ECE-124 Lab-4 Submission Form – Winter 2018

GROUP NUMBER: 7		Lab4 Demo	Lab4 Report	
SESSION NUMBER: 202		Out of 10	Out of 10	
Partner A: Yifan Yu	yf3yu			
Partner B: Daguang Ruan	d2ruan			

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**	

LAB4 GRP7 SESS202 REPORT

VHDL SOURCE CODE 1: TOP LEVEL STRUCTURAL

```
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

COMPONENT SevenSegment is port (

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

COMPONENT segment7_mux is
   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 MealyStatemachine IS
Port(

 clk_input, resetButton : IN std_logic;

 X_EQ, X_GT, X_LT : IN std_logic; --comparing DESIRED TO ACTUAL
 Y_EQ, Y_GT, Y_LT : IN std_logic; --comparing DESIRED TO ACTUAL
 extenderOut : IN std_logic;

 X_MOTION, Y_MOTION : IN std_logic;

 X_Clk_en, X_UPorDOWN: OUT std_logic;
 Y_Clk_en, Y_UPorDOWN: OUT std_logic;

 ExtenderEnable: OUT std_logic;

 isError : OUT std_logic


 );
END COMPONENT;

COMPONENT Bin_Counter4bit is port
(
               Main_clk                      : in std_logic;
               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;

COMPONENT FourBitComparator is port (

        bitA0, bitA1, bitA2, bitA3, bitB0, bitB1, bitB2, bitB3 : in std_logic;

        AGTB    : out std_logic;
        AEQB    : out std_logic;
        ALTB    : out std_logic

        );
end COMPONENT;

COMPONENT Grappler IS Port (
 clk_input, rst_n, controlButton, enable                                        : IN
std_logic;
 grappleControl
               : OUT std_logic
 );
end COMPONENT;
```

```
COMPONENT 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 COMPONENT;

COMPONENT Extender IS Port
(
 clk_input, rst_n, controlButton, enable                                          : IN
std_logic;
 currentShiftValue
              : IN std_logic_vector(3 downto 0);
 bitShifting, extenderOut, bitShiftDirection, grapplerEnable              : OUT std_logic
 );
END COMPONENT;


COMPONENT mux is port (

   hex_in1, hex_in2    : in std_logic_vector(6 downto 0);
       mux_select :in std_logic;
       hex_out :       out std_logic_vector(6 downto 0)


);
end COMPONENT;

COMPONENT muxSingle is port (

   hex_in1, hex_in2    : in std_logic;
       mux_select :in std_logic;
       hex_out :       out std_logic


);
end COMPONENT;

COMPONENT FlashCounter is port
       (
              Main_clk                     : in std_logic;
              rst_n                        : in std_logic := '0';
              clk_en               : in std_logic := '0';
              up1_down0            : in std_logic := '0';
              counter_bits   : out std_logic
       );
       end COMPONENT;
```

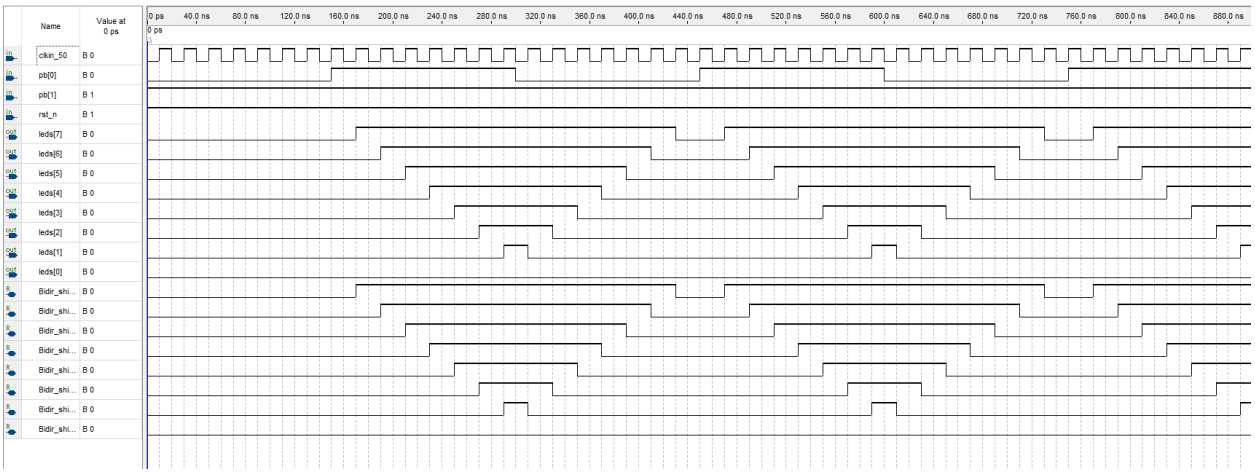
```
------------------------------------------------------------------------------------------------
---
       CONSTANT       SIM                                                     :  boolean := FALSE;
       -- set to TRUE for simulation runs otherwise keep at 0.
   CONSTANT CLK_DIV_SIZE                              :      INTEGER := 26;    -- 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

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

       SIGNAL XTARGET7seg, YTARGET7seg : std_logic_vector(6 downto 0);
       SIGNAL XCURRENT, YCURRENT : std_logic_vector(3 downto 0);
       SIGNAL XCURRENT7seg, YCURRENT7seg : std_logic_vector(6 downto 0);

       SIGNAL XMUX7seg, YMUX7seg: std_logic_vector(6 downto 0);

       SIGNAL XTargLTCurr, XTargEQCurr, XTargGTCurr : std_logic;
       SIGNAL YTargLTCurr, YTargEQCurr, YTargGTCurr : std_logic;

       SIGNAL bitShiftDirControl, bitShiftEnable : std_logic;
       SIGNAL currentShiftValue : std_logic_vector(3 downto 0);

       SIGNAL extenderOutSignal : std_logic;

       SIGNAL X_ClockEnable, Y_ClockEnable : std_logic;
       SIGNAL X_Direction, Y_Direction : std_logic;

       SIGNAL GrappleEnableSignal : std_logic;

       SIGNAL extenderEnableSignal : std_logic;
       SIGNAL ERROR7seg, NOTHING7seg, ERROR7segOutput : std_logic_vector(6 downto 0);
       SIGNAL X_ERROR_AND_VALUE7seg, Y_ERROR_AND_VALUE7seg:std_logic_vector(6 downto 0);

       SIGNAL ERROR : std_logic;

       SIGNAL MUX_CLOCK : std_logic;

       SIGNAL FLASH: std_logic;



------------------------------------------------------------------------------------------------
---
```

```
BEGIN

-- 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

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

leds (7 downto 4) <= currentShiftValue (3 downto 0);
leds (0) <= ERROR;
ERROR7seg <= "1111001";
NOTHING7seg <= "0000000";


-------------------------------------------------------------------------------------------------
--
INST1XTARGET: SevenSegment PORT MAP (sw(7 downto 4),XTARGET7Seg);
INST2YTARGET: SevenSegment PORT MAP (sw(3 downto 0),YTARGET7Seg);
INST11XCURRENT: SevenSegment PORT MAP (XCURRENT,XCURRENT7seg);
INST12YCURRENT: SevenSegment PORT MAP (YCURRENT,YCURRENT7seg);

INST13MUX_X:mux PORT MAP (

   XCURRENT7seg, XTARGET7Seg,
        pb(3),
        XMUX7seg);

INST14MUX_Y:mux PORT MAP (

   YCURRENT7seg, YTARGET7Seg,
        pb(2),
        YMUX7seg);

INST15X_MUX_ERROR :mux PORT MAP (

   XMUX7seg, ERROR7segOutput,
        ERROR,
        X_ERROR_AND_VALUE7seg);

INST16Y_MUX_ERROR :mux PORT MAP (

   YMUX7seg,ERROR7segOutput,
        ERROR,
        Y_ERROR_AND_VALUE7seg);


INST18ERROR_AND_NOTHING: mux PORT MAP(

        ERROR7seg, NOTHING7seg,
        FLASH,

        ERROR7segOutput);
```

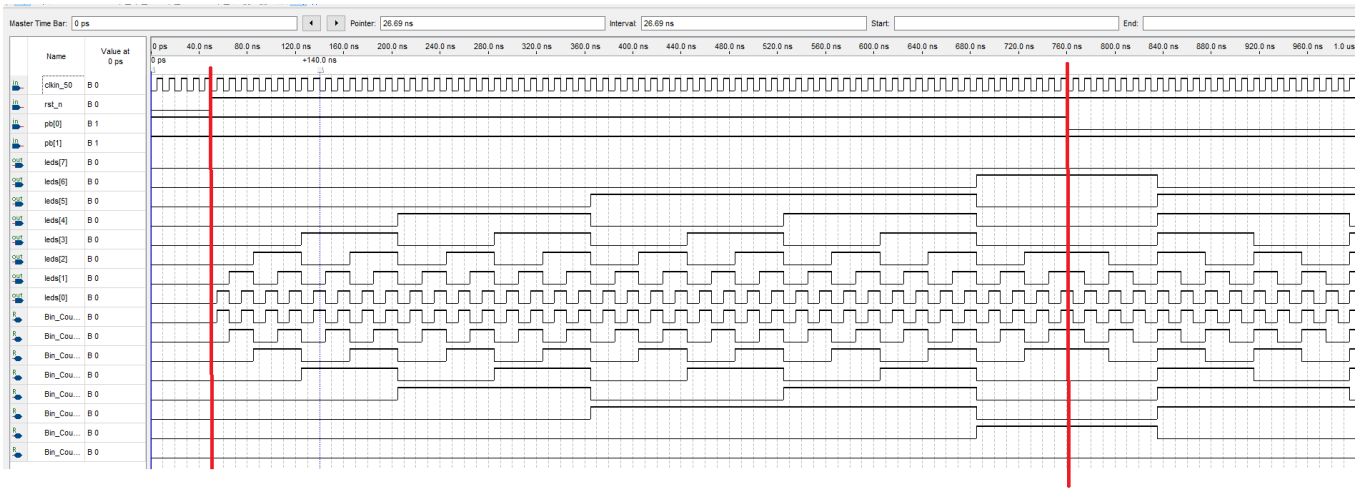
```
INST19FLASH: FlashCounter PORT MAP
	(
		std_logic(bin_counter(23)),
		rst_n,
		'1',
		'1',
		FLASH
	);



INST3: segment7_mux PORT MAP (clkin_50, X_ERROR_AND_VALUE7seg(6 downto 0),
Y_ERROR_AND_VALUE7seg(6 downto 0), seg7_data(6 downto 0), seg7_char1, seg7_char2);

INST4X: Bin_Counter4bit PORT MAP (Main_Clk,rst_n,X_ClockEnable,X_Direction,XCURRENT(3 downto 0)
);

INST5Y: Bin_Counter4bit PORT MAP (Main_Clk,rst_n, Y_ClockEnable,Y_Direction,YCURRENT(3 downto 0)
);


INST6X: FourBitComparator PORT MAP (

	sw(4),sw(5), sw(6), sw(7),
	XCURRENT(0), XCURRENT(1), XCURRENT(2), XCURRENT(3),
	XTargGTCurr, XTargEQCurr, XTargLTCurr);

INST7Y: FourBitComparator PORT MAP (

	sw(0),sw(1), sw(2), sw(3),
	YCURRENT(0), YCURRENT(1), YCURRENT(2), YCURRENT(3),
	YTargGTCurr, YTargEQCurr, YTargLTCurr);


INST8GrapplerSM: Grappler PORT MAP (Main_Clk, rst_n, pb(0) , GrappleEnableSignal, leds(3));


INST9: Bidir_shift_reg PORT MAP(
	Main_Clk, rst_n,
	bitShiftEnable, bitShiftDirControl,
	currentShiftValue (3 downto 0) );
--
INST10: Extender PORT MAP (

	Main_Clk, rst_n, pb(1), extenderEnableSignal,
	currentShiftValue(3 downto 0),

	bitShiftEnable, extenderOutSignal, bitShiftDirControl,GrappleEnableSignal
);


INSTMEALY: MealyStatemachine PORT MAP (

 Main_Clk, rst_n,

 XTargEQCurr, XTargGTCurr, XTargLTCurr,
 YTargEQCurr, YTargGTCurr, YTargLTCurr,
 extenderOutSignal,
 pb(3), pb(2),

 X_ClockEnable, X_Direction,
 Y_ClockEnable, Y_Direction,

 extenderEnableSignal,
 ERROR

 );


 --INST13ERROR_MUX: mux PORT MAP (ERROR7seg,
```

```
END SimpleCircuit;
```

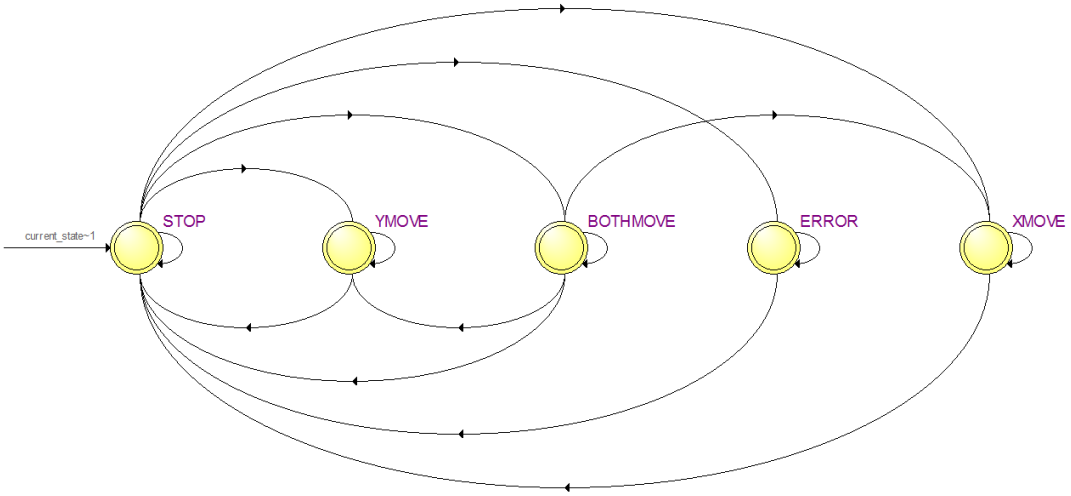
Simulation 1: 8-Bits Shift Register



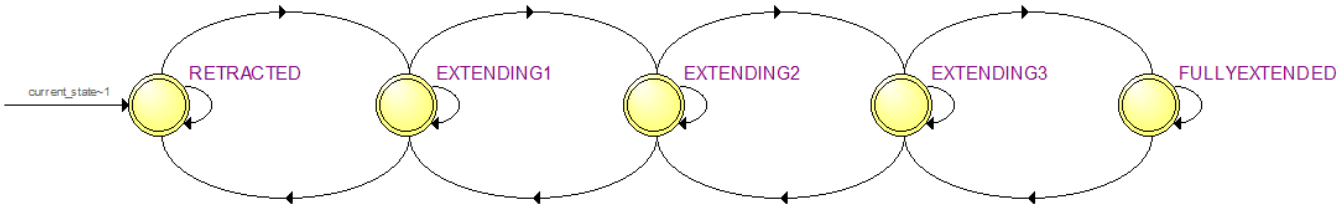
Simulation 2: 8-Bits Counter



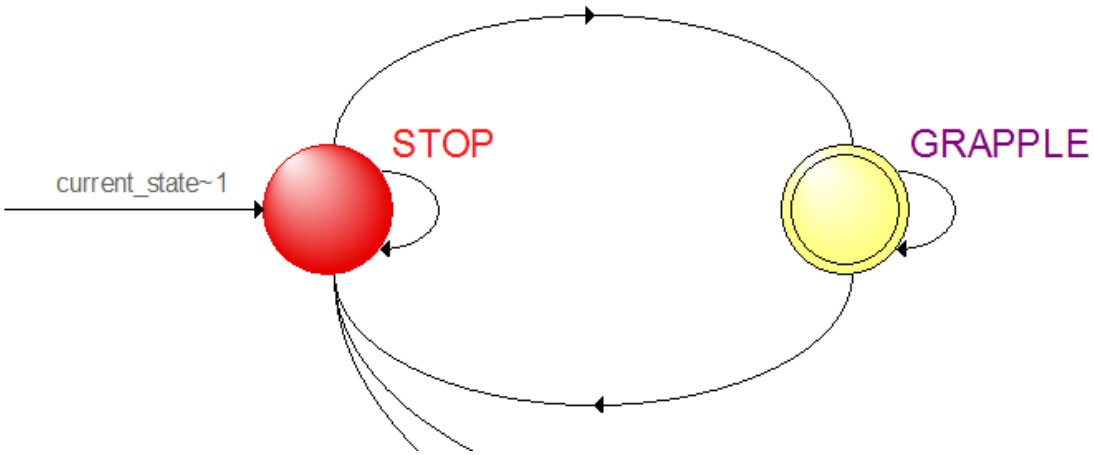
State Diagram 1 Mealy State Machine



State Diagram 2: Extender Moore State Machine



State Diagram 3: Grappler Moore State Machine



Form of the SM

STATE MACHINE 1: MEALY

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

Entity MealyStatemachine IS Port
(
 clk_input, resetButton : IN std_logic;

 X_EQ, X_GT, X_LT : IN std_logic; --comparing DESIRED TO ACTUAL
 Y_EQ, Y_GT, Y_LT : IN std_logic; --comparing DESIRED TO ACTUAL
 extenderOut : IN std_logic;



 X_MOTION, Y_MOTION : IN std_logic;

 X_Clk_en, X_UPorDOWN: OUT std_logic;
 Y_Clk_en, Y_UPorDOWN: OUT std_logic;

 ExtenderEnable: OUT std_logic;

 isError : OUT std_logic := '0'-- need to put that somewhere







 );
END ENTITY;


 Architecture behaviour of MealyStatemachine is


 -- state for the X control and Y control

 TYPE STATE IS (XMOVE, STOP, YMOVE, BOTHMOVE, ERROR);



 SIGNAL current_state, next_state     : STATE; -- our signal for currentState and nextSate


 BEGIN

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




 ------------------------------CONTROL FOR X-------------------------------------
 -- REGISTER_LOGIC PROCESS:

Register_Section: PROCESS (clk_input, resetButton, next_state)  -- this process synchronizes the
activity to a clock
BEGIN
        IF (resetButton = '0') THEN
                current_state <= STOP;


        ELSIF(rising_edge(clk_input)) THEN
                current_state <= next_State;
        END IF;
END PROCESS;
```

```
-- TRANSITION LOGIC PROCESS

Transition_Section: PROCESS (current_state, X_EQ, X_GT, X_LT,Y_EQ, Y_GT,
Y_LT,extenderOut,X_MOTION, Y_MOTION)


BEGIN
     CASE current_state IS

          --switching states depending on the value of the comparaison
         WHEN STOP =>

                                IF( X_EQ = '0' AND Y_EQ = '0' AND (X_MOTION='0') AND (Y_MOTION='0')
and extenderout = '0') THEN
                                       -- both are bigger or smaller AND we ve got both buttons at
active low
                                       next_state <= BOTHMOVE;

                                ELSIF((X_GT='1' or X_LT='1') AND (X_MOTION='0') and extenderout =
'0' ) THEN

                                       next_state <= XMOVE;

                                ELSIF((Y_GT='1' or Y_LT='1') AND (Y_MOTION='0')  and extenderout =
'0' ) THEN

                                       next_state <= YMOVE;

                                ELSIF( (X_EQ = '0' or Y_EQ = '0')  and ((X_MOTION='0') or
(Y_MOTION='0')) and extenderOut = '1') THEN

                                       next_state <= ERROR;


                                ELSE
                                       next_state <= STOP;

                                END IF;



         WHEN XMOVE =>

                                IF( X_EQ = '1') THEN

                                       next_state <= STOP;


                                ELSE
                                       next_state <= XMOVE;

                                END IF;



         WHEN YMOVE =>

                                IF( Y_EQ = '1') THEN

                                       next_state <= STOP;


                                ELSE
                                       next_state <= YMOVE;

                                END IF;
```

```
          WHEN BOTHMOVE =>

                                IF(Y_EQ = '1' and  X_EQ = '1') THEN

                                        next_state <= STOP;

                                ELSIF( Y_EQ = '1' and  X_EQ = '0') THEN

                                        next_state <= XMOVE;

                                ELSIF( Y_EQ = '0' and  X_EQ = '1')THEN

                                        next_state <= YMOVE;

                                ELSE

                                        next_state <= BOTHMOVE;

                                END IF;



                        WHEN ERROR =>

                                IF( extenderOut = '0') THEN
                                        next_state <= STOP;
                                ELSE
                                        next_state <= ERROR;
                                END IF;

                        WHEN OTHERS =>
                                next_state <= STOP;
                        END CASE;

 END PROCESS;

-- DECODER SECTION PROCESS

Decoder_Section: PROCESS (current_state, X_EQ, X_GT, X_LT,Y_EQ, Y_GT, Y_LT,extenderOut,X_MOTION,
Y_MOTION)

BEGIN

        CASE current_state IS

                WHEN STOP =>
                                X_CLK_en <= '0';
                                Y_CLK_en <= '0';
                                isError <= '0';

                        IF (X_EQ='1'AND Y_EQ='1') THEN

                                ExtenderEnable <= '1';
                        ELSE

                                ExtenderEnable <= '0';

                        END IF;




                WHEN XMOVE =>


                        IF (X_GT='1') THEN
```

```
                X_CLK_en <= '1';
                X_UPorDOWN <= '1';



        ELSIF (X_LT = '1') THEN
         -- count forward

                X_CLK_en <= '1';
                X_UPorDOWN <= '0';

        ELSE
                X_CLK_en <= '0';

        END IF;



WHEN YMOVE =>
        Y_CLK_en <= '0';

        IF (Y_GT='1') THEN
         -- count backward
                Y_CLK_en <= '1';
                Y_UPorDOWN <= '1';

        ELSIF (Y_LT = '1') THEN
         -- count forward
                Y_CLK_en <= '1';
                Y_UPorDOWN <= '0';


        ELSE
                Y_CLK_en <= '0';

        END IF;




WHEN BOTHMOVE =>
        IF (X_GT='1') THEN

                X_CLK_en <= '1';
                X_UPorDOWN <= '1';



        ELSIF (X_LT = '1') THEN
         -- count forward

                X_CLK_en <= '1';
                X_UPorDOWN <= '0';

        ELSE
                X_CLK_en <= '0';

        END IF;


        IF (Y_GT='1') THEN
         -- count backward
                Y_CLK_en <= '1';
                Y_UPorDOWN <= '1';

        ELSIF (Y_LT = '1') THEN
         -- count forward
                Y_CLK_en <= '1';
                Y_UPorDOWN <= '0';
```

```
                        ELSE
                                Y_CLK_en <= '0';

                        END IF;

                WHEN ERROR =>

                        --IF ( (Y_EQ = '0' OR X_EQ = '0') AND (X_MOTION = '0' OR Y_MOTION = '0')
)THEN

                                isError <= '1';
                                X_CLK_en <= '0';
                                Y_CLK_en <= '0';
                                ExtenderEnable <= '1';
                        --ELSE
                                --isError <= '0';

                        --END IF;

                END CASE;


 END PROCESS;

 END ARCHITECTURE behaviour;
```

STATE MACHINE 2: EXTENDER MOORE

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

Entity Extender IS Port
(
 clk_input, rst_n, controlButton, enable                                          : IN
std_logic;
 currentShiftValue
              : IN std_logic_vector(3 downto 0);
 bitShifting, extenderOut, bitShiftDirection, grapplerEnable              : OUT std_logic
 );
END ENTITY;


 Architecture SM of Extender is



 TYPE STATE_NAMES IS (RETRACTED,EXTENDING1, EXTENDING2, EXTENDING3, FULLYEXTENDED);   -- list all
the STATE_NAMES values


 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 <= RETRACTED;
       ELSIF(rising_edge(clk_input)) THEN
              current_state <= next_State;
       END IF;
END PROCESS;



-- TRANSITION LOGIC PROCESS

Transition_Section: PROCESS ( current_state,controlButton, enable, currentShiftValue)

BEGIN

 CASE current_state IS
        WHEN RETRACTED =>
                              IF(controlButton = '0' AND enable = '1') THEN
                                     next_state <= EXTENDING1;
                              ELSE
                                     next_state <= RETRACTED;
                              END IF;

        WHEN EXTENDING1 =>
                              IF(currentShiftValue = "0000") THEN
                                     next_state <= EXTENDING2;
                              ELSIF(currentShiftValue = "1000") THEN
```

```
                                next_state <= RETRACTED;
                            ELSE
                                next_state <= EXTENDING1;
                            END IF;

        WHEN EXTENDING2 => --forward stays here twice
                            IF(currentShiftValue = "1000") THEN
                                next_state <= EXTENDING3;
                            ELSIF(currentShiftValue = "1110") THEN
                                next_state <= EXTENDING1;
                            ELSE
                                next_state <= EXTENDING2;
                            END IF;

        WHEN EXTENDING3 =>
                            IF(currentShiftValue = "1110") THEN
                                next_state <= FULLYEXTENDED;
                            ELSIF(currentShiftValue = "1111") THEN
                                next_state <= EXTENDING2;
                            ELSE
                                next_state <= EXTENDING3;
                            END IF;

        WHEN FULLYEXTENDED =>
                            IF(controlButton = '0' AND enable = '1' ) THEN
                                next_state <= EXTENDING3;
                            ELSE
                                next_state <= FULLYEXTENDED;
                            END IF;


                    WHEN OTHERS =>
           next_state <= RETRACTED;
              END CASE;
 END PROCESS;

-- DECODER SECTION PROCESS

Decoder_Section: PROCESS ( current_state,controlButton, enable, currentShiftValue)

BEGIN
     CASE current_state IS
         WHEN RETRACTED =>
                            bitShifting <= '0';
                            extenderOut<= '0';
                            grapplerEnable <= '0';
                            bitShiftDirection <= '1';

         WHEN EXTENDING1 =>

                            bitShifting <= '1';
                            extenderOut<= '1';
                            grapplerEnable <= '0';

         WHEN EXTENDING2 =>

                            bitShifting <='1';
                            extenderOut<= '1';
                            grapplerEnable <= '0';
         WHEN EXTENDING3 =>

                            bitShifting <= '1';
                            extenderOut<= '1';
                            grapplerEnable <= '0';

         WHEN FULLYEXTENDED =>
                            bitShifting <= '0';
                            extenderOut<= '1';
                            grapplerEnable <= '1';
                            bitShiftDirection <= '0';
```

```
        WHEN others =>
                            bitShifting <= '0';
                            extenderOut<= '0';
                            grapplerEnable <= '0';
                            bitShiftDirection <= '1';

         END CASE;
END PROCESS;

END ARCHITECTURE SM;
```

STATE MACHINE 3: GRAPPLER MOORE

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

Entity Grappler IS Port
(
 clk_input, rst_n, controlButton, enable                              : IN std_logic;
 grappleControl
               : OUT std_logic
 );
END ENTITY;


 Architecture STATEMACHINE of Grappler is



 TYPE STATE_NAMES IS (STOP, GRAPPLE);   -- list all the STATE_NAMES values


 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 <= STOP;
        ELSIF(rising_edge(clk_input)) THEN
                current_state <= next_State;
        END IF;
END PROCESS;



-- TRANSITION LOGIC PROCESS

Transition_Section: PROCESS (current_state,controlButton, enable)

BEGIN
     CASE current_state IS
         WHEN STOP =>
                                IF( controlButton = '0' AND enable= '1' ) THEN --pressed
                                        next_state <= GRAPPLE;

                                ELSE
                                        next_state <= STOP;
                                END IF;

                        WHEN GRAPPLE =>
                                IF( controlButton = '0' OR enable= '0') THEN --pressed
                                        next_state <= STOP;
                                ELSE
                                        next_state <= GRAPPLE;
                                END IF;

WHEN OTHERS =>
                next_state <= STOP;
        END CASE;
```

```
 END PROCESS;

-- DECODER SECTION PROCESS

Decoder_Section: PROCESS (current_state,controlButton, enable)

BEGIN
     CASE current_state IS
         WHEN STOP =>

                               grappleControl <= '0';


         WHEN GRAPPLE =>
                               grappleControl <= '1';

         WHEN others =>
                               grappleControl <= '0';

          END CASE;
 END PROCESS;

 END ARCHITECTURE STATEMACHINE;
```

Difference between a Mealy and a Moore

The output of a Mealy machine is dependent on both the input and the current state while the output of a Moore machine is only dependent on the current state.

Fitter Report on Resources Utilization by Entity

Fitter Resource Utilization by Entity

	Compilation Hierarchy Node	Logic Cells	Dedicated Logic Registers	I/O Registers	Memory Bits
1	⌄ |LogicalStep_Lab4_top	153 (25)	48 (24)	0 (0)	0
1	|Bidir_shift_reg:INST9|	4 (4)	4 (4)	0 (0)	0
2	|Bin_Counter4bit:INST4X|	7 (7)	4 (4)	0 (0)	0
3	|Bin_Counter4bit:INST5Y|	6 (6)	4 (4)	0 (0)	0
4	|Extender:INST10|	16 (16)	5 (5)	0 (0)	0
5	|FlashCounter:INST19FLASH|	1 (1)	1 (1)	0 (0)	0
6	|FourBitComparator:INST6X|	6 (6)	0 (0)	0 (0)	0
7	|FourBitComparator:INST7Y|	4 (4)	0 (0)	0 (0)	0
8	|Grappler:INST8GrapplerSM|	2 (2)	1 (1)	0 (0)	0
9	|MealyStatemachine:INSTMEALY|	34 (34)	5 (5)	0 (0)	0
10	|SevenSegment:INST11XCURRENT|	7 (7)	0 (0)	0 (0)	0
11	|SevenSegment:INST12YCURRENT|	7 (7)	0 (0)	0 (0)	0
12	|SevenSegment:INST1XTARGET|	7 (7)	0 (0)	0 (0)	0
13	|SevenSegment:INST2YTARGET|	7 (7)	0 (0)	0 (0)	0
14	|segment7_mux:INST3|	22 (22)	0 (0)	0 (0)	0