Part A (20 points). Paste the VHDL for your modified calculator below. Highlight your modifications to the code by making them **bold**. Note that the next paragraph is formatted using the “code style” which uses a fixed-width font and has appropriately spaced tabs. Please always use this paragraph style for your VHDL code. Also, be sure to format your code so that lines do not wrap-around in the lab report. Points will be deducted for code that is badly formatted or difficult to read.

---

-- Simple Binary Calculator
-- Jon Turner, 12/2007
-- modified 5/2010 for newer prototype boards
--
-- This circuit implements a simple binary calculator with three
-- operations
--
-- clear stored value
-- enter new value
-- add to stored value
--
-- The input data and the result are both 16 bit values.

-- Modified 12/2013 by Jon Turner
--
-- This version also includes a mode input and an error output.
-- The calculator's internal error bit is set if an addition operation
-- causes an arithmetic error. When mode=0, the addition is treated as
-- unsigned addition. When mode=1, the addition is treated as signed.
---

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.commonDefs.all;

entity calculator is port ( 
    clk: in std_logic; -- signals to enable operations
    clear, load, add: in std_logic;  -- 0 for unsigned, 1 for signed
    mode: in std_logic; 
    dIn: in word;  -- input data
    result: out word; -- output result
    error: out std_logic); -- high when error detected
end calculator;

architecture a1 of calculator is 
signal dReg: word;
signal sum: word;
signal errBit: std_logic;
begin
  sum <= dIn + dReg;
  error <= errBit;
  process (clk) begin
    if rising_edge(clk) then
      if clear = '1' then
        dReg <= (wordsize-1 downto 0 => '0');
        errBit <= '0';
      elsif errBit = '0' then
        if load = '1' then
          dReg <= dIn;
        elsif add = '1' then
          dReg <= sum;
        end if;
      end if;
      end if;
    end process;
    result <= dReg;
  end process;
end a1;
**Part B.** (10 points) Draw a block diagram of the calculator including your modifications to it. Include a block labeled *ErrorDetector* that represents the logic that determines if an error has occurred. Your diagram need not show how this block is implemented, but it should show all the signals that connect to it. Your diagram should also include a flip flop top hold the error bit. It’s ok to submit a hand-drawn figure on the printed copy only, but please, make it neat, well-organized and legible.
Part C. (20 points). Modify the testCalculator testbench to include additional tests to verify the error detection feature added to the calculator. You must verify that it detects errors for both values of the mode input, that error bit is cleared when a clear operation is performed and that other operations are blocked when the error bit is set. Highlight the tests you added by making them bold.

-- Testbench for calculator module
-- Jon Turner, 12/2007
-- Modified 12/2013, Jon Turner
-- Added tests to verify modifications to the calculator circuit that
-- check for arithmetic errors.
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;
use work.commonDefs.all;

entity testCalculator is
end testCalculator;

architecture a1 of testCalculator is
component calculator port(
  clk : in std_logic;
  clear, load, add : in std_logic;
  mode: in std_logic;
  dIn : in word;
  result : out word;
  error: out std_logic);
end component;

signal clk :  std_logic := '0';
signal clear :  std_logic := '0';
signal load :  std_logic := '0';
signal add :  std_logic := '0';
signal mode :  std_logic := '0';
signal dIn :  word := (others=>'0');
signal result :  word;
signal error: std_logic;
begin
  -- create instance of calculator circuit
  uut: calculator port map(
    clk => clk, clear => clear, load => load,
    add => add, mode => mode, dIn => dIn, result => result, error => error
  );

  process begin -- clock process for clk
    clk_loop : loop
      clk <= '0'; wait for 10 ns;
      clk <= '1'; wait for 10 ns;
    end loop clk_loop;
  end process;

  tb : process begin -- test inputs
mode <= '0';
clear <= '1'; load <= '1'; add <= '1'; dIn <= x"ffff"; wait for 20 ns;
clear <= '0'; load <= '1'; add <= '0'; dIn <= x"ffff"; wait for 20 ns;
clear <= '0'; load <= '1'; add <= '1'; dIn <= x"ffff"; wait for 20 ns;
clear <= '0'; load <= '0'; add <= '1'; dIn <= x"0001"; wait for 20 ns;
clear <= '0'; load <= '1'; add <= '1'; dIn <= x"0001"; wait for 20 ns;
clear <= '1'; load <= '0'; add <= '0'; dIn <= x"0001"; wait for 20 ns;
clear <= '0'; load <= '0'; add <= '1'; dIn <= x"0002"; wait for 20 ns;
clear <= '0'; load <= '0'; add <= '1'; dIn <= x"0003"; wait for 20 ns;
clear <= '0'; load <= '0'; add <= '1'; dIn <= x"0004"; wait for 20 ns;
clear <= '0'; load <= '0'; add <= '1'; dIn <= x"0100"; wait for 20 ns;
clear <= '0'; load <= '0'; add <= '1'; dIn <= x"0200"; wait for 20 ns;
clear <= '0'; load <= '0'; add <= '1'; dIn <= x"0300"; wait for 20 ns;

mode <= '1';
clear <= '0'; load <= '1'; add <= '0'; dIn <= x"ffff"; wait for 20 ns;
clear <= '0'; load <= '0'; add <= '1'; dIn <= x"0001"; wait for 20 ns;
clear <= '0'; load <= '0'; add <= '1'; dIn <= x"0001"; wait for 20 ns;
clear <= '0'; load <= '0'; add <= '1'; dIn <= x"7fff"; wait for 20 ns;
clear <= '0'; load <= '0'; add <= '1'; dIn <= x"0001"; wait for 20 ns;
clear <= '0'; load <= '0'; add <= '1'; dIn <= x"0001"; wait for 20 ns;
clear <= '1'; load <= '0'; add <= '0'; dIn <= x"0001"; wait for 20 ns;
wait for 20 ns;
assert (false) report "Simulation ended normally." severity failure;
end process;
end a1;
Run the simulator using the `testCalculator` testbench and paste a screenshot of the waveform window below. Set the radix to hexadecimal for the `dIn` and `result` signals. Note that by default, the simulator displays signals in white on a black background. This works poorly when printed. You can change this using the Preferences item in the Edit menu of the waveform window. Please make the waveform area white and make the signal colors and all text black. You should only have to do this one time. After that, your preferences will be remembered. If your screenshot extends over a longer time period than 400 ns, split it into two screenshots and paste them one after the other. That is, first paste a screenshot that covers the first 400 ns of your simulation run, then paste a second screenshot for the next 400 ns. Make sure that all text is clearly legible on the printed copy. If not reduce the time period covered by each screenshot until it is legible.

Find the place in your simulation where `mode`=0 and `error` goes high for the first time. What are the values of the two operands at this moment? Explain why these input values should trigger an arithmetic error.

The operand values are `dReg=ffff` and `dIn=0001`. Since `ffff` is the largest value that can be represented as a 16-bit unsigned value, adding 1 to it produces an arithmetic error.

Find the place in your simulation where `mode`=1 and `error` goes high for the first time. What are the values of the two operands at this moment? Explain why these input values should trigger an arithmetic error.

The operand values are `dReg=0001` and `dIn=7fff`. Since `7fff` is the largest positive value that can be represented in a 16-bit 2's-complement system, adding 1 to it results in an arithmetic error.
Part D. (20 points) Paste a copy of the modified version of top below. Also, a copy of the modified binaryOutMod.

-- Top module for simple calculator on S3 board
-- Jon Turner 8/2010
-- This version uses just the buttons and switches plus the LEDs.
-- This version modified for arithmetic error detection by calculator
-- Modified 12/2013, Jon Turner
-- Modified to work with new version of calculator circuit that
-- supports error detection. Added error and mode signals to
-- calculator and error signal to output module.

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.commonDefs.all;

entity top is port(
  clk: in std_logic;
  -- S3 board buttons, knob, switches and LEDs
  btn: in buttons;
  knob: in knobSigs;
  swt: in switches;
  led: out leds;
  -- signals for controlling LCD display
  lcd: out lcdSigs);
end top;

architecture a1 of top is

component calculator port ( 
  clk: in std_logic;
  clear, load, add: in std_logic;
  mode: in std_logic;
  din : in word;
  result: out word;
  error: out std_logic);
end component;

component binaryInMod port(
  clk: in std_logic;
  btn: in buttons;
  knob: in knobSigs;
  resetOut: out std_logic;
  dBtn: out std_logic_vector(3 downto 1);
  pulse: out std_logic_vector(3 downto 1);
  inBits: out word);
end component;

component binaryOutMod port(
  clk, reset: in std_logic;
  error: in std_logic;
  topRow, botRow: in word;
)
lcd: out lcdSigs);
end component binaryOutMod;

signal reset, clear, load, add: std_logic;
signal dBn, pulse: std_logic_vector(3 downto 1);
signal inBits, outBits: word;
signal error, mode: std_logic;

begin

    -- connect the sub-components
    imod: binaryInMod port map(clk,btn,knob,reset,dBtn,pulse,inBits);
calc: calculator port map(clk,clear,load,add,mode,inBits,outBits,error);
omod: binaryOutMod port map(clk,reset,error,inBits,outBits,lcd);

    -- define internal control signals
    clear <= dBtn(1) or reset;
    load <= pulse(2);
    add <= pulse(3);
    mode <= swt(0);

    led(0) <= mode; led(1) <= error;
    led(7 downto 2) <= "000000";
end a1;
The modified version of `binaryOutMod` appears below

---
-- Simple binary output module
-- Jon Turner - 10/2011
--
-- The two data inputs topRow and botRow are displayed on the S3 board's LCD
-- display in binary, with the high order bit at the left.
--
-- This version includes an error input. When the error input is high,
-- the bottom row displays dashes
---

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;
use work.commonDefs.all;

entity binaryOutMod is port(
    clk, reset: in std_logic;
    error: in std_logic;
    topRow, botRow: in word;
    -- signals for controlling LCD display
    lcd: out lcdSigs);
end entity binaryOutMod;

architecture a1 of binaryOutMod is

component lcdDisplay port(
    clk, reset : in std_logic;
    -- internal interface for controlling display
    update: in std_logic; -- update a stored value
    selekt: in std_logic_vector(4 downto 0); -- character to replace
    nuChar: in std_logic_vector(7 downto 0); -- new character value
    -- connections to external pins
    lcd: out lcdSigs);
end component;

-- counter for controlling when to update lcdDisplay
constant CNTR_LENGTH: integer := 20;
signal counter: std_logic_vector(CNTR_LENGTH-1 downto 0);
signal lowBits: std_logic_vector(CNTR_LENGTH-6 downto 0);

-- signals for controlling lcdDisplay
signal update: std_logic;
signal selekt: std_logic_vector(4 downto 0);
signal nuChar: std_logic_vector(7 downto 0);

begin
    disp: lcdDisplay port map(clk, reset, update, selekt, nuChar, lcd);

    lowBits <= counter(CNTR_LENGTH-6 downto 0);
    update <= '1' when lowBits = (lowBits'range => '0') else '0';
    selekt <= counter(CNTR_LENGTH-1 downto CNTR_LENGTH-5);

    nuChar <= x"2d" when error = '1' and selekt(4) = '1' else
            x"30" when (selekt(4) = '0' and
```
topRow((wordSize-1)-int(selekt(3 downto 0))) = '0') or
(selekt(4) = '1' and
botRow((wordSize-1)-int(selekt(3 downto 0))) = '0') else x"31";

process(clk) begin
  if rising_edge(clk) then
    counter <= counter + 1;
    if reset = '1' then
      counter <= (others => '0');
    end if;
  end if;
end process;
end a1;
Part E. (10 points) Draw a block diagram of your top level circuit below. It should include the input module, output module and calculator and all signals connecting them. Label all signals with the names used by the top module.
**Part F.** (15 points) Run the simulator using the provided *testTop* testbench. Do not modify the testbench. When you run the simulation, you will notice that all the interesting stuff happens near the very end of the simulation. Focus on this part when you are checking to make sure your circuit works correctly. Add signals to the waveform display to help you verify that the circuit works correctly. Specifically, be sure to include all of the calculator’s internal signals and the *nuChar* signal in the output module. Organize the signals so that related signals are grouped together and use dividers to label different groups of signals. Paste a screenshot of your simulation output showing the time period from 11,001 microseconds to 11,003 microseconds. Make sure that all text is clearly legible on the printed output.

What are the operand values for the first addition operation in this time period? What is the result of this addition? Is it correct? Is an error detected at this point?

The operands are ffff and 0001 and the sum is 0000. It does result in an error since mode=0 at this point, so this is an unsigned addition that produces an overflow.

What is the value of *nuChar* at this point? Is it what you expect? Are the LEDs correct?

The value of *nuChar* is 2d, which is the ASCII character code for the dash character. This is the character used to indicate the presence of an error. This is what is expected, since at this point the output module is generating the characters that are displayed in the bottom row of the display. The LEDs are correct. They show led(1)=‘1’ which is expected when an error occurs. Also, led(0)=‘0’ which is correct since the mode is 0 at this point.
What are the operands for the first addition operation in this time range? What result is produced? Is an error detected at this point?

The operands are ffff and 8001. Since mode=1 at this point, we are doing signed addition and ffff represents -1. Adding 1 to it produces a result of 8000, which is also a negative number, so no error should be detected at this point.