LAB 4 GRP 7 SESS 201 REPORT

Structural VHDL for Top Level VHDL (Signals, Components, Instances)

ARCHITECTURE SimpleCircuit OF LogicalStep\_Lab4\_top IS

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

-- SIGNALS

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 EXT\_OUT : std\_logic;

signal EXT\_EN : std\_logic;

signal shift\_enable : std\_logic;

signal shift\_dirout : std\_logic;

signal ext\_light : std\_logic\_vector(3 downto 0);

signal Grapple\_en : std\_logic;

signal seg7\_A : std\_logic\_vector(6 downto 0);

signal seg7\_B : std\_logic\_vector(6 downto 0);

signal Digit1 : std\_logic\_vector(6 downto 0);

signal Digit2 : std\_logic\_vector(6 downto 0);

signal Extend : std\_logic;

signal Grapple : std\_logic;

signal G\_led : std\_logic;

signal button : std\_logic;

signal XTAR : std\_logic\_vector(3 downto 0);

signal Xcurr : std\_logic\_vector(3 downto 0) := "0000";

signal Xdrive : std\_logic;

signal XOUT : std\_logic\_vector(3 downto 0);

signal Xup1\_down0 : std\_logic;

signal Xgrtr : std\_logic;

signal Xeq : std\_logic;

signal Xless : std\_logic;

signal on\_countX : std\_logic;

signal errorLEDx : std\_logic;

signal YTAR : std\_logic\_vector(3 downto 0);

signal Ycurr : std\_logic\_vector(3 downto 0) := "0000";

signal YOUT : std\_logic\_vector(3 downto 0);

signal Ydrive : std\_logic;

signal Yup1\_down0 : std\_logic;

signal Ygrtr : std\_logic

signal Yeq : std\_logic;

signal Yless : std\_logic;

signal on\_countY : std\_logic;

signal errorLEDy : std\_logic;

signal ERR\_SIG : std\_logic;

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

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

-- COMPONENTS USED

--

-- COMPONENT 1 (SevenSegment)

component SevenSegment 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 2 (7-Bit Digit Mux)

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 segment7\_mux;

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

-- COMPONENT 3 (Bidirectional Up Down Shift Counter)

component Bin\_Counter4bit port (

Main\_CLK : in std\_logic := '0';

rst\_n : in std\_logic := '0';

enable : in std\_logic := '0';

up1\_down0 : in std\_logic := '0';

counter\_bits : out std\_logic\_vector(3 downto 0)

);

End component Bin\_Counter4bit;

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

-- COMPONENT 4 (Extender Moore State Machine)

component Moore\_SM port (

clk\_input, rst\_n, exten, button : IN std\_logic;

leds : IN std\_logic\_vector(3 downto 0);

shift\_en, shift\_dir, exten\_out, grap\_en : OUT std\_logic

);

End component Moore\_SM;

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

-- COMPONENT 5 (Grappler Moore State Machine)

component Moore\_SM2 Port (

clk\_input, rst\_n, button, enable : IN std\_logic;

led : OUT std\_logic

);

END component Moore\_SM2;

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

-- COMPONENT 7 (Mealy State Machine)

component Mealy\_SM port (

clk\_input,XMOTION, YMOTION, EX\_OUT,XEQ,XGT,XLT,YEQ,YGT,YLT : IN std\_logic;

rst\_n : IN std\_logic;

clk\_enX,clk\_enY, Xcount,Ycount,ERRORled,EXTD\_EN : OUT std\_logic

);

End component Mealy\_SM;

-------------------------------------------------------------------------------------------------------------------------------------------- COMPONENT 8 (Bidirectional Left Right Shift Register)

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 Bidir\_shift\_reg;

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

-- COMPONENT 9 (4-Bit Comparator)

component Compx4 port (

a0 : in std\_logic;

a1 : in std\_logic;

a2 : in std\_logic;

a3 : in std\_logic;

b0 : in std\_logic;

b1 : in std\_logic;

b2 : in std\_logic;

b3 : in std\_logic;

grtr : out std\_logic;

eq : out std\_logic;

less : out std\_logic

);

end component Compx4;

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

-- COMPONENT 10 (System Error Digit Flash)

component ErrorFlash port (

clk\_input : in std\_logic;

rst\_n : in std\_logic;

INPUT1 : in std\_logic\_vector(6 downto 0); --when button 3 pressed

INPUT2 : in std\_logic\_vector(6 downto 0);

Selector : in std\_logic;

OUTPUT : out std\_logic\_vector(6 downto 0)

);

end component ErrorFlash;

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

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

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

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

-- IMPLEMENTATION AND INSTANCES --

XDRIVE <= pb(3);

YDRIVE <= pb(2);

button <= pb(0);

-- Mealy State Machine --

MEALY0: Mealy\_SM port map

(Main\_CLK,XDRIVE,YDRIVE,EXT\_OUT,Xeq,Xgrtr,Xless,Yeq,Ygrtr,Yless,rst\_n,on\_countX,on\_countY,Xup1\_down0,Yup1\_down0,ERR\_SIG,EXT\_EN);

-- UP/DOWN counting and Comparing of X-Coordinate -- For Motion of X

INST1: Bin\_Counter4bit port map (Main\_CLK,rst\_n,on\_countX,Xup1\_down0,Xcurr);

INST2: Compx4 port map (Xcurr(0),Xcurr(1),Xcurr(2),Xcurr(3),sw(4),sw(5),sw(6),sw(7),Xgrtr,Xeq,Xless); --compares target X coordinate with current;

-- UP/DOWN counting and Comparing of Y-Coordinate -- For Motion of Y

INST3: Bin\_Counter4bit port map (Main\_CLK,rst\_n,on\_countY,Yup1\_down0,Ycurr);

INST4: Compx4 port map (Ycurr(0),Ycurr(1),Ycurr(2),Ycurr(3),sw(0),sw(1),sw(2),sw(3),Ygrtr,Yeq,Yless); --compares target X coordinate with current;

-- Extender Moore State Machine & Motion of Extender with Bit Shifter--

INST5: Moore\_SM port map(Main\_CLK,rst\_n,EXT\_EN,pb(1),ext\_light,shift\_enable,shift\_dirout,EXT\_OUT,Grapple\_en);

INST6: Bidir\_shift\_reg port map(Main\_CLK,rst\_n,shift\_enable,shift\_dirout,ext\_light);

-- Grappler Moore State Machine --

INST7: Moore\_SM2 port map (Main\_CLK, rst\_n, button, Grapple\_en, G\_led);

-- Extender Arm Output Lights --

leds(7) <= ext\_light(3);

leds(6) <= ext\_light(2);

leds(5) <= ext\_light(1);

leds(4) <= ext\_light(0);

-- Grappler Output Light --

leds(3) <= G\_led;

-- Switch Inputs for Target --

YTAR <= sw(3 downto 0);

XTAR <= sw(7 downto 4);

with XDRIVE select

XOUT <= Xcurr when '0',

XTAR when '1';

with YDRIVE select

YOUT <= Ycurr when '0',

YTAR when '1';

-- 7 Segment Display Decoder --

INST20: SevenSegment port map(XOUT, seg7\_A);

INST30: SevenSegment port map(YOUT, seg7\_B);

-- SYSTEM ERROR Digit Display Flash Sate Machine --

INST35: ErrorFlash port map (Main\_CLK, rst\_n, "0000000", seg7\_A, ERR\_SIG, Digit1);

INST36: ErrorFlash port map (Main\_CLK, rst\_n, "0000000", seg7\_B, ERR\_SIG, Digit2);

leds(0) <= ERR\_SIG;

-- Display the Current Or Target Digits on the 7-Bit Display --

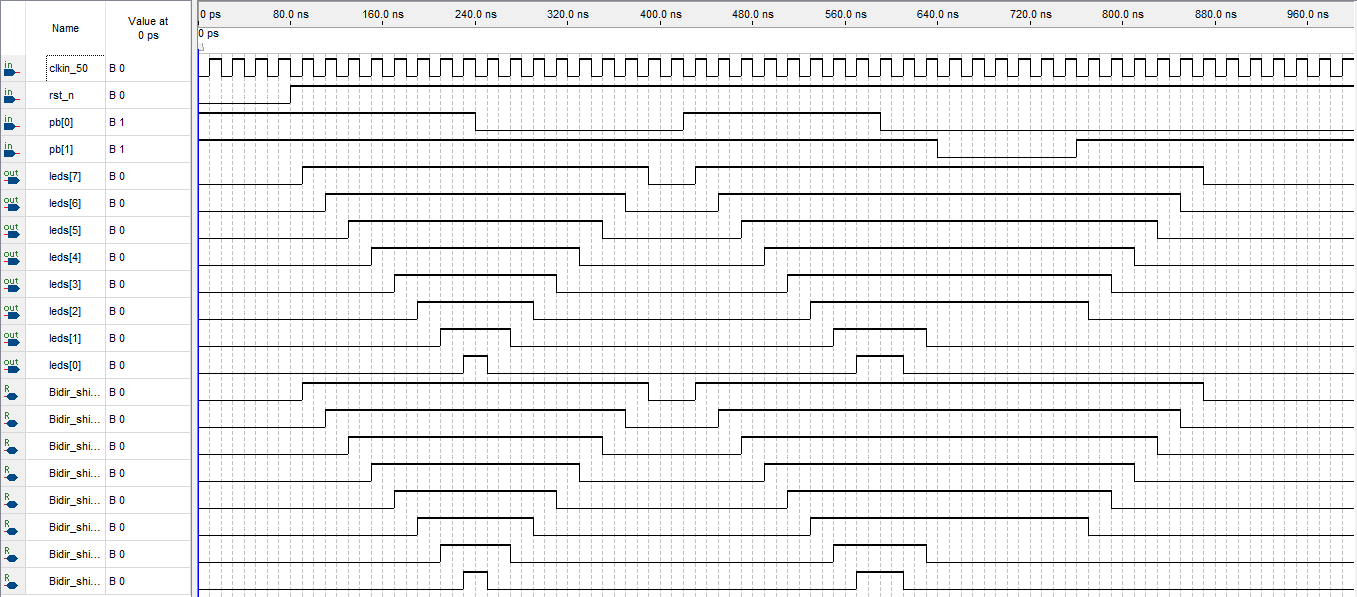
INST40: segment7\_mux port map(clkin\_50, Digit1, Digit2, seg7\_data, seg7\_char1, seg7\_char2);

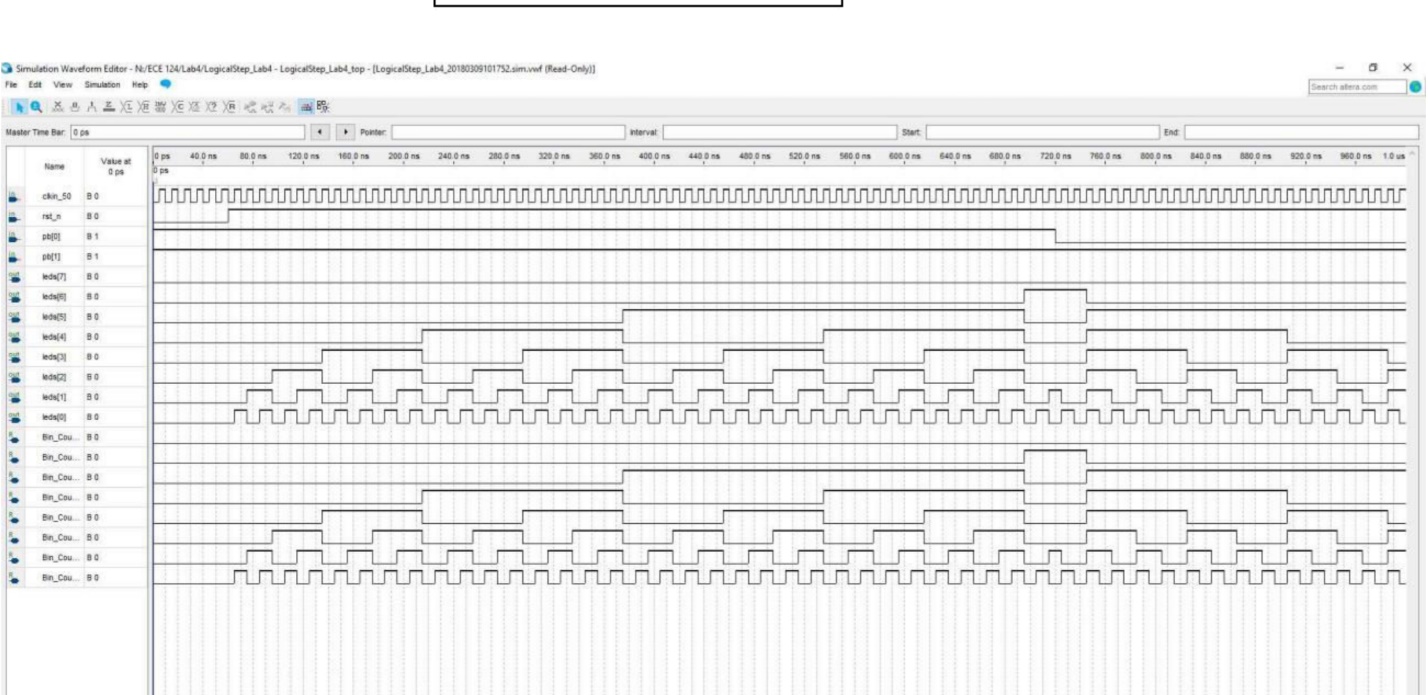
---------------------------------------------------------------------------------------------------

END SimpleCircuit;

SIMULATIONS

Simulation of 8bit Shift Register -

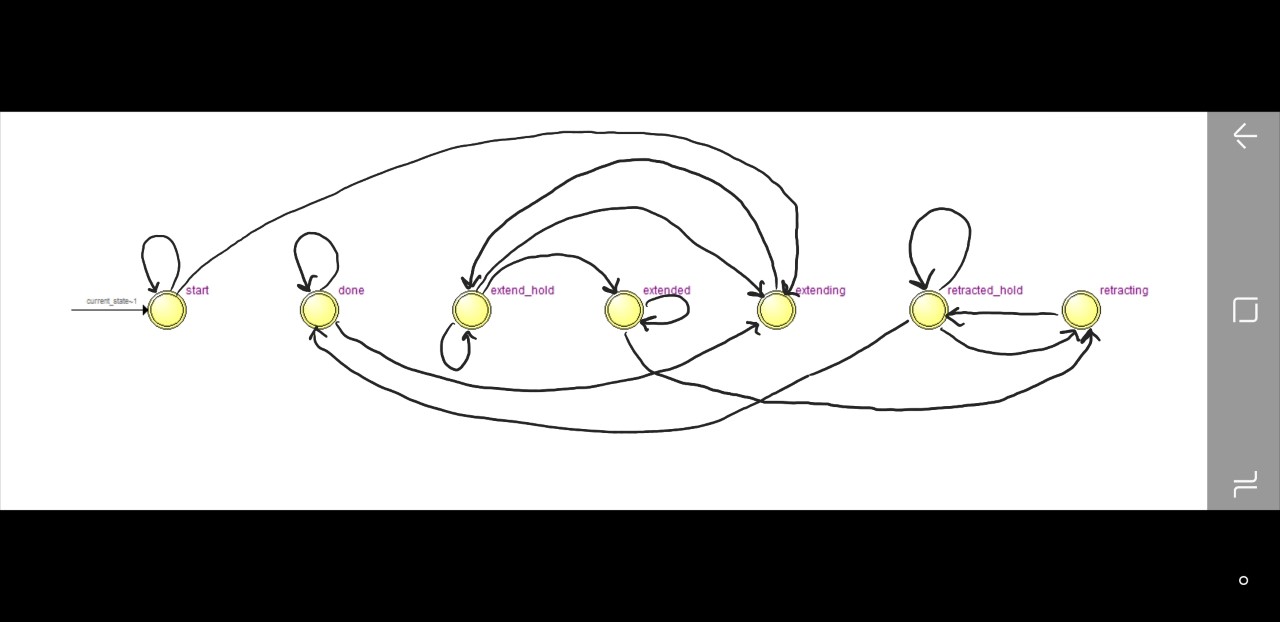


and 8 bit Binary counter

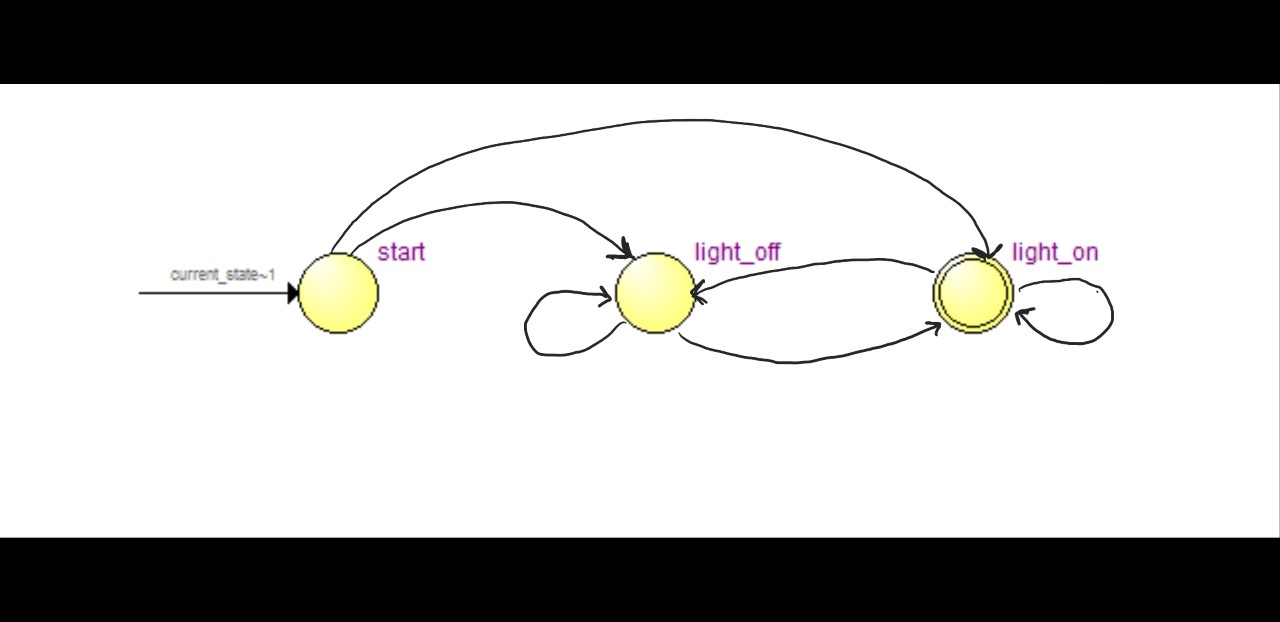
------------------------------------------------------------------------------------------------------------------------------------------

STATE MACHINE DIAGRAMS

Moore State Machine 2 (Grappler)

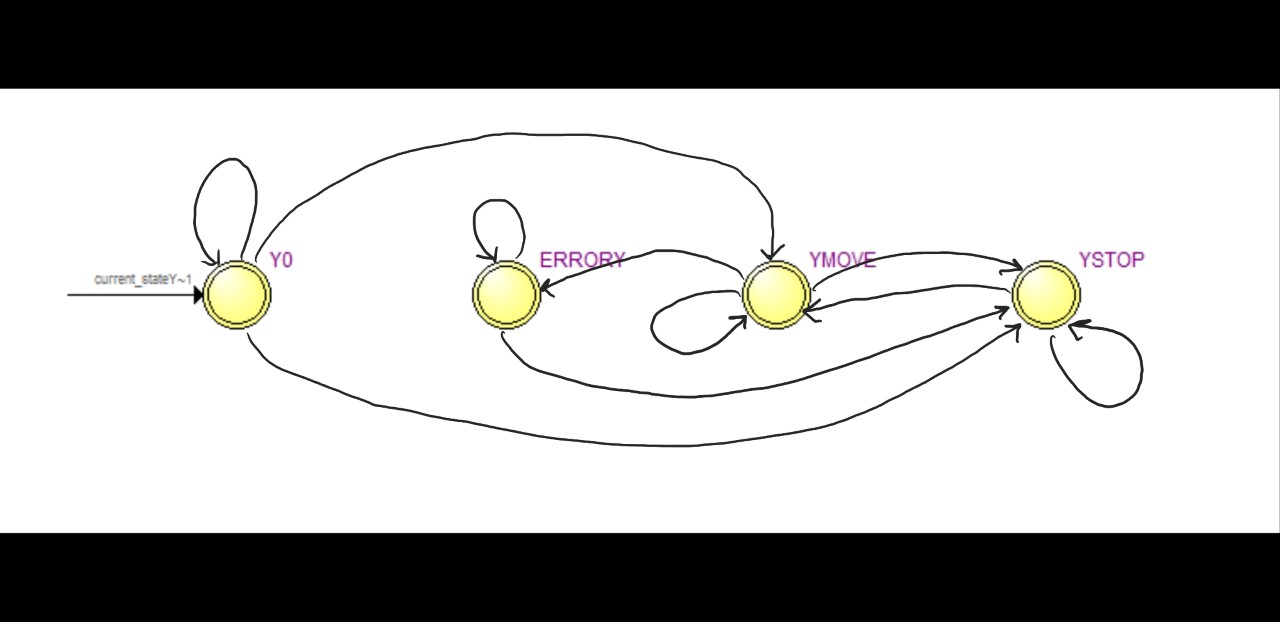


Moore State Machine (Extender)



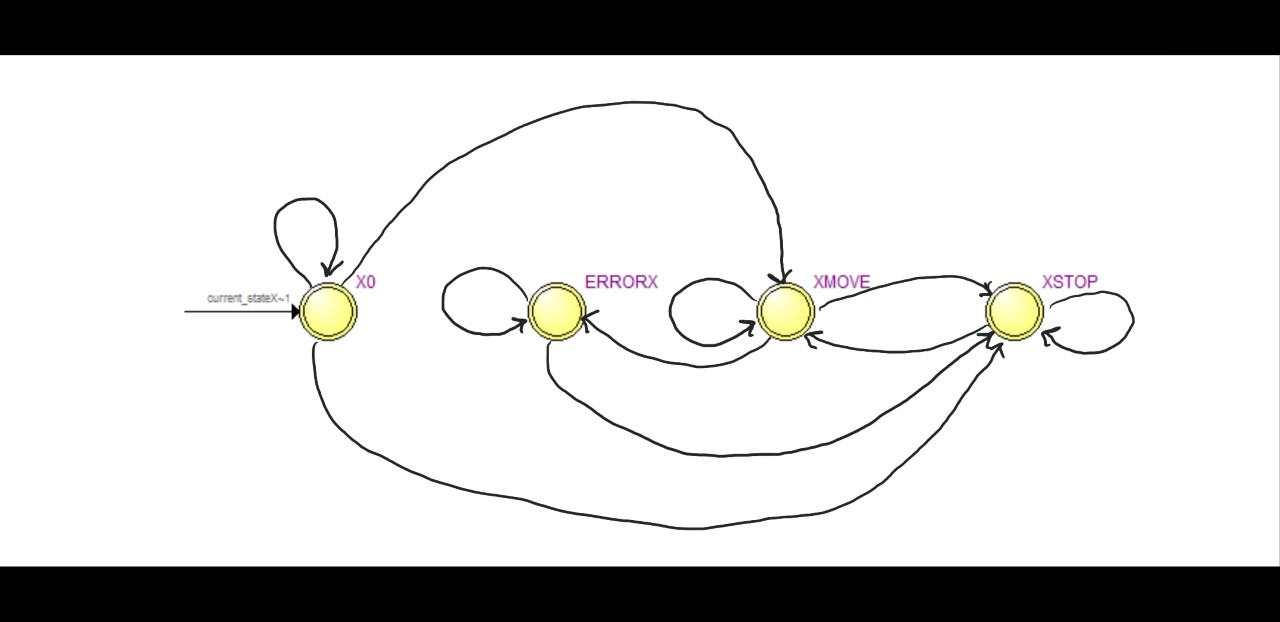
Mealy State Machine

(Y Motion)



Mealy State Machine

(X Motion)



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

Moore State Machine (VHDL Code) (Extender)

-- We Programmed our extender such that it continues to extend as we hold down the push button

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

Entity Moore\_SM IS Port

(

clk\_input, rst\_n, exten, button : IN std\_logic;

leds : IN std\_logic\_vector(3 downto 0);

shift\_en, shift\_dir, exten\_out, grap\_en : OUT std\_logic

);

END ENTITY;

Architecture SM of Moore\_SM is

TYPE STATE\_NAMES IS (start,extending,extend\_hold, extended, retracting, retracted\_hold, done); -- 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 <= start;

ELSIF(rising\_edge(clk\_input)) THEN

current\_state <= next\_State;

END IF;

END PROCESS;

-- TRANSITION LOGIC PROCESS

Transition\_Section: PROCESS (exten, button, leds, current\_state)

BEGIN

CASE current\_state IS

WHEN start =>

IF(exten ='1' AND button='0') THEN

next\_state <= extending;

ELSE

next\_state <= start;

END IF;

WHEN extending =>

IF(button='1') THEN

next\_state <= extend\_hold;

END IF;

WHEN extend\_hold =>

IF(leds="1111") THEN

next\_state <= extended;

ELSIF(button='0') THEN

next\_state <= extending;

ELSE

next\_state <= extend\_hold;

END IF;

WHEN extended =>

IF(button='0') THEN

next\_state <= retracting;

ELSE

next\_state <= extended;

END IF;

WHEN retracting =>

IF(button='1')THEN

next\_state <= retracted\_hold;

END IF;

WHEN retracted\_hold =>

IF(leds="0000") THEN

next\_state <= done;

ELSIF(button='0') THEN

next\_state <= retracting;

ELSE

next\_state <= retracted\_hold;

END IF;

WHEN done =>

IF(button='0' AND exten ='1') THEN

next\_state <= extending;

ELSE

next\_state <= done;

END IF;

END CASE;

END PROCESS;

-- DECODER SECTION PROCESS

Decoder\_Section: PROCESS (current\_state)

BEGIN

CASE current\_state IS

WHEN start =>

shift\_en <= '0';

shift\_dir <= '0';

exten\_out <= '0';

grap\_en <= '0';

WHEN extending =>

shift\_en <= '1';

shift\_dir <= '1';

exten\_out <= '1';

grap\_en <= '0';

WHEN retracting =>

shift\_en <= '1';

shift\_dir <= '0';

exten\_out <= '1';

grap\_en <= '0';

WHEN extended =>

shift\_en <= '0';

shift\_dir <= '0';

exten\_out <= '1';

grap\_en <= '1';

WHEN done =>

shift\_en <= '0';

shift\_dir <= '0';

exten\_out <= '0';

grap\_en <= '0';

WHEN others =>

shift\_en <= '0';

shift\_dir <= '0';

exten\_out <= '1';

grap\_en <= '0';

END CASE;

END PROCESS;

END ARCHITECTURE SM;

Moore State Machine 2 (VHDL Code) (Grappler Code)

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

Entity Moore\_SM2 IS Port

(

clk\_input, rst\_n, button, enable : IN std\_logic;

led : OUT std\_logic

);

END ENTITY;

Architecture SM2 of Moore\_SM2 is

TYPE STATE\_NAMES IS (start, light\_on, light\_off); -- 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 <= start;

ELSIF(rising\_edge(clk\_input)) THEN

current\_state <= next\_State;

END IF;

END PROCESS;

-- TRANSITION LOGIC PROCESS

Transition\_Section: PROCESS (button, enable, current\_state)

BEGIN

CASE current\_state IS

WHEN start =>

IF (enable = '1' AND button = '0') then

next\_state <= light\_on;

ELSIF (enable ='0' OR button = '1') then

next\_state <= light\_off;

END IF;

WHEN light\_on =>

IF (enable = '1' AND button = '0') then

next\_state <= light\_on;

ELSIF (enable ='0' OR button = '1') then

next\_state <= light\_off;

END IF;

WHEN light\_off =>

IF (enable = '1' AND button = '0') then

next\_state <= light\_on;

ELSIF (enable ='0' OR button = '1') then

next\_state <= light\_off;

END IF;

END CASE;

END PROCESS;

-- DECODER SECTION PROCESS

Decoder\_Section: PROCESS (current\_state)

BEGIN

CASE current\_state IS

WHEN start =>

led <= '0';

WHEN light\_on =>

If (enable = '1' AND button = '0') then

led <= '1';

End if;

WHEN light\_off =>

led <= '0';

END CASE;

END PROCESS;

END ARCHITECTURE SM2;

Mealy State Machine (VHDL Code) (XY Motion)

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

Entity Mealy\_SM IS Port

(

clk\_input,XMOTION, YMOTION, EX\_OUT,XEQ,XGT,XLT,YEQ,YGT,YLT : IN std\_logic;

rst\_n : in std\_logic := '0';

clk\_enX,clk\_enY, Xcount,Ycount,ERRORled,EXTD\_EN : OUT std\_logic

);

END ENTITY;

Architecture SM of Mealy\_SM is

TYPE STATE\_NAMESX IS (X0,XMOVE,XSTOP,ERRORX); -- list all the STATE\_NAMES values

TYPE STATE\_NAMESY IS (Y0,YMOVE,YSTOP,ERRORY);

SIGNAL current\_stateX, next\_stateX : STATE\_NAMESX; -- signals of type STATE\_NAMES

SIGNAL current\_stateY, next\_stateY : STATE\_NAMESY; -- signals of type STATE\_NAMES

SIGNAL EXTDX : std\_logic;

SIGNAL EXTDY : std\_logic;

SIGNAL ledX : std\_logic := '0';

SIGNAL ledY : std\_logic := '0';

signal EXTD\_OUT :std\_logic;

BEGIN

EXTD\_OUT <= EX\_OUT;

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

--State Machine:

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

-- REGISTER\_LOGIC PROCESS:

Register\_Section: PROCESS (clk\_input, rst\_n, next\_stateX,next\_stateY) -- this process synchronizes the activity to a clock

BEGIN

IF (rst\_n = '0') THEN

current\_stateX <= X0;

current\_stateY <= Y0;

ELSIF(rising\_edge(clk\_input)) THEN

current\_stateX <= next\_StateX;

current\_stateY <= next\_StateY;

END IF;

END PROCESS;

-- TRANSITION LOGIC PROCESS

Transition\_Section: PROCESS (XMOTION, EXTD\_OUT, XEQ, XGT,XLT, current\_stateX)

BEGIN

CASE current\_stateX IS

WHEN X0 =>

IF(XMOTION = '0' AND XEQ = '0' AND EXTD\_OUT = '0') THEN

next\_stateX <= XMOVE;

ELSIF (XEQ = '1') THEN

next\_stateX <= XSTOP;

ELSE

next\_stateX <= X0;

END IF;

WHEN XMOVE =>

IF(EXTD\_OUT = '1') THEN

next\_stateX <= ERRORX;

ELSIF (XMOTION='0' AND XLT = '1') THEN

next\_stateX <= XMOVE;

ELSIF(XMOTION='0' AND XGT = '1') THEN

next\_stateX <= XMOVE;

ELSIF(XMOTION='1' OR XEQ = '1') THEN

next\_stateX <= XSTOP;

END IF;

WHEN XSTOP =>

IF(XMOTION='0') THEN

next\_stateX <= XMOVE;

ELSE

next\_stateX <= XSTOP;

END IF;

WHEN ERRORX =>

IF(EXTD\_OUT = '0') THEN

next\_stateX <= XSTOP;

ELSE

next\_stateX <= ERRORX;

END IF;

END CASE;

END PROCESS;

-- DECODER SECTION PROCESS

Decoder\_Section: PROCESS (XMOTION, EXTD\_OUT, XEQ, XGT,XLT, current\_stateX)

BEGIN

CASE current\_stateX IS

WHEN X0 =>

EXTDX <= '0';

WHEN XMOVE =>

EXTDX <= '0';

clk\_enX <= '1';

IF(XMOTION='0' AND XGT = '1') THEN

Xcount <= '0';

ELSIF(XMOTION='1' OR XEQ = '1') THEN

clk\_enX <= '0';

ELSIF (XMOTION='0' AND XLT = '1') THEN

Xcount <= '1';

END IF;

WHEN XSTOP =>

IF (XEQ = '1') THEN

EXTDX <= '1';

ELSE

EXTDX <= '0';

END IF;

WHEN ERRORX =>

ledX <= '1';

IF(EXTD\_OUT = '0') THEN

ledX <= '0';

END IF;

END CASE;

END PROCESS;

Transition\_Section2: PROCESS (YMOTION, EXTD\_OUT,YEQ, YGT,YLT, current\_stateY)

BEGIN

CASE current\_stateY IS

WHEN Y0 =>

IF(YMOTION = '0' AND YEQ = '0' AND EXTD\_OUT = '0') THEN

next\_stateY <= YMOVE;

ELSIF (YEQ = '1') THEN

next\_stateY <= YSTOP;

ELSIF (EXTD\_OUT = '1') THEN

next\_stateY <= ERRORY;

ELSE

next\_stateY <= Y0;

END IF;

WHEN YMOVE =>

IF (EXTD\_OUT = '1') THEN

next\_stateY <= ERRORY;

ELSIF(YMOTION='0' AND YLT = '1') THEN

next\_stateY <= YMOVE;

ELSIF(YMOTION='0' AND YGT = '1') THEN

next\_stateY <= YMOVE;

ELSIF(YMOTION='1' OR YEQ = '1') THEN

next\_stateY <= YSTOP;

END IF;

WHEN YSTOP =>

IF(YMOTION='0') THEN

next\_stateY <= YMOVE;

ELSE

next\_stateY <= YSTOP;

END IF;

WHEN ERRORY =>

IF(EXTD\_OUT = '0') THEN

next\_stateY <= YSTOP;

ELSE

next\_stateY <= ERRORY;

END IF;

END CASE;

END PROCESS;

-- DECODER SECTION PROCESS

Decoder\_Section2: PROCESS (YMOTION, EXTD\_OUT,YEQ, YGT,YLT, current\_stateY)

BEGIN

CASE current\_stateY IS

WHEN Y0 =>

EXTDY <= '0';

WHEN YMOVE =>

EXTDY <= '0';

clk\_enY <= '1';

IF(YMOTION='0' AND YGT = '1') THEN

Ycount <= '0';

ELSIF(YMOTION='1' OR YEQ = '1') THEN

clk\_enY <= '0';

ELSIF (YMOTION='0' AND YLT = '1') THEN

Ycount <= '1';

END IF;

WHEN YSTOP =>

IF (YEQ = '1') THEN

EXTDY <= '1';

ELSE

EXTDY <= '0';

END IF;

WHEN ERRORY =>

ledY <= '1';

IF(EXTD\_OUT = '0') THEN

ledY <= '0';

END IF;

END CASE;

END PROCESS;

ERRORled <= ledX OR ledY;

EXTD\_EN <= EXTDX AND EXTDY;

END ARCHITECTURE SM;

Moore vs Mealy State Machine

Mealy State Machine is a finite-state machine whose output values are determined both by its current state and the current inputs.

Moore State Machine is a state machine whose output values are determined only by its current state.

In our VHDL code implementation we have used the similar logic to build the Moore and Mealy State Machines.

Fitter Report

