Lab 4 Report

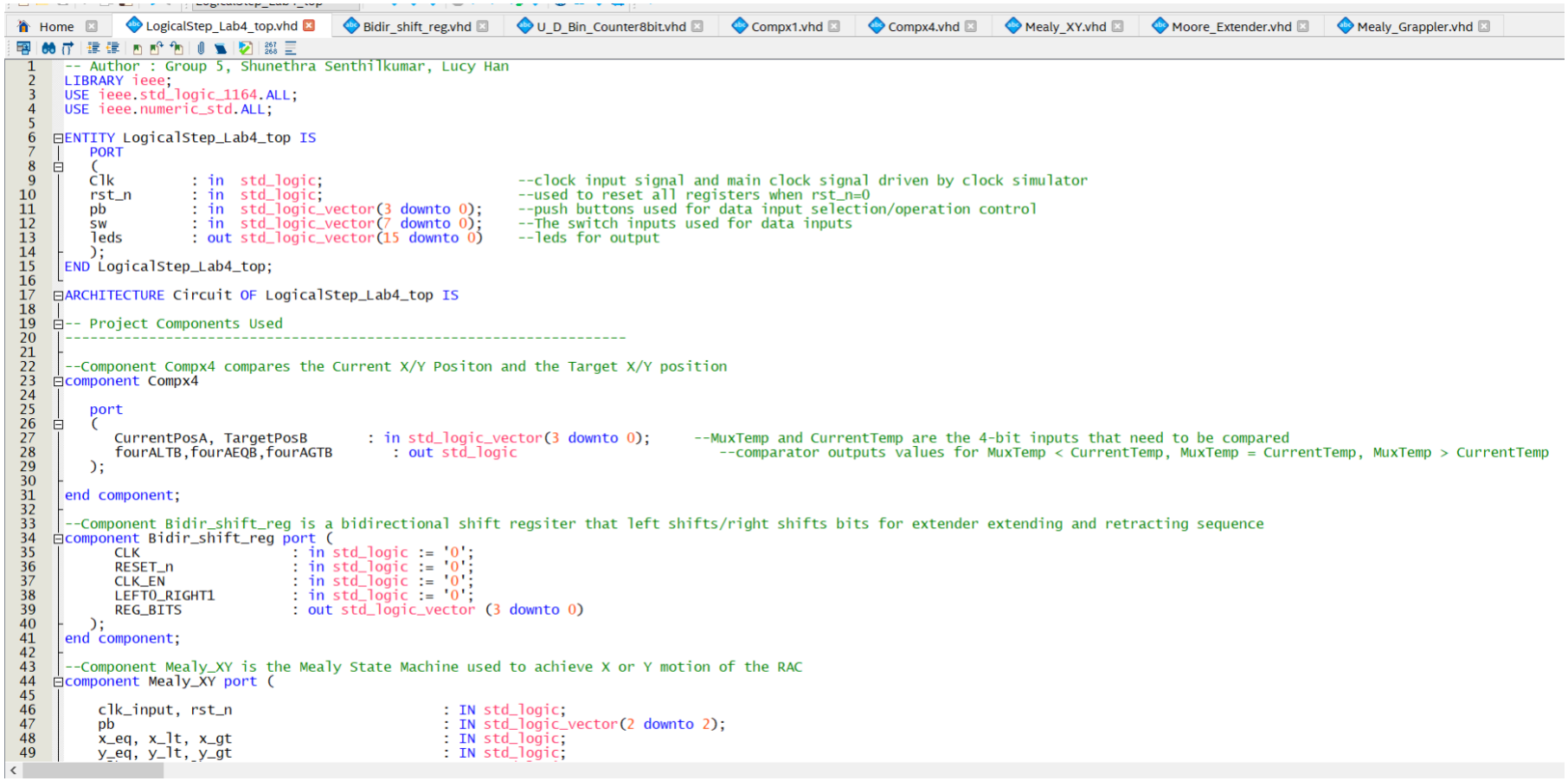
ECE 124

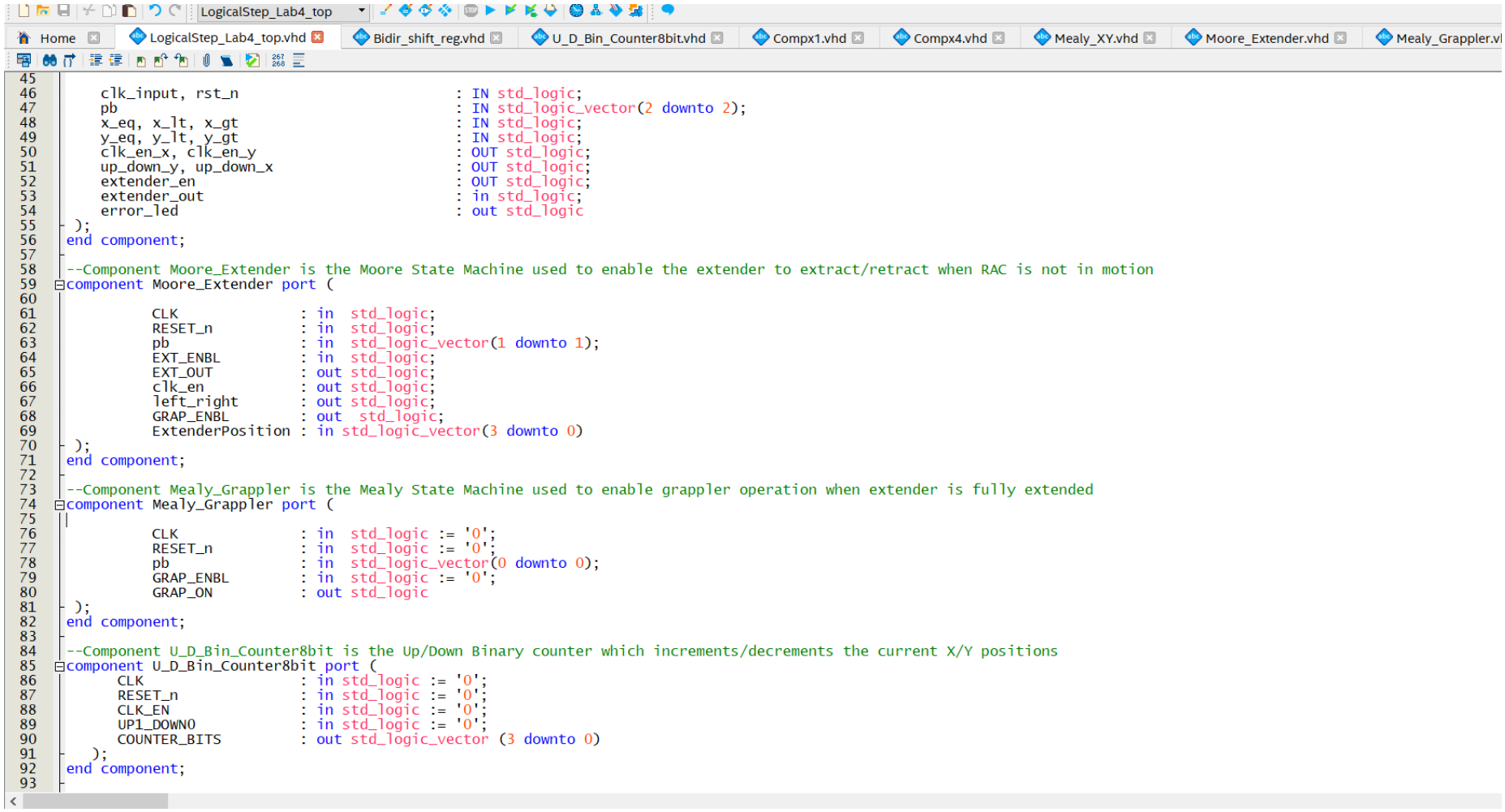
Group 5, Session 203

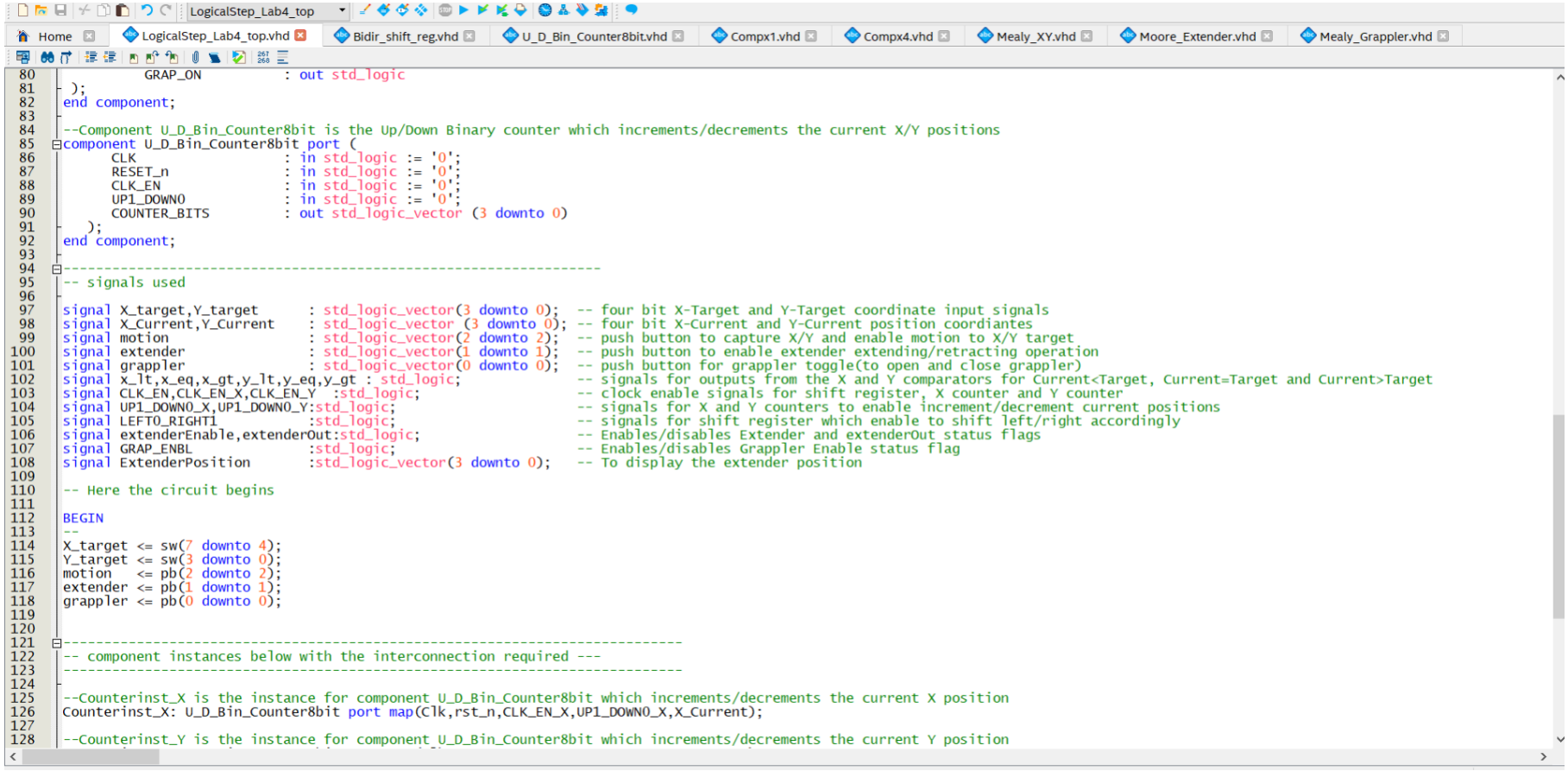
Authors/Team members: Shunethra Senthilkumar, Lucy Han

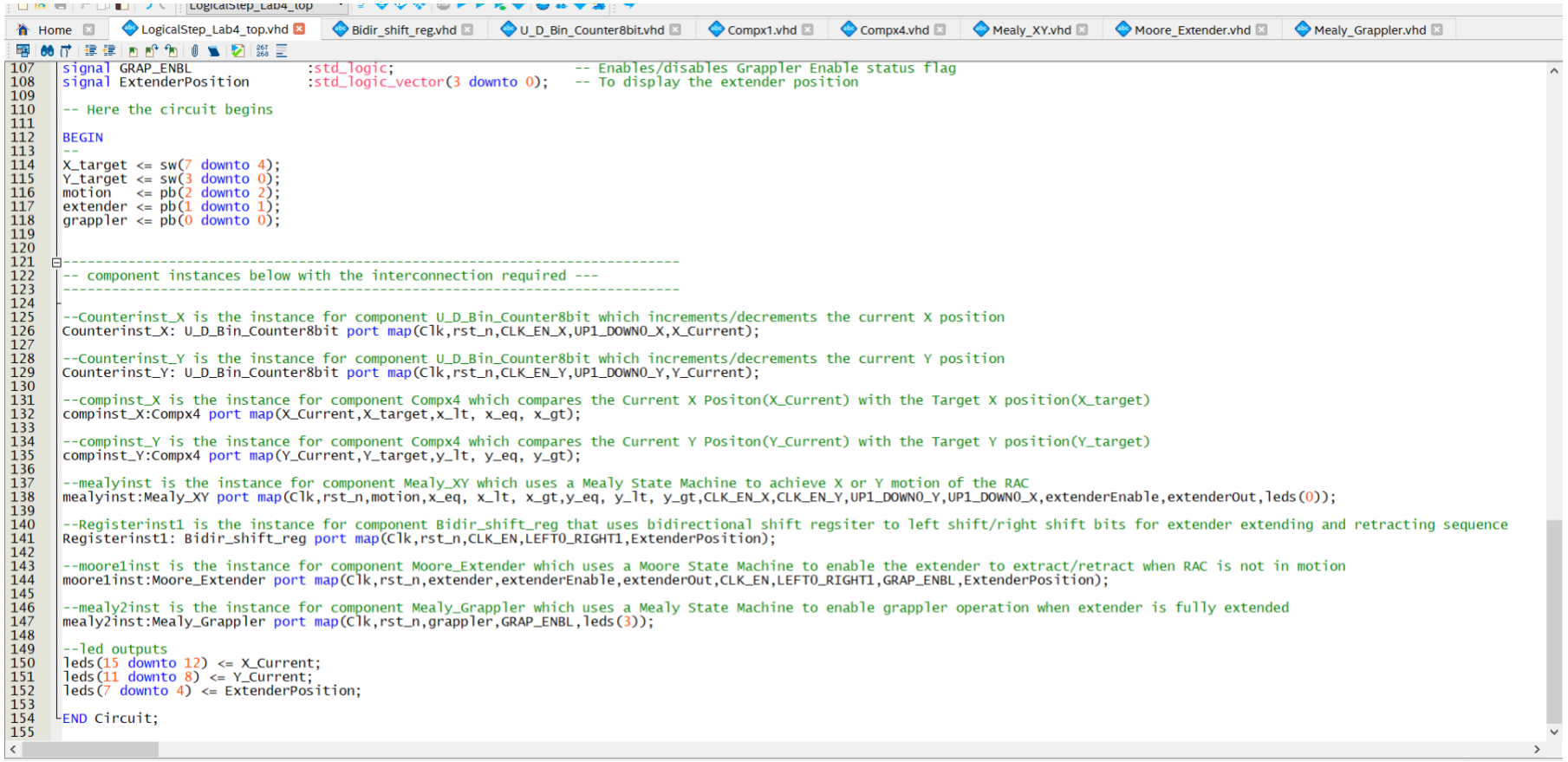
VHDL DESIGN

LogicalStep_Lab4_top.vhd




LogicalStep_Lab4_top
Home
LogicalStep_Lab4_top.vhd
Bidir_shift_reg.vhd
U_D_Bin_Counter8bit.vhd
Compx1.vhd
Compx4.vhd
Mealy_XY.vhd
Moore_Extender.vhd
Mealy_Grappler.vh
45
46 clk_input, rst_n : IN std_logic;
47 pb : IN std_logic_vector(2 downto 2);
48 x_eq, x_lt, x_gt : IN std_logic;
49 y_eq, y_lt, y_gt : IN std_logic;
50 clk_en_x, clk_en_y : OUT std_logic;
51 up_down_y, up_down_x : OUT std_logic;
52 extender_en : OUT std_logic;
53 extender_out : in std_logic;
54 error_led : out std_logic
55);
56 end component;
57
58 --Component Moore_Extender is the Moore State Machine used to enable the extender to extract/retract when RAC is not in motion
59 component Moore_Extender port (
60
61 CLK : in std_logic;
62 RESET_n : in std_logic;
63 pb : in std_logic_vector(1 downto 1);
64 EXT_ENBL : in std_logic;
65 EXT_OUT : out std_logic;
66 clk_en : out std_logic;
67 left_right : out std_logic;
68 GRAP_ENBL : out std_logic;
69 ExtenderPosition : in std_logic_vector(3 downto 0)
70);
71 end component;
72
73 --Component Mealy_Grappler is the Mealy State Machine used to enable grappler operation when extender is fully extended
74 component Mealy_Grappler port (
75
76 CLK : in std_logic := '0';
77 RESET_n : in std_logic := '0';
78 pb : in std_logic_vector(0 downto 0);
79 GRAP_ENBL : in std_logic := '0';
80 GRAP_ON : out std_logic
81);
82 end component;
83
84 --Component U_D_Bin_Counter8bit is the Up/Down Binary counter which increments/decrements the current X/Y positions
85 component U_D_Bin_Counter8bit port (
86 CLK : in std_logic := '0';
87 RESET_n : in std_logic := '0';
88 CLK_EN : in std_logic := '0';
89 UP1_DOWN0 : in std_logic := '0';
90 COUNTER_BITS : out std_logic_vector (3 downto 0)
91);
92 end component;
93


LogicalStep_Lab4_top
Home
LogicalStep_Lab4_top.vhd
Bidir_shift_reg.vhd
U_D_Bin_Counter8bit.vhd
Compx1.vhd
Compx4.vhd
Mealy_XY.vhd
Moore_Extender.vhd
Mealy_Grappler.vhd
GRAP_ON : out std_logic
);
end component;
--Component U_D_Bin_Counter8bit is the Up/Down Binary counter which increments/decrements the current X/Y positions
component U_D_Bin_Counter8bit port (
CLK : in std_logic := '0';
RESET_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;
-- signals used
signal X_target,Y_target : std_logic_vector(3 downto 0); -- four bit X-Target and Y-Target coordinate input signals
signal X_Current,Y_Current : std_logic_vector (3 downto 0); -- four bit X-Current and Y-Current position coordiantes
signal motion : std_logic_vector(2 downto 2); -- push button to capture X/Y and enable motion to X/Y target
signal extender : std_logic_vector(1 downto 1); -- push button to enable extender extending/retracting operation
signal grappler : std_logic_vector(0 downto 0); -- push button for grappler toggle(to open and close grappler)
signal x_lt,x_eq,x_gt,y_lt,y_eq,y_gt : std_logic; -- signals for outputs from the X and Y comparators for Current<Target, Current=Target and Current>Target
signal CLK_EN,CLK_EN_X,CLK_EN_Y :std_logic; -- clock enable signals for shift register, X counter and Y counter
signal UP1_DOWN0_X,UP1_DOWN0_Y:std_logic; -- signals for X and Y counters to enable increment/decrement current positions
signal LEFT0_RIGHT1 :std_logic; -- signals for shift register which enable to shift left/right accordingly
signal extenderEnable,extenderOut:std_logic; -- Enables/disables Extender and extenderOut status flags
signal GRAP_ENBL :std_logic; -- Enables/disables Grappler Enable status flag
signal ExtenderPosition :std_logic_vector(3 downto 0); -- To display the extender position
-- Here the circuit begins
BEGIN
--
X_target <= sw(7 downto 4);
Y_target <= sw(3 downto 0);
motion <= pb(2 downto 2);
extender <= pb(1 downto 1);
grappler <= pb(0 downto 0);
-- component instances below with the interconnection required ---
--Counterinst_X is the instance for component U_D_Bin_Counter8bit which increments/decrements the current X position
Counterinst_X: U_D_Bin_Counter8bit port map(Clk,rst_n,CLK_EN_X,UP1_DOWN0_X,X_Current);
--Counterinst_Y is the instance for component U_D_Bin_Counter8bit which increments/decrements the current Y position


Home
LogicalStep_Lab4_top.vhd
Bidir_shift_reg.vhd
U_D_Bin_Counter8bit.vhd
Compx1.vhd
Compx4.vhd
Mealy_XY.vhd
Moore_Extender.vhd
Mealy_Grappler.vhd
signal GRAP_ENBL :std_logic; -- Enables/disables Grappler Enable status flag
signal ExtenderPosition :std_logic_vector(3 downto 0); -- To display the extender position
-- Here the circuit begins
BEGIN
--
X_target <= sw(7 downto 4);
Y_target <= sw(3 downto 0);
motion <= pb(2 downto 2);
extender <= pb(1 downto 1);
grappler <= pb(0 downto 0);
-- component instances below with the interconnection required ---
--Counterinst_X is the instance for component U_D_Bin_Counter8bit which increments/decrements the current X position
Counterinst_X: U_D_Bin_Counter8bit port map(Clk,rst_n,CLK_EN_X,UP1_DOWNO_X,X_Current);
--Counterinst_Y is the instance for component U_D_Bin_Counter8bit which increments/decrements the current Y position
Counterinst_Y: U_D_Bin_Counter8bit port map(Clk,rst_n,CLK_EN_Y,UP1_DOWNO_Y,Y_Current);
--compinst_X is the instance for component Compx4 which compares the Current X Positon(X_Current) with the Target X position(X_target)
compinst_X:Compx4 port map(X_Current,X_target,x_lt, x_eq, x_gt);
--compinst_Y is the instance for component Compx4 which compares the Current Y Positon(Y_Current) with the Target Y position(Y_target)
compinst_Y:Compx4 port map(Y_Current,Y_target,y_lt, y_eq, y_gt);
--mealyinst is the instance for component Mealy_XY which uses a Mealy State Machine to achieve X or Y motion of the RAC
mealyinst:Mealy_XY port map(Clk,rst_n,motion,x_eq, x_lt, x_gt,y_eq, y_lt, y_gt,CLK_EN_X,CLK_EN_Y,UP1_DOWNO_Y,UP1_DOWNO_X,extenderEnable,extenderOut,leds(0));
--Registerinst1 is the instance for component Bidir_shift_reg that uses bidirectional shift regsiter to left shift/right shift bits for extender extending and retracting sequence
Registerinst1: Bidir_shift_reg port map(Clk,rst_n,CLK_EN,LEFTO_RIGHT1,ExtenderPosition);
--moore1inst is the instance for component Moore_Extender which uses a Moore State Machine to enable the extender to extract/retract when RAC is not in motion
moore1inst:Moore_Extender port map(Clk,rst_n,extender,extenderEnable,extenderOut,CLK_EN,LEFTO_RIGHT1,GRAP_ENBL,ExtenderPosition);
--mealy2inst is the instance for component Mealy_Grappler which uses a Mealy State Machine to enable grappler operation when extender is fully extended
mealy2inst:Mealy_Grappler port map(Clk,rst_n,grappler,GRAP_ENBL,leds(3));
--led outputs
leds(15 downto 12) <= X_Current;
leds(11 downto 8) <= Y_Current;
leds(7 downto 4) <= ExtenderPosition;
END Circuit;

Compx4.vhd

Home | LogicalStep_Lab4_top.vhd | Compx4.vhd | Bidir_shift_reg.vhd | U_D_Bin_Counter8bit.vhd | Compx1.vhd | Mealy_XY.vhd | Moore_Extender.vhd

```
1  -- Author : Group 5, Shunethra Senthilkumar, Lucy Han
2  library ieee;
3  use ieee.std_logic_1164.all;
4  use ieee.numeric_std.all;
5
6  --Compx4 compares the Current X/Y Positon and the Target X/Y position
7  entity Compx4 is
8
9     port
10    (
11       CurrentPosA, TargetPosB       : in std_logic_vector(3 downto 0);
12       fourALTB,fourAEQB,fourAGTB       : out std_logic
13    );
14
15 end entity;
16
17 architecture fourbitcomparator of Compx4 is
18
19 --component Compx1 is a single bit magnitude comparator used to compare single bit inputs for A(MuxTemp) and B(CurrentTemp)
20 component Compx1
21 port
22    (
23       A,B                  : in std_logic;
24       AGTB,AEQB,ALTB       : out std_logic
25    );
26 end component;
27
28 -- signals used
29 signal AGTB,AEQB,ALTB    : std_logic_vector(3 downto 0);                  -- AGTB is A greater than B
30                                                                           -- AEQB is A equal to B
31                                                                           -- ALTB is A lesser than B
32
33 begin
34
35
36 -- 4 instances of Compx1 to compare individual/single bit inputs of Current X/Y Positon and the Target X/Y position(bit by bit comparison)
37 bit3: Compx1 port map (CurrentPosA(3),TargetPosB(3),AGTB(3),AEQB(3),ALTB(3));
38
39 bit2: Compx1 port map (CurrentPosA(2),TargetPosB(2),AGTB(2),AEQB(2),ALTB(2));
40
41 bit1: Compx1 port map (CurrentPosA(1),TargetPosB(1),AGTB(1),AEQB(1),ALTB(1));
42
43 bit0: Compx1 port map (CurrentPosA(0),TargetPosB(0),AGTB(0),AEQB(0),ALTB(0));
44
45
46 --Value of Current X/Y Positon lesser than Target X/Y position
47 fourALTB <=  (ALTB(3)) OR (AEQB(3) AND ALTB(2)) OR (AEQB(3) AND AEQB(2) AND ALTB(1)) OR (AEQB(3) AND AEQB(2) AND AEQB(1) AND ALTB(0));
48
49 --Value of Current X/Y Positon equal to Target X/Y position
```

```vhdl
entity Compx4 is

    port
    (
        CurrentPosA, TargetPosB       : in std_logic_vector(3 downto 0);
        fourALTB,fourAEQB,fourAGTB       : out std_logic
    );

end entity;

architecture fourbitcomparator of Compx4 is

--component Compx1 is a single bit magnitude comparator used to compare single bit inputs for A(MuxTemp) and B(CurrentTemp)
component Compx1
port
    (
        A,B                 : in std_logic;
        AGTB,AEQB,ALTB      : out std_logic
    );
end component;

-- signals used
signal AGTB,AEQB,ALTB   : std_logic_vector(3 downto 0);                    -- AGTB is A greater than B
                                                                           -- AEQB is A equal to B
                                                                           -- ALTB is A lesser than B

begin


-- 4 instances of Compx1 to compare individual/single bit inputs of Current X/Y Positon and the Target X/Y position(bit by bit comparison)
bit3: Compx1 port map (CurrentPosA(3),TargetPosB(3),AGTB(3),AEQB(3),ALTB(3));

bit2: Compx1 port map (CurrentPosA(2),TargetPosB(2),AGTB(2),AEQB(2),ALTB(2));

bit1: Compx1 port map (CurrentPosA(1),TargetPosB(1),AGTB(1),AEQB(1),ALTB(1));

bit0: Compx1 port map (CurrentPosA(0),TargetPosB(0),AGTB(0),AEQB(0),ALTB(0));


--Value of Current X/Y Positon lesser than Target X/Y position
fourALTB <=  (ALTB(3)) OR (AEQB(3) AND ALTB(2)) OR (AEQB(3) AND AEQB(2) AND ALTB(1)) OR (AEQB(3) AND AEQB(2) AND AEQB(1) AND ALTB(0));

--Value of Current X/Y Positon equal to Target X/Y position
fourAEQB <=  (AEQB(3) AND AEQB(2) AND AEQB(1) AND AEQB(0));

--Current X/Y Positon greater than Target X/Y position
fourAGTB <=  (AGTB(3)) OR (AEQB(3) AND AGTB(2)) OR (AEQB(3) AND AEQB(2) AND AGTB(1)) OR (AEQB(3) AND AEQB(2) AND AEQB(1) AND AGTB(0));

end fourbitcomparator;
```

Compx1.vhd

```vhdl
-- Author : Group 5, Shunethra Senthilkumar, Lucy Han
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

--Compx1 is a single bit magnitude comparator used to compare single bit inputs for A(X/Y Current position) and B(X/YTarget Position)
entity Compx1 is

    port
    (
        A,B                 : in std_logic;         --single bit inputs A and B from 4-bit inputs
        AGTB,AEQB,ALTB      : out std_logic         --single bit comparator output values for A>B, A=B, A<B
    );

end entity;

architecture comparator of Compx1 is



begin

--Value for A greater than B using AND and NOT gates
AGTB <= A AND (NOT B);

--Value for A equal to B using XNOR gate
AEQB <= A XNOR B;

--Value of A lesser than B using AND and NOT gates
ALTB <= B AND (NOT A);

end comparator;
```

Bidir_shift_reg.vhd

Home | LogicalStep_Lab4_top.vhd | Compx4.vhd | Bidir_shift_reg.vhd | U_D_Bin_Counter8bit.vhd | Compx1.vhd | Mealy_XY.vhd | Moore_Extender.vhd | Mealy_G

```vhdl
-- Author : Group 5, Shunethra Senthilkumar, Lucy Han
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

--Bidir_shift_reg is a bidirectional shift regsiter that left shifts/right shifts bits for extender extending and retracting sequence
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);  --4 bit signal which is used to left/right shift the bits

begin


reg:process (CLK, RESET_n) is
    begin
        if (RESET_n = '0') then
            sreg <= "0000";                                 --sreg is set to "0000" when RESET_n is active/0

        elsif ((rising_edge(CLK)) AND (CLK_EN ='1')) then     --during rising edge of clock and when clock is enabled, shifting of bits can happen

            if (LEFT0_RIGHT1 = '1')then
                sreg (3 downto 0) <= '1' & sreg (3 downto 1);   --right shift of bits

            elsif (LEFT0_RIGHT1 = '0')then
                sreg (3 downto 0) <= sreg (2 downto 0) & '0';   --left shift of bits

            end if;

        end if;

end process;
REG_BITS <= sreg;

end one;
```

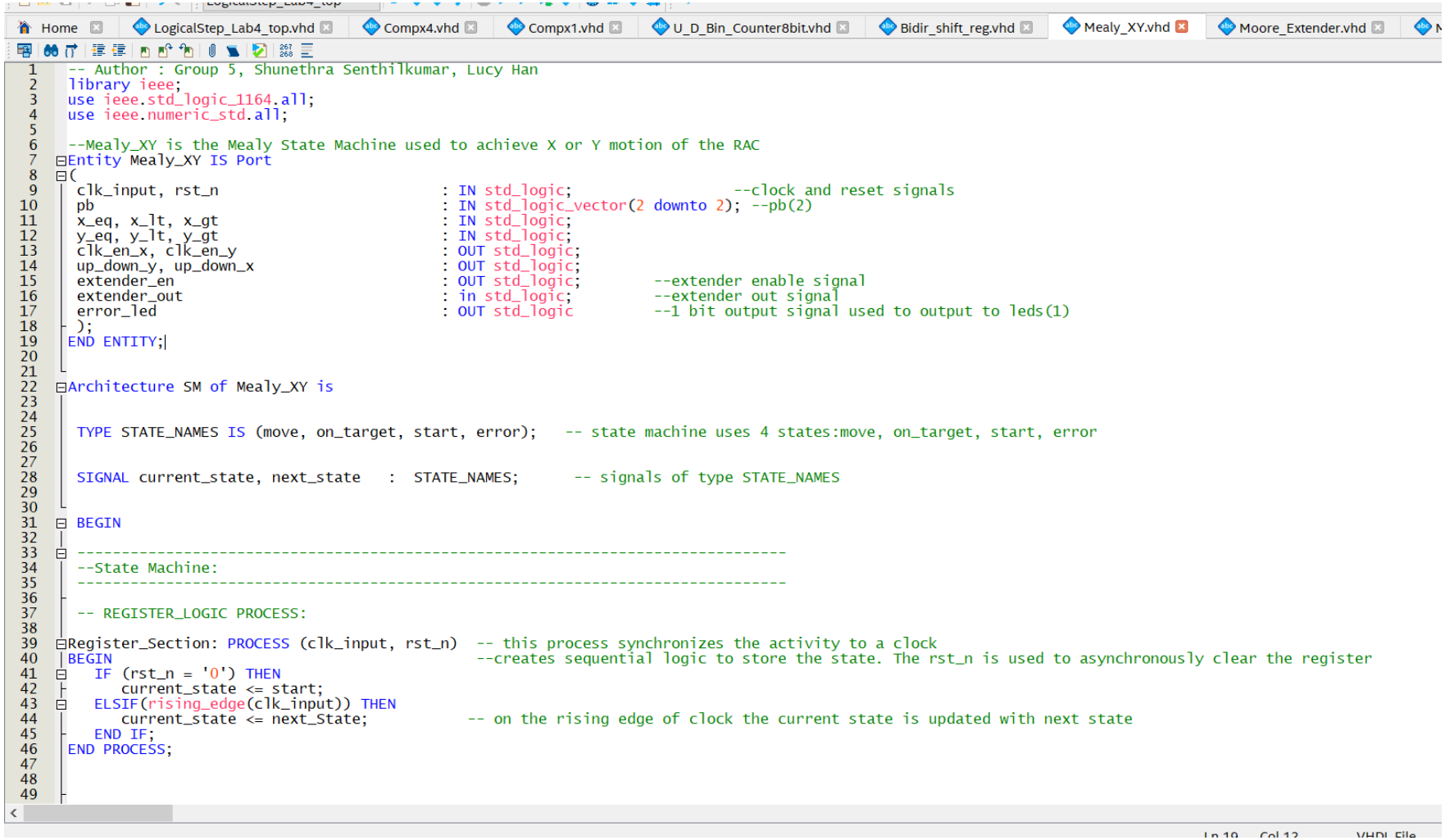
U_D_Bin_Counter8bit.vhd

```vhdl
-- Author : Group 5, Shunethra Senthilkumar, Lucy Han
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

--U_D_Bin_Counter8bit is the Up/Down Binary counter which increments/decrements the current X/Y positions
entity U_D_Bin_Counter8bit is

   port
   (
      CLK                  : in std_logic := '0';
      RESET_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 U_D_Bin_Counter8bit is

signal ud_bincounter    : UNSIGNED(3 downto 0);    --4 bit signal which is incremented/decremented

begin


process (CLK, RESET_n) is
   begin
      if (RESET_n = '0') then
         ud_bincounter <= "0000";

      elsif (rising_edge(CLK)) then          --during rising edge of clock, counter can function

         --both increment/decrement are enabled when CLK_EN clock signal is active
         if ((UP1_DOWN0 = '1') AND (CLK_EN ='1'))then
            ud_bincounter <=(ud_bincounter+1);  --functions as an upcounter

         elsif ((UP1_DOWN0 = '0') AND (CLK_EN ='1'))then
            ud_bincounter <=(ud_bincounter-1);  --functions a down counter

         end if;

      end if;

end process;
COUNTER_BITS <= std_logic_vector(ud_bincounter);

end;
```

State Machine VHDL design files:

There are three state machines used in our design. We used two Mealy state machines and one Moore state machine.

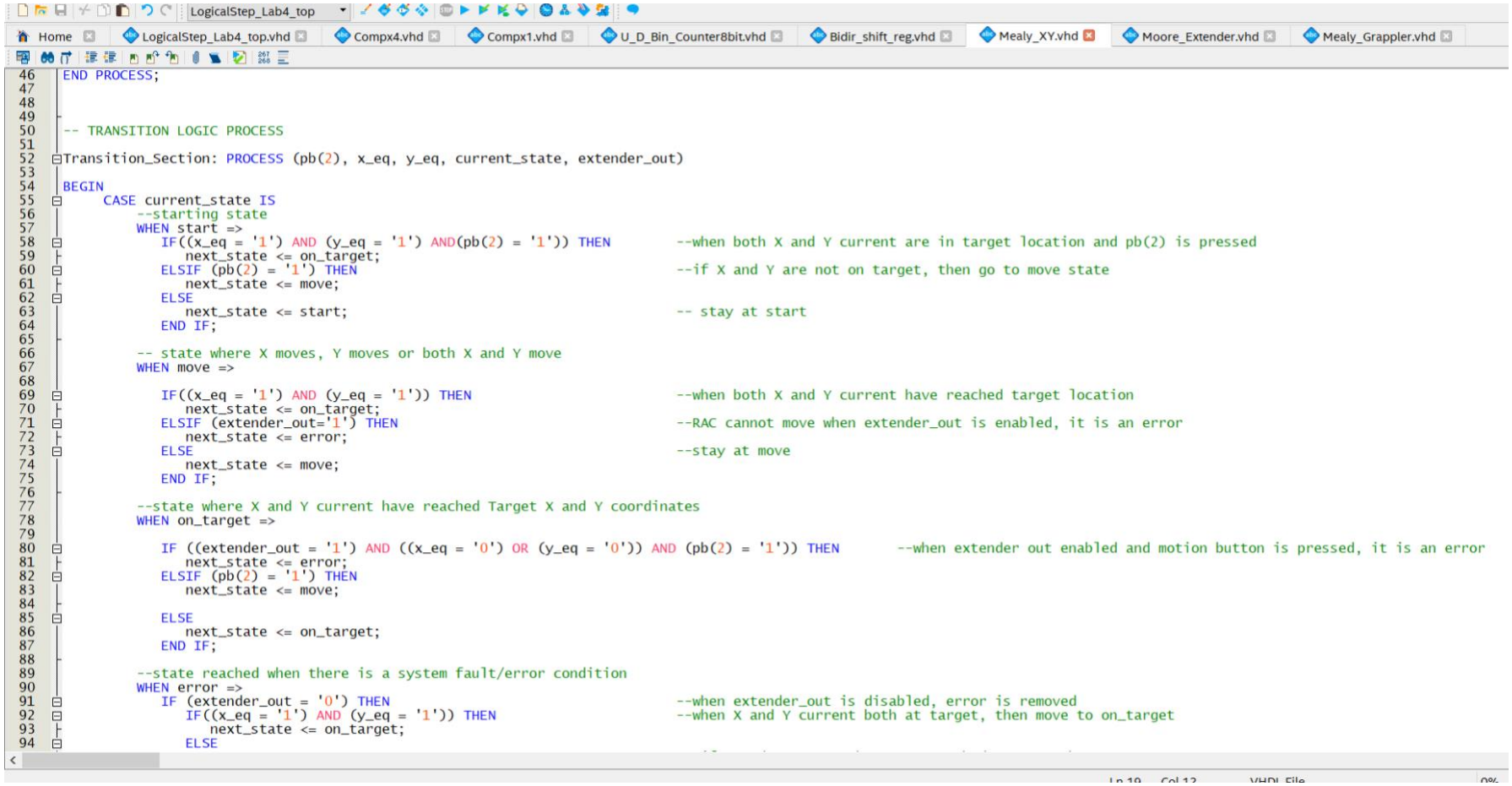
Mealy_XY.vhd



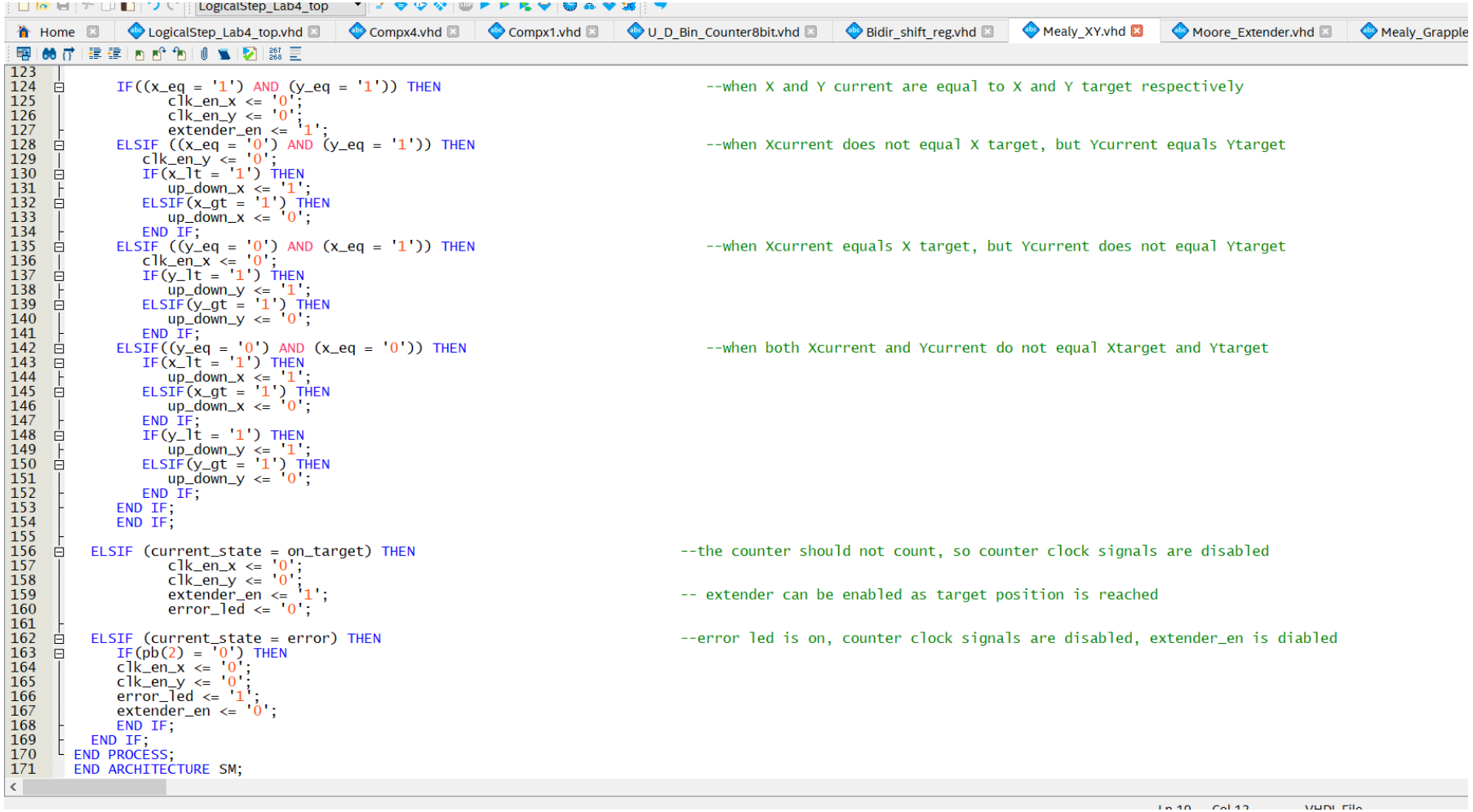
Home | LogicalStep_Lab4_top.vhd | Compx4.vhd | Compx1.vhd | U_D_Bin_Counter8bit.vhd | Bidir_shift_reg.vhd | Mealy_XY.vhd | Moore_Extender.vhd

```
1  -- Author : Group 5, Shunethra Senthilkumar, Lucy Han
2  library ieee;
3  use ieee.std_logic_1164.all;
4  use ieee.numeric_std.all;
5
6  --Mealy_XY is the Mealy State Machine used to achieve X or Y motion of the RAC
7  Entity Mealy_XY IS Port
8  (
9   clk_input, rst_n                    : IN std_logic;                  --clock and reset signals
10  pb                                  : IN std_logic_vector(2 downto 2); --pb(2)
11  x_eq, x_lt, x_gt                    : IN std_logic;
12  y_eq, y_lt, y_gt                    : IN std_logic;
13  clk_en_x, clk_en_y                  : OUT std_logic;
14  up_down_y, up_down_x                : OUT std_logic;
15  extender_en                         : OUT std_logic;        --extender enable signal
16  extender_out                        : in std_logic;         --extender out signal
17  error_led                           : OUT std_logic         --1 bit output signal used to output to leds(1)
18  );
19 END ENTITY;
20
21
22 Architecture SM of Mealy_XY is
23
24
25  TYPE STATE_NAMES IS (move, on_target, start, error);   -- state machine uses 4 states:move, on_target, start, error
26
27
28  SIGNAL current_state, next_state   :  STATE_NAMES;      -- signals of type STATE_NAMES
29
30
31  BEGIN
32
33  ----------------------------------------------------------------------------
34  --State Machine:
35  ----------------------------------------------------------------------------
36
37  -- REGISTER_LOGIC PROCESS:
38
39 Register_Section: PROCESS (clk_input, rst_n)  -- this process synchronizes the activity to a clock
40 BEGIN                                         --creates sequential logic to store the state. The rst_n is used to asynchronously clear the register
41    IF (rst_n = '0') THEN
42       current_state <= start;
43    ELSIF(rising_edge(clk_input)) THEN
44       current_state <= next_State;           -- on the rising edge of clock the current state is updated with next state
45    END IF;
46 END PROCESS;
47
48
49
```

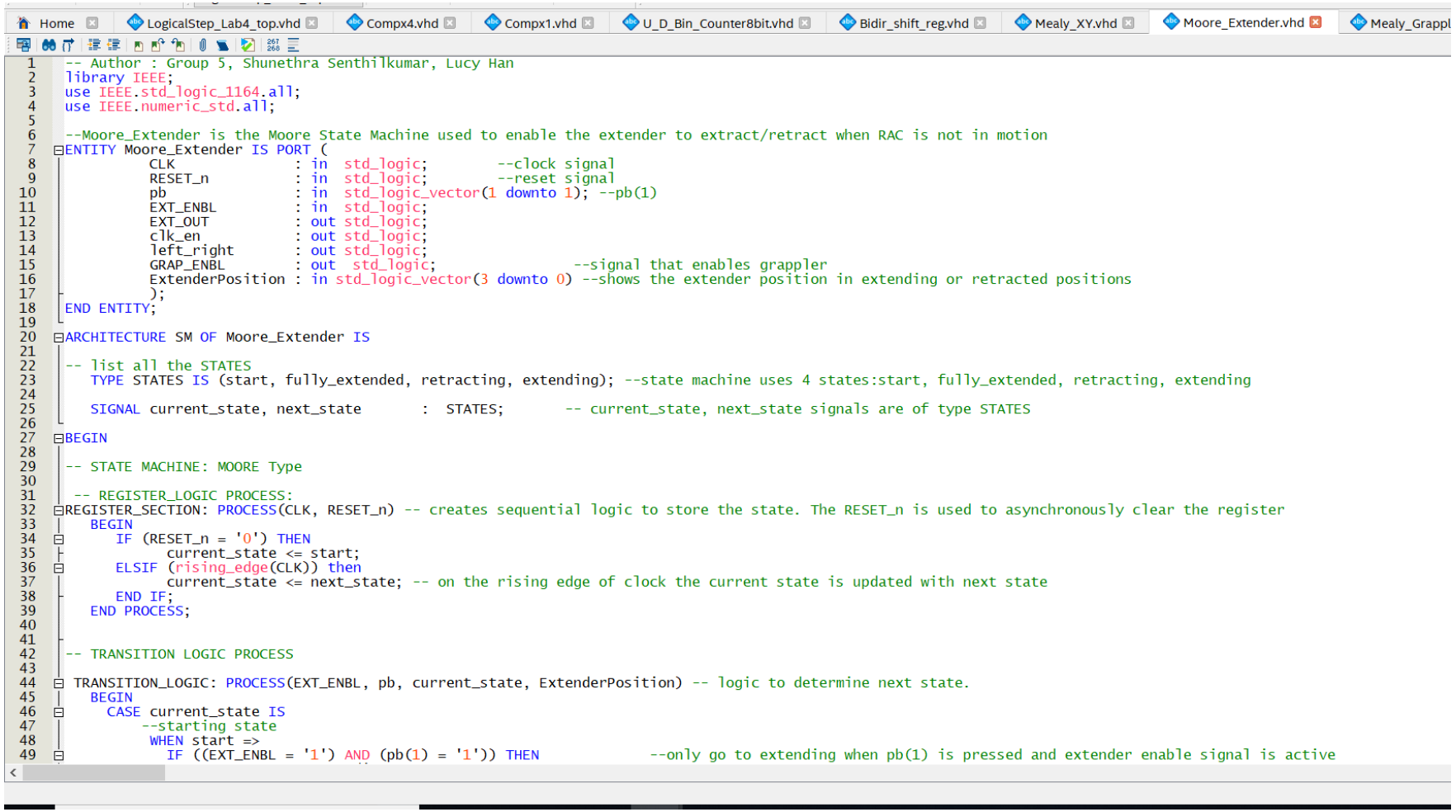
Ln 19 Col 13 VHDL File




```
46  END PROCESS;
47
48
49
50  -- TRANSITION LOGIC PROCESS
51
52  Transition_Section: PROCESS (pb(2), x_eq, y_eq, current_state, extender_out)
53
54  BEGIN
55      CASE current_state IS
56          --starting state
57          WHEN start =>
58             IF((x_eq = '1') AND (y_eq = '1') AND(pb(2) = '1')) THEN        --when both X and Y current are in target location and pb(2) is pressed
59                next_state <= on_target;
60             ELSIF (pb(2) = '1') THEN                                       --if X and Y are not on target, then go to move state
61                next_state <= move;
62             ELSE
63                next_state <= start;                                        -- stay at start
64             END IF;
65
66          -- state where X moves, Y moves or both X and Y move
67          WHEN move =>
68
69             IF((x_eq = '1') AND (y_eq = '1')) THEN                         --when both X and Y current have reached target location
70                next_state <= on_target;
71             ELSIF (extender_out='1') THEN                                  --RAC cannot move when extender_out is enabled, it is an error
72                next_state <= error;
73             ELSE                                                           --stay at move
74                next_state <= move;
75             END IF;
76
77          --state where X and Y current have reached Target X and Y coordinates
78          WHEN on_target =>
79
80             IF ((extender_out = '1') AND ((x_eq = '0') OR (y_eq = '0')) AND (pb(2) = '1')) THEN       --when extender out enabled and motion button is pressed, it is an error
81                next_state <= error;
82             ELSIF (pb(2) = '1') THEN
83                next_state <= move;
84
85             ELSE
86                next_state <= on_target;
87             END IF;
88
89          --state reached when there is a system fault/error condition
90          WHEN error =>
91             IF (extender_out = '0') THEN                                   --when extender_out is disabled, error is removed
92                IF((x_eq = '1') AND (y_eq = '1')) THEN                      --when X and Y current both at target, then move to on_target
93                   next_state <= on_target;
94                ELSE
```

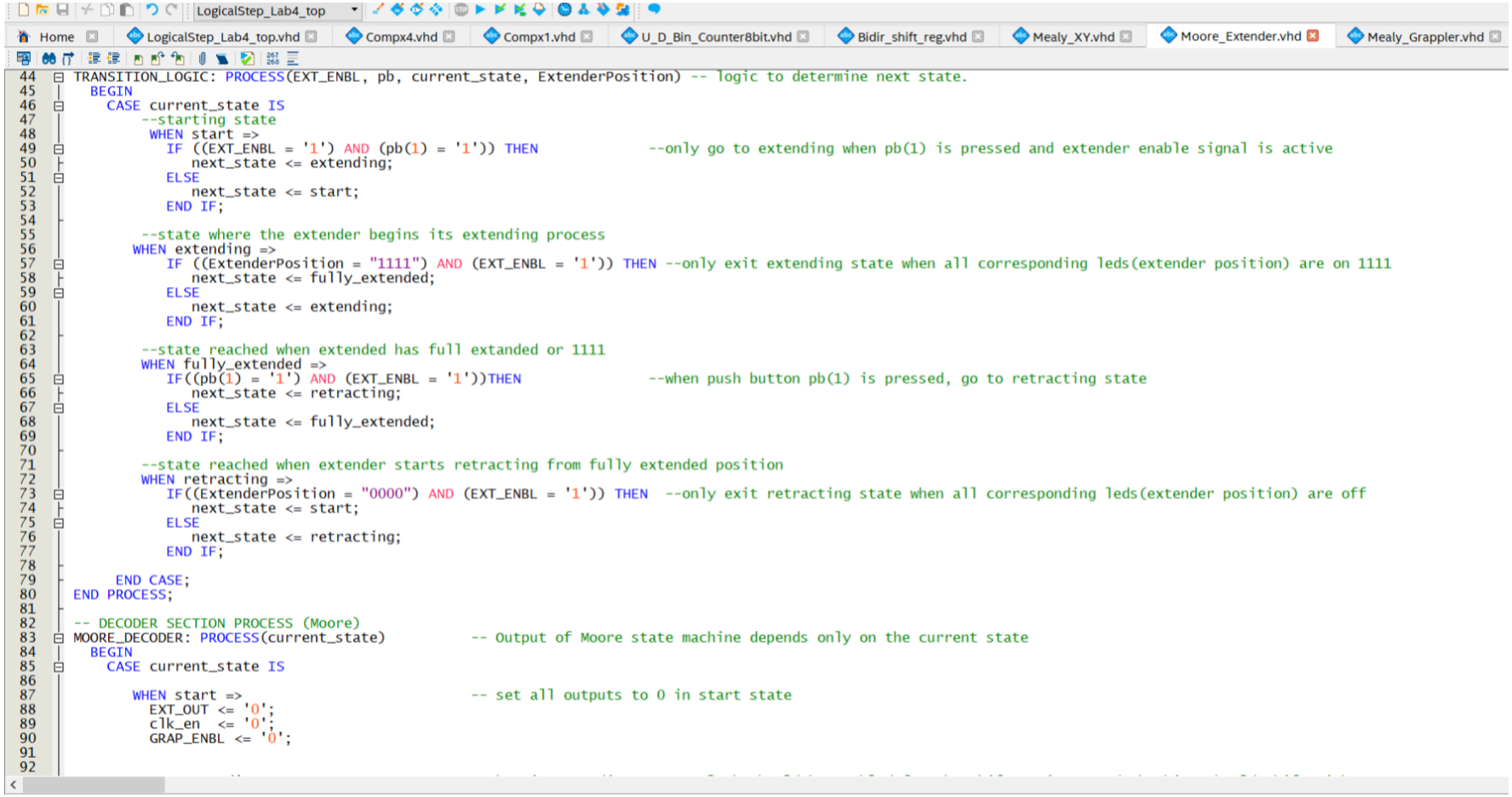

LogicalStep_Lab4_top
Home
LogicalStep_Lab4_top.vhd
Compx4.vhd
Compx1.vhd
U_D_Bin_Counter8bit.vhd
Bidir_shift_reg.vhd
Mealy_XY.vhd
Moore_Extender.vhd
Mealy_Grapple
123
124 IF((x_eq = '1') AND (y_eq = '1')) THEN --when X and Y current are equal to X and Y target respectively
125 clk_en_x <= '0';
126 clk_en_y <= '0';
127 extender_en <= '1';
128 ELSIF ((x_eq = '0') AND (y_eq = '1')) THEN --when Xcurrent does not equal X target, but Ycurrent equals Ytarget
129 clk_en_y <= '0';
130 IF(x_lt = '1') THEN
131 up_down_x <= '1';
132 ELSIF(x_gt = '1') THEN
133 up_down_x <= '0';
134 END IF;
135 ELSIF ((y_eq = '0') AND (x_eq = '1')) THEN --when Xcurrent equals X target, but Ycurrent does not equal Ytarget
136 clk_en_x <= '0';
137 IF(y_lt = '1') THEN
138 up_down_y <= '1';
139 ELSIF(y_gt = '1') THEN
140 up_down_y <= '0';
141 END IF;
142 ELSIF((y_eq = '0') AND (x_eq = '0')) THEN --when both Xcurrent and Ycurrent do not equal Xtarget and Ytarget
143 IF(x_lt = '1') THEN
144 up_down_x <= '1';
145 ELSIF(x_gt = '1') THEN
146 up_down_x <= '0';
147 END IF;
148 IF(y_lt = '1') THEN
149 up_down_y <= '1';
150 ELSIF(y_gt = '1') THEN
151 up_down_y <= '0';
152 END IF;
153 END IF;
154 END IF;
155
156 ELSIF (current_state = on_target) THEN --the counter should not count, so counter clock signals are disabled
157 clk_en_x <= '0';
158 clk_en_y <= '0';
159 extender_en <= '1'; -- extender can be enabled as target position is reached
160 error_led <= '0';
161
162 ELSIF (current_state = error) THEN --error led is on, counter clock signals are disabled, extender_en is diabled
163 IF(pb(2) = '0') THEN
164 clk_en_x <= '0';
165 clk_en_y <= '0';
166 error_led <= '1';
167 extender_en <= '0';
168 END IF;
169 END IF;
170 END PROCESS;
171 END ARCHITECTURE SM;

Moore_Extender.vhd



Home | LogicalStep_Lab4_top.vhd | Compx4.vhd | Compx1.vhd | U_D_Bin_Counter8bit.vhd | Bidir_shift_reg.vhd | Mealy_XY.vhd | Moore_Extender.vhd | Mealy_Grappl

```
1  -- Author : Group 5, Shunethra Senthilkumar, Lucy Han
2  library IEEE;
3  use IEEE.std_logic_1164.all;
4  use IEEE.numeric_std.all;
5
6  --Moore_Extender is the Moore State Machine used to enable the extender to extract/retract when RAC is not in motion
7  ENTITY Moore_Extender IS PORT (
8            CLK              : in  std_logic;        --clock signal
9            RESET_n          : in  std_logic;        --reset signal
10           pb               : in  std_logic_vector(1 downto 1); --pb(1)
11           EXT_ENBL         : in  std_logic;
12           EXT_OUT          : out std_logic;
13           clk_en           : out std_logic;
14           left_right       : out std_logic;
15           GRAP_ENBL        : out  std_logic;               --signal that enables grappler
16           ExtenderPosition : in std_logic_vector(3 downto 0) --shows the extender position in extending or retracted positions
17           );
18 END ENTITY;
19
20 ARCHITECTURE SM OF Moore_Extender IS
21
22 -- list all the STATES
23    TYPE STATES IS (start, fully_extended, retracting, extending); --state machine uses 4 states:start, fully_extended, retracting, extending
24
25    SIGNAL current_state, next_state       :  STATES;       -- current_state, next_state signals are of type STATES
26
27 BEGIN
28
29 -- STATE MACHINE: MOORE Type
30
31  -- REGISTER_LOGIC PROCESS:
32 REGISTER_SECTION: PROCESS(CLK, RESET_n) -- creates sequential logic to store the state. The RESET_n is used to asynchronously clear the register
33    BEGIN
34       IF (RESET_n = '0') THEN
35             current_state <= start;
36       ELSIF (rising_edge(CLK)) then
37             current_state <= next_state; -- on the rising edge of clock the current state is updated with next state
38       END IF;
39    END PROCESS;
40
41
42 -- TRANSITION LOGIC PROCESS
43
44  TRANSITION_LOGIC: PROCESS(EXT_ENBL, pb, current_state, ExtenderPosition) -- logic to determine next state.
45    BEGIN
46      CASE current_state IS
47          --starting state
48           WHEN start =>
49             IF ((EXT_ENBL = '1') AND (pb(1) = '1')) THEN            --only go to extending when pb(1) is pressed and extender enable signal is active
```


LogicalStep_Lab4_top
Home
LogicalStep_Lab4_top.vhd
Compx4.vhd
Compx1.vhd
U_D_Bin_Counter8bit.vhd
Bidir_shift_reg.vhd
Mealy_XY.vhd
Moore_Extender.vhd
Mealy_Grappler.vhd
TRANSITION_LOGIC: PROCESS(EXT_ENBL, pb, current_state, ExtenderPosition) -- logic to determine next state.
BEGIN
CASE current_state IS
--starting state
WHEN start =>
IF ((EXT_ENBL = '1') AND (pb(1) = '1')) THEN --only go to extending when pb(1) is pressed and extender enable signal is active
next_state <= extending;
ELSE
next_state <= start;
END IF;
--state where the extender begins its extending process
WHEN extending =>
IF ((ExtenderPosition = "1111") AND (EXT_ENBL = '1')) THEN --only exit extending state when all corresponding leds(extender position) are on 1111
next_state <= fully_extended;
ELSE
next_state <= extending;
END IF;
--state reached when extended has full extanded or 1111
WHEN fully_extended =>
IF((pb(1) = '1') AND (EXT_ENBL = '1'))THEN --when push button pb(1) is pressed, go to retracting state
next_state <= retracting;
ELSE
next_state <= fully_extended;
END IF;
--state reached when extender starts retracting from fully extended position
WHEN retracting =>
IF((ExtenderPosition = "0000") AND (EXT_ENBL = '1')) THEN --only exit retracting state when all corresponding leds(extender position) are off
next_state <= start;
ELSE
next_state <= retracting;
END IF;
END CASE;
END PROCESS;
-- DECODER SECTION PROCESS (Moore)
MOORE_DECODER: PROCESS(current_state) -- Output of Moore state machine depends only on the current state
BEGIN
CASE current_state IS
WHEN start => -- set all outputs to 0 in start state
EXT_OUT <= '0';
clk_en <= '0';
GRAP_ENBL <= '0';

Home | LogicalStep_Lab4_top.vhd | Compx4.vhd | Compx1.vhd | U_D_Bin_Counter8bit.vhd | Bidir_shift_reg.vhd | Mealy_XY.vhd | Moore_Extender.vhd | Mealy_G

```
71          --state reached when extender starts retracting from fully extended position
72          WHEN retracting =>
73             IF((ExtenderPosition = "0000") AND (EXT_ENBL = '1')) THEN  --only exit retracting state when all corresponding leds(extender position) are off
74                next_state <= start;
75             ELSE
76                next_state <= retracting;
77             END IF;
78
79       END CASE;
80    END PROCESS;
81
82    -- DECODER SECTION PROCESS (Moore)
83    MOORE_DECODER: PROCESS(current_state)          -- Output of Moore state machine depends only on the current state
84      BEGIN
85        CASE current_state IS
86
87           WHEN start =>                          -- set all outputs to 0 in start state
88             EXT_OUT <= '0';
89             clk_en  <= '0';
90             GRAP_ENBL <= '0';
91
92
93           WHEN extending =>                      --when in extending state, clock should be enabled for the shift register and the bits should shift right
94           IF(pb(1) ='0') THEN                    --extender starts extending only after after pb(1) is pressed and released
95             clk_en  <= '1';
96             left_right <= '1';
97             EXT_OUT <= '1';
98             GRAP_ENBL <= '0';
99           END IF;
100
101          WHEN fully_extended =>                 --clock should be disabled, and extender out and grappler are enabled for grappler's operations to begin
102             clk_en <= '0';
103             EXT_OUT <= '1';
104             GRAP_ENBL <= '1';
105
106          WHEN retracting =>                     --when in retracting state, clock should be enabled for the shift register and the bits should shift left
107          IF(pb(1) ='0') THEN                    --extender starts retracting only after after pb(1) is pressed and released
108            clk_en  <= '1';
109            left_right <= '0';
110            EXT_OUT <= '1';
111            GRAP_ENBL <= '0';
112          END IF;
113
114       END CASE;
115
116   END PROCESS;
117
118   END SM;
119
```

Mealy_Grappler.vhd

Home | LogicalStep_Lab4_top.vhd | Compx4.vhd | Compx1.vhd | U_D_Bin_Counter8bit.vhd | Bidir_shift_reg.vhd | Mealy_XY.vhd | Moore_Extender.vhd

```
1  -- Author : Group 5, Shunethra Senthilkumar, Lucy Han
2  library IEEE;
3  use IEEE.std_logic_1164.all;
4  use IEEE.numeric_std.all;
5
6  --Mealy_Grappler is the Mealy State Machine used to enable grappler operation when extender is fully extended
7  ENTITY Mealy_Grappler IS PORT (
8           CLK               : in  std_logic := '0';    --clock signal
9           RESET_n           : in  std_logic := '0';    --reset signal
10          pb                : in  std_logic_vector(0 downto 0);
11          GRAP_ENBL         : in  std_logic := '0';    --grappler enable
12          GRAP_ON           : out std_logic            --grappler on
13          );
14 END ENTITY;
15
16 ARCHITECTURE SM OF Mealy_Grappler IS
17
18 -- list all the STATES
19    TYPE STATES IS (INIT, GRAP_OPEN, GRAP_CLOSED);           -- state machine uses 3 states:INIT, GRAP_OPEN, GRAP_CLOSED
20
21    SIGNAL current_state, next_state       :  STATES;    |   -- current_state, next_state signals are of type STATES
22
23 BEGIN
24
25
26 -- STATE MACHINE: Mealy Type
27
28  -- REGISTER_LOGIC PROCESS:
29 REGISTER_SECTION: PROCESS(CLK, RESET_n) -- creates sequential logic to store the state. The RESET_n is used to asynchronously clear the register
30    BEGIN
31       IF (RESET_n = '0') THEN
32             current_state <= INIT;
33       ELSIF (rising_edge(CLK)) then
34             current_state <= next_state; -- on the rising edge of clock the current state is updated with next state
35       END IF;
36    END PROCESS;
37
38
39 -- TRANSITION LOGIC PROCESS
40  TRANSITION_LOGIC: PROCESS(GRAP_ENBL, pb(0), current_state)    -- logic to determine next state.
41    BEGIN
42      CASE current_state IS
43          --starting state
44          WHEN INIT =>
45            IF ((GRAP_ENBL='1') AND (pb(0) = '1')) THEN        --reaches GRAP_OPEN only when grappler enable is active and push button is pressed
46               next_state <= GRAP_OPEN;
47            ELSE
48               next_state <= INIT;
49            END IF;
```

LogicalStep_Lab4_top

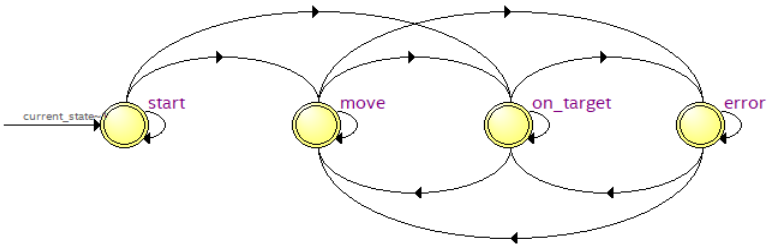
Home | LogicalStep_Lab4_top.vhd | Compx4.vhd | Compx1.vhd | U_D_Bin_Counter8bit.vhd | Bidir_shift_reg.vhd | Mealy_XY.vhd | Moore_Extender.vhd

```
47              ELSE
48                 next_state <= INIT;
49              END IF;
50
51           --state entered when grappler needs to be open
52          WHEN GRAP_OPEN =>
53              IF ((GRAP_ENBL='1') AND (pb(0) = '1')) THEN       --reaches GRAP_CLOSED only when grappler enable is active and push button is pressed
54                 next_state <= GRAP_CLOSED;
55              ELSE
56                 next_state <= GRAP_OPEN;
57              END IF;
58
59           --state entered when grappler needs to be closed
60           WHEN GRAP_CLOSED =>
61              IF ((GRAP_ENBL='1') AND (pb(0) = '1')) THEN       --reaches GRAP_OPEN only when grappler enable is active and push button is pressed
62                 next_state <= GRAP_OPEN;
63              ELSE
64                 next_state <= GRAP_CLOSED;
65              END IF;
66
67            WHEN OTHERS =>                                     --else go back to starting state INIT
68                 next_state <= INIT;
69
70        END CASE;
71    END PROCESS;
72
73
74   -- DECODER SECTION PROCESS (Mealy)
75   Mealy_DECODER: PROCESS(current_state, pb(0))         -- outputs of mealy state machine depend on both current state and circuit inputs
76     BEGIN
77
78     IF (current_state = INIT) THEN
79           GRAP_ON <= '0';                              -- grappler on is set to 0 initially
80
81     ELSIF (current_state = GRAP_OPEN) THEN
82           IF(pb(0) = '0') THEN                         -- grappler opens only after pb(0) is pressed and released
83              GRAP_ON  <= '1';                          -- grappler opens
84           END IF;
85
86      ELSIF (current_state = GRAP_CLOSED) THEN
87          IF(pb(0) = '0') THEN                         --grappler closes only after pb(0) is pressed and released
88            GRAP_ON <= '0';                            --grappler closes
89           END IF;
90
91     END IF;
92    END PROCESS;
93
94   END SM;
95
```

State Diagram for State Machine

There are three state machines used in our design. We used two mealy state machines and one moore state machine.

X/Y Transport State diagram (Mealy Type)

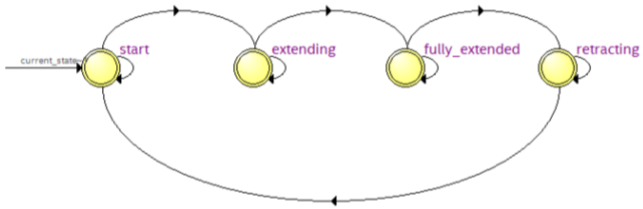


State Table

	Source State	Destination State	Condition
1	error	error	(extender_out)
2	error	move	(!x_eq).(!extender_out) + (x_eq).(!y_eq).(!extender_out)
3	error	on_target	(x_eq).(y_eq).(!extender_out)
4	move	error	(extender_out).(!x_eq) + (extender_out).(x_eq).(!y_eq)
5	move	move	(!x_eq).(!extender_out) + (x_eq).(!y_eq).(!extender_out)
6	move	on_target	(x_eq).(y_eq)
7	on_target	error	(extender_out).(pb[2]).(!x_eq) + (extender_out).(pb[2]).(x_eq).(!y_eq)
8	on_target	move	(!x_eq).(!extender_out).(pb[2]) + (x_eq).(!y_eq).(!extender_out).(pb[2]) + (x_eq).(y_eq).(pb[2])
9	on_target	on_target	(!pb[2])
10	start	move	(!x_eq).(pb[2]) + (x_eq).(!y_eq).(pb[2])
11	start	start	(!pb[2])
12	start	on_target	(x_eq).(y_eq).(pb[2])

Transitions | Encoding

Extender State Diagram (Moore type):

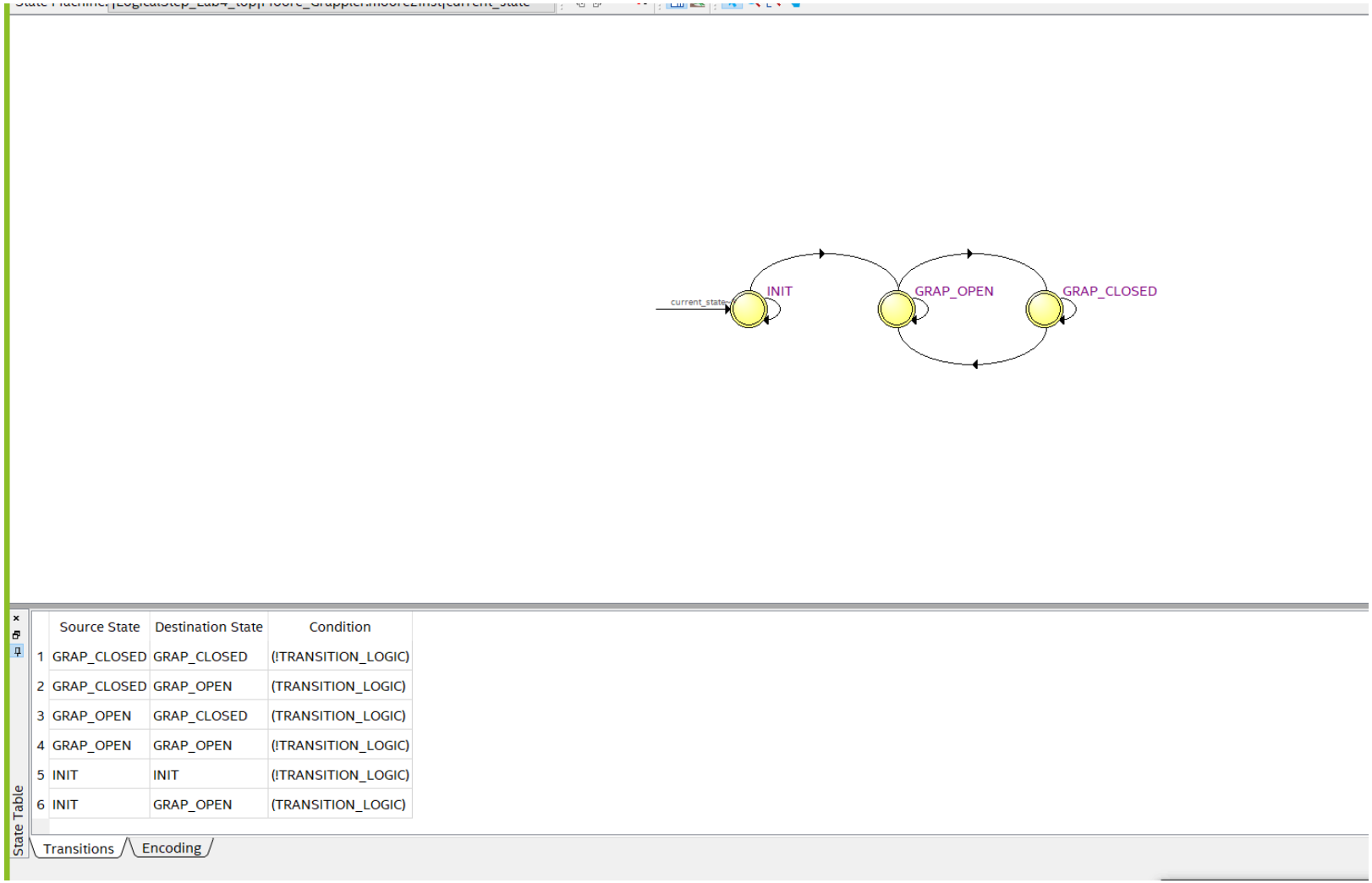


State Table

	Source State	Destination State	Condition
1	extending	extending	(!ExtenderPosition[0]) + (ExtenderPosition[0]).(!ExtenderPosition[1]) + (ExtenderPosition[0]).(ExtenderPosition[1]).(!ExtenderPosition[2]) + (ExtenderPosition[0]).(ExtenderPosition[1]).(ExtenderPosition[2]).(!ExtenderPosition[3]) + (Extender
2	extending	fully_extended	(EXT_ENBL).(ExtenderPosition[0]).(ExtenderPosition[1]).(ExtenderPosition[2]).(ExtenderPosition[3])
3	fully_extended	fully_extended	(!pb[1]) + (pb[1]).(!EXT_ENBL)
4	fully_extended	retracting	(EXT_ENBL).(pb[1])
5	retracting	start	(!ExtenderPosition[0]).(!ExtenderPosition[1]).(!ExtenderPosition[2]).(!ExtenderPosition[3]).(EXT_ENBL)
6	retracting	retracting	(!ExtenderPosition[0]).(!ExtenderPosition[1]).(!ExtenderPosition[2]).(!ExtenderPosition[3]).(!EXT_ENBL) + (!ExtenderPosition[0]).(!ExtenderPosition[1]).(!ExtenderPosition[2]).(ExtenderPosition[3]) + (!ExtenderPosition[0]).(!ExtenderPosition[1
7	start	extending	(EXT_ENBL).(pb[1])
8	start	start	(!EXT_ENBL) + (EXT_ENBL).(!pb[1])

Transitions / Encoding /

Grappler State Diagram (Mealy type):

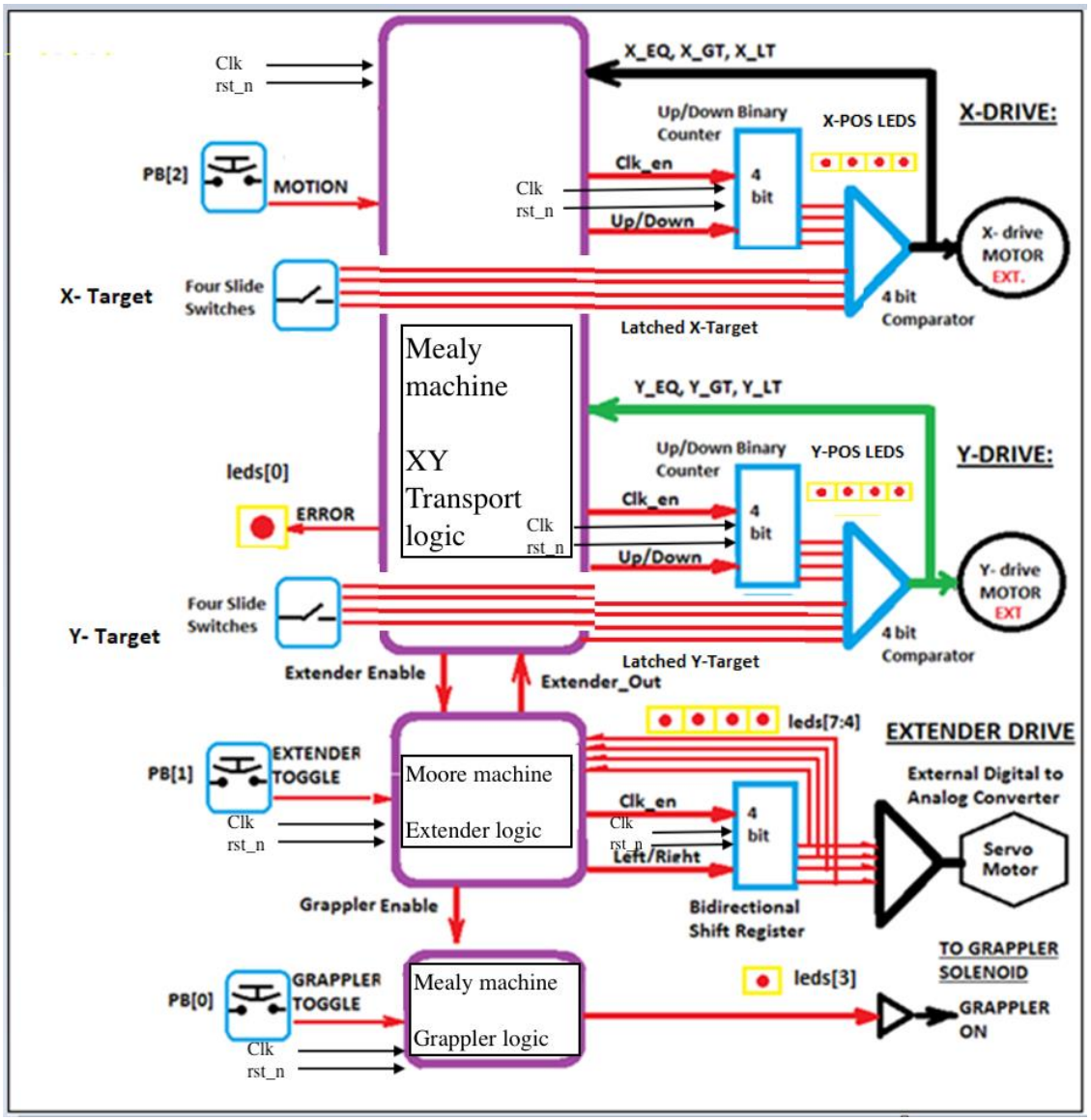


State Table

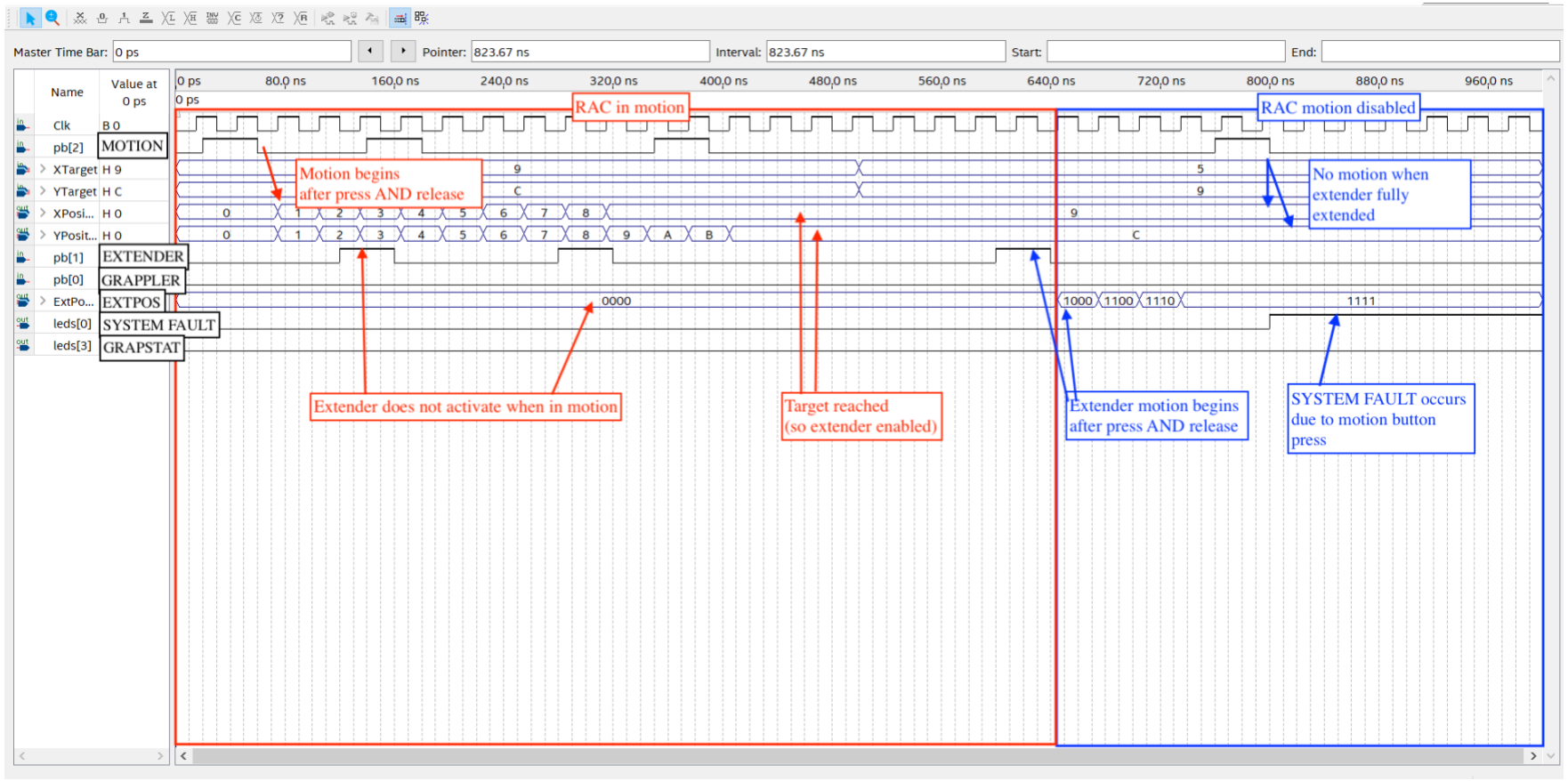
	Source State	Destination State	Condition
1	GRAP_CLOSED	GRAP_CLOSED	(!TRANSITION_LOGIC)
2	GRAP_CLOSED	GRAP_OPEN	(TRANSITION_LOGIC)
3	GRAP_OPEN	GRAP_CLOSED	(TRANSITION_LOGIC)
4	GRAP_OPEN	GRAP_OPEN	(!TRANSITION_LOGIC)
5	INIT	INIT	(!TRANSITION_LOGIC)
6	INIT	GRAP_OPEN	(TRANSITION_LOGIC)

Transitions | Encoding

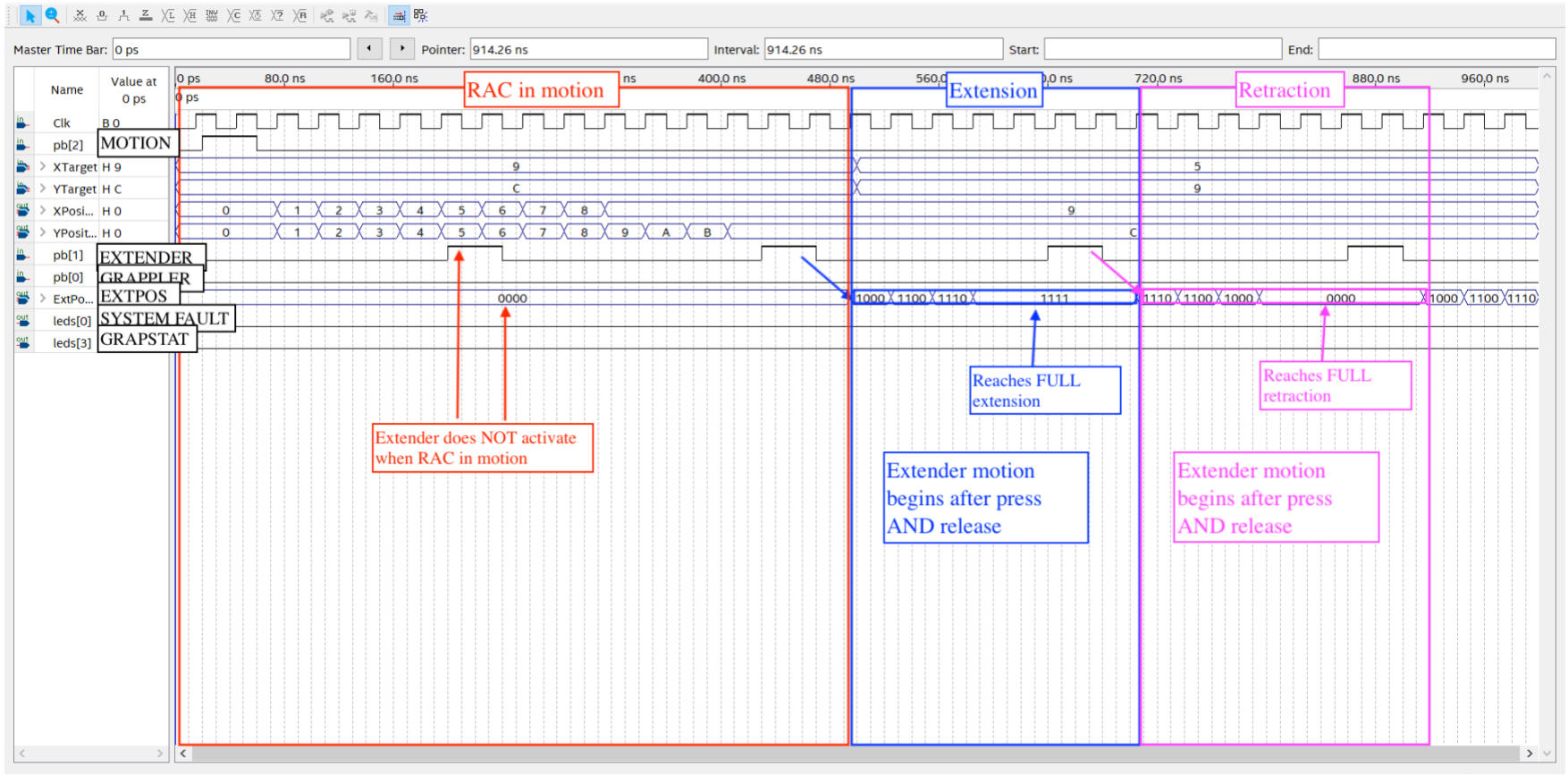
RAC BLOCK DIAGRAM



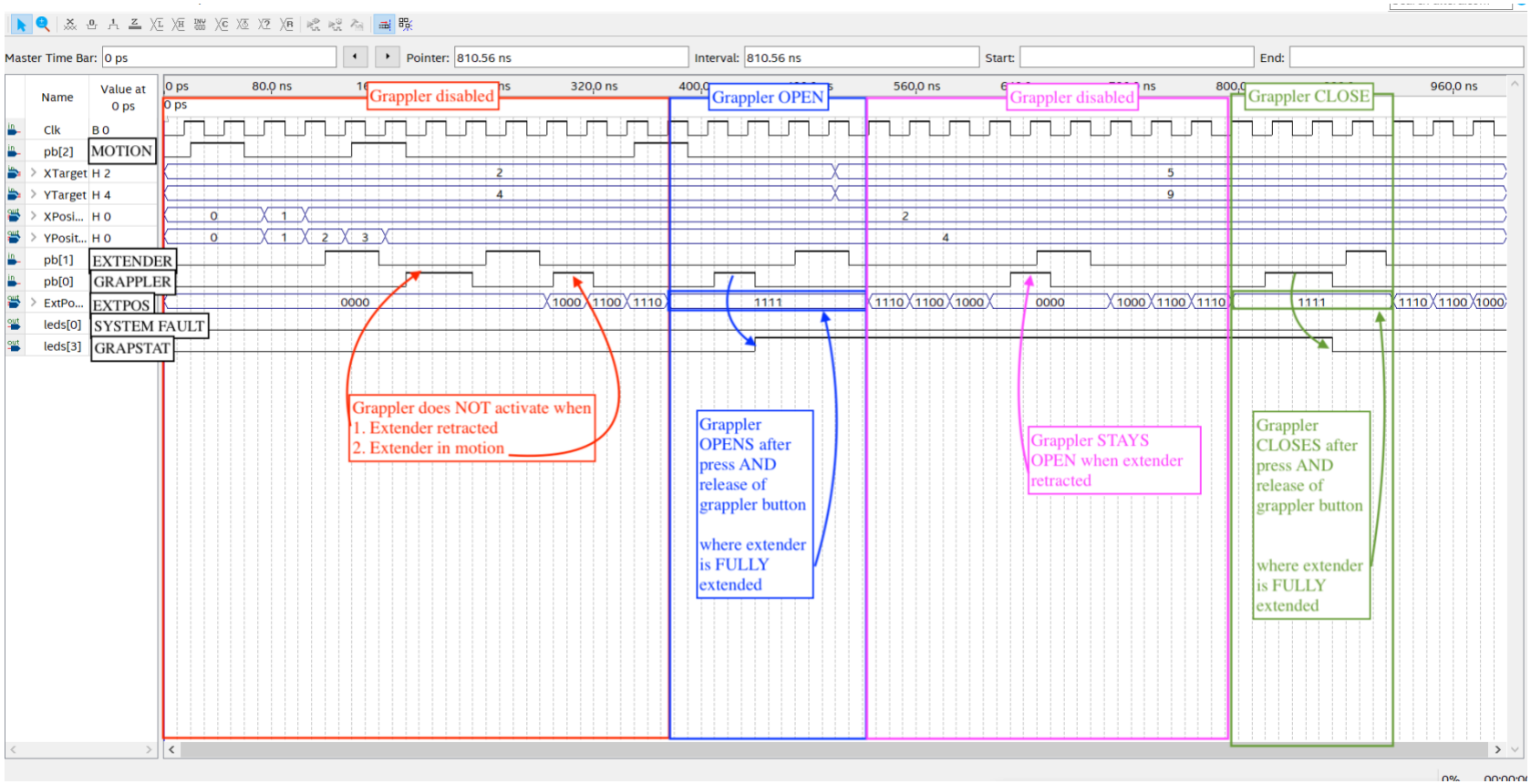
X/Y TRANSPORT OPERATIONS



EXTENDER OPERATIONS



GRAPPLER OPERATIONS



ALL REGISTERS ARE RESET WHEN rst_n is Active