Lab #2 Example, Center LEDs

Sean Graham

Kennesaw State University

CPE 3020: VHDL Design with FPGAs

Professor Scott Tippens

Spring 2025

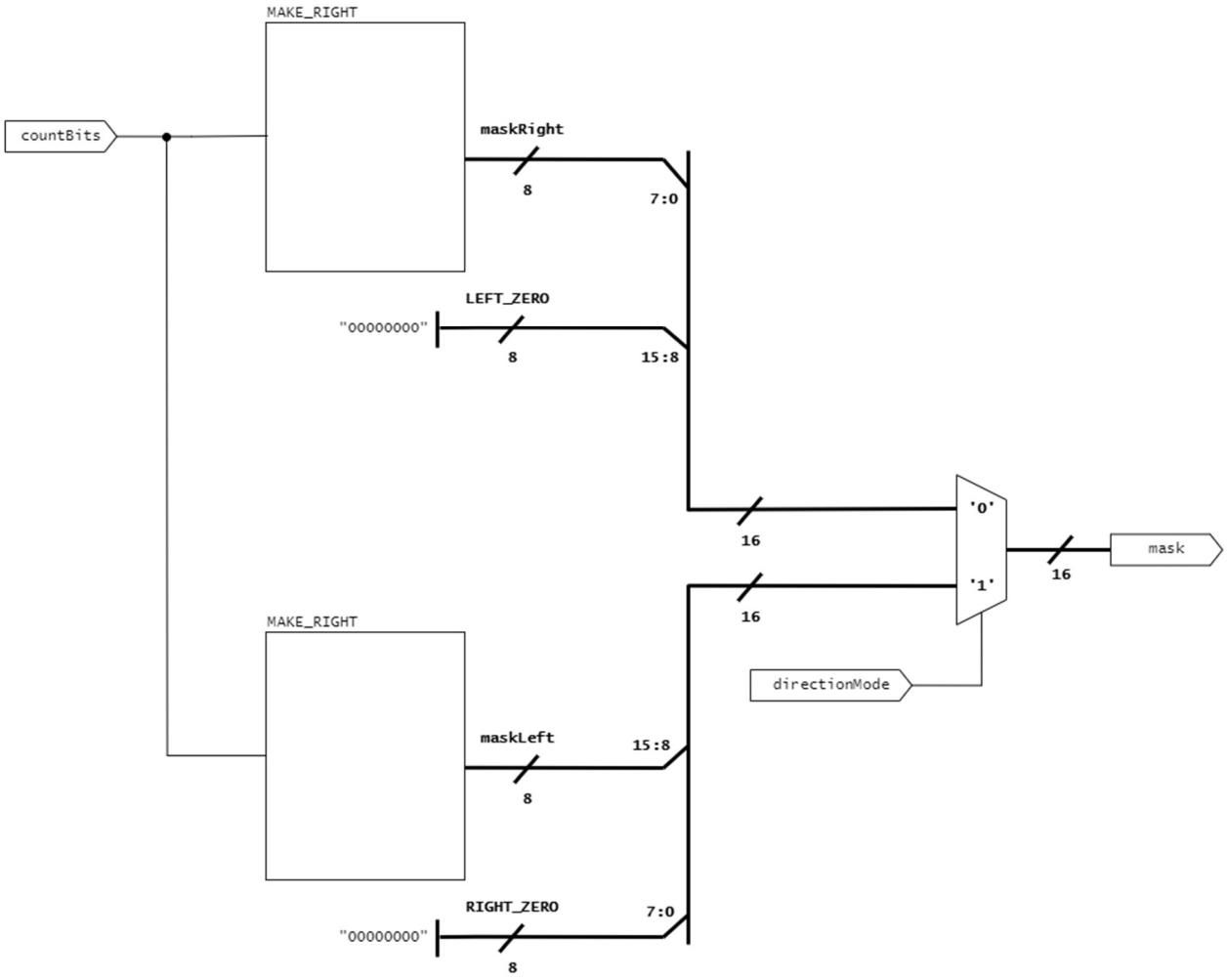

Fig. 1. Design diagram

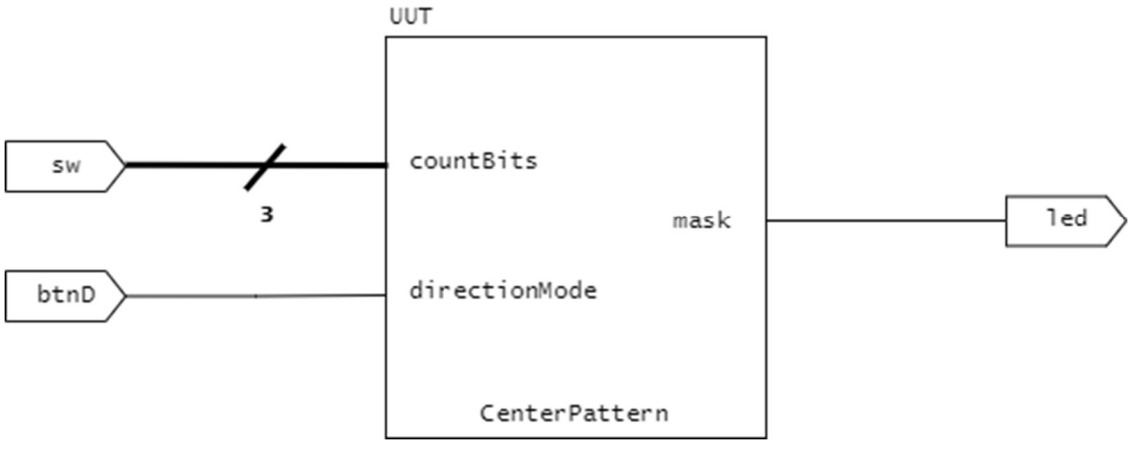

Fig. 2. Wrapper diagram

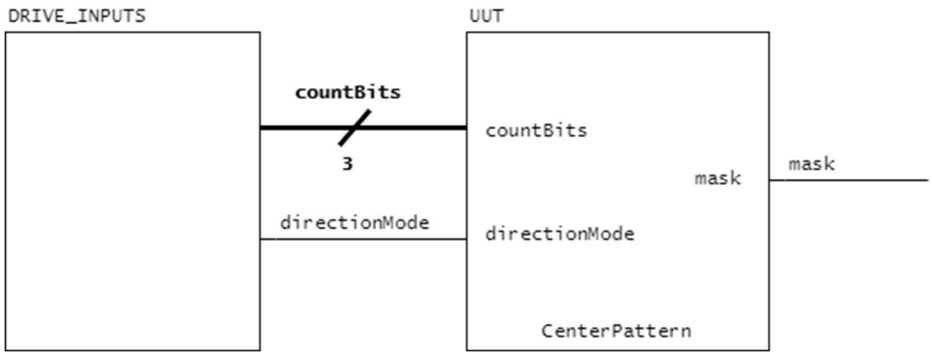

Fig. 3. Testbench diagram

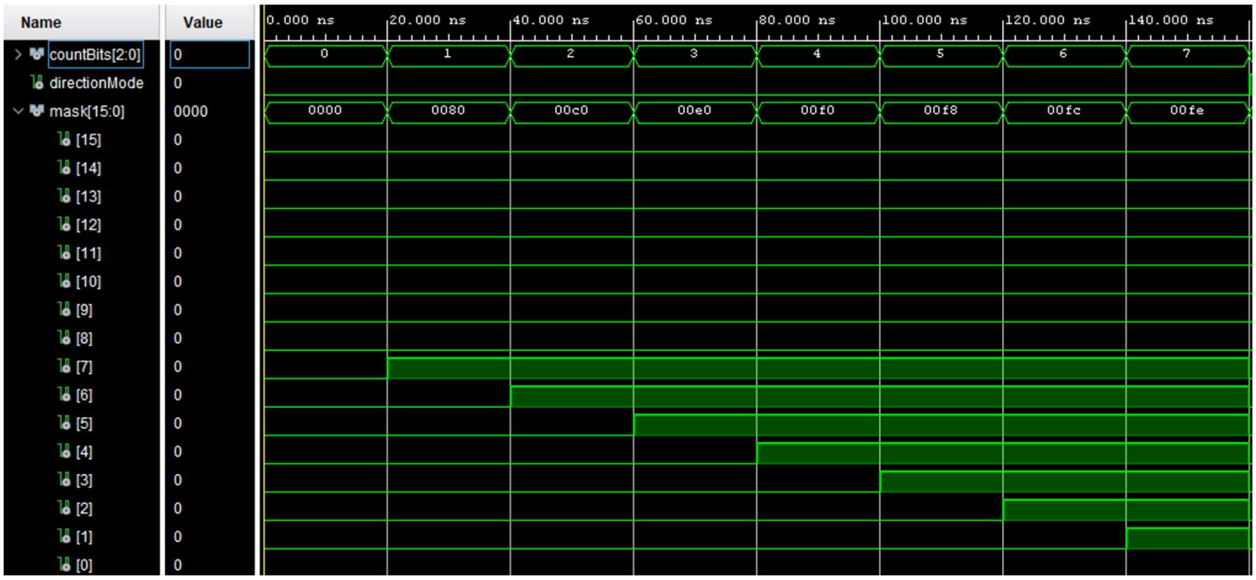

Fig. 4. Simulation results, right direction

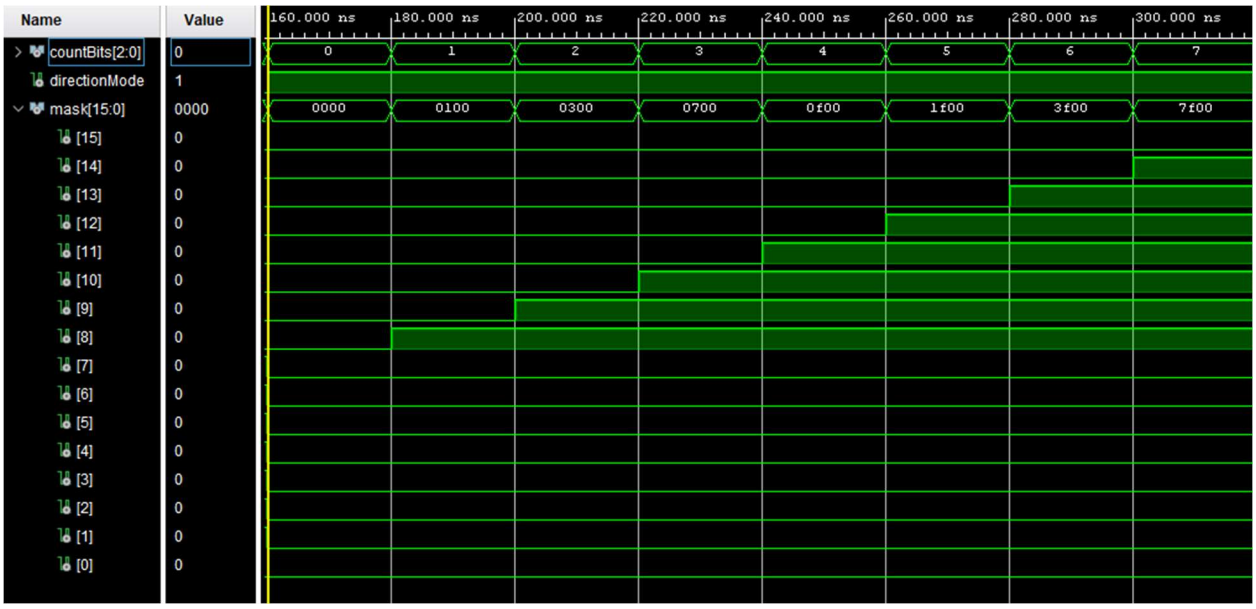

Fig. 5. Simulation results, left direction

```
--------------------------------------------------------------------------------
--
-- 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;
```