library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL; use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

package mem is

constant width :integer := 4; constant memsize :integer := 8 ;

type meory is array(0 to (memsize-1))of std\_logic\_vector ((width-1) downto 0); end package mem;

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL; use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

use work.mem.all;

entity fifo is

Port ( clk : in STD\_LOGIC; rst : in STD\_LOGIC;

rd : in STD\_LOGIC; wr : in STD\_LOGIC;

datain : in STD\_LOGIC\_VECTOR (3 downto 0); memfull : out STD\_LOGIC;

mememp : out STD\_LOGIC;

dataout : out STD\_LOGIC\_VECTOR (3 downto 0)); end fifo;

architecture Behavioral of fifo is

signal cdiv : std\_logic\_vector(25 downto 0); signal clk : std\_logic;

signal ramtmp : meory;

signal cntr : std\_logic\_vector(2 downto 0);

signal rdptr , wrptr : std\_logic\_vector(2 downto 0);

begin

P1: process(mclk,rst) -- system clk begin

if rst = '1' then

cdiv <= ( others => '0');

elsif mclk'event and mclk = '1' then

<= cdiv + '1' ;

if ( cdiv = "10111110101111000010000000" ) then

clk <= '1';

cdiv <= (others =>'0');

end if;

else end if;

clk <= '0';

end process;

memoryfull\_empty : process(cntr,rdptr,wrptr) -- memoryfull\_empty process begin

if(cntr = "000")and(rdptr = wrptr) then memfull <= '0';

mememp <= '1';

elsif(cntr = "111")and(rdptr = wrptr) then memfull <= '1';

mememp <= '0';

else

end if; end process;

memfull <= '0';

mememp <= '0';

p2: process (rst, clk, rd, wr) begin

if rst = '1' then

cntr <= "000";

elsif clk'event and clk = '1' then

if wr = '1' and rd = '0' then cntr <= cntr + '1';

if cntr <= "111" then

cntr <= "000";

end if;

elsif wr = '0' and rd = '1' then cntr <= cntr - '1';

if cntr <= "000" then

cntr <= "111"; else NULL;

end if;

end if;

end if; end process;

p3: process(clk,rst,rd,wr,wrptr,rdptr) begin

if rst = '1' then

wrptr <= (others => '0'); rdptr <= (others => '0');

elsif clk'event and clk = '1' then

if wr = '1' and rd = '0' then

wrptr <= wrptr + '1'; rdptr <= rdptr;

elsif wr = '0' and rd = '1' then rdptr <= rdptr + '1'; wrptr <= wrptr;

else NULL; end if;

end if; end process;

p4: process(clk,rst,rd,wr,cntr) begin

if clk'event and clk = '1' then if cntr /= "000" then

if rd = '1' and wr = '0' then

dataout <= ramtmp(conv\_integer(rdptr));

end if;

elsif cntr /= "111" then

if wr = '1' and rd = '0' then

ramtmp(conv\_integer(wrptr)) <= datain;

end if; end process; end Behavioral;

end if;

end if;

Test Bench

-- Name:

-- Roll No.:

-- Experiment No. 4: To write VHDL code with test bench, synthesis, implementation on PLD, 4X4 FIFO.

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

ENTITY tb3 IS END tb3;

ARCHITECTURE behavior OF tb3 IS

-- Component Declaration for the Unit Under Test (UUT) COMPONENT fifo

PORT(

clk : IN std\_logic;

rst : IN std\_logic; rd : IN std\_logic; wr : IN std\_logic;

datain : IN std\_logic\_vector(3 downto 0); memfull : OUT std\_logic;

mememp : OUT std\_logic;

dataout : OUT std\_logic\_vector(3 downto 0)

);

END COMPONENT;

--Inputs

signal clk : std\_logic := '0'; signal rst : std\_logic := '0'; signal rd : std\_logic := '0'; signal wr : std\_logic := '0';

signal datain : std\_logic\_vector(3 downto 0) := (others => '0');

--Outputs

signal memfull : std\_logic; signal mememp : std\_logic;

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

-- Clock period definitions

constant clk\_period : time := 10 ns; BEGIN

-- Instantiate the Unit Under Test (UUT) uut: fifo PORT MAP (

clk => clk, rst => rst, rd => rd, wr => wr,

datain => datain, memfull => memfull, mememp => mememp, dataout => dataout

);

-- Clock process definitions clk\_process :process

begin

clk <= '0';

wait for clk\_period/2; clk <= '1';

wait for clk\_period/2;

end process;

-- Stimulus process stim\_proc: process begin

rst <= '1';

wait for 10 ns; rst <= '0';

wait for 10 ns; wr <= '1';

rd <= '0'

datain <= "1011"; wait for 10 ns; rst <= '0';

wr <= '1';

rd <= '0';

datain <= "1010"; wait for 10 ns; rst <= '0';

wr <= '1';

rd <= '0';

datain <= "1001"; wait for 10 ns; datain <= "1000"; wait for 10 ns; datain <= "0111"; wait for 10 ns; datain <= "0110"; wait for 10 ns; datain <= "0101"; wait for 10 ns; datain <= "0100"; wait for 10 ns;

wr <= '0';

rd <= '1';

wait for 30 ns; wr <= '0';

rd <= '1';

wait for 10 ns; wr <= '0';

rd <= '1';

wait for 10 ns; wr <= '0';

rd <= '1';

wait for 30 ns; wr <= '0';

rd <= '1';

wait for 10 ns;

wr <= '0';

rd <= '1';

wait for 10 ns; wr <= '0';

rd <= '1';

wait for 30 ns;

end process; END;

wr <= '0';

rd <= '1';

wait for 10 ns; rst <= '1';

wait;

Code

-- Name

-- Experiment No. 5: To write VHDL code, simulate with test bench, synthesis, implement on PLC, LCD.

With clk divisions:

library IEEE;

use IEEE.std\_logic\_1164.all; use IEEE.numeric\_std.all;

use IEEE.std\_logic\_unsigned.all;

entity lcd is

Port ( mclk,rst : in STD\_LOGIC;

lcd\_data : out STD\_LOGIC\_VECTOR (7 downto 0); lcd\_rw : out STD\_LOGIC;

lcd\_rs : out STD\_LOGIC; lcd\_en : out STD\_LOGIC);

end lcd;

architecture Behavioral of lcd is

signal cnt : std\_logic\_vector(20 downto 0); signal clk : std\_logic;

type states is(rst\_state,first\_cwd,first\_cwd1,second\_cwd,second\_cwd1, third\_cwd,third\_cwd1,fourth\_cwd,fourth\_cwd1, first\_char,first\_char1,second\_char,second\_char1,third\_char, third\_char1,fourth\_char,fourth\_char1,fifth\_char,fifth\_char1, sixth\_char,sixth\_char1,seventh\_char,seventh\_char1);

signal p\_state,n\_state: states;

begin

clk\_div : process(mclk,rst) -- system clk begin

if rst = '1' then

cnt <= (others => '0');

elsif mclk'event and mclk = '1' then cnt <= cnt + '1';

if ( cnt = "011110100001001000000" ) then

clk <= '1';

cnt <= (others =>'0');

end if; end process;

else end if;

clk <= '0';

mem\_elemt:process(clk,rst) begin

if rst = '1' then

p\_state <= rst\_state; elsif clk'event and clk = '1' then

p\_state <= n\_state;

end if; end process;

next\_state:process(p\_state) begin

case p\_state is

when rst\_state =>

lcd\_rw <= '0';

lcd\_rs <= '0';

lcd\_en <= '0'; n\_state <= first\_cwd;

lcd\_data <= "00111100"; -- 3CH when first\_cwd =>

lcd\_rw <= '0';

lcd\_rs <= '0';

lcd\_en <= '1';

n\_state <= first\_cwd1;

lcd\_data <= "00111100"; -- 3CH 8 bit mode 2 lines 5x8 dots when first\_cwd1 => \

lcd\_rw <= '0';

lcd\_rs <= '0';

lcd\_en <= '0';

n\_state <= second\_cwd; lcd\_data <= "00001100"; -- 0CH

when second\_cwd =>

lcd\_rw <= '0';

lcd\_rs <= '0';

lcd\_en <= '1';

n\_state <= second\_cwd1;

lcd\_data <= "00001100"; -- 0CH Display on Cursor off when second\_cwd1 =>

lcd\_rw <= '0';

lcd\_rs <= '0';

lcd\_en <= '0';

n\_state <= third\_cwd; lcd\_data <= "00000001"; -- 01H

when third\_cwd =>

lcd\_rw <= '0';

lcd\_rs <= '0';

lcd\_en <= '1';

n\_state <= third\_cwd1;

lcd\_data <= "00000001"; -- 01H Clear Display when third\_cwd1 =>

lcd\_rw <= '0';

lcd\_rs <= '0';

lcd\_en <= '0';

n\_state <= fourth\_cwd; lcd\_data <= "00000010"; -- 02H

when fourth\_cwd =>

lcd\_rw <= '0';

lcd\_rs <= '0';

lcd\_en <= '1';

n\_state <= fourth\_cwd1;

lcd\_data <= "00000010"; -- 02H Return Cursor to Home when fourth\_cwd1 =>

lcd\_rw <= '0';

lcd\_rs <= '1';

lcd\_en <= '0';

n\_state <= first\_char; lcd\_data <= "01010011";

when first\_char =>

lcd\_rw <= '0';

lcd\_rs <= '1';

lcd\_en <= '1';

n\_state <= first\_char1; lcd\_data <= "01010011";

when first\_char1 =>

lcd\_rw <= '0';

lcd\_rs <= '1';

lcd\_en <= '0'; n\_state <= second\_char; lcd\_data <= "01001000";

when second\_char =>

lcd\_rw <= '0';

lcd\_rs <= '1';

lcd\_en <= '1'; n\_state <= second\_char1; lcd\_data <= "01001000";

when second\_char1 =>

lcd\_rw <= '0';

lcd\_rs <= '1';

lcd\_en <= '0'; n\_state <= third\_char; lcd\_data <= "01010101";

when third\_char =>

lcd\_rw <= '0';

lcd\_rs <= '1';

lcd\_en <= '1'; n\_state <= third\_char1; lcd\_data <= "01010101";

when third\_char1 =>

lcd\_rw <= '0';

lcd\_rs <= '1';

lcd\_en <= '0'; n\_state <= fourth\_char; lcd\_data <= "01000010";

when fourth\_char =>

lcd\_rw <= '0';

lcd\_rs <= '1';

lcd\_en <= '1'; n\_state <= fourth\_char1; lcd\_data <= "01000010";

when fourth\_char1 =>

lcd\_rw <= '0';

lcd\_rs <= '1';

lcd\_en <= '0'; n\_state <= fifth\_char; lcd\_data <= "01001000";

when fifth\_char =>

lcd\_rw <= '0';

lcd\_rs <= '1';

lcd\_en <= '1'; n\_state <= fifth\_char1; lcd\_data <= "01001000";

when fifth\_char1 =>

lcd\_rw <= '0';

lcd\_rs <= '1';

lcd\_en <= '0'; n\_state <= sixth\_char; lcd\_data <= "01000001";

when sixth\_char =>

lcd\_rw <= '0';

lcd\_rs <= '1';

lcd\_en <= '1'; n\_state <= sixth\_char1; lcd\_data <= "01000001";

when sixth\_char1 =>

lcd\_rw <= '0';

lcd\_rs <= '1';

lcd\_en <= '0'; n\_state <= seventh\_char; lcd\_data <= "01001101";

when seventh\_char =>

lcd\_rw <= '0';

lcd\_rs <= '1';

lcd\_en <= '1'; n\_state <= seventh\_char1; lcd\_data <= "01001101";

when others =>

NULL;

end case; end process;

end Behavioral;

Without clk divisions:

library IEEE;

use IEEE.std\_logic\_1164.all; use IEEE.numeric\_std.all;

use IEEE.std\_logic\_unsigned.all;

entity lcd is

Port ( clk,rst : in STD\_LOGIC;

lcd\_data : out STD\_LOGIC\_VECTOR (7 downto 0); lcd\_rw : out STD\_LOGIC;

lcd\_rs : out STD\_LOGIC; lcd\_en : out STD\_LOGIC);

end lcd;

architecture Behavioral of lcd is

signal cnt : std\_logic\_vector(20 downto 0); signal clk : std\_logic;

begin

type states is(rst\_state,first\_cwd,first\_cwd1,second\_cwd,second\_cwd1, third\_cwd,third\_cwd1,fourth\_cwd,fourth\_cwd1, first\_char,first\_char1,second\_char,second\_char1,third\_char, third\_char1,fourth\_char,fourth\_char1,fifth\_char,fifth\_char1, sixth\_char,sixth\_char1,seventh\_char,seventh\_char1);

signal p\_state,n\_state: states;

clk\_div : process(mclk,rst) -- system clk begin

if rst = '1' then

cnt <= (others => '0'); elsif mclk'event and mclk = '1' then

cnt <= cnt + '1';

if ( cnt = "011110100001001000000" ) then

clk <= '1';

cnt <= (others =>'0');

end if; end process;

mem\_elemt:process(clk,rst) begin

else end if;

clk <= '0';

if rst = '1' then

p\_state <= rst\_state; elsif clk'event and clk = '1' then

p\_state <= n\_state;

end if; end process;

next\_state:process(p\_state) begin

case p\_state is

when rst\_state =>

lcd\_rw <= '0';

lcd\_rs <= '0';

lcd\_en <= '0'; n\_state <= first\_cwd;

lcd\_data <= "00111100"; -- 3CH when first\_cwd =>

lcd\_rw <= '0';

lcd\_rs <= '0';

lcd\_en <= '1';

n\_state <= first\_cwd1; lcd\_data <= "00111100";

when first\_cwd1 =>

lcd\_rw <= '0';

lcd\_rs <= '0';

lcd\_en <= '0';

n\_state <= second\_cwd; lcd\_data <= "00001100";

when second\_cwd =>

lcd\_rw <= '0';

lcd\_rs <= '0';

lcd\_en <= '1'; n\_state <= second\_cwd1; lcd\_data <= "00001100";

when second\_cwd1 => --

lcd\_rw <= '0';

lcd\_rs <= '0';

lcd\_en <= '0'; n\_state <= third\_cwd;

lcd\_data <= "00000001"; -- 01H when third\_cwd =>

lcd\_rw <= '0';

lcd\_rs <= '0';

lcd\_en <= '1'; n\_state <= third\_cwd1; lcd\_data <= "00000001";

when third\_cwd1 =>

lcd\_rw <= '0';

lcd\_rs <= '0';

lcd\_en <= '0'; n\_state <= fourth\_cwd; lcd\_data <= "00000010";

when fourth\_cwd =>

lcd\_rw <= '0';

lcd\_rs <= '0';

lcd\_en <= '1'; n\_state <= fourth\_cwd1; lcd\_data <= "00000010";

when fourth\_cwd1 =>

lcd\_rw <= '0';

lcd\_rs <= '1';

lcd\_en <= '0'; n\_state <= first\_char; lcd\_data <= "01010011";

when first\_char =>

lcd\_rw <= '0';

lcd\_rs <= '1';

lcd\_en <= '1';

n\_state <= first\_char1; lcd\_data <= "01010011";

when first\_char1 =>

lcd\_rw <= '0';

lcd\_rs <= '1';

lcd\_en <= '0'; n\_state <= second\_char; lcd\_data <= "01001000";

when second\_char =>

lcd\_rw <= '0';

lcd\_rs <= '1';

lcd\_en <= '1'; n\_state <= second\_char1; lcd\_data <= "01001000";

when second\_char1 => --

send 3rd Character Data

lcd\_rw <= '0';

lcd\_rs <= '1';

lcd\_en <= '0'; n\_state <= third\_char; lcd\_data <= "01010101";

when third\_char =>

lcd\_rw <= '0';

lcd\_rs <= '1';

lcd\_en <= '1'; n\_state <= third\_char1; lcd\_data <= "01010101";

when third\_char1 =>

lcd\_rw <= '0';

lcd\_rs <= '1';

lcd\_en <= '0'; n\_state <= fourth\_char; lcd\_data <= "01000010";

when fourth\_char =>

lcd\_rw <= '0';

lcd\_rs <= '1';

lcd\_en <= '1'; n\_state <= fourth\_char1; lcd\_data <= "01000010";

when fourth\_char1 =>

lcd\_rw <= '0';

lcd\_rs <= '1';

lcd\_en <= '0'; n\_state <= fifth\_char; lcd\_data <= "01001000";

when fifth\_char =>

lcd\_rw <= '0';

lcd\_rs <= '1';

lcd\_en <= '1'; n\_state <= fifth\_char1; lcd\_data <= "01001000";

when fifth\_char1 =>

lcd\_rw <= '0';

lcd\_rs <= '1';

lcd\_en <= '0'; n\_state <= sixth\_char; lcd\_data <= "01000001";

when sixth\_char =>

lcd\_rw <= '0';

lcd\_rs <= '1';

lcd\_en <= '1'; n\_state <= sixth\_char1; lcd\_data <= "01000001";

when sixth\_char1 =>

lcd\_rw <= '0';

lcd\_rs <= '1';

lcd\_en <= '0';

n\_state <= seventh\_char; lcd\_data <= "01001101";

when seventh\_char =>

lcd\_rw <= '0';

lcd\_rs <= '1';

lcd\_en <= '1'; n\_state <= seventh\_char1; lcd\_data <= "01001101";

when others => NULL;

end case; end process;

end Behavioral;

Test Bench

-- Name:

-- Experiment No. 5: To write VHDL code, simulate with test bench, synthesis, implement on PLC, LCD.

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

ENTITY fortb\_lcdtb2 IS END fortb\_lcdtb2;

ARCHITECTURE behavior OF fortb\_lcdtb2 IS

-- Component Declaration for the Unit Under Test (UUT) COMPONENT lcd

PORT(

clk : IN std\_logic; rst : IN std\_logic;

lcd\_data : OUT std\_logic\_vector(7 downto 0); lcd\_rw : OUT std\_logic;

lcd\_rs : OUT std\_logic; lcd\_en : OUT std\_logic

);

END COMPONENT;

--Inputs

signal clk : std\_logic := '0'; signal rst : std\_logic := '0';

--Outputs

signal lcd\_data : std\_logic\_vector(7 downto 0); signal lcd\_rw : std\_logic;

signal lcd\_rs : std\_logic; signal lcd\_en : std\_logic;

-- Clock period definitions

constant clk\_period : time := 10 ns; BEGIN

-- Instantiate the Unit Under Test (UUT) uut: lcd PORT MAP (

clk => clk, rst => rst,

lcd\_data => lcd\_data, lcd\_rw => lcd\_rw, lcd\_rs => lcd\_rs, lcd\_en => lcd\_en

);

-- Clock process definitions clk\_process :process

begin

clk <= '0';

wait for clk\_period/2; clk <= '1';

wait for clk\_period/2;

end process;

-- Stimulus process stim\_proc: process begin

rst <= '1';

wait for 10 ns; rst <= '0';

wait for 200 ns;

wait;

end process;

END;

-- Name:

# clock pin for Basys2 Board

#NET "mclk" LOC = "B8"; # Bank = 0, Signal name = MCLK #NET "uclk" LOC = "M6"; # Bank = 2, Signal name = UCLK #NET "mclk" CLOCK\_DEDICATED\_ROUTE = FALSE;

#NET "uclk" CLOCK\_DEDICATED\_ROUTE = FALSE;

# Pin assignment for LEDs

#NET "mememp" LOC = "G1" ; # Bank = 3, Signal name = LD7 #NET "memfull" LOC = "P4" ; # Bank = 2, Signal name = LD6 #NET "Led<5>" LOC = "N4" ; # Bank = 2, Signal name = LD5 #NET "Led<4>" LOC = "N5" ; # Bank = 2, Signal name = LD4

NET "dataout<3>" LOC = "G1" ; # Bank = 2, Signal name = LD3 NET "dataout<2>" LOC = "P4" ; # Bank = 3, Signal name = LD2 NET "dataout<1>" LOC = "N4" ; # Bank = 2, Signal name = LD1 NET "dataout<0>" LOC = "N5" ; # Bank = 2, Signal name = LD0

#Pin assignment for SWs

NET "rd" LOC = "N3"; # Bank = 2, Signal name = SW7 NET "wr" LOC = "E2"; # Bank = 3, Signal name = SW6 #NET "sw<5>" LOC = "F3"; # Bank = 3, Signal name = SW5

#NET "sw<4>" LOC = "G3"; # Bank = 3, Signal name = SW4

NET "datain<3>" LOC = "B4"; # Bank = 3, Signal name = SW3 NET "datain<2>" LOC = "K3"; # Bank = 3, Signal name = SW2 NET "datain<1>" LOC = "L3"; # Bank = 3, Signal name = SW1 NET "datain<0>" LOC = "P11"; # Bank = 2, Signal name = SW0

# Pin assignment for RESET SW

NET "rst" LOC = "A7"; # Bank = 1, Signal name = BTN3

#NET "btn<2>" LOC = "M4"; # Bank = 0, Signal name = BTN2 #NET "btn<1>" LOC = "C11"; # Bank = 2, Signal name = BTN1 #NET "btn<0>" LOC = "G12"; # Bank = 0, Signal name = BTN0

-- Name:

# clock pin for Basys2 Board

NET "mclk" LOC = "B8"; # Bank = 0, Signal name = MCLK #NET "uclk" LOC = "M6"; # Bank = 2, Signal name = UCLK #NET "mclk" CLOCK\_DEDICATED\_ROUTE = FALSE;

#NET "uclk" CLOCK\_DEDICATED\_ROUTE = FALSE;

# Pin assignment for RESET SWs

NET "rst" LOC = "A7"; # Bank = 1, Signal name = BTN3 #NET " " LOC = "M4"; # Bank = 0, Signal name = BTN2 #NET " " LOC = "C11"; # Bank = 2, Signal name = BTN1 #NET " " LOC = "G12"; # Bank = 0, Signal name = BTN0

# Loop Back only tested signals connect keypad, lcd

#NET "PIO<72>" LOC = "B2" | DRIVE = 2 | PULLUP ; # Bank = 1, Signal name = JA1 #NET "PIO<73>" LOC = "A3" | DRIVE = 2 | PULLUP ; # Bank = 1, Signal name = JA2 #NET "PIO<74>" LOC = "J3" | DRIVE = 2 | PULLUP ; # Bank = 1, Signal name = JA3 #NET "PIO<75>" LOC = "B5" | DRIVE = 2 | PULLUP ; # Bank = 1, Signal name = JA4

# LCD Control

NET "lcd\_rs" LOC = "C6" | DRIVE = 2 | PULLUP ; # Bank = 1, Signal name = JB1 NET "lcd\_rw" LOC = "B6" | DRIVE = 2 | PULLUP ; # Bank = 1, Signal name = JB2 NET "lcd\_en" LOC = "C5" | DRIVE = 2 | PULLUP ; # Bank = 1, Signal name = JB3 #NET "row<0>" LOC = "B7" | DRIVE = 2 | PULLUP ; # Bank = 1, Signal name = JB4

# LCD Data Lines

NET "lcd\_data<7>" LOC = "A9" | DRIVE = 2 | PULLUP ; # Bank = 1, Signal name = JC1 NET "lcd\_data<6>" LOC = "B9" | DRIVE = 2 | PULLUP ; # Bank = 1, Signal name = JC2 NET "lcd\_data<5>" LOC = "A10" | DRIVE = 2 | PULLUP ; # Bank = 1, Signal name = JC3 NET "lcd\_data<4>" LOC = "C9" | DRIVE = 2 | PULLUP ; # Bank = 1, Signal name = JC4

NET "lcd\_data<3>" LOC = "C12" | DRIVE = 2 | PULLUP ; # Bank = 1, Signal name = JD1 NET "lcd\_data<2>" LOC = "A13" | DRIVE = 2 | PULLUP ; # Bank = 2, Signal name = JD2 NET "lcd\_data<1>" LOC = "C13" | DRIVE = 2 | PULLUP ; # Bank = 1, Signal name = JD3 NET "lcd\_data<0>" LOC = "D12" | DRIVE = 2 | PULLUP ; # Bank = 2, Signal name = JD4