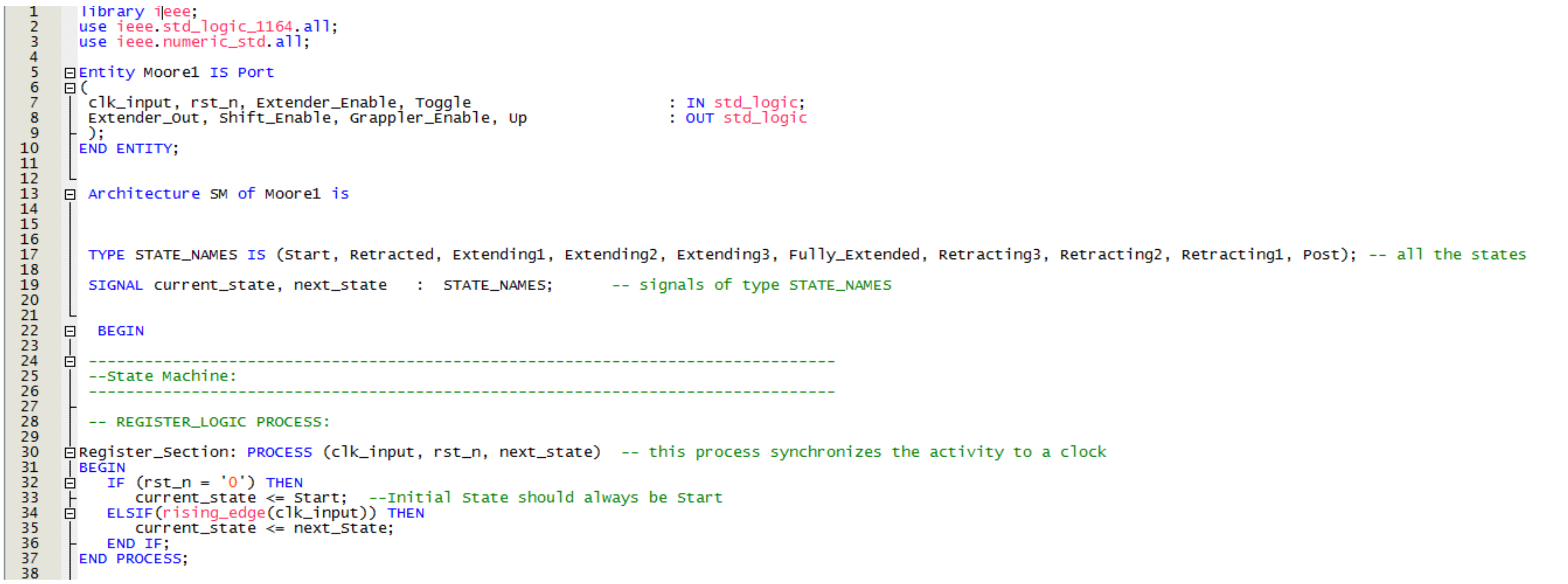
```vhdl
 BEGIN
      CASE current_state IS
         WHEN Not_moving =>                                    --When in Not_moving, next state is Not_moving if no x and y motion is present, goes to
            IF(ExtenderOut = '1') THEN                         -- Extender_Out if extender is out, else goes to Moving
               next_state <= Extender_Out;
            ELSIF(X_Motion ='0' AND Y_Motion = '0' AND ExtenderOut = '0') THEN
               next_state <= Not_moving;
            ELSE
               next_state <= Moving;
            End if;

         WHEN Moving =>                                         --When in Moving next state is Not_moving if no x and y motion is present, if extender is out goes to
            IF(ExtenderOut = '1') THEN                          --Error_State else stays in Moving
               next_state <= Error_State;
            ELSIF(X_Motion ='0' AND Y_Motion = '0' AND ExtenderOut = '0') THEN
               next_state <= Not_moving;
            ELSE
               next_state <= Moving;
            End if;

         WHEN Extender_Out =>                                   --When in Extender_Out, stay in Extender_Out if we are not moving and extender is still out, go to
            IF(ExtenderOut = '1' AND X_Motion = '0' AND Y_Motion = '0') THEN --Not_Moving if extender is no longer out and not moving, go to Moving if extender is
               next_state <= Extender_Out;                                     --out and moving in x or y, else Error_State
            ELSIF(X_Motion ='0' AND Y_Motion = '0' AND ExtenderOut = '0') THEN
               next_state <= Not_moving;
            ELSIF((X_MOTION = '1' OR Y_Motion = '1')  AND ExtenderOut = '0') Then
               next_state <= Moving;
            ELSE
               next_state <= Error_State;
            End if;
         WHEN Error_State =>                                   --When in Error_State, stay in Error_State if extender is out, go to Not_moving if no motion in x and
            IF(ExtenderOut = '1') THEN                         --y direction and extender is not out, else go to moving
               next_state <= Error_State;
            ELSIF(X_Motion ='0' AND Y_Motion = '0' AND ExtenderOut = '0') THEN
               next_state <= Not_moving;
            ELSE
               next_state <= Moving;
            END IF;
       END CASE;
  END PROCESS;

```



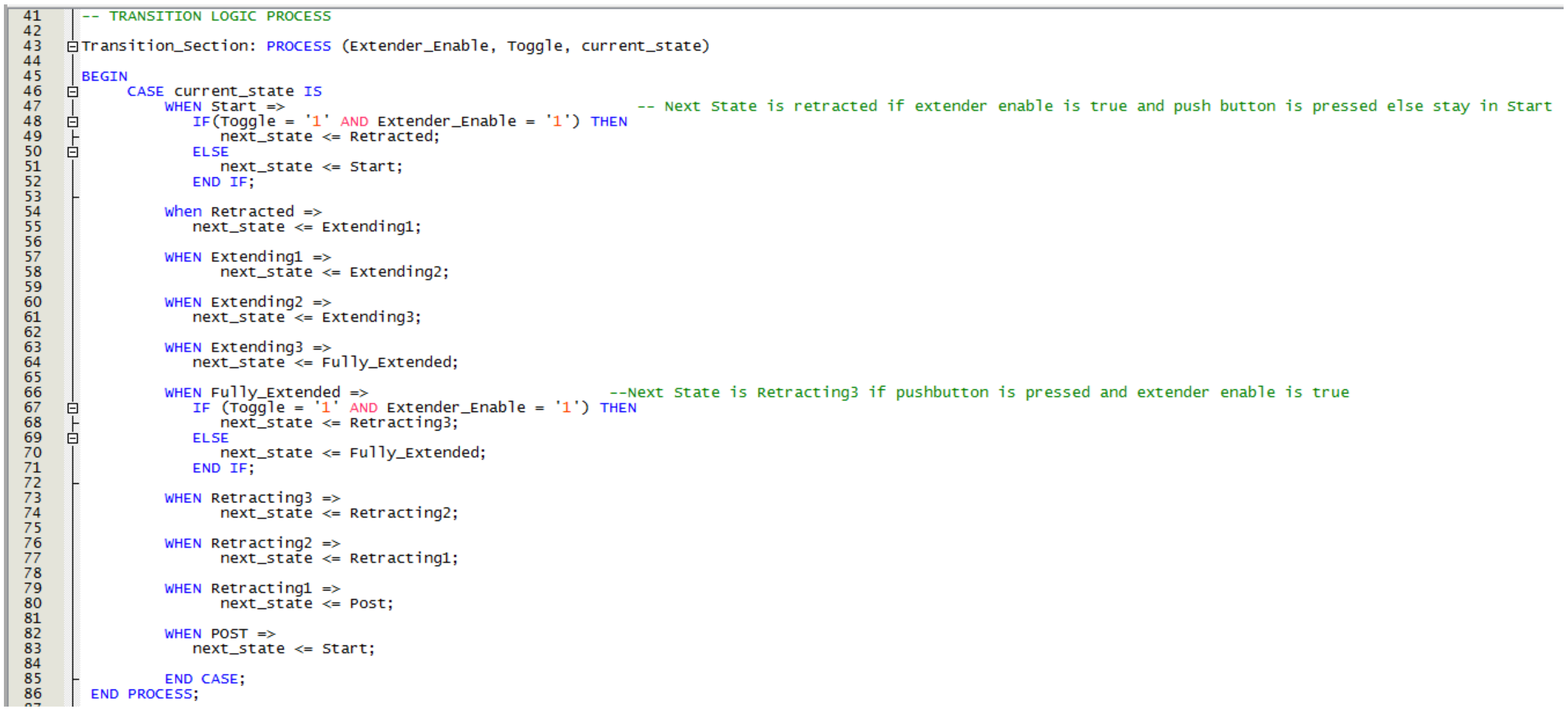
```
Decoder_Section: PROCESS (X_Motion, Y_Motion, X_EQ, X_GT, X_LT, Y_EQ, Y_GT, Y_LT, ExtenderOut, current_state)

BEGIN
    CASE current_state IS
        WHEN Not_moving =>                  --In Not_moving, there can never be an error, we look to see if we need to extender based on x and y position and
        Error <= '0';                       -- count up/down/don't count based on x and y postion
        Extender_Enable <= (X_EQ AND Y_EQ) ;
        XCount_Up <= X_GT;
        XCount_Enable <= '0';

        YCount_Up <= Y_GT;
        YCount_Enable <= '0';

        WHEN Moving =>
        Error <= ExtenderOut;               --In moving, there is an error if extender is out, and do not enable extender if moving
        Extender_Enable <= (NOT ExtenderOut) AND (X_EQ AND Y_EQ);
        XCount_Up <= X_GT;
        XCount_Enable <= Not X_EQ;

        YCount_Up <= Y_GT;  |
        YCount_Enable <= NOT Y_EQ;

        WHEN Extender_Out =>                --In Extender_Out if attempt to move while the extender is out, extender is always enabled when the extender is out
        Error <= (X_MOTION OR Y_Motion)  AND ExtenderOut;
        Extender_Enable <='1';

        XCount_Up <= '0';
        XCount_Enable <= '0';

        YCount_Up <= '0';
        YCount_Enable <= '0';

        WHEN Error_State =>               -- All counting disabled when in Error_State and there will be an error as long as the extender is out
        Error <= ExtenderOut;
        Extender_Enable <= '1';

        XCount_Up <= '0';
        XCount_Enable <= '0';

        YCount_Up <= '0';
        YCount_Enable <= '0';

    END CASE;
END PROCESS;

END ARCHITECTURE SM;
```



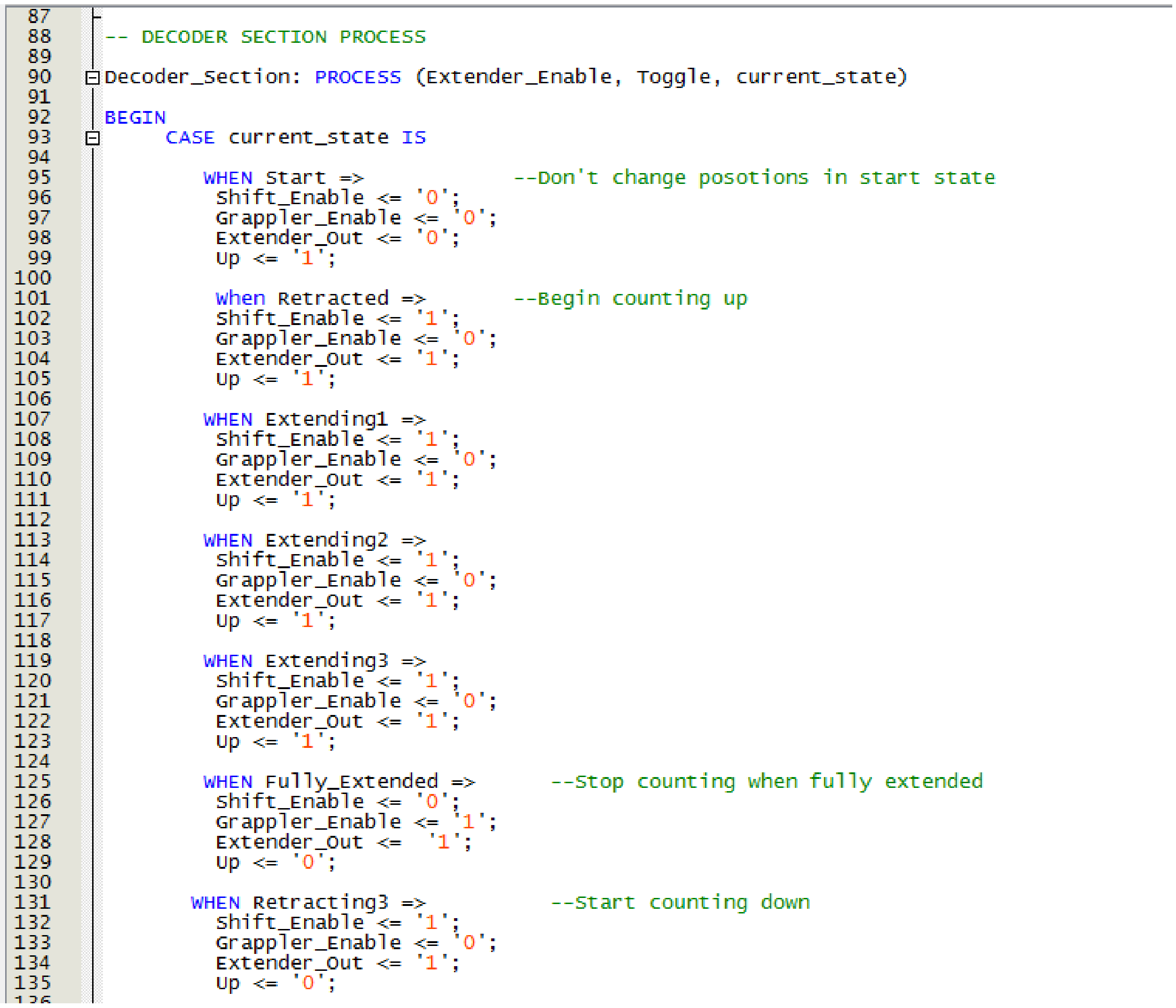
Extender State Machine (Moore 1):



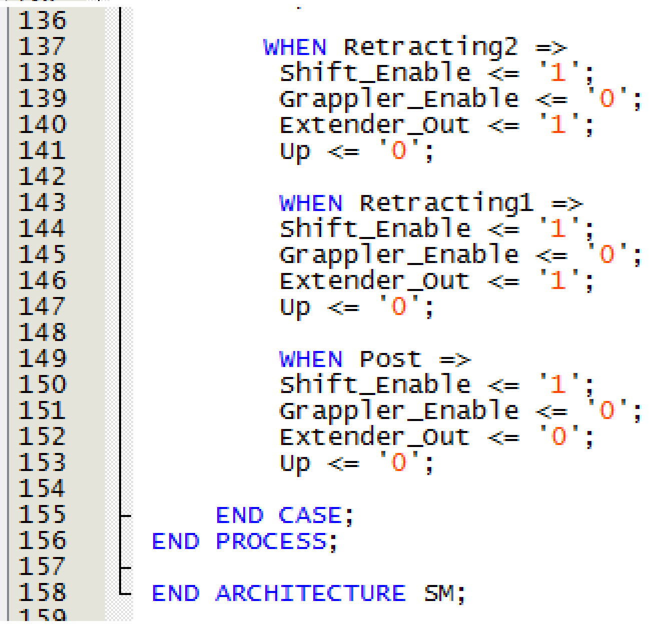
```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

Entity Moore1 IS Port
(
 clk_input, rst_n, Extender_Enable, Toggle                   : IN std_logic;
 Extender_Out, Shift_Enable, Grappler_Enable, Up             : OUT std_logic
 );
END ENTITY;


 Architecture SM of Moore1 is



 TYPE STATE_NAMES IS (Start, Retracted, Extending1, Extending2, Extending3, Fully_Extended, Retracting3, Retracting2, Retracting1, Post); -- all the states

 SIGNAL current_state, next_state    :  STATE_NAMES;      -- signals of type STATE_NAMES


  BEGIN

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

 -- REGISTER_LOGIC PROCESS:

Register_Section: PROCESS (clk_input, rst_n, next_state)  -- this process synchronizes the activity to a clock
BEGIN
   IF (rst_n = '0') THEN
      current_state <= Start;  --Initial State should always be Start
   ELSIF(rising_edge(clk_input)) THEN
      current_state <= next_State;
   END IF;
END PROCESS;
```



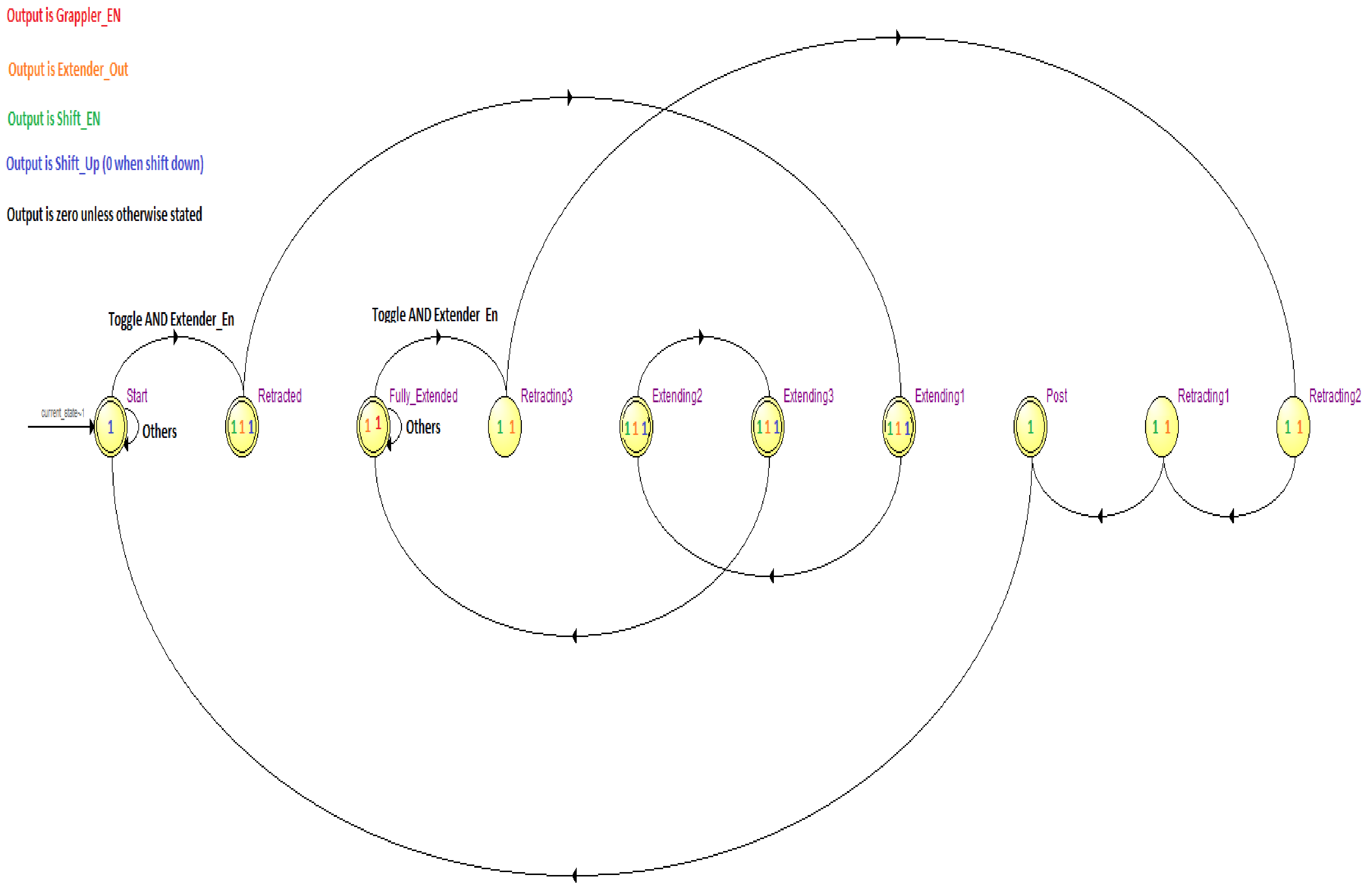
```vhdl
-- TRANSITION LOGIC PROCESS

Transition_Section: PROCESS (Extender_Enable, Toggle, current_state)

BEGIN
     CASE current_state IS
         WHEN Start =>                                          -- Next State is retracted if extender enable is true and push button is pressed else stay in Start
            IF(Toggle = '1' AND Extender_Enable = '1') THEN
               next_state <= Retracted;
            ELSE
               next_state <= Start;
            END IF;

         When Retracted =>
            next_state <= Extending1;

         WHEN Extending1 =>
               next_state <= Extending2;

         WHEN Extending2 =>
            next_state <= Extending3;

         WHEN Extending3 =>
            next_state <= Fully_Extended;

         WHEN Fully_Extended =>                              --Next State is Retracting3 if pushbutton is pressed and extender enable is true
            IF (Toggle = '1' AND Extender_Enable = '1') THEN
               next_state <= Retracting3;
            ELSE
               next_state <= Fully_Extended;
            END IF;

         WHEN Retracting3 =>
               next_state <= Retracting2;

         WHEN Retracting2 =>
               next_state <= Retracting1;

         WHEN Retracting1 =>
               next_state <= Post;

         WHEN POST =>
            next_state <= Start;

         END CASE;
 END PROCESS;
```



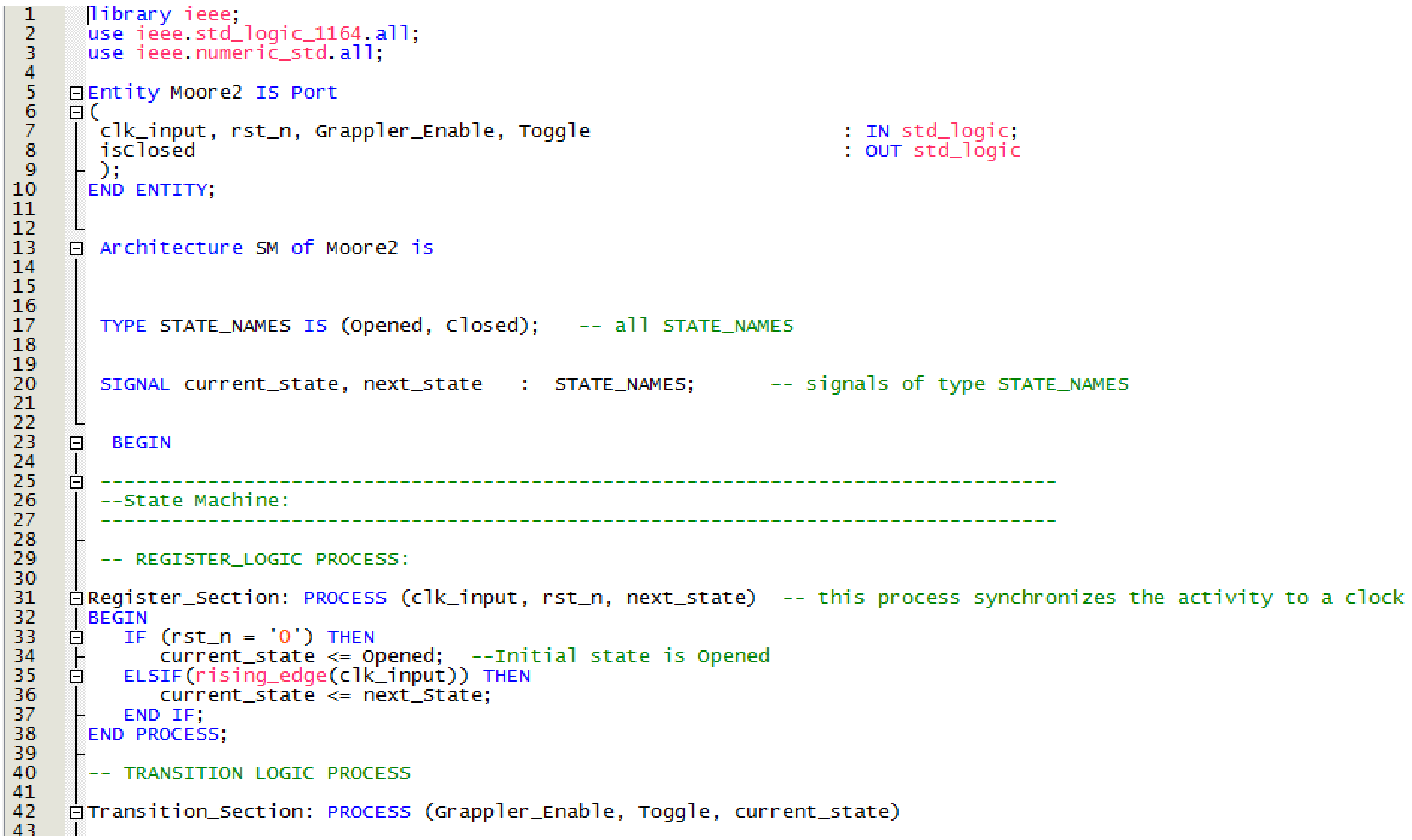
```
-- DECODER SECTION PROCESS

Decoder_Section: PROCESS (Extender_Enable, Toggle, current_state)

BEGIN
     CASE current_state IS

        WHEN Start =>          --Don't change posotions in start state
         Shift_Enable <= '0';
         Grappler_Enable <= '0';
         Extender_Out <= '0';
         Up <= '1';

         when Retracted =>       --Begin counting up
         Shift_Enable <= '1';
         Grappler_Enable <= '0';
         Extender_Out <= '1';
         Up <= '1';

        WHEN Extending1 =>
         Shift_Enable <= '1';
         Grappler_Enable <= '0';
         Extender_Out <= '1';
         Up <= '1';

        WHEN Extending2 =>
         Shift_Enable <= '1';
         Grappler_Enable <= '0';
         Extender_Out <= '1';
         Up <= '1';

        WHEN Extending3 =>
         Shift_Enable <= '1';
         Grappler_Enable <= '0';
         Extender_Out <= '1';
         Up <= '1';

        WHEN Fully_Extended =>      --Stop counting when fully extended
         Shift_Enable <= '0';
         Grappler_Enable <= '1';
         Extender_Out <=  '1';
         Up <= '0';

       WHEN Retracting3 =>          --Start counting down
         Shift_Enable <= '1';
         Grappler_Enable <= '0';
         Extender_Out <= '1';
         Up <= '0';
```



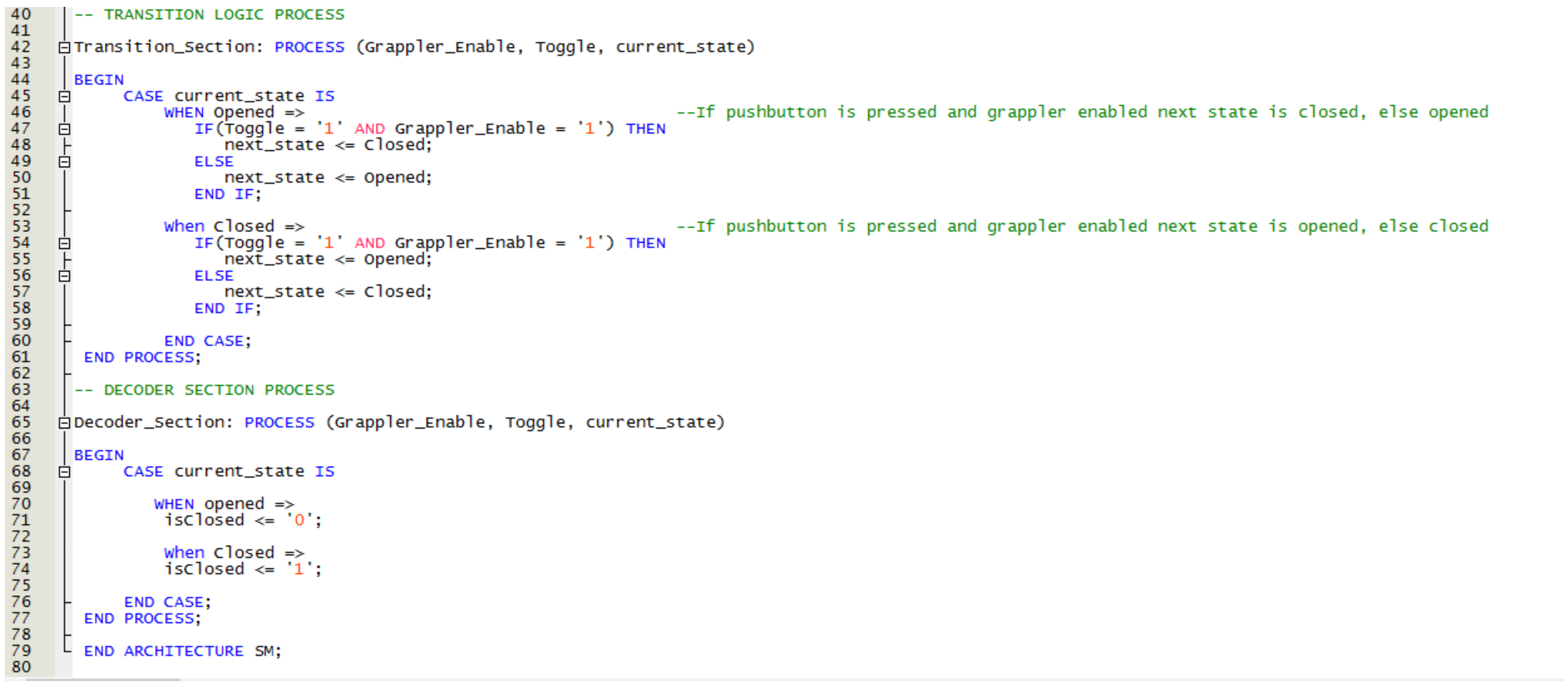
```
       WHEN Retracting2 =>
        Shift_Enable <= '1';
        Grappler_Enable <= '0';
        Extender_Out <= '1';
        Up <= '0';

        WHEN Retracting1 =>
        Shift_Enable <= '1';
        Grappler_Enable <= '0';
        Extender_Out <= '1';
        Up <= '0';

        WHEN Post =>
        Shift_Enable <= '1';
        Grappler_Enable <= '0';
        Extender_Out <= '0';
        Up <= '0';

    END CASE;
END PROCESS;

END ARCHITECTURE SM;
```


Output is Grappler_EN
Output is Extender_Out
Output is Shift_EN
Output is Shift_Up (0 when shift down)
Output is zero unless otherwise stated
current_state~1
Start
1
Others
Toggle AND Extender_En
Retracted
1 1 1
Fully_Extended
1 1
Others
Toggle AND Extender En
Retracting3
1 1
Extending2
1 1 1
Extending3
1 1 1
Extending1
1 1 1
Post
1
Retracting1
1 1
Retracting2
1 1

Grappler State Machine (Moore 2):



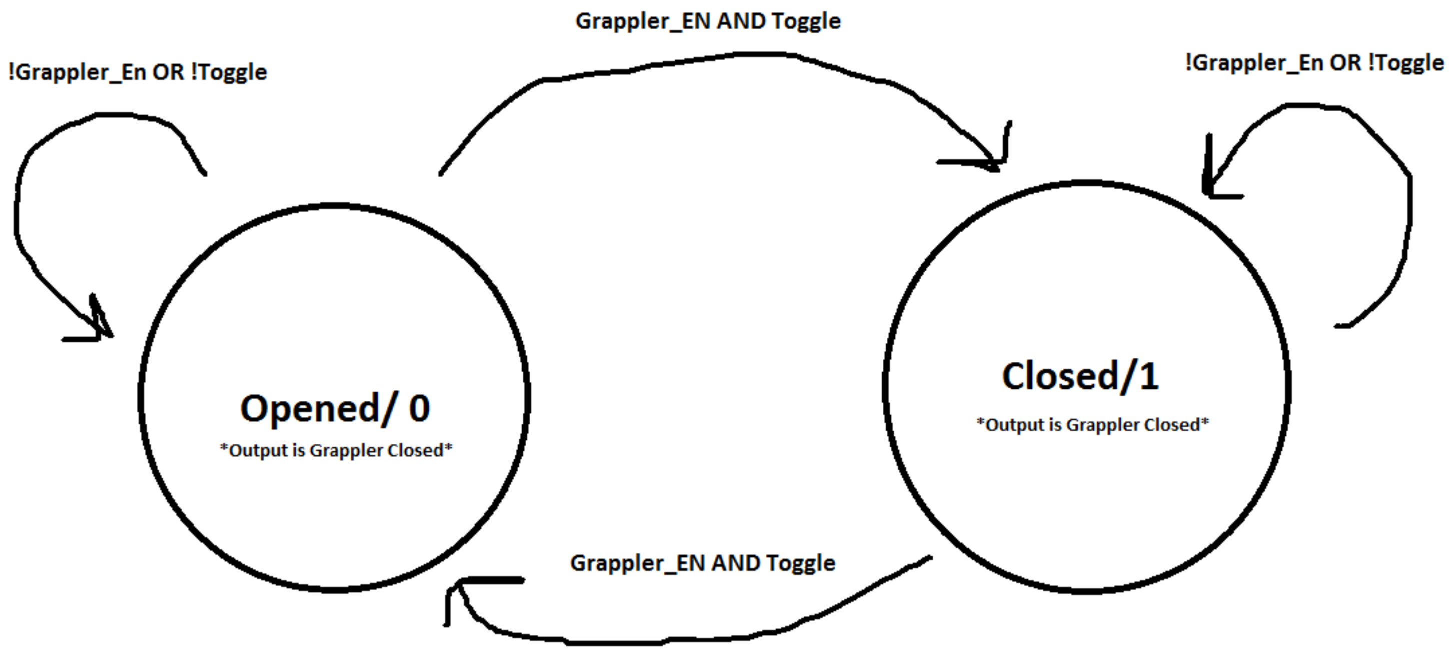
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

Entity Moore2 IS Port
(
 clk_input, rst_n, Grappler_Enable, Toggle                           : IN std_logic;
 isClosed                                                            : OUT std_logic
 );
END ENTITY;


 Architecture SM of Moore2 is



 TYPE STATE_NAMES IS (Opened, Closed);    -- all STATE_NAMES


 SIGNAL current_state, next_state   :  STATE_NAMES;       -- signals of type STATE_NAMES


  BEGIN

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

 -- REGISTER_LOGIC PROCESS:

Register_Section: PROCESS (clk_input, rst_n, next_state)  -- this process synchronizes the activity to a clock
BEGIN
   IF (rst_n = '0') THEN
      current_state <= Opened;   --Initial state is Opened
   ELSIF(rising_edge(clk_input)) THEN
      current_state <= next_State;
   END IF;
END PROCESS;

-- TRANSITION LOGIC PROCESS

Transition_Section: PROCESS (Grappler_Enable, Toggle, current_state)
```



-- TRANSITION LOGIC PROCESS

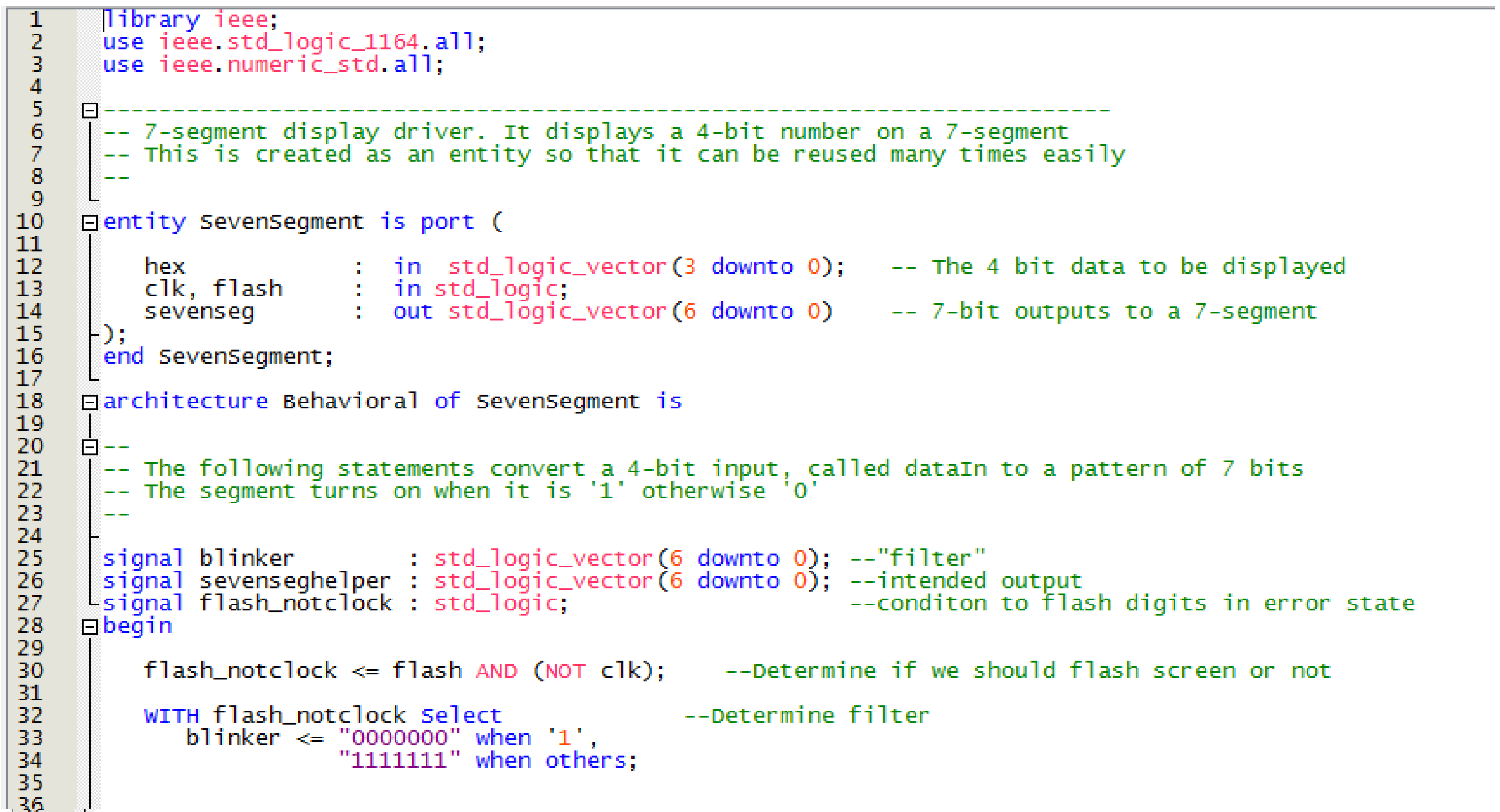
Transition_Section: PROCESS (Grappler_Enable, Toggle, current_state)

```

BEGIN
     CASE current_state IS
         WHEN Opened =>                                          --If pushbutton is pressed and grappler enabled next state is closed, else opened
            IF(Toggle = '1' AND Grappler_Enable = '1') THEN
               next_state <= Closed;
            ELSE
               next_state <= Opened;
            END IF;

         when Closed =>                                          --If pushbutton is pressed and grappler enabled next state is opened, else closed
            IF(Toggle = '1' AND Grappler_Enable = '1') THEN
               next_state <= Opened;
            ELSE
               next_state <= Closed;
            END IF;

         END CASE;
 END PROCESS;

-- DECODER SECTION PROCESS

Decoder_Section: PROCESS (Grappler_Enable, Toggle, current_state)

BEGIN
     CASE current_state IS

        WHEN opened =>
         isClosed <= '0';

         when Closed =>
         isClosed <= '1';

     END CASE;
 END PROCESS;

 END ARCHITECTURE SM;

```

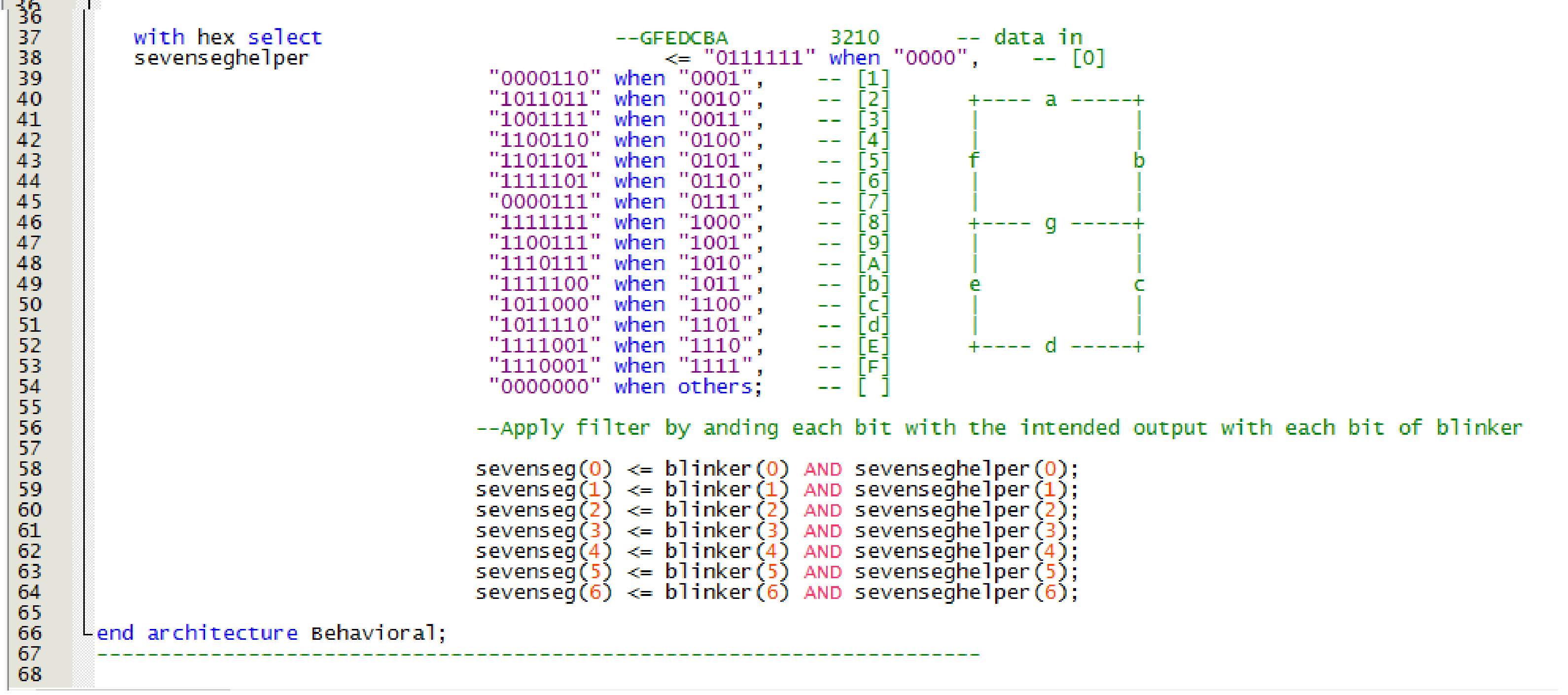


State Machine was not generated by Quartus, so we made it in paint

Seven Segment (For Error State – Flashing):

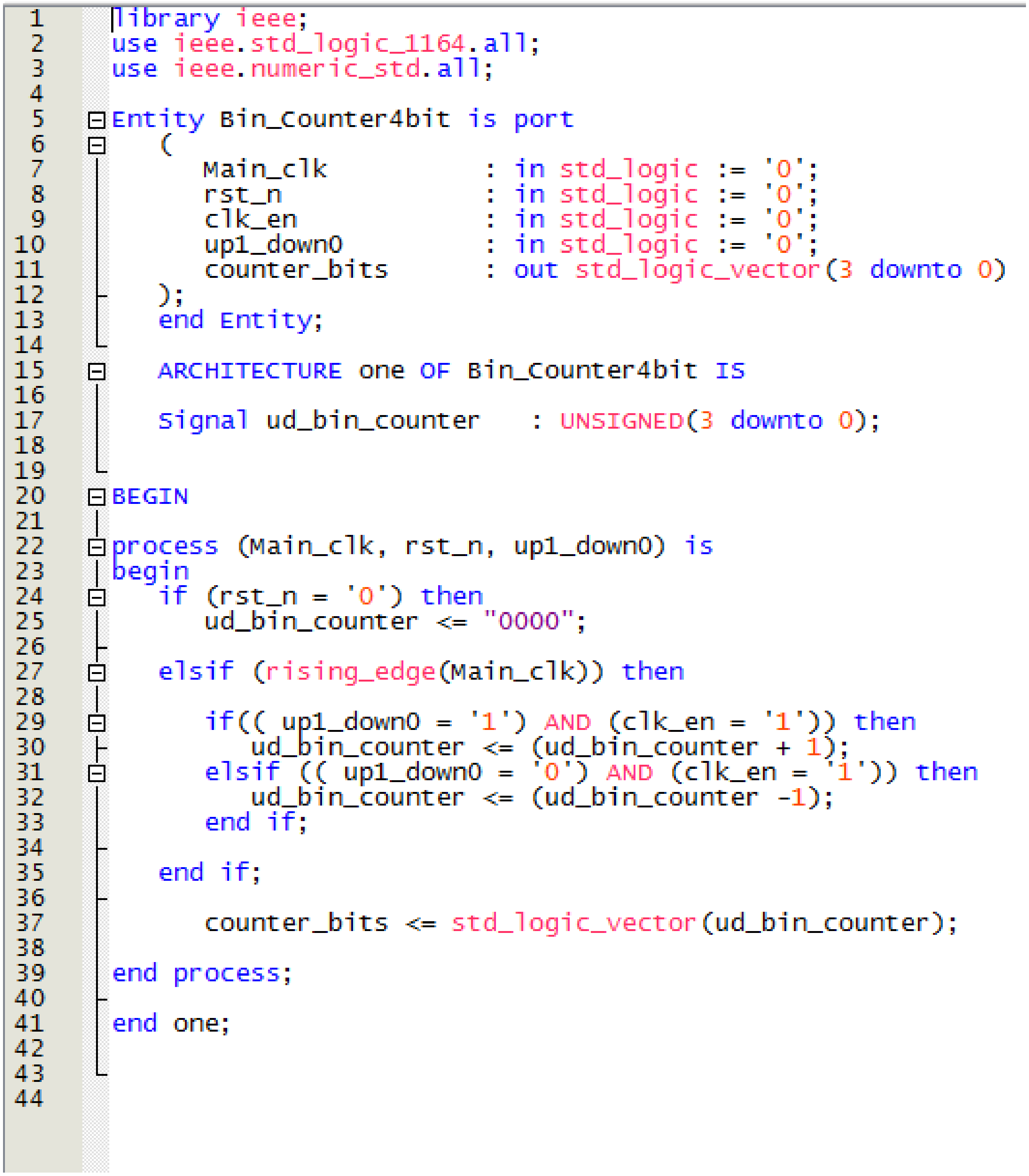


```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

-------------------------------------------------------------------------
-- 7-segment display driver. It displays a 4-bit number on a 7-segment
-- This is created as an entity so that it can be reused many times easily
--

entity SevenSegment is port (

   hex            :  in  std_logic_vector(3 downto 0);   -- The 4 bit data to be displayed
   clk, flash     :  in std_logic;
   sevenseg       :  out std_logic_vector(6 downto 0)    -- 7-bit outputs to a 7-segment
);
end SevenSegment;

architecture Behavioral of SevenSegment is

--
-- The following statements convert a 4-bit input, called dataIn to a pattern of 7 bits
-- The segment turns on when it is '1' otherwise '0'
--

signal blinker        : std_logic_vector(6 downto 0); --"filter"
signal sevenseghelper : std_logic_vector(6 downto 0); --intended output
signal flash_notclock : std_logic;                    --conditon to flash digits in error state
begin

   flash_notclock <= flash AND (NOT clk);    --Determine if we should flash screen or not

   WITH flash_notclock Select             --Determine filter
      blinker  <= "0000000" when '1',
                  "1111111" when others;

```



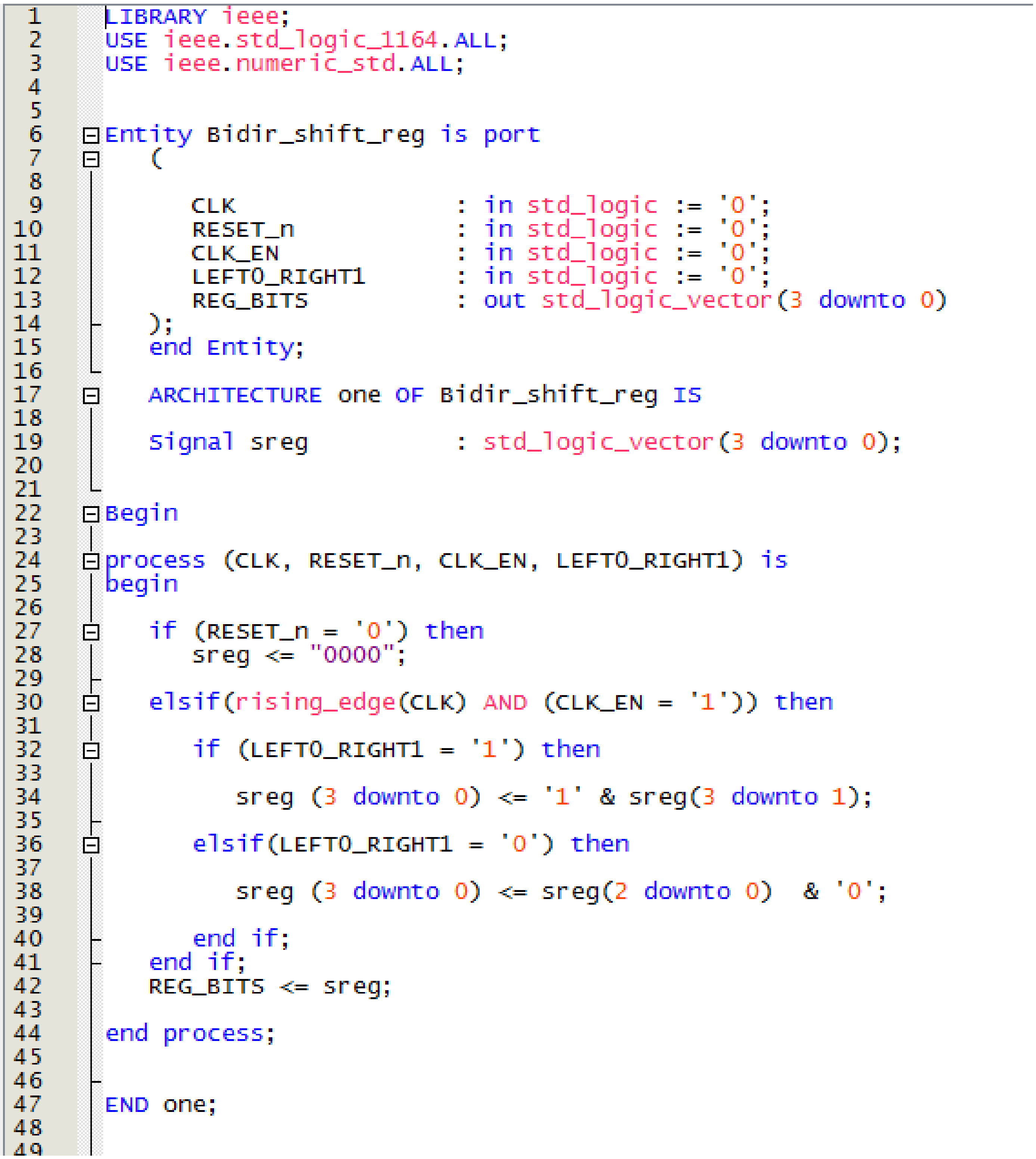
```vhdl

     with hex select                     --GFEDCBA        3210       -- data in
     sevenseghelper                         <= "0111111" when "0000",     -- [0]
                          "0000110" when "0001",      -- [1]
                          "1011011" when "0010",      -- [2]       +---- a -----+
                          "1001111" when "0011",      -- [3]       |            |
                          "1100110" when "0100",      -- [4]       |            |
                          "1101101" when "0101",      -- [5]       f            b
                          "1111101" when "0110",      -- [6]       |            |
                          "0000111" when "0111",      -- [7]       |            |
                          "1111111" when "1000",      -- [8]       +---- g -----+
                          "1100111" when "1001",      -- [9]       |            |
                          "1110111" when "1010",      -- [A]       |            |
                          "1111100" when "1011",      -- [b]       e            c
                          "1011000" when "1100",      -- [c]       |            |
                          "1011110" when "1101",      -- [d]       |            |
                          "1111001" when "1110",      -- [E]       +---- d -----+
                          "1110001" when "1111",      -- [F]
                          "0000000" when others;      -- [ ]

                         --Apply filter by anding each bit with the intended output with each bit of blinker

                         sevenseg(0) <= blinker(0) AND sevenseghelper(0);
                         sevenseg(1) <= blinker(1) AND sevenseghelper(1);
                         sevenseg(2) <= blinker(2) AND sevenseghelper(2);
                         sevenseg(3) <= blinker(3) AND sevenseghelper(3);
                         sevenseg(4) <= blinker(4) AND sevenseghelper(4);
                         sevenseg(5) <= blinker(5) AND sevenseghelper(5);
                         sevenseg(6) <= blinker(6) AND sevenseghelper(6);

end architecture Behavioral;
---------------------------------------------------------------
```

4 Bit Counter:



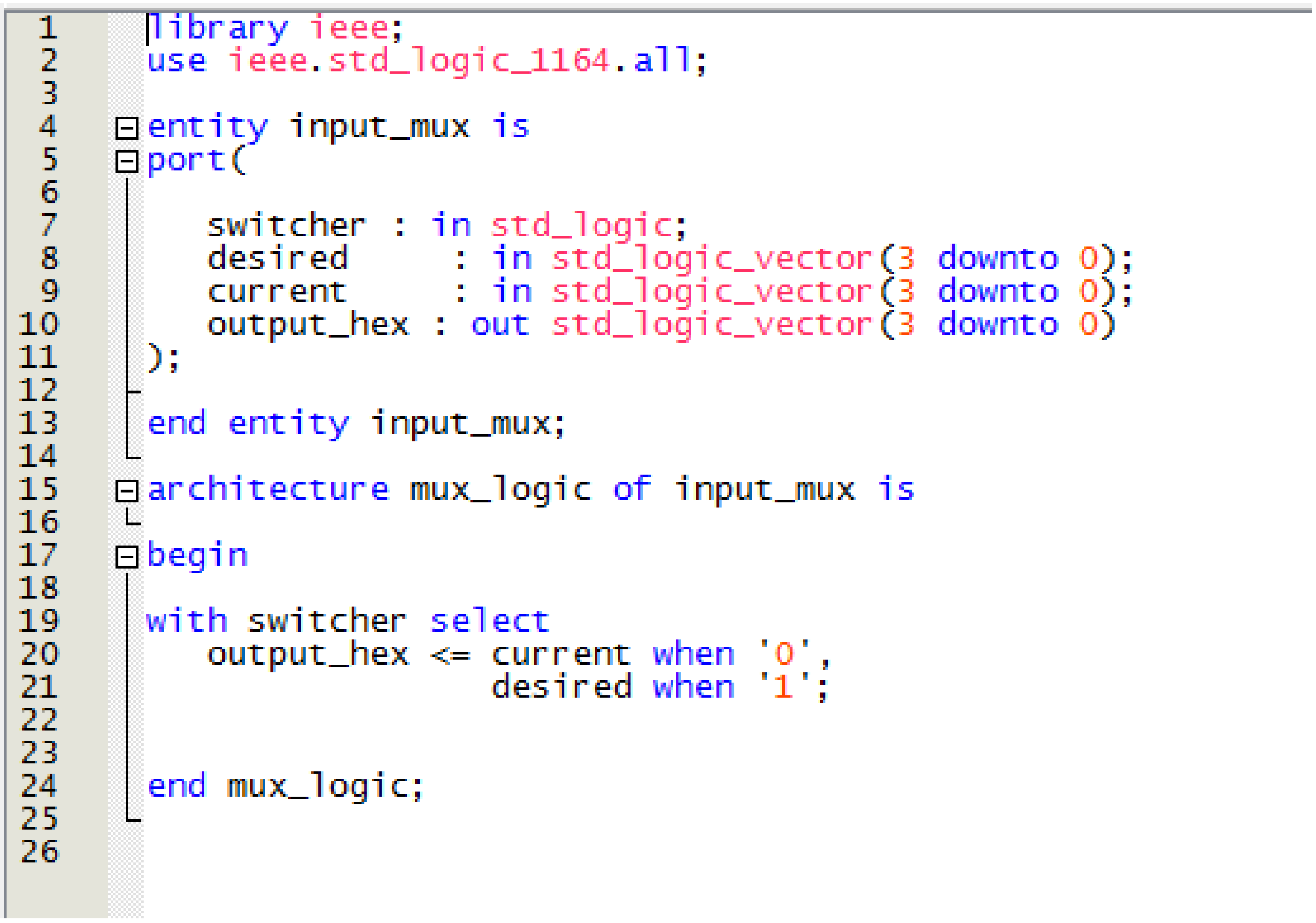
```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

Entity Bin_Counter4bit is port
   (
      Main_clk            : in std_logic := '0';
      rst_n               : in std_logic := '0';
      clk_en              : in std_logic := '0';
      up1_down0           : in std_logic := '0';
      counter_bits        : out std_logic_vector(3 downto 0)
   );
   end Entity;

   ARCHITECTURE one OF Bin_Counter4bit IS

   Signal ud_bin_counter    : UNSIGNED(3 downto 0);


BEGIN

process (Main_clk, rst_n, up1_down0) is
begin
   if (rst_n = '0') then
      ud_bin_counter <= "0000";

   elsif (rising_edge(Main_clk)) then

      if(( up1_down0 = '1') AND (clk_en = '1')) then
         ud_bin_counter <= (ud_bin_counter + 1);
      elsif (( up1_down0 = '0') AND (clk_en = '1')) then
         ud_bin_counter <= (ud_bin_counter -1);
      end if;

   end if;

      counter_bits <= std_logic_vector(ud_bin_counter);

end process;

end one;
```

4 Bit Shift Register:



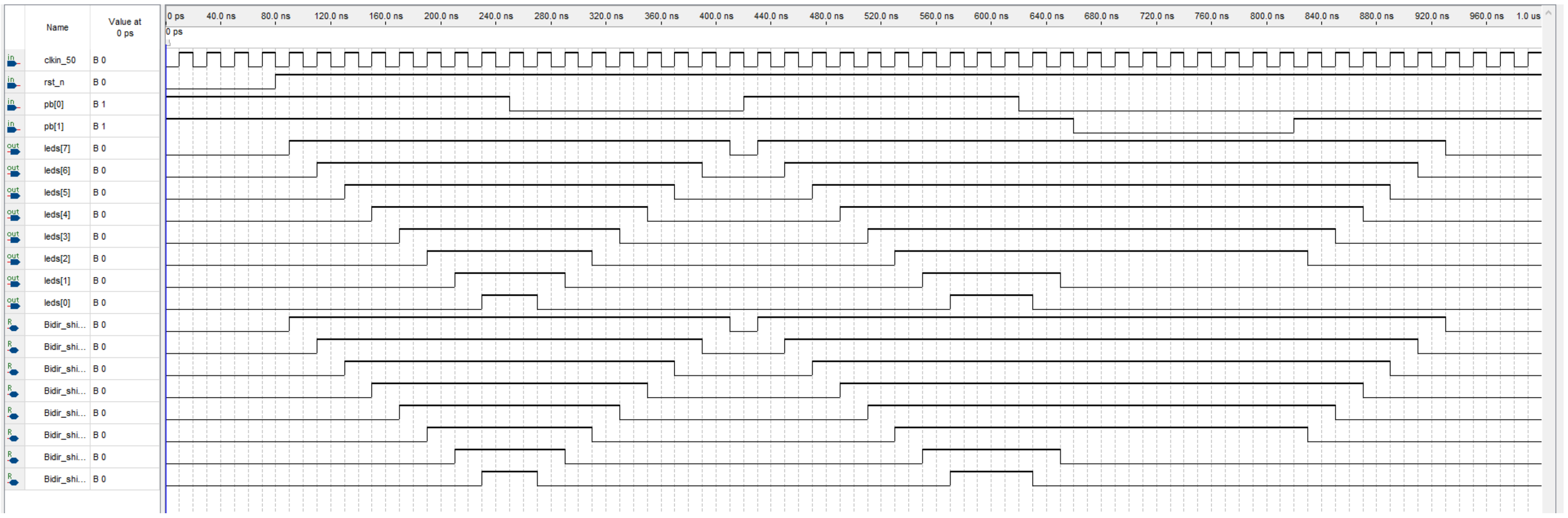
```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;


Entity Bidir_shift_reg is port
   (

      CLK                 : in std_logic := '0';
      RESET_n             : in std_logic := '0';
      CLK_EN              : in std_logic := '0';
      LEFT0_RIGHT1        : in std_logic := '0';
      REG_BITS            : out std_logic_vector(3 downto 0)
   );
   end Entity;

   ARCHITECTURE one OF Bidir_shift_reg IS

   Signal sreg            : std_logic_vector(3 downto 0);


Begin

process (CLK, RESET_n, CLK_EN, LEFT0_RIGHT1) is
begin

   if (RESET_n = '0') then
      sreg <= "0000";

   elsif(rising_edge(CLK) AND (CLK_EN = '1')) then

      if (LEFT0_RIGHT1 = '1') then

         sreg (3 downto 0) <= '1' & sreg(3 downto 1);

      elsif(LEFT0_RIGHT1 = '0') then

         sreg (3 downto 0) <= sreg(2 downto 0)  & '0';

      end if;
   end if;
   REG_BITS <= sreg;

end process;


END one;
```

Input Mux:

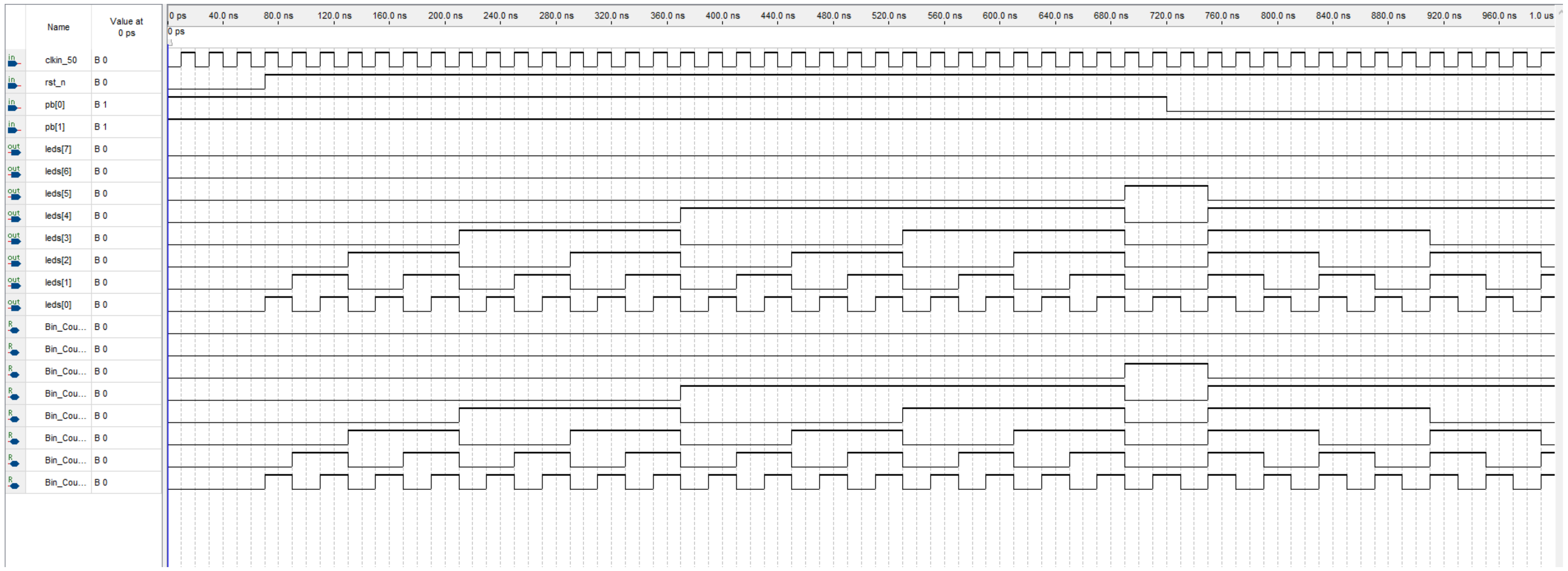


```
library ieee;
use ieee.std_logic_1164.all;

entity input_mux is
port(

	switcher : in std_logic;
	desired	 : in std_logic_vector(3 downto 0);
	current	 : in std_logic_vector(3 downto 0);
	output_hex : out std_logic_vector(3 downto 0)
);

end entity input_mux;

architecture mux_logic of input_mux is

begin

with switcher select
	output_hex <= current when '0',
			  desired when '1';


end mux_logic;
```

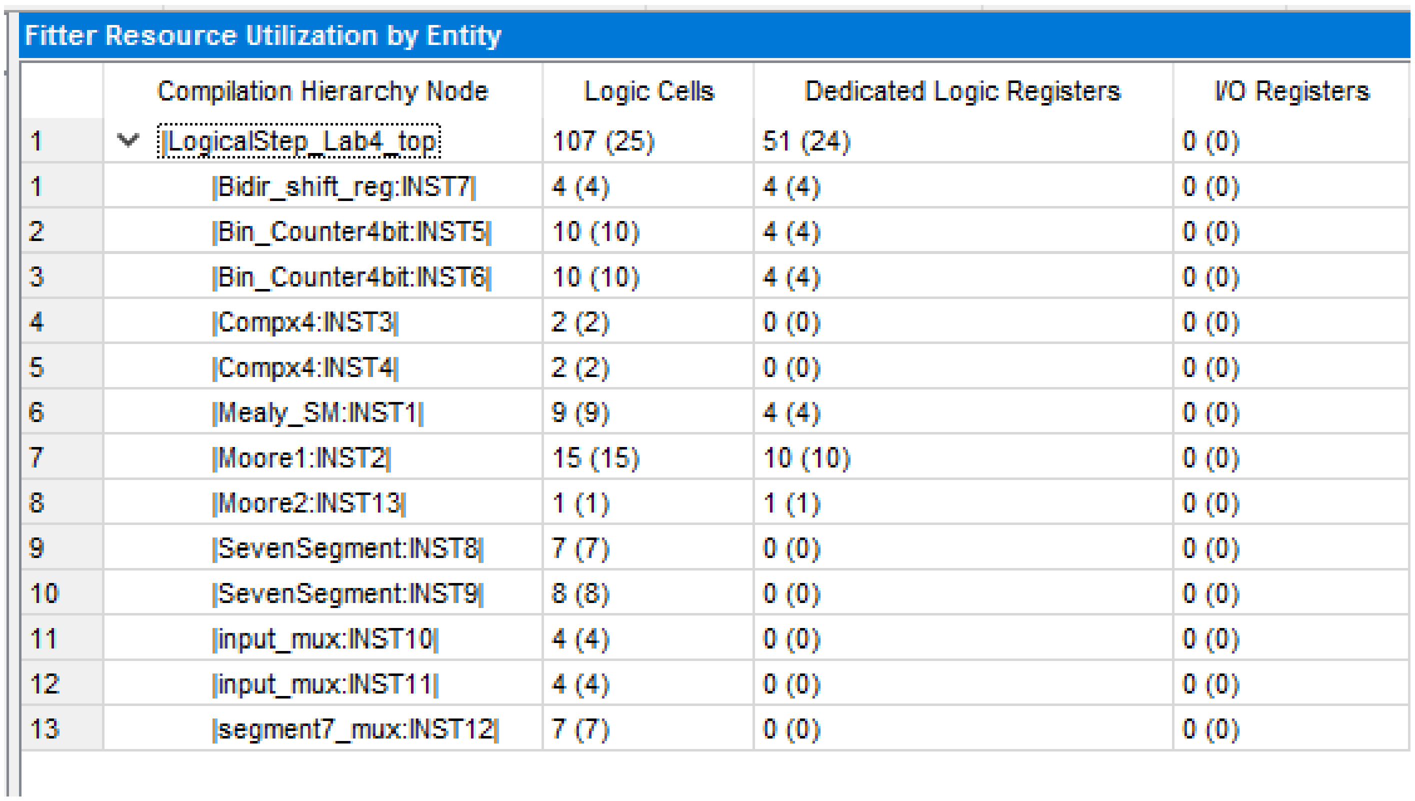
Simulations for Shift Register:



Simulations for Counter:



Fitter Report:



Fitter Resource Utilization by Entity

	Compilation Hierarchy Node	Logic Cells	Dedicated Logic Registers	I/O Registers
1	|LogicalStep_Lab4_top	107 (25)	51 (24)	0 (0)
1	|Bidir_shift_reg:INST7|	4 (4)	4 (4)	0 (0)
2	|Bin_Counter4bit:INST5|	10 (10)	4 (4)	0 (0)
3	|Bin_Counter4bit:INST6|	10 (10)	4 (4)	0 (0)
4	|Compx4:INST3|	2 (2)	0 (0)	0 (0)
5	|Compx4:INST4|	2 (2)	0 (0)	0 (0)
6	|Mealy_SM:INST1|	9 (9)	4 (4)	0 (0)
7	|Moore1:INST2|	15 (15)	10 (10)	0 (0)
8	|Moore2:INST13|	1 (1)	1 (1)	0 (0)
9	|SevenSegment:INST8|	7 (7)	0 (0)	0 (0)
10	|SevenSegment:INST9|	8 (8)	0 (0)	0 (0)
11	|input_mux:INST10|	4 (4)	0 (0)	0 (0)
12	|input_mux:INST11|	4 (4)	0 (0)	0 (0)
13	|segment7_mux:INST12|	7 (7)	0 (0)	0 (0)