**Lab #2 Example, Center LEDs**

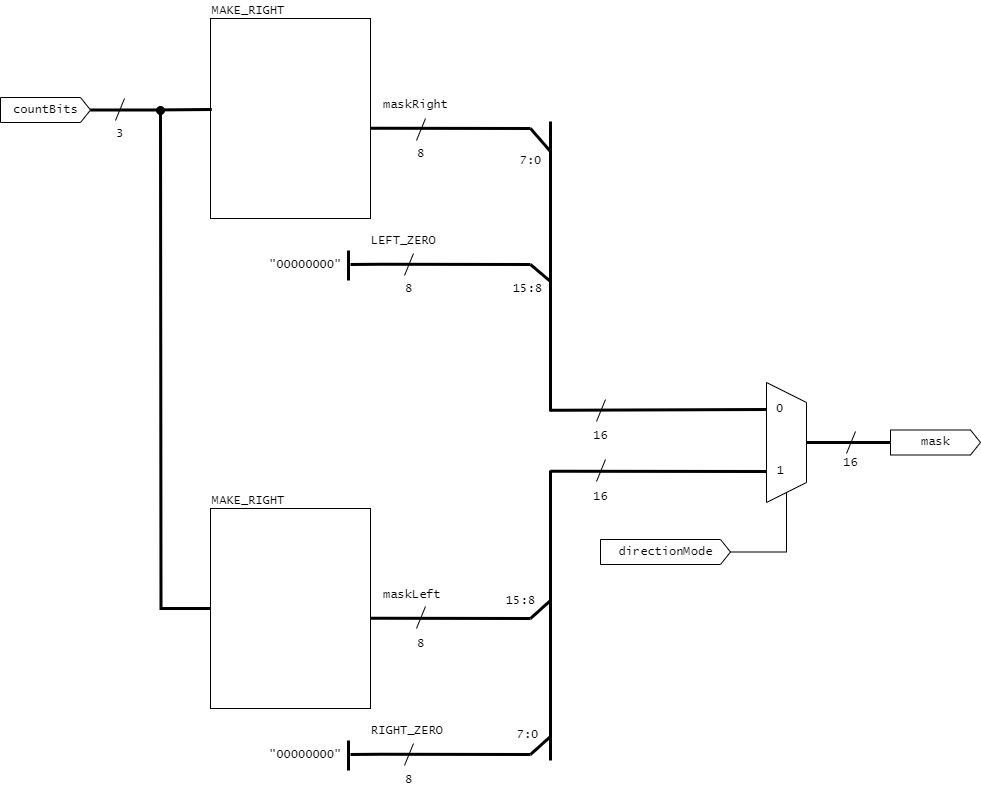
Sean Graham

Kennesaw State University

CPE 3020: VHDL Design with FPGAs

Professor Scott Tippens

Spring 2025

  
Fig. 1. Design diagram

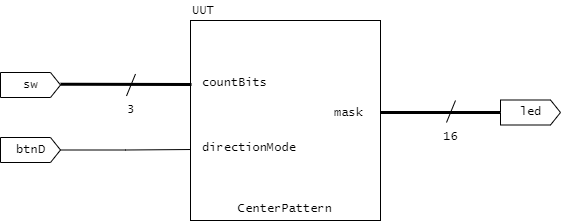
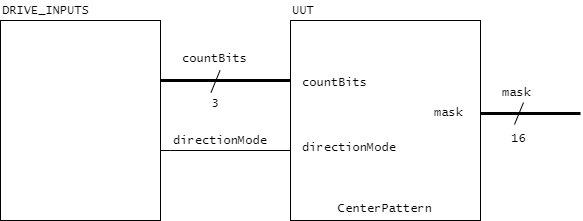


Fig. 2. Wrapper diagram

  
Fig. 3. Testbench diagram

A screen shot of a graph

Description automatically generated

Fig. 4. Simulation results, right direction

A screen shot of a graph

Description automatically generated  
Fig. 5. Simulation results, left direction

*--------------------------------------------------------------------------------*

*--*

*-- Lab 02 Demo: CenterPattern*

*-- Sean Graham*

*--*

*--     Generates a 16-bit pattern given a number of bits and a direction.*

*--*

*--     'countBits' is a 3-bit binary signal encoding the number of active bits,*

*--       between 0 to 7.*

*--*

*--     When 'directionMode' is 0, counts down from bit 7 down to 0*

*--     When 'directionMode' is 1, counts up from bit 15 to 8.*

*--*

*--     The generated pattern is output on 'mask'.*

*--*

*--------------------------------------------------------------------------------*

library ieee;

use ieee.std\_logic\_1164.all;

entity CenterPattern is

    port(

        countBits: in std\_logic\_vector(2 downto 0);

        directionMode: in std\_logic;

        mask: out std\_logic\_vector(15 downto 0)

    );

end CenterPattern;

architecture CenterPattern\_ARCH of CenterPattern is

*-----------------------------------------------------------------CONSTANTS--*

    constant ACTIVE: std\_logic := '1';

*-- integer bit representations*

    constant BITS\_ZERO : std\_logic\_vector(2 downto 0) := "000";

    constant BITS\_ONE  : std\_logic\_vector(2 downto 0) := "001";

    constant BITS\_TWO  : std\_logic\_vector(2 downto 0) := "010";

    constant BITS\_THREE: std\_logic\_vector(2 downto 0) := "011";

    constant BITS\_FOUR : std\_logic\_vector(2 downto 0) := "100";

    constant BITS\_FIVE : std\_logic\_vector(2 downto 0) := "101";

    constant BITS\_SIX  : std\_logic\_vector(2 downto 0) := "110";

    constant BITS\_SEVEN: std\_logic\_vector(2 downto 0) := "111";

*-- pattern literals*

*-- note, could be replaced with arrays*

    constant RIGHT\_ZERO : std\_logic\_vector(7 downto 0) := "00000000";

    constant RIGHT\_ONE  : std\_logic\_vector(7 downto 0) := "10000000";

    constant RIGHT\_TWO  : std\_logic\_vector(7 downto 0) := "11000000";

    constant RIGHT\_THREE: std\_logic\_vector(7 downto 0) := "11100000";

    constant RIGHT\_FOUR : std\_logic\_vector(7 downto 0) := "11110000";

    constant RIGHT\_FIVE : std\_logic\_vector(7 downto 0) := "11111000";

    constant RIGHT\_SIX  : std\_logic\_vector(7 downto 0) := "11111100";

    constant RIGHT\_SEVEN: std\_logic\_vector(7 downto 0) := "11111110";

    constant LEFT\_ZERO  : std\_logic\_vector(7 downto 0) := "00000000";

    constant LEFT\_ONE   : std\_logic\_vector(7 downto 0) := "00000001";

    constant LEFT\_TWO   : std\_logic\_vector(7 downto 0) := "00000011";

    constant LEFT\_THREE : std\_logic\_vector(7 downto 0) := "00000111";

    constant LEFT\_FOUR  : std\_logic\_vector(7 downto 0) := "00001111";

    constant LEFT\_FIVE  : std\_logic\_vector(7 downto 0) := "00011111";

    constant LEFT\_SIX   : std\_logic\_vector(7 downto 0) := "00111111";

    constant LEFT\_SEVEN : std\_logic\_vector(7 downto 0) := "01111111";

*-------------------------------------------------------------------SIGNALS--*

    signal maskLeft: std\_logic\_vector(7 downto 0);

    signal maskRight: std\_logic\_vector(7 downto 0);

begin

*-- generate pattern on left bits, as if (directionMode = 0)*

    MAKE\_LEFT: with countBits select

        maskLeft  <= LEFT\_ZERO   when BITS\_ZERO,

                     LEFT\_ONE    when BITS\_ONE,

                     LEFT\_TWO    when BITS\_TWO,

                     LEFT\_THREE  when BITS\_THREE,

                     LEFT\_FOUR   when BITS\_FOUR,

                     LEFT\_FIVE   when BITS\_FIVE,

                     LEFT\_SIX    when BITS\_SIX,

                     LEFT\_SEVEN  when others;

*-- generate pattern on right bits, as if (directionMode = 1)*

    MAKE\_RIGHT: with countBits select

        maskRight <= RIGHT\_ZERO  when BITS\_ZERO,

                     RIGHT\_ONE   when BITS\_ONE,

                     RIGHT\_TWO   when BITS\_TWO,

                     RIGHT\_THREE when BITS\_THREE,

                     RIGHT\_FOUR  when BITS\_FOUR,

                     RIGHT\_FIVE  when BITS\_FIVE,

                     RIGHT\_SIX   when BITS\_SIX,

                     RIGHT\_SEVEN when others;

*-- select correct mask based on actual value of directionMode*

    MUX: with directionMode select

        mask <= (maskLeft & RIGHT\_ZERO) when ACTIVE,

                (LEFT\_ZERO & maskRight) when others;

end CenterPattern\_ARCH;

*--------------------------------------------------------------------------------*

*--*

*-- Lab 02 Demo: CenterPattern\_BASYS3*

*-- Sean Graham*

*--*

*--     Wrapper for the CenterPattern entity on the BASYS3 board.*

*--*

*--     Rightmost 3 switches encode the number of active bits.*

*--     When the down button is held, direction of the pattern is reversed.*

*--     Outputs are displayed on the on-board LEDs.*

*--*

*--------------------------------------------------------------------------------*

library ieee;

use ieee.std\_logic\_1164.all;

entity CenterPattern\_BASYS3 is

    port(

        sw: in std\_logic\_vector(2 downto 0);

        btnD: in std\_logic;

        led: out std\_logic\_vector(15 downto 0)

    );

end CenterPattern\_BASYS3;

architecture CenterPattern\_BASYS3\_ARCH of CenterPattern\_BASYS3 is

*-----------------------------------------------------------------COMPONENT--*

*-- uut*

    component CenterPattern is

        port(

            countBits: in std\_logic\_vector(2 downto 0);

            directionMode: in std\_logic;

            mask: out std\_logic\_vector(15 downto 0)

        );

    end component;

begin

*-- map design to hardware*

    UUT: CenterPattern port map(

        countBits => sw,

        directionMode => btnD,

        mask => led

    );

end CenterPattern\_BASYS3\_ARCH;

*--------------------------------------------------------------------------------*

*--*

*-- Lab 02 Demo: CenterPattern\_TB*

*-- Sean Graham*

*--*

*--     Testbench for the CenterPattern entity.*

*--*

*--     Tests all permutations of countBits in the default direction (left),*

*--       then again in the active direction (right). Reports to the console on*

*--       unexpected output.*

*--     Uses array and record types to automate tests.*

*--*

*--------------------------------------------------------------------------------*

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity CenterPattern\_TB is

*-- note, a testbench will typically have an empty entity declaration*

end CenterPattern\_TB;

architecture CenterPattern\_TB\_ARCH of CenterPattern\_TB is

*----------------------------------------------------------TYPE DEFINITIONS--*

*-- grouped inputs to CenterMask with their expected combinational output*

    type t\_TEST is record

        count: integer;

        direction: std\_logic;

        result: std\_logic\_vector(15 downto 0);

    end record t\_TEST;

*-- a generic list of at least one test*

    type t\_TEST\_ARRAY is array (positive range <>) of t\_TEST;

*-----------------------------------------------------------------CONSTANTS--*

    constant ACTIVE: std\_logic := '1';

*-- time width of each test*

    constant STEP\_TIME: time := 20 ns;

*-- directions*

    constant RIGHT: std\_logic := '0';

    constant LEFT : std\_logic := '1';

*-- tests*

    constant ALL\_TESTS: t\_TEST\_ARRAY(1 to 16) := (

        ( 0, RIGHT, "0000000000000000" ),

        ( 1, RIGHT, "0000000010000000" ),

        ( 2, RIGHT, "0000000011000000" ),

        ( 3, RIGHT, "0000000011100000" ),

        ( 4, RIGHT, "0000000011110000" ),

        ( 5, RIGHT, "0000000011111000" ),

        ( 6, RIGHT, "0000000011111100" ),

        ( 7, RIGHT, "0000000011111110" ),

        ( 0, LEFT,  "0000000000000000" ),

        ( 1, LEFT,  "0000000100000000" ),

        ( 2, LEFT,  "0000001100000000" ),

        ( 3, LEFT,  "0000011100000000" ),

        ( 4, LEFT,  "0000111100000000" ),

        ( 5, LEFT,  "0001111100000000" ),

        ( 6, LEFT,  "0011111100000000" ),

        ( 7, LEFT,  "0111111100000000" )

    );

*-----------------------------------------------------------------COMPONENT--*

*-- uut*

    component CenterPattern is

        port(

            countBits: in std\_logic\_vector(2 downto 0);

            directionMode: in std\_logic;

            mask: out std\_logic\_vector(15 downto 0)

        );

    end component;

*-----------------------------------------------------------------FUNCTIONS--*

*-- convert std\_logic\_vector to string for logging*

    function print\_bits (

        bits: std\_logic\_vector

    ) return string is

        variable bitsString: string(1 to bits'length);

    begin

*-- set bits*

        for i in bits'range loop

*-- note std\_logic'image takes the form "'X'", so extract 2nd char*

            bitsString(i + 1) := std\_logic'image(bits(i))(2);

        end loop;

*-- return between quotes*

        return '"' & bitsString & '"';

    end function;

*-- convert t\_TEST to a descriptive name*

    function print\_test(

        test: t\_TEST

    ) return string is

    begin

        if (test.direction = RIGHT) then

            return "RIGHT\_" & integer'image(test.count);

        else

            return "LEFT\_" & integer'image(test.count);

        end if;

    end function;

*-------------------------------------------------------------------SIGNALS--*

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

    signal directionMode: std\_logic;

    signal mask: std\_logic\_vector(15 downto 0);

begin

    DRIVE\_INPUTS: process is

    begin

*-- initialize signals*

        countBits <= (others => '0');

        directionMode <= RIGHT;

*-- run all test cases*

        for i in ALL\_TESTS'range loop

            countBits <= std\_logic\_vector(to\_unsigned(ALL\_TESTS(i).count, 3));

            directionMode <= ALL\_TESTS(i).direction;

*-- alert if results do not match*

            wait for STEP\_TIME;

            assert (mask = ALL\_TESTS(i).result)

                report "FAILED TEST #" & integer'image(i)

                & " (" & print\_test(ALL\_TESTS(i)) & ")"

                & ". Expected " & print\_bits(ALL\_TESTS(i).result)

                & ", received " & print\_bits(mask);

        end loop;

        wait;

    end process;

    UUT: CenterPattern port map(

        countBits => countBits,

        directionMode => directionMode,

        mask => mask

    );

end CenterPattern\_TB\_ARCH;