Whack-an-LED
Final Project, CPE 133, Fall 2017
Arthur Knatt, Troy Knatt

Final Project: Whack-An-LED

Description:

This project is a unique rendition of the classic childhood game “Whack-A-Mole.” It uses switches and LEDs instead of a hammer and a mole. The SSEG display shows a timer, which counts down from 20. While the counter ticks down, one of the eight different LEDs will light up, and the user must flip the corresponding switch up before the LED moves, then down. The LEDs flash in random order. When the timer reaches zero, the 7-segment display will show how many “moles” were whacked, and the LEDs will all be lit. The reset button is used to begin the timer from 20 again, and the game will start fresh.

Inputs and Outputs:

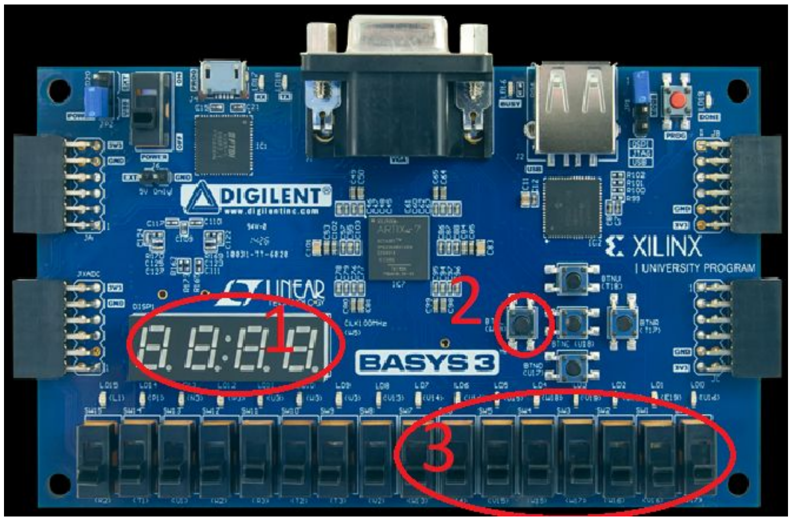

Figure 1: Annotated figure of Basys 3 board

The above figure 1 shows the three components of our board. This includes the seven segment display (1), the reset button (2), and the eight switches with accompanying LEDs (3). The display will begin at 20 seconds, and count down. During this time, the LEDs will flash one at a time, randomly, while the user is to switch the corresponding switch on and off. When the timer reaches 0, the display will then display the number of times the user “whacked” an LED, and the reset button may be used to start the timer over at 20, and play again.

1. Seven segment display	Displays a downward counting timer, from 20 to 0. After 0, it no longer displays a time. It then displays the point value the user earned, and awaits the press of the reset button. During this waiting state, all LEDs are lit.
2. Reset button	A press of this button will set the timer back to 20, no matter what it is currently at, reset the point value, and begin the LEDs randomly flashing again.
3. Switches and LEDs	A random LED will turn on for a set amount of time, and different LEDs will continue flashing one at a time until the timer reaches 0. The switches directly below the LEDs are the ones to be switches to signify a “whack.” There are eight switches and eight LEDs.

Table 1: Explains the inputs and outputs annotated in Figure 1

Whack-an-LED
Final Project, CPE 133, Fall 2017
Arthur Knatt, Troy Knatt

Structural Model:

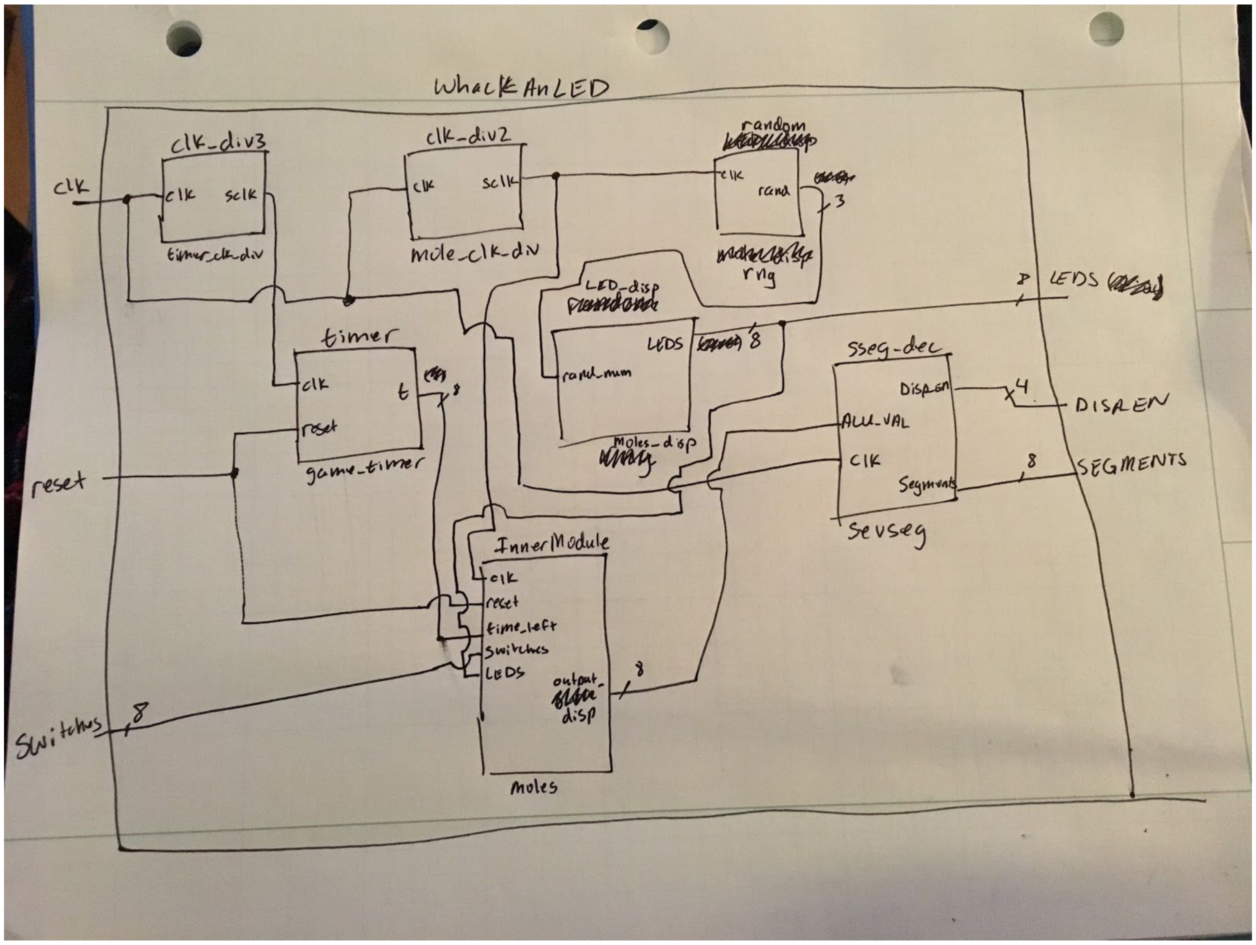

Figure 2: Structural Diagram for “Whack an LED”

Code:

```
----------------------------------------------------------------------------------
-- Engineer: Arthur Knatt, Troy Knatt
--
-- Create Date:    Fall 2017
-- Module Name:    WhackAnLED.vhd
-- Description: Top Level Module for our final project, Whack An LED
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity WhackAnLED is
```

```
    Port ( CLK      : in STD_LOGIC;
           RESET    : in  STD_LOGIC;
           SWITCHES : in  STD_LOGIC_VECTOR (7 downto 0);
           DISP_EN  : out  STD_LOGIC_VECTOR (3 downto 0);
           LEDS     : out  STD_LOGIC_VECTOR (7 downto 0);
           SEGMENTS : out  STD_LOGIC_VECTOR (7 downto 0));
end WhackAnLED;


architecture Behavioral of WhackAnLED is

  component clk_div2 is
     Port (  clk : in std_logic;
             sclk : out std_logic);
  end component;

  component clk_div3 is
     Port (  clk : in std_logic;
             sclk : out std_logic);
  end component;

  component timer is
     Port ( CLK       : in  STD_LOGIC;
            RESET     : in  STD_LOGIC;
            TIME_LEFT : out  STD_LOGIC_VECTOR (7 downto 0));
  end component;

  component sseg_dec is
    Port (      ALU_VAL : in std_logic_vector(7 downto 0);
                 SIGN : in std_logic;
                VALID : in std_logic;
                  CLK : in std_logic;
              DISP_EN : out std_logic_vector(3 downto 0);
             SEGMENTS : out std_logic_vector(7 downto 0));
  end component;

  component InnerModule is
     Port ( leds         : in  STD_LOGIC_VECTOR(7 downto 0);
            clk          : in  STD_LOGIC;
            reset        : in  STD_LOGIC;
            time_left    : in  STD_LOGIC_VECTOR (7 downto 0);
            switches     : in  STD_LOGIC_VECTOR (7 downto 0);
            score        : out  STD_LOGIC_VECTOR (7 downto 0));
  end component;

  component LED_disp is
    Port ( rn : in  STD_LOGIC_VECTOR (2 downto 0);
        LEDS     : out  STD_LOGIC_VECTOR (7 downto 0);
        time_left: in STD_LOGIC_VECTOR (7 downto 0));
  end component;

  component random is
    generic ( width : integer :=  3 );
    Port (    clk        : in std_logic;
          rand_num : out std_logic_vector (width-1 downto 0)
    );
  end component;

  signal randNum : std_logic_vector(2 downto 0);
  signal moles_clock, timer_clock : std_logic;
  signal l1, the_score, t1 : std_logic_vector(7 downto 0);

begin

  mole_clk_div : clk_div2
  port map ( clk  => CLK,
         sclk => moles_clock
        );

  rng : random
  port map (  clk        => moles_clock,
          rand_num => randNum
        );

  mole_disp : LED_disp
  port map ( rn => randNum,
         LEDS     => l1,
         time_left => t1
        );

  timer_clk_div : clk_div3
  port map ( clk  => CLK,
         sclk => timer_clock
        );

  game_timer : timer
port map ( CLK       => timer_clock,
           reset     => reset,
           time_left => t1
         );

moles : InnerModule
port map ( LEDS      => l1,
           reset     => reset,
           CLK       => moles_clock,
           time_left => t1,
           switches  => switches,
           score     => the_score
         );

sevseg : sseg_dec
port map ( ALU_VAL  => the_score,
           SIGN     => '0',
           VALID    => '1',
           CLK      => CLK,
           DISP_EN  => DISP_EN,
           SEGMENTS => SEGMENTS
         );

LEDS <= l1;

end Behavioral;


----------------------------------------------------------------------------------
-- Engineer: Arthur Knatt, Troy Knatt
--
-- Create Date:    Fall 2017
-- Module Name:    clk_div2.vhd
-- Description: clock divider module that lets our mole LED's pop up almost
--              twice per second
----------------------------------------------------------------------------------

-----------------------------------------------------------------------
-----------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
-----------------------------------------------------------------------
-- Module to divide the clock
-----------------------------------------------------------------------
entity clk_div2 is
    Port (  clk : in std_logic;
           sclk : out std_logic);
end clk_div2;

architecture my_clk_div of clk_div2 is
   constant max_count : integer := (35000000);
   signal tmp_clk : std_logic := '0';
begin
   my_div: process (clk,tmp_clk)
      variable div_cnt : integer := 0;
   begin
      if (rising_edge(clk)) then
         if (div_cnt = MAX_COUNT) then
            tmp_clk <= not tmp_clk;
            div_cnt := 0;
         else
            div_cnt := div_cnt + 1;
         end if;
      end if;
      sclk <= tmp_clk;
   end process my_div;
end my_clk_div;



-- random number generator retrieved from "linuxlizard":
-- https://github.com/linuxlizard/vhdl/blob/master/random.vhdl
-- http://vhdlguru.blogspot.com/2010/03/random-number-generator-in-vhdl.html
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity random is
    generic ( width : integer :=  32 );
port (
      clk : in std_logic;
      rand_num : out std_logic_vector (width-1 downto 0)   --output vector
    );
end random;
```

```vhdl
architecture Behavioral of random is
begin
   process(clk)
      variable rand_temp : std_logic_vector(width-1 downto 0):=(others=>'1');
--       variable rand_temp : std_logic_vector(width-1 downto 0):=(width-1 => '1',others => '0');
      variable temp : std_logic := '0';
   begin
      if(rising_edge(clk)) then
         temp := rand_temp(width-1) xor rand_temp(width-2);
         rand_temp(width-1 downto 1) := rand_temp(width-2 downto 0);
         rand_temp(0) := temp;
      end if;
      rand_num <= rand_temp(2 downto 0);
   end process;

end architecture Behavioral;




----------------------------------------------------------------------------------
-- Engineer: Arthur Knatt, Troy Knatt
--
-- Create Date:    Fall 2017
-- Module Name:    LED_disp.vhd
-- Description: module that drives the LED's above the switches. when the game
-- is not running, all 8 of the LED's are on. While the game is running, the
-- random number generator decides which LED will be on
--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity LED_disp is
   Port ( rn : in  STD_LOGIC_VECTOR (2 downto 0);
          LEDS     : out  STD_LOGIC_VECTOR (7 downto 0);
          time_left: in STD_LOGIC_VECTOR (7 downto 0)
);
end LED_disp;

architecture Behavioral of LED_disp is

begin
            -- sets all 8 LEDs on when game is not in play
```

```
    LEDS <= "11111111" when time_left = "00000000" else
         "00000001" when rn = "000" else
         "00000010" when rn = "001" else
         "00000100" when rn = "010" else
         "00001000" when rn = "011" else
         "00010000" when rn = "100" else
         "00100000" when rn = "101" else
         "01000000" when rn = "110" else
         "10000000" when rn = "111";
        -- The first line means no lights will be on when
        -- the timer says "0"
end Behavioral;

----------------------------------------------------------------------------------
-- Engineer: Arthur Knatt, Troy Knatt
--
-- Create Date:    Fall 2017
-- Module Name:    clk_div3.vhd
-- Description: clock divider module that allows our timer to count down
--              by seconds
----------------------------------------------------------------------------------


-----------------------------------------------------------------------
-----------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
-----------------------------------------------------------------------
-- Module to divide the clock
-----------------------------------------------------------------------
entity clk_div3 is
   Port (  clk : in std_logic;
           sclk : out std_logic);
end clk_div3;

architecture my_clk_div of clk_div3 is
  constant max_count : integer := (50000000);
  signal tmp_clk : std_logic := '0';
begin
  my_div: process (clk,tmp_clk)
    variable div_cnt : integer := 0;
  begin
```

```vhdl
    if (rising_edge(clk)) then
      if (div_cnt = MAX_COUNT) then
        tmp_clk <= not tmp_clk;
        div_cnt := 0;
      else
        div_cnt := div_cnt + 1;
      end if;
    end if;
    sclk <= tmp_clk;
  end process my_div;
end my_clk_div;

----------------------------------------------------------------------------------
-- Engineer: Arthur Knatt, Troy Knatt
--
-- Create Date:    Fall 2017
-- Module Name:    timer.vhd
-- Description: Module that counts down from 20, giving players enough time to
--              whack some LED's
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity timer is
   Port ( clk       : in  STD_LOGIC;
          reset     : in  STD_LOGIC;
          time_left : out  STD_LOGIC_VECTOR (7 downto 0));
end timer;

architecture Behavioral of timer is
   -- initialize the timer to 20 seconds
  signal t : STD_LOGIC_VECTOR(7 downto 0) := "00010100";

begin
  timer: process (clk, reset, t)
  begin
    if (t > "00000000") then
      if (rising_edge(CLK)) then
        t <= t - 1;
      end if;
    end if;
```

```
      if (RESET = '1') then
          -- resets timer to 20 secs
        t <= "00010100";
      end if;

    end process timer;

    time_left <= t;

end Behavioral;

----------------------------------------------------------------------------------
-- Engineer: Arthur Knatt, Troy Knatt
--
-- Create Date:    Fall 2017
-- Module Name:    InnerModule.vhd
-- Description: Inner module that keeps track of the score, and tells
-- the 7-seg display what to show. Either the score or the timer
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity InnerModule is
    Port ( leds          : in  STD_LOGIC_VECTOR(7 downto 0);
           reset         : in  STD_LOGIC;
           clk           : in  STD_LOGIC;
           time_left     : in  STD_LOGIC_VECTOR (7 downto 0);
           switches      : in  STD_LOGIC_VECTOR (7 downto 0);
           output_disp   : out  STD_LOGIC_VECTOR (7 downto 0));
end InnerModule;

architecture Behavioral of InnerModule is
    signal score : STD_LOGIC_VECTOR(7 downto 0)  := "00000000";
begin

    count: process (LEDS, reset, switches, clk)
    begin

      if (rising_edge(clk)) then
        if (time_left > "00000000") then
          if (LEDS = switches) then
            score <= score + 1;
          end if;
          output_disp <= TIME_LEFT;
        else
          output_disp <= score;
        end if;
      end if;

      if (RESET = '1') then
        score <= "00000000";
      end if;

    end process count;

end Behavioral;

-----------------------------------------------------------------------
-- CPE 133 VHDL File: sseg_dec.vhd
-- Description: Special seven segment display driver;
--
--  two special inputs:
--
--      VALID: if valid = 0, four dashes will be display
--             if valid = 1, decimal number appears on display
--
--       SIGN: if sign = 1, a minus sign appears in left-most digit
--             if sign = 0, no minus sign appears
--
--
-- Author: bryan mealy (12-16-10)
--
-- revisions:
-----------------------------------------------------------------------

----------------------------------------------------------------------
----------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-------------------------------------------------------------
-- 4 digit seven-segment display driver. Outputs are active
-- low and configured ABCEDFG in "segment" output.
```

```
-----------------------------------------------------------------
entity sseg_dec is
    Port (      ALU_VAL : in std_logic_vector(7 downto 0);
                   SIGN : in std_logic;
                  VALID : in std_logic;
                    CLK : in std_logic;
                DISP_EN : out std_logic_vector(3 downto 0);
               SEGMENTS : out std_logic_vector(7 downto 0));
end sseg_dec;

---------------------------------------------------------------
-- description of ssegment decoder
---------------------------------------------------------------
architecture my_sseg of sseg_dec is

   -- declaration of 8-bit binary to 2-digit BCD converter --
   component bin2bcdconv
      Port ( BIN_CNT_IN : in std_logic_vector(7 downto 0);
                LSD_OUT : out std_logic_vector(3 downto 0);
                MSD_OUT : out std_logic_vector(3 downto 0);
               MMSD_OUT : out std_logic_vector(3 downto 0));
   end component;

   component clk_div
      Port (  clk : in std_logic;
             sclk : out std_logic);
   end component;

   -- intermediate signal declaration -----------------------
   signal   cnt_dig : std_logic_vector(1 downto 0);
   signal   digit : std_logic_vector (3 downto 0);
   signal   lsd,msd,mmsd : std_logic_vector(3 downto 0);
   signal   sclk : std_logic;

begin

   -- instantiation of bin to bcd converter -----------------
   my_conv: bin2bcdconv
   port map ( BIN_CNT_IN => ALU_VAL,
                 LSD_OUT => lsd,
                 MSD_OUT => msd,
                MMSD_OUT => mmsd);
```

```vhdl
my_clk: clk_div
port map (clk => clk,
       sclk => sclk );

-- advance the count (used for display multiplexing) -----
process (SCLK)
begin
  if (rising_edge(SCLK)) then
    cnt_dig <= cnt_dig + 1;
  end if;
end process;


-- select the display sseg data abcdefg (active low) -----
segments <= "00000011" when digit = "0000"  else
        "10011111" when digit = "0001"  else
        "00100101" when digit = "0010"  else
        "00001101" when digit = "0011"  else
        "10011001" when digit = "0100"  else
        "01001001" when digit = "0101"  else
        "01000001" when digit = "0110"  else
        "00011111" when digit = "0111"  else
        "00000001" when digit = "1000"  else
        "00001001" when digit = "1001"  else
        "11111101" when digit = "1110" else   -- dash
        "11111111" when digit = "1110" else   -- blank
        "11111111";

-- actuate the correct display --------------------------
disp_en <= "1110" when cnt_dig = "00" else
        "1101" when cnt_dig = "01" else
        "1011" when cnt_dig = "10" else
        "0111" when cnt_dig = "11" else
        "1111";


process (cnt_dig, lsd, msd, mmsd, sign, valid)
  variable mmsd_v, msd_v : std_logic_vector(3 downto 0);
begin
  mmsd_v := mmsd;
  msd_v := msd;

  -- do the lead zero blanking for two msb's
```

```
    if (mmsd_v = X"0") then
      if (msd_v = X"0") then
        msd_v := X"F";
      end if;
      mmsd_v := X"F";
    end if;

    if (valid = '1') then
      if (sign = '0') then
        case cnt_dig is
          when "00" => digit <= "1111";
          when "01" => digit <= mmsd_v;
          when "10" => digit <= msd_v;
          when "11" => digit <= lsd;
          when others => digit <= "0000";
        end case;
      else
        case cnt_dig is
          when "00" => digit <= "1110";
          when "01" => digit <= mmsd_v;
          when "10" => digit <= msd_v;
          when "11" => digit <= lsd;
          when others => digit <= "0000";
        end case;
      end if;
    else digit <= "1110";
    end if;
  end process;


end my_sseg;




-----------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
-----------------------------------------------------------------------
-- interface description for bin to bcd converter
-----------------------------------------------------------------------
```

```
entity bin2bcdconv is
   Port ( BIN_CNT_IN : in std_logic_vector(7 downto 0);
          LSD_OUT : out std_logic_vector(3 downto 0);
          MSD_OUT : out std_logic_vector(3 downto 0);
         MMSD_OUT : out std_logic_vector(3 downto 0));
end bin2bcdconv;

---------------------------------------------------------------------
-- description of 8-bit binary to 3-digit BCD converter
---------------------------------------------------------------------
architecture my_ckt of bin2bcdconv is
begin
  process(bin_cnt_in)
     variable cnt_tot : INTEGER range 0 to 255 := 0;
     variable lsd,msd,mmsd : INTEGER range 0 to 9 := 0;
  begin

   -- convert input binary value to decimal
   cnt_tot := 0;
   if (bin_cnt_in(7) = '1') then cnt_tot := cnt_tot + 128;  end if;
   if (bin_cnt_in(6) = '1') then cnt_tot := cnt_tot +  64;  end if;
   if (bin_cnt_in(5) = '1') then cnt_tot := cnt_tot +  32;  end if;
   if (bin_cnt_in(4) = '1') then cnt_tot := cnt_tot +  16;  end if;
   if (bin_cnt_in(3) = '1') then cnt_tot := cnt_tot +   8;  end if;
   if (bin_cnt_in(2) = '1') then cnt_tot := cnt_tot +   4;  end if;
   if (bin_cnt_in(1) = '1') then cnt_tot := cnt_tot +   2;  end if;
   if (bin_cnt_in(0) = '1') then cnt_tot := cnt_tot +   1;  end if;

   -- initialize intermediate signals
   msd  := 0;
   mmsd := 0;
   lsd  := 0;

   --  calculate the MMSB
   for I in 1 to 2 loop
      exit when (cnt_tot >= 0 and cnt_tot < 100);
      mmsd := mmsd + 1; -- increment the mmds count
      cnt_tot := cnt_tot - 100;
   end loop;

    --  calculate the MSB
   for I in 1 to 9 loop
      exit when (cnt_tot >= 0 and cnt_tot < 10);
   msd := msd + 1; -- increment the mds count
   cnt_tot := cnt_tot - 10;
end loop;

lsd := cnt_tot;   -- lsd is what is left over

-- convert lsd to binary
case lsd is
   when 9 =>  lsd_out <= "1001";
   when 8 =>  lsd_out <= "1000";
   when 7 =>  lsd_out <= "0111";
   when 6 =>  lsd_out <= "0110";
   when 5 =>  lsd_out <= "0101";
   when 4 =>  lsd_out <= "0100";
   when 3 =>  lsd_out <= "0011";
   when 2 =>  lsd_out <= "0010";
   when 1 =>  lsd_out <= "0001";
   when 0 =>  lsd_out <= "0000";
   when others =>  lsd_out <= "0000";
end case;

-- convert msd to binary
case msd is
   when 9 =>  msd_out <= "1001";
   when 8 =>  msd_out <= "1000";
   when 7 =>  msd_out <= "0111";
   when 6 =>  msd_out <= "0110";
   when 5 =>  msd_out <= "0101";
   when 4 =>  msd_out <= "0100";
   when 3 =>  msd_out <= "0011";
   when 2 =>  msd_out <= "0010";
   when 1 =>  msd_out <= "0001";
   when 0 =>  msd_out <= "0000";
   when others =>  msd_out <= "0000";
  end case;

-- convert msd to binary
case mmsd is
   when 2 =>  mmsd_out <= "0010";
   when 1 =>  mmsd_out <= "0001";
   when 0 =>  mmsd_out <= "0000";
   when others =>  mmsd_out <= "0000";
end case;
```

```
  end process;
end my_ckt;



-----------------------------------------------------------------------
-----------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
-----------------------------------------------------------------------
-- Module to divide the clock
-----------------------------------------------------------------------
entity clk_div is
   Port (  clk : in std_logic;
           sclk : out std_logic);
end clk_div;

architecture my_clk_div of clk_div is
  constant max_count : integer := (2200);
  signal tmp_clk : std_logic := '0';
begin
  my_div: process (clk,tmp_clk)
    variable div_cnt : integer := 0;
  begin
    if (rising_edge(clk)) then
      if (div_cnt = MAX_COUNT) then
        tmp_clk <= not tmp_clk;
        div_cnt := 0;
      else
        div_cnt := div_cnt + 1;
      end if;
    end if;
    sclk <= tmp_clk;
  end process my_div;
end my_clk_div;
```