**Lab #3 Example, Counter**

Sean Graham

Kennesaw State University

CPE 3020: VHDL Design with FPGAs

Professor Scott Tippens

Spring 2025

A black and white square with white squares

Description automatically generated  
Fig. 1. Design diagram

A white rectangle on a black background

Description automatically generated

Fig. 2. Wrapper diagram

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

A screenshot of a computer

Description automatically generated  
Fig. 3. Testbench diagram

A screen shot of a graph

Description automatically generated

Fig. 4. Simulation results, right direction

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

*--*

*-- 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;

*-------------------------------------------------------------------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;