

# 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,

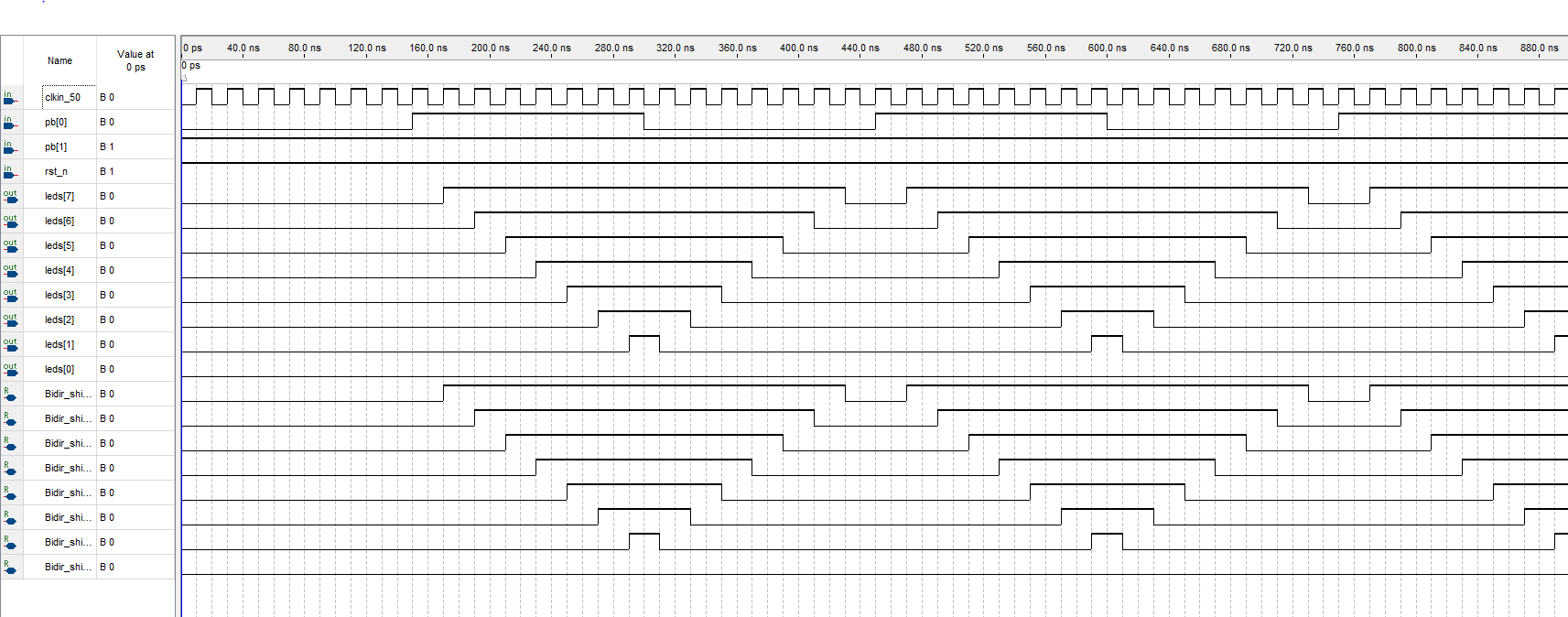
ERROR

);

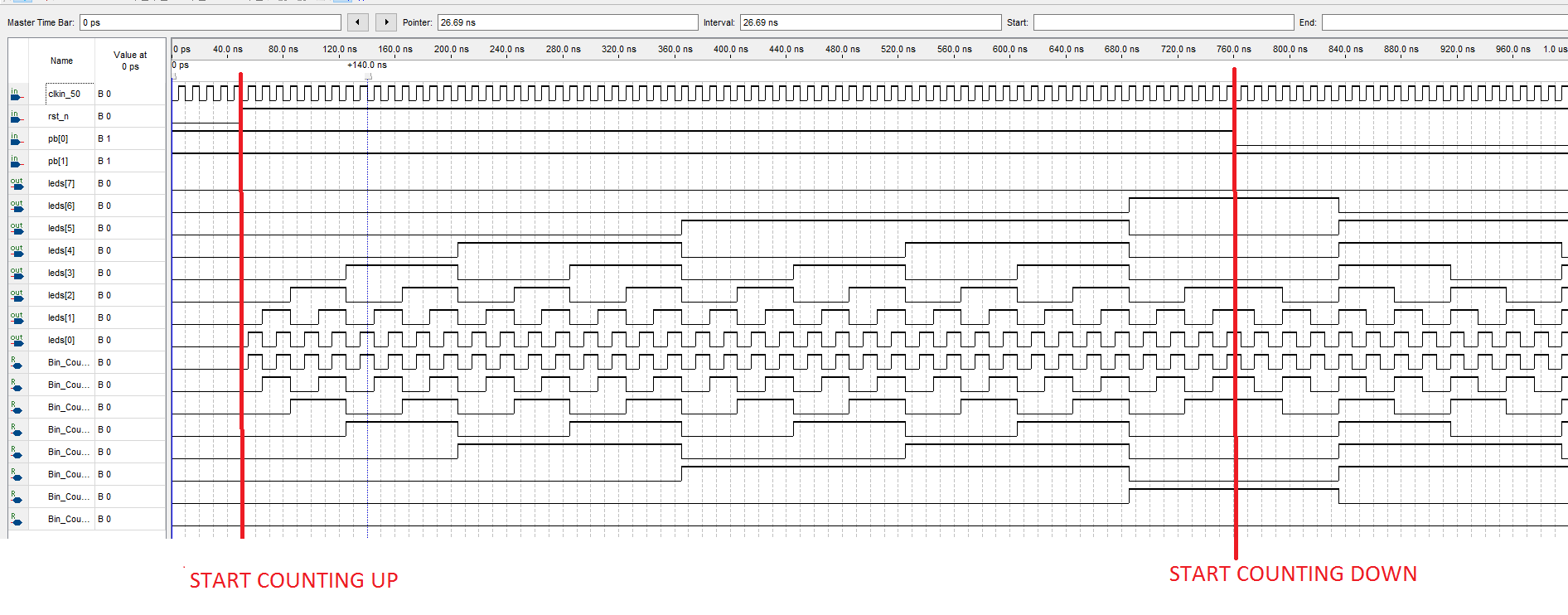
--INST13ERROR\_MUX: mux PORT MAP (ERROR7seg,

END SimpleCircuit;

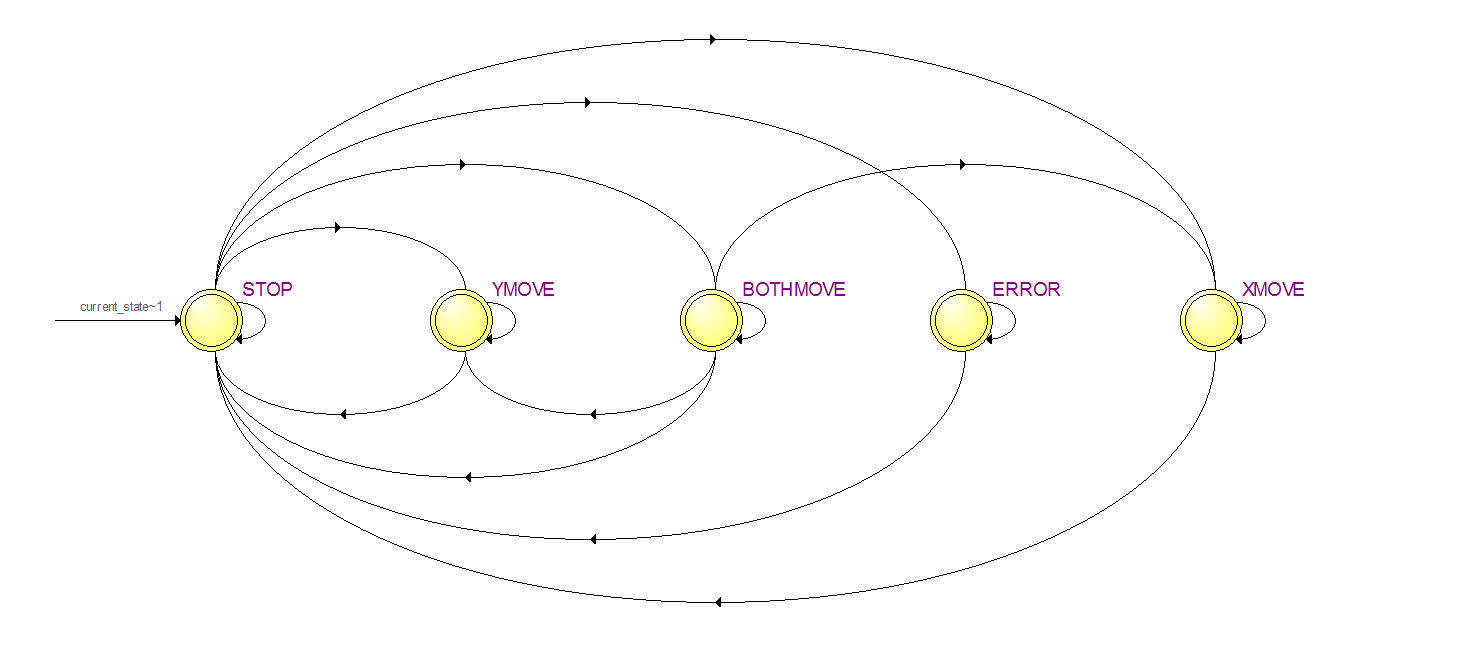
## Simulation 1: 8-Bits Shift Register



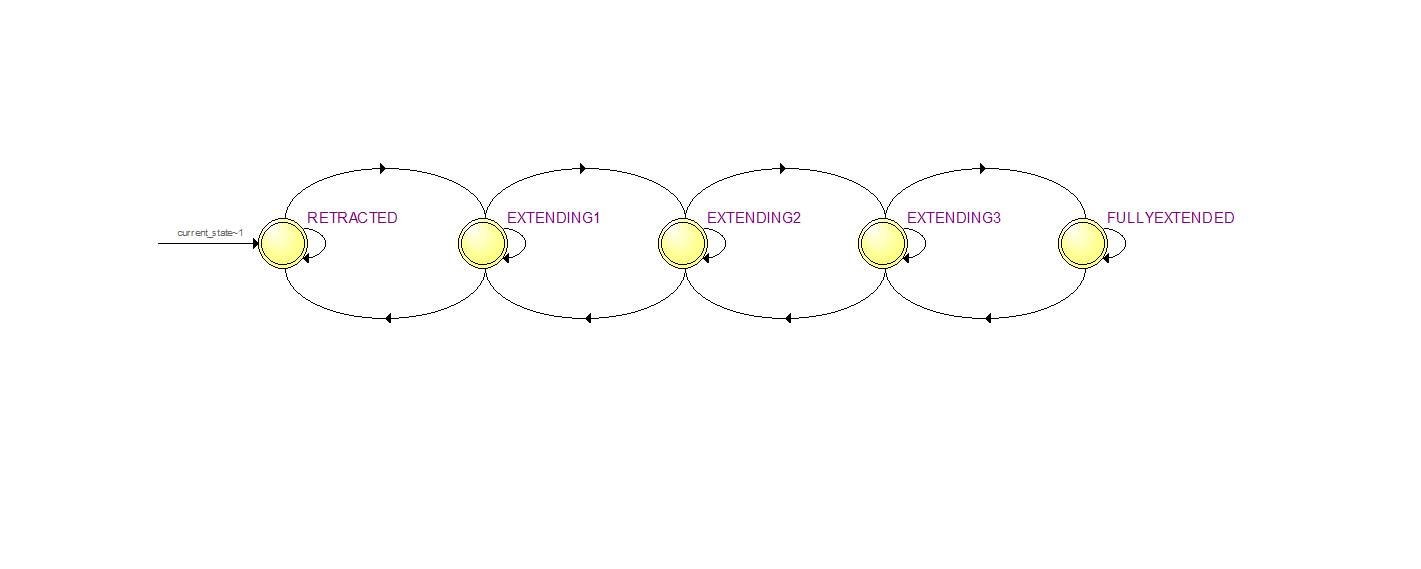
## Simulation 2: 8-Bits Counter



## 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

