Lab #3 Example, Counter

Sean Graham

Kennesaw State University

CPE 3020: VHDL Design with FPGAs

Professor Scott Tippens

Spring 2025

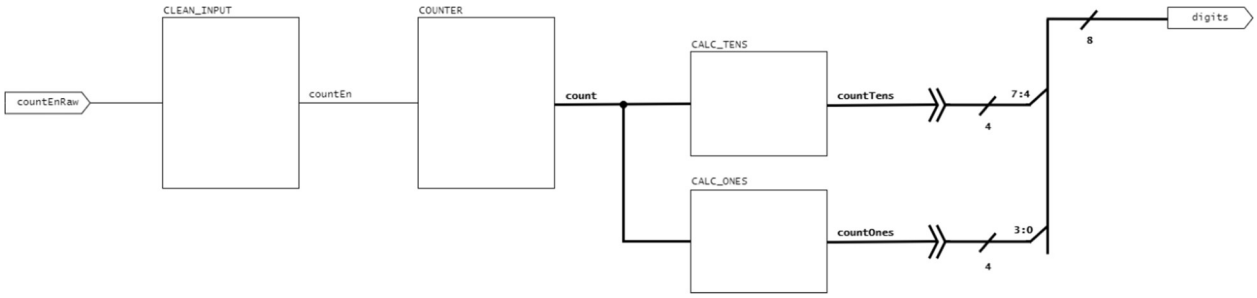

Fig. 1. Design diagram

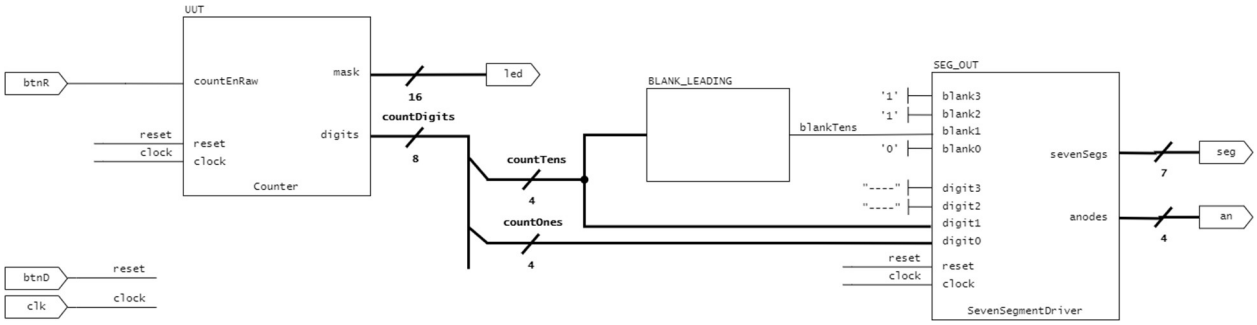

Fig. 2. Wrapper diagram

Note, SevenSegmentDriver is a component from the external package *physical_io_package*. Its diagrams and description have not been included.

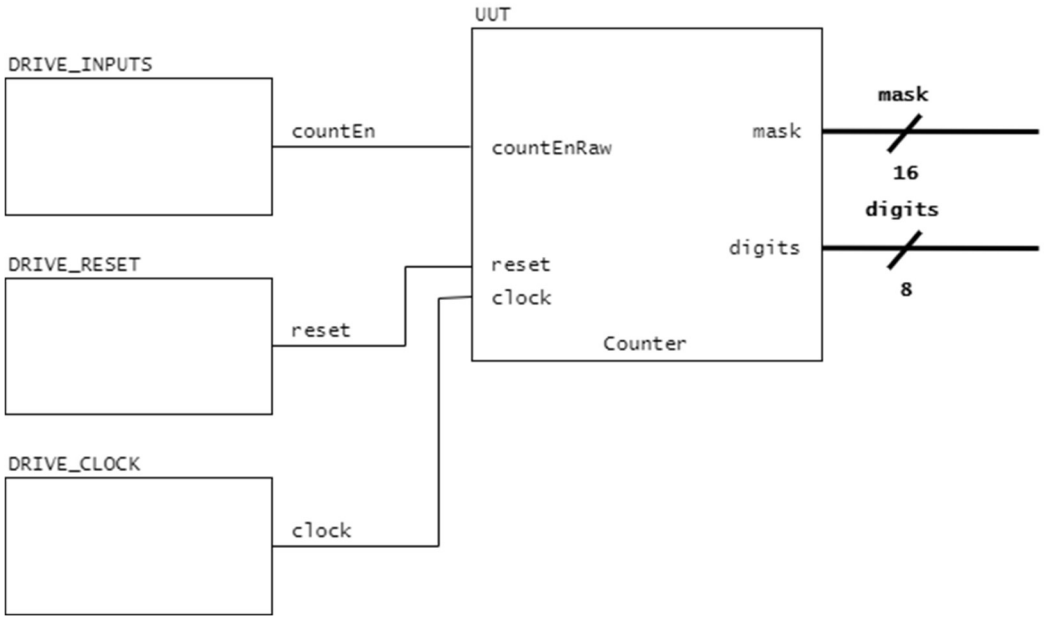

Fig. 3. Testbench diagram

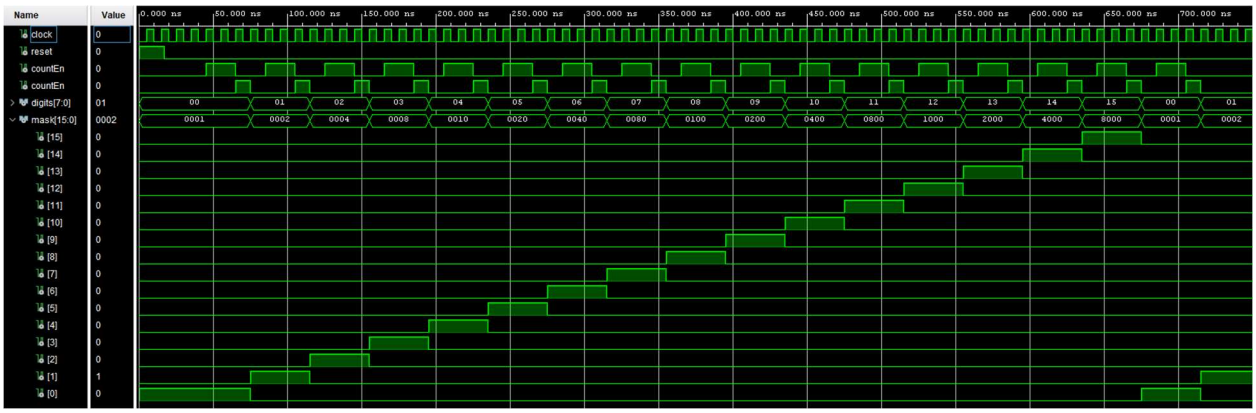

Fig. 4. Simulation results, right direction

```vhdl
--------------------------------------------------------------------------------
--
-- Lab 03 Demo: Counter
-- Sean Graham
--
--     Simple 4-bit synchronous counter. On reset, count is cleared to 0.
--     Counts to 15 on countEn then overflows to 0 again.
--
--     Count is output as an 8-bit BCD string, countBits.
--     A 16-bit mask with the corresponding bit high is also output.
--
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Counter is
    port(
        clock: in std_logic;
        reset: in std_logic;

        countEnRaw: in std_logic;

        digits: out std_logic_vector(7 downto 0); -- bcd tens (15:8), ones (7:0)
        mask: out std_logic_vector(15 downto 0)
    );
end Counter;

architecture Counter_ARCH of Counter is
    ------------------------------------------------------------------CONSTANTS--
    constant ACTIVE: std_logic := '1';

    -- when a low input is followed by a high, signal has just gone high
    constant FIRST_ACTIVE: std_logic_vector(1 downto 0) := (ACTIVE, not ACTIVE);

    -------------------------------------------------------------------SIGNALS--
    signal count: integer range 0 to 15;

    -- synchronized input
    signal countEn: std_logic;

    -- digits
    signal countTens: integer range 0 to 1;
    signal countOnes: integer range 0 to 9;
begin
    -- synchronize async button input and trigger on first cycle high
    -- note, physical_io_package has components for this
    CLEAN_INPUT: process (reset, clock) is
        -- in : countEnRaw
        -- out: countEn

        -- buffer of the last 4 inputs
        -- introduces a small delay for inputs to propogate
        variable inputs: std_logic_vector(3 downto 0);
    begin
        if (reset = ACTIVE) then
            -- on reset, saturate the buffer with low inputs
            countEn <= not ACTIVE;
            inputs := (others => not ACTIVE);
        elsif rising_edge(clock) then
            -- shift next input into buffer
            inputs := countEnRaw & inputs(3 downto 1);

            -- count only when the button is first pressed
            -- note, effectively shortens the synchronizer chain by 2
            if (inputs(1 downto 0) = FIRST_ACTIVE) then
                countEn <= ACTIVE;
            end if;
        end if;
    end process;
```

```
    -- store and update count
    MAKE_COUNT: process (reset, clock) is
        -- in : countEn
        -- out: count
    begin
        if (reset = ACTIVE) then
            -- reset to 0
            count <= 0;
        elsif rising_edge(clock) then
            -- increment on count
            if (countEn = ACTIVE) then
                if (count < 15) then
                    count <= count + 1;
                else
                    count <= 0;
                end if;
            end if;
        end if;
    end process;

    -- convert to decimal digits
    CALC_TENS: countTens <= (count / 10);
    CALC_ONES: countOnes <= (count mod 10);

    -- merge and convert to bcd
    BCD_TENS: digits(7 downto 4) <= std_logic_vector(to_unsigned(countTens, 4));
    BCD_ONES: digits(3 downto 0) <= std_logic_vector(to_unsigned(countOnes, 4));

    -- create mask pattern
    MAKE_MASK: process (count) is
    begin
        mask <= (others => not ACTIVE);
        mask(count) <= ACTIVE;
    end process;
end Counter_ARCH;
```

```
--------------------------------------------------------------------------------
--
-- Lab 03 Demo: Counter_BASYS3
-- Sean Graham
--
--     Wrapper implementing the Counter entity on the BASYS3 board.
--
--     When the right button is pressed, the counter increments.
--     When the down button is pressed, the counter is reset.
--
--     Current count is shown in decimal on the board's seven-segment display,
--       and the corresponding LED (count mod 16) is lit.
--
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

entity Counter_BASYS3 is
    port(
        clk: in std_logic;

        btnR: in std_logic;
        btnD: in std_logic;

        led: out std_logic_vector(15 downto 0);
        seg: out std_logic_vector(6 downto 0);
        an: out std_logic_vector(3 downto 0)
    );
end Counter_BASYS3;

architecture Counter_BASYS3_ARCH of Counter_BASYS3 is
    ------------------------------------------------------------------CONSTANTS--
    constant ACTIVE: std_logic := '1';

    -- bcd literals
    constant BCD_BLANK: std_logic_vector(3 downto 0) := "----";

    ------------------------------------------------------------------COMPONENTS--
    -- uut: counter
    component Counter is
        port(
            clock: in std_logic;
            reset: in std_logic;

            countEnRaw: in std_logic;

            digits: out std_logic_vector(7 downto 0); -- bcd tens (15:8), ones (7:0)
            mask: out std_logic_vector(15 downto 0)
        );
    end component;

    -- seven segment driver
    component SevenSegmentDriver is
        port(
            reset: in std_logic;
            clock: in std_logic;

            digit3: in std_logic_vector(3 downto 0);    --leftmost digit
            digit2: in std_logic_vector(3 downto 0);    --2nd from left digit
            digit1: in std_logic_vector(3 downto 0);    --3rd from left digit
            digit0: in std_logic_vector(3 downto 0);    --rightmost digit

            blank3: in std_logic;    --leftmost digit
            blank2: in std_logic;    --2nd from left digit
            blank1: in std_logic;    --3rd from left digit
            blank0: in std_logic;    --rightmost digit

            sevenSegs: out std_logic_vector(6 downto 0);    --MSB=g, LSB=a
            anodes:    out std_logic_vector(3 downto 0)    --MSB=leftmost digit
        );
    end component;
```

```vhdl
    ------------------------------------------------------------------SIGNALS--
    -- asynchronous inputs
    signal clock: std_logic;
    signal reset: std_logic;

    -- counter
    signal countEn: std_logic;
    signal countDigits: std_logic_vector(7 downto 0);

    -- bcd digits
    signal countTens: std_logic_vector(3 downto 0);
    signal countOnes: std_logic_vector(3 downto 0);

    -- seven seg
    signal blankTens: std_logic;
begin
    -- map async inputs
    clock <= clk;
    reset <= btnD;

    -- map counter component
    UUT: Counter port map(
        clock => clock,
        reset => reset,

        countEnRaw => btnR,

        digits => countDigits,
        mask => led -- drive led output
    );

    -- split BCD count into digits
    countTens <= countDigits(7 downto 4);
    countOnes <= countDigits(3 downto 0);

    -- blank leading digit when it is zero
    BLANK_LEADING: with countTens select
        blankTens <= (not ACTIVE) when "0000",
                     ACTIVE       when others;

    -- map seven segment component
    SEG_OUT: SevenSegmentDriver port map(
        reset => reset,
        clock => clock,

        digit3 => BCD_BLANK, -- don't care, will be blanked
        digit2 => BCD_BLANK, -- don't care, will be blanked
        digit1 => countTens,
        digit0 => countOnes,

        blank3 => ACTIVE, -- always blanked
        blank2 => ACTIVE, -- always blanked
        blank1 => blankTens,
        blank0 => (not ACTIVE), -- always visible

        sevenSegs => seg,
        anodes => an
    );
end Counter_BASYS3_ARCH;
```

```
--------------------------------------------------------------------------------
--
-- Lab 03 Demo: Counter_TB
-- Sean Graham
--
--     Testbench for the Counter entity.
--
--     Tests the first 16 states of the counter, at which point behavior is
--       expected to loop. For each check, increments the counter then checks
--       state.
--
--     Note, code written in the wrapper is *not* tested by the testbench.
--       so you want to keep as much logic as reasonable in the UUT.
--
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Counter_TB is
end Counter_TB;

architecture Counter_TB_ARCH of Counter_TB is
    ------------------------------------------------------------TYPE DEFINITIONS--
    -- grouped expected outputs of Counter
    type t_TEST is record
        digits: std_logic_vector(7 downto 0);
        mask: 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';

    constant CLOCK_PERIOD: time := 10 ns;
    constant STEP_TIME: time := 50 ns;

    -- tests
    constant ALL_TESTS: t_TEST_ARRAY(1 to 17) := (
        ( "00000000", "0000000000000001" ),
        ( "00000001", "0000000000000010" ),
        ( "00000010", "0000000000000100" ),
        ( "00000011", "0000000000001000" ),
        ( "00000100", "0000000000010000" ),
        ( "00000101", "0000000000100000" ),
        ( "00000110", "0000000001000000" ),
        ( "00000111", "0000000010000000" ),
        ( "00001000", "0000000100000000" ),
        ( "00001001", "0000001000000000" ),
        ( "00010000", "0000010000000000" ),
        ( "00010001", "0000100000000000" ),
        ( "00010010", "0001000000000000" ),
        ( "00010011", "0010000000000000" ),
        ( "00010100", "0100000000000000" ),
        ( "00010101", "1000000000000000" ),
        ( "00000000", "0000000000000001" )
    );

    ------------------------------------------------------------------COMPONENT--
    -- uut: counter
    component Counter is
        port(
            clock: in std_logic;
            reset: in std_logic;

            countEn: in std_logic;

            digits: out std_logic_vector(7 downto 0); -- bcd tens (15:8), ones (7:0)
            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);
        variable i: integer range 1 to bits'length;
    begin
        -- track position in string separately
        -- to show ranges in both directions correctly
        i = 1;

        -- set bits
        for j in bits'range loop
            -- note, std_logic'image takes the form "'x'", so extract 2nd char
            bitsString(i) := bitsString & std_logic'image(bits(j))(2);
            i := i + 1;
        end loop;

        -- return between quotes
        return '"' & bitsString & '"';
    end function;

    ------------------------------------------------------------------SIGNALS--
    signal clock: std_logic;
    signal reset: std_logic;

    signal countEn: std_logic;
    signal digits: std_logic_vector(7 downto 0);
    signal mask: std_logic_vector(15 downto 0);
begin
    DRIVE_RESET: process is
    begin
        reset <= ACTIVE;
        wait for 17 ns;

        reset <= not ACTIVE;
        wait;
    end process;

    DRIVE_CLOCK: process is
    begin
        clock <= not ACTIVE;
        wait for (CLOCK_PERIOD / 2);

        clock <= ACTIVE;
        wait for (CLOCK_PERIOD / 2);

        -- note, processes unterminated by a wait will repeat indefinitely
    end process;

    DRIVE_INPUTS: process is
    begin
        -- initalize signals
        countEn <= (not ACTIVE);

        -- wait until the first clock cycle after reset
        wait until (reset = not ACTIVE);
        wait until rising_edge(clock);

        -- wait a little longer
        wait for STEP_TIME;

        -- run all tests
        for i in ALL_TESTS'range loop
            -- alert if results do not match
            assert (digits = ALL_TESTS(i).digits)
                report "FAILED TEST #" & integer'image(i)
                & " (BCD). Expected " & print_bits(ALL_TESTS(i).digits)
```

```
                & ", received " & print_bits(digits);

            assert (mask = ALL_TESTS(i).mask)
                report "FAILED TEST #" & integer'image(i)
                & " (MASK). Expected " & print_bits(ALL_TESTS(i).mask)
                & ", received " & print_bits(mask);

            -- keep countEn active until next clock cycle to increment counter
            countEn <= ACTIVE;
            wait for CLOCK_PERIOD;

            -- keep countEn low for rest of step
            countEn <= (not ACTIVE);
            wait for (STEP_TIME - CLOCK_PERIOD);
        end loop;

        wait;
    end process;

    UUT: Counter port map(
        clock => clock,
        reset => reset,

        countEn => countEn,

        digits => digits,
        mask => mask
    );
end Counter_TB_ARCH;
```