Part A. (20 points) Paste the VHDL for the stack component.

-- Stack circuit
-- Jon Turner, 1/2014
--
-- This circuit implements a simple stack with a generic size and a
-- generic word size. It has three generic parameters.
--
-- stkSiz is actually one more than the true stack size; it must be
-- a power of 2
-- lgSiz is the base 2 logarithm of stkSiz
-- wordSiz is the number of bits in each stack word
--
-- The stack has two control signals, push and pop. When push is high,
-- the value on the data input is added to the stack. When pop is high
-- the top entry on the stack is removed. When both push and pop are
-- high, the input value replaces the value on the top of the stack.
-- This is called a swap.
--
-- Any attempt to add a value to a full stack is ignored.
-- A pop or swap on an empty stack is also ignored.

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
use work.commonDefs.all;

entity stack is
  generic(
    stkSiz: integer := 8;
    lgSiz: integer := 3;
    wordSiz: integer := 16);
  port(
    clk, reset: in std_logic;
    push, pop: in std_logic;
    dIn: in std_logic_vector(wordSiz-1 downto 0);
    top: out std_logic_vector(wordSiz-1 downto 0);
    empty, full: out std_logic);
end stack;

architecture a1 of stack is

  type stackType is array(0 to stkSiz-1) of std_logic_vector(wordSiz-1 downto 0);

  signal stak: stackType;

  signal sp: std_logic_vector(lgSiz-1 downto 0); -- stack pointer
begin
  process(clk) begin
    if rising_edge(clk) then
      if reset = '1' then
        sp <= (others => '0'); stak(stakSiz-1) <= (others => '0');
      else
        if push > pop then
          if sp /= (sp'range => '1') then
            stak(int(sp)) <= dIn; sp <= sp + 1;
          end if;
        elsif push < pop then
          if sp /= (sp'range => '0') then
            sp <= sp - 1;
          end if;
        elsif push = '1' then -- pop='1' also
          if sp /= (sp'range => '0') then
            stak(int(sp-1)) <= dIn;
          end if;
        end if;
      end if;
    end if;
  end process;
top <= stak(int(sp-1));
  full <= '1' when sp = (sp'range => '1') else '0';
  empty <= '1' when sp = (sp'range => '0') else '0';
end a1;
Part B. (15 points) Write a testbench to verify the operation of your stack module and paste the source for your testbench below. Your testbench should add a series of unique values that fill the stack and should then pop items off the stack. You can save yourself some time and effort by writing some simple utility procedures to control the stack input signals when performing the push, pop and swap operations. Include assertions in your testbench to check that the value on the top of the stack is correct at all times, and that the full and empty signals are correct. Be sure to check that a push on a full stack has no effect, and that a pop on an empty stack has no effect. Also be sure to check that the swap operation works correctly. Include comments in your testbench to explain what you’re doing.

--- Testbench for stack
-- Jon Turner, 2/2014

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
use work.txt_util.all;
use work.commonDefs.all;

entity testStack is end testStack;
architecture a1 of testStack is

component stack
    generic(
        stakSiz: integer :=  8;
        lgSiz: integer :=  3;
        wordSiz: integer := 16);
    port(
        clk, reset: in std_logic;
        push, pop: in std_logic;
        dIn: in std_logic_vector(wordSiz-1 downto 0);
        top: out std_logic_vector(wordSiz-1 downto 0);
        full, empty: out std_logic);
end component;

signal clk, reset : std_logic := '0';
signal push, pop : std_logic := '0';
signal inbits: word := (others => '0');

signal empty, full: std_logic;
signal outbits: word;

-- Clock period definitions
constant clk_period : time := 20 ns;
constant pause: time := 5*clk_period;

BEGIN
    uut: stack generic map(8,3,wordSize)
        port map(clk,reset,push,pop,inBits,outBits,full,empty);

    process begin
        clk <= '0'; wait for clk_period/2;
        clk <= '1'; wait for clk_period/2;
    end process;

process
-- reset the circuit
procedure restart is begin
    reset <= '1'; wait for pause; reset <= '0';
end;

-- pop the stack
procedure popIt is begin
    pop <= '1'; wait for clk_period; pop <= '0'; wait for pause;
end;

-- push a value onto the stack
procedure pushIt(inval: integer) is begin
    inbits <= slv(inval,wordSize);
    push <= '1'; wait for clk_period; push <= '0'; wait for pause;
end;

-- swap the input value with the value on the top of the stack
procedure swapIt(inval: integer) is begin
    inbits <= slv(inval,wordSize);
    push <= '1'; pop <= '1'; wait for clk_period;
    push <= '0'; pop <= '0'; wait for pause;
end;

begin
    wait for 100 ns;

    restart;
    assert empty = '1' report "empty signal low when stack is empty";

    -- push until stack is full
    for i in 1 to 7 loop
        assert full = '0' report "full signal high when stack is not full";
        pushIt(i);
        assert int(outbits) = i report "incorrect stack top when pushing, i=" & str(i);
        assert empty = '0' report "empty signal high when stack is not empty";
    end loop;

    assert full = '1' report "full signal low when stack is full";

    -- attempt to push again (should be ignored)
    pushIt(8);
    assert int(outbits) = 7 report "incorrect stack top " & str(outbits);

    -- and swap (should work)
    swapIt(9);
    assert int(outbits) = 9 report "incorrect stack top " & str(outbits);

    -- pop until stack is empty
    for i in 7 downto 1 loop
        assert empty = '0'
            report "empty signal high when stack is not empty";
        assert int(outbits) = i
            report "incorrect stack top when popping, i=" & str(i);
        popIt;
        assert full = '0' report "full signal high when stack is not full";
    end loop;
end;
end loop;
assert empty = '1' report "empty signal low when stack is empty";
-- attempt to pop again (should be ignored)
popIt;
assert empty = '1' report "empty signal low when stack is empty";
assert full = '0' report "full signal high when stack is not full";
-- attempt to swap again (should be ignored)
swapIt(13);
assert empty = '1' report "empty signal low when stack is empty";
assert full = '0' report "full signal high when stack is not full";
assert false report "normal termination" severity failure;
end process;
end;
Part C. (15 points) Paste one or more screenshots from your stack simulation below. Be sure to include the stack contents (expand the stack to show each stack value as a separate waveform) and stack pointer values in your simulation output. Make sure all text is easily legible.

The first 1.3 microseconds of the stack simulation appears below.
At what time in the simulation does the stack first become full? *At 900 ns*

At what time is the first push on a full stack? *At 1400 ns*

At what time does the stack first become empty after being non-empty? *At 2000 ns*

When is the first successful swap operation? *At 1160 ns.*

When is the first pop on an empty stack? *At 2100 ns.*
Part D. (20 points). Paste your source for stackCalc below

-- Stack Calculator
-- Jon Turner, 1/2014
--
-- Operations
-- x0 - clear top value on stack (set to zero)
-- x1 - clear stack
-- x2 - remove value from top of stack
-- x3 - place input value on stack
-- x4 - add input value to the value on top of stack and replace value
--   on stack
-- x5 - add top two values on stack together and put replace them with sum
-- x6 - subtract input value from value on top of stack and replace stack top
--   with the difference
-- x7 - subtract the top two values on stack and put replace them
--   with difference
--
-- Most operations can be done in a single clock tick, but the two operations
-- that operate on the top two stack elements require two clock ticks.
--
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
use work.commonDefs.all;

entity stackCalc is
  generic(
    stakSiz: integer :=  8;
    lgSiz:   integer :=  3;
    wordSiz: integer := 16);
  port(
    clk, reset: in std_logic;
    op: in nibble;
    doOp: in std_logic;
    dIn: in std_logic_vector(wordSiz-1 downto 0);
    result: out std_logic_vector(wordSiz-1 downto 0));
end stackCalc;

architecture a1 of stackCalc is
  component stack
    generic(
      stakSiz: integer :=  8;
      lgSiz:   integer :=  3;
      wordSiz: integer := 16);
    port(
      clk, reset: in std_logic;
      push, pop: in std_logic;
      dIn: in std_logic_vector(wordSiz-1 downto 0);
      top: out std_logic_vector(wordSiz-1 downto 0);
      empty, full: out std_logic);
  end component;

  -- stack interface signals
  signal push, pop, resetStak, empty, full: std_logic;
  signal stakIn, stakTop: std_logic_vector(wordSiz-1 downto 0);
signal cont5, cont7: std_logic; -- continuation signals for ops 5, 7
signal temp: std_logic_vector(wordSiz-1 downto 0); -- previous stack top

begin
stak: stack generic map(stakSiz, lgSiz, wordSiz)
port map(clk, resetStak, push, pop, 
stakIn, stakTop, empty, full);

-- process for controlling the operations that take two steps
process(clk) begin
if rising_edge(clk) then
if reset = '1' then
cont5 <= '0'; cont7 <= '0';
temp <= (others => '0');
else
cont5 <= '0'; cont7 <= '0';
if doOp = '1' then
if op = x"5" then
    temp <= stakTop; cont5 <= '1';
elsif op = x"7" then
    temp <= stakTop; cont7 <= '1';
end if;
end if;
end if;
end if;
end if;
end process;

-- asynchronous process for controlling the stack
process (reset, op, doOp, dIn, stakTop, temp) begin
push <= '0'; pop <= '0';
stakIn <= dIn; result <= stakTop;
resetStak <= reset;
if doOp = '1' and cont5 = '0' and cont7 = '0' then
    case op is
    when x"0" => -- clear top value on stack
        stakIn <= (others => '0'); push <= '1'; pop <= '1';
    when x"1" => resetStak <= '1';
    when x"2" => pop <= '1';
    when x"3" => push <= '1';
    when x"4" => stakIn <= dIn + stakTop; push <= '1'; pop <= '1';
    when x"5" => pop <= '1';
    when x"6" => stakIn <= stakTop - dIn; push <= '1'; pop <= '1';
    when x"7" => pop <= '1';
    when others =>
        end case;
end if;
if cont5 = '1' or cont7 = '1' then -- completing ops 5,7
if empty = '0' then
    push <= '1'; pop <= '1';
    if cont5 = '1' then stakIn <= temp + stakTop;
    else stakIn <= temp - stakTop;
    end if;
else -- restore stack to original state
    stakIn <= temp; push <= '1';
end if;
end if;
end process;
end al;
Part E (10 points). Draw a block diagram of your stackCalc circuit below. You can show most of the control logic as a large block labeled control. However, the stack component and any additional flip flops or registers used by the stackCalc component should be shown as separate blocks. Also, you should show how different data values can be input to the stack and you should label signals appropriately to make your diagram as clear an unambiguous as you can.
**Part F.** (15 points) Simulate your stackCalc circuit using the provided testStackCalc testbench. Verify that there are no error reports on the simulation console from the assert statements in the testbench. Paste a screenshot of the first 3 microseconds of the simulation below. Include all internal state from your stackCalc component and all the stack signals in your waveform display. Show the stack in expanded form. Use the unsigned integer radix for all the numeric values.

What values are on the stack at time 850 nanoseconds? 1,2,3,4,5,6.

What values are on the stack at 1.7 nanoseconds? 1,2.
Paste a copy of the portion of the simulation from 3 microseconds to the end below.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputs</td>
<td>0</td>
</tr>
<tr>
<td>reset</td>
<td>5</td>
</tr>
<tr>
<td>doop</td>
<td>0</td>
</tr>
<tr>
<td>inbits[15:0]</td>
<td>0</td>
</tr>
<tr>
<td>clk</td>
<td>1</td>
</tr>
<tr>
<td>internal state and o</td>
<td></td>
</tr>
<tr>
<td>cont5</td>
<td>0</td>
</tr>
<tr>
<td>cont7</td>
<td>0</td>
</tr>
<tr>
<td>temp[15:0]</td>
<td>10</td>
</tr>
<tr>
<td>outbits[15:0]</td>
<td>10</td>
</tr>
<tr>
<td>stack signals</td>
<td></td>
</tr>
<tr>
<td>push</td>
<td>0</td>
</tr>
<tr>
<td>pop</td>
<td>0</td>
</tr>
<tr>
<td>stak[0:7]</td>
<td>[1, 1]</td>
</tr>
<tr>
<td>[0]</td>
<td>10</td>
</tr>
<tr>
<td>[1]</td>
<td>11</td>
</tr>
<tr>
<td>[2]</td>
<td>9</td>
</tr>
<tr>
<td>[3]</td>
<td>12</td>
</tr>
<tr>
<td>[4]</td>
<td>8</td>
</tr>
<tr>
<td>[5]</td>
<td>13</td>
</tr>
<tr>
<td>[6]</td>
<td>7</td>
</tr>
<tr>
<td>[7]</td>
<td>0</td>
</tr>
<tr>
<td>sp[2:0]</td>
<td>1</td>
</tr>
<tr>
<td>empty</td>
<td>0</td>
</tr>
<tr>
<td>full</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time (ns)</th>
<th>3,000</th>
<th>3,500</th>
<th>4,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>op[3:0]</td>
<td>5</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>inbits[15:0]</td>
<td>0</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>internal state and o</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cont5</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>cont7</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>temp[15:0]</td>
<td>10</td>
<td>13</td>
<td>3</td>
</tr>
<tr>
<td>outbits[15:0]</td>
<td>10</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>stack signals</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>push</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pop</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>stak[0:7]</td>
<td>[1, 1]</td>
<td>[1, 1]</td>
<td>[1, 1]</td>
</tr>
<tr>
<td>[0]</td>
<td>10</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>[1]</td>
<td>11</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>[2]</td>
<td>9</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>[3]</td>
<td>12</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>[4]</td>
<td>8</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>[5]</td>
<td>13</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>[6]</td>
<td>7</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>[7]</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sp[2:0]</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>empty</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>full</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

What calculator operation is being performed at time 3090 ns? How does this operation change the values on the stack? Is the resulting stack state correct?

This is a subtract operation which subtracts the top two values on the stack and replaces them with their difference. Before the operation, the top two values are 13 and 5, so their difference is 8, which is the new value at the top of the stack. This appears as the outbits output from the calculator component.

What calculator operation is being performed at time 3200 ns? How does this operation change the values on the stack? Is the resulting stack state correct?

This is an add operation that adds the input value to the value on top of the stack. The value on the top of the stack is 8, while the input value is 13, so the new value on top of the stack is 21, and the simulation output shows this, so the resulting stack state is correct.
Part G. (15 points) Simulate your completed circuit using the provided testTop testbench. Include the inputs and output to the stackCalc component in your waveform window, plus the stack and stack pointer signals from the stack component (in this case, you need not expand the stack). In addition, include the selekt and nuChar signals from outMod component, plus the cb signal from the lcdDisplay component. Show the nuChar and cb signals in ASCII format. Paste a screenshot showing the portion of the simulation from 0 microseconds to 6 microseconds below.

What calculator operations are performed during this period?

A series of push operations are performed.

How many items are on the stack at the end of this period?

Seven.
Paste a screenshot showing the portion of the simulation from 29 to 31 microseconds.

What operation is performed at 29.2 microseconds?
This is a subtract on the top two values on the stack.

What operation is performed at 30.8 microseconds?
This operation adds the input (2) to the top value on the stack (1), which gives a result of 3.
Paste a screenshot showing the period from 15 to 20 milliseconds (not microseconds) below.

What would you expect to see on the LCD display at this point? Why?

In the top row, the string “in 0002 out 0003”. In the bottom row, the string “add to top “, since this is what appears in the lcdDisplay component’s character buffer at the end of the simulation run.
Part H. (10 points) Proceed to this part only after you have completed the simulation in Part G and have convinced yourself that your circuit works correctly. Prototype your circuit using the prototype board available in Bryan 316. Once you have your circuit loaded onto the board and you have convinced yourself that it works correctly, fill in your name below on the printed copy and have one of the TAs check it and sign their name below, after assigning the appropriate number of demo points.

Student name: _____________________________ has successfully demonstrated the stakCalc circuit on the prototype board.

TA name: _________________________________

TA signature: _____________________________

Demo points (out of 10): ________________

Comments (if the circuit does not work 100% correctly, make a note of all issues below):