**Part A** (10 points). Draw a state transition diagram for your pattern matcher. Label the states with symbolic names. Label the transitions clearly with the conditions that trigger the transitions and with any actions that should be taken when the transition occurs. You may omit “restart” transitions. Note that this is a fairly complicated diagram. Think carefully about each state and each transition. Organize the diagram to make it easy to follow. Draw it neatly and legibly. You are strongly urged to do this part before you write any VHDL code.
Part B. (20 points) Paste the VHDL for your pattern matcher below. Make sure that your VHDL is consistent with your state diagram. 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.

```
-- Pattern Matcher
-- Jon Turner, 12/2013
--
-- This circuit matches a simple regular expression: a+(b^n | c) d.
-- Here, a, b, c and d are symbols (from an alphabet of 16 symbols),
-- the plus sign indicates one or more copies of the symbol a and
-- n is a repeat count. So for example, if n=3, the pattern matcher
-- will detect a match when it sees either of the strings aaabbd or aacd.
--
-- The repeat count is captured when the circuit is reset.
-- The circuit counts the total number of times a pattern has been
-- matched and outputs this value.
--
library IEEE;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
use work.commonDefs.all;

entity patternMatcher is port(
    clk, restart, valid: in std_logic;
    inSym: in nibble;
    repCount: in nibble;
    patCount: out byte);
end patternMatcher;

architecture a1 of patternMatcher is

  type stateType is (nil, a, ab, ac, axd);

  signal state: stateType;
  signal bNum, bCount:nibble;
  signal matchCount: byte;

begin

  patCount <= matchCount;

  process(clk) begin
    if rising_edge(clk) then
      if restart = '1' then
        state <= nil;
        bCount <= repCount;
        matchCount <= (others => '0');
      elsif valid = '1' then
        case state is
          when nil =>
            if inSym = x"0" then state <= a;
            end if;
          when a =>
            if inSym = x"1" then state <= ab; bNum <= x"1";
            elsif inSym = x"2" then state <= ac;
            elsif inSym /= x"0" then state <= nil;
            end if;
```
when ab =>
  if inSym = x"1" then
    if bNum /= bCount then bNum <= bNum + 1;
    else state <= nil;
  end if;
  elsif inSym = x"3" then
    if bNum = bCount then
      state <= axd; matchCount <= matchCount + 1;
      else
      state <= nil;
    end if;
  elsif inSym = x"0" then state <= a;
  else state <= nil;
  end if;
when ac =>
  if inSym = x"3" then
    state <= axd; matchCount <= matchCount + 1;
  elsif inSym = x"0" then
    state <= a;
  else
    state <= nil;
  end if;
when axd =>
  if inSym = x"0" then state <= a;
  else state <= nil;
  end if;
end case;
end if;
end if;
end process;
end a1
Part C. (10 points) Run the simulator using the provided testPatMatch testbench and paste a screenshot of the waveform window below. Set the radix to decimal for the patCount signal. Be sure to include your state signal in the waveform display, as well as any other registers that are defined by your patMatch code. Paste a screenshot showing the portion of the simulation from 100 ns to 800 ns. Make sure that all text is clearly visible and legible on the printed copy.

How many patterns are matched during this period? Exactly what patterns are matched?

Three patterns are matched. They are abbbd, abbbd and aacd.

Paste a screenshot below showing the portion of the simulation from 1000 ns to 1800 ns.

How many patterns are matched during this period? How many times is a pattern match started but then the match fails part way through? At what times are failed matches detected? What are the partial patterns, including the first symbol that doesn’t match?

There are no successful matches during this period. Four failed matches are detected during this period. They are detected at time 1150 ns, 1230, 1430 and 1630. The failed matches are ad, ah, abbbba and abbb.
Part D. (10 points) In your repository, you will find a `binaryInMod`, an output module called `outMod` and a partial implementation of a `top` module. You are to complete the `top` module so that it connects the `patternMatcher` to an instance of `binaryInMod` and an instance of `outMod`. The `restart` input of `patternMatch` should be connected to output `pulse(1)` of the `binaryInMod` circuit, `valid` should be connected to `pulse(2)`, `inSym` should be connected to `inBits(3..0)` and `repCount` should be connected to `inBits(11..8)`. The `outMod`'s `valid` and `inSym` inputs should be connected to the corresponding inputs of `patternMatcher` and its `patCount` input should be connected to the `patCount` output of `patternMatcher`. You should also connect the `inSym` and `repCount` inputs of `patternMatcher` to `led(3..0)` and `led(7..4)` respectively. The `outMod` circuit displays the current `inSym` value as an ASCII character at the right end of the top row of the display. Recently input symbols appear to the left of the current symbol. The value of `patCount` is displayed in hex in the bottom row of the display. Paste a copy of your `top` circuit below.

```
-- Top module for pattern matcher
-- Jon Turner 12/2013
--
-- This circuit implements a simple pattern matcher on the prototype
-- boards. An instance of the binaryInMod circuit is used to provide
-- inputs to an instance of patternMatcher and an instance of the
-- outMod circuit is used to display information about the operation
-- of the patternMatcher.
--
library IEEE;
use ieee.std_logic_1164.all;
use IEEE.numeric_std.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 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 outMod is port(
  clk, reset: in std_logic;
  inSym: in nibble;
  valid: in std_logic;
end component;
```
component patternMatcher port(
    clk, restart, valid: in std_logic;
    inSym: in nibble;
    repCount: in nibble;
    patCount: out byte);
end component;

signal reset, restart, valid: std_logic;
signal dBtn, pulse: std_logic_vector(3 downto 1);
signal inBits: word;

signal inSym, repCount: nibble;
signal patCount: byte;

begin
    -- define inputs to patternMatch
    restart <= pulse(1); valid <= pulse(2);
inSym <= inBits(3 downto 0);
repCount <= inBits(11 downto 8);

    -- instantiate and connect the sub-components
    imod: binaryInMod port map(clk,btn,knob,reset,dBtn,pulse,inBits);
    omod: outMod port map(clk,reset,inSym,valid,patCount,lcd);
    pmat: patternMatcher port map(clk,restart, valid, inSym,
                                 repCount, patCount);

    -- connect signals to leds to facilitate checking
    led <= inSym & repCount;
end a1;
Part E. (10 points). You will find an incomplete testTop testbench in your repository. Add tests to the testbench following the instructions in the comments. Make sure that your tests cover every transition in your state transition diagram. Include a 50 microsecond delay in your testbench, after the last test input. Paste the code for your tests below.

```c
-- start with a set of tests using a repeat count of 3
-- first test all the cases where the input matches the pattern
-- be sure to use all the state-machine transitions that
-- lead to successful matches
restart(x"3");
nextSymVec(x"12fffffff");
nextSymVec(x"01113fffff");
nextSymVec(x"011135ffff");
nextSymVec(x"0023fffffff");
nextSymVec(x"0023213fff");

-- next add cases that fail after matching one or more
-- initial 'a' characters
nextSymVec(x"03fffffff");
nextSymVec(x"07fffffff");

-- now cases that fail after matching ab
nextSymVec(x"01110fffff");
nextSymVec(x"01111fffff");
nextSymVec(x"01115fffff");
nextSymVec(x"0113fffffff");
nextSymVec(x"0110ffffff");

-- and finally, cases that fail after matching ac
nextSymVec(x"020fffffff");
nextSymVec(x"021fffffff");
nextSymVec(x"02bfffffffff");

-- now, a few more tests using a repeat count of 1
restart(x"1");
nextSymVec(x"013fffffff");
nextSymVec(x"003fffffff");
nextSymVec(x"011fffffff");
nextSymVec(x"0113fffffff");
nextSymVec(x"023fffffff");
nextSymVec(x"231fffffff");

wait for 50 us;
```
**Part F.** (15 points) Simulate your completed circuit using your testTop testbench. Remove the default signals that are supplied by the simulator and include the following instead, using the indicated radix setting.

- the clk, btn (binary) and knob (binary) inputs to top, along with the led (hex) outputs
- the resetOut, pulse (hex) and inbits (hex) outputs from binaryInMod
- all signals from patternMatcher, including the state signal; use hex for the repCount and inSym inputs and use the unsigned integer radix for the patCount output
- the top (ASCII) and bot (unsigned integer) signals from outMod

Organize the signals in the waveform window in appropriate groups, with dividers labeling the different groups. Make sure that the operationMode constant in commonDefs.vhd is set to 0 for this part. Paste a screenshot showing a portion of the simulation where at least three patterns are matched. Make sure that the patternMatcher signals are all clearly readable. If you need to, split the screenshot into two parts to improve legibility.

What are the first three patterns that are matched during this part of the simulation? At what times does the pattern matcher recognize that a match has occurred?

*The first three patterns matched are abbbd, abbbd and aacd. The matches are recognized at times 6.5 microseconds, time 10 and time 15.*
Paste a screenshot below showing the final portion of your simulation, including the final values of the top and bot signals from outMod.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>60 us</th>
<th>65 us</th>
<th>70 us</th>
<th>75 us</th>
</tr>
</thead>
<tbody>
<tr>
<td>top signals</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>trn[3:0]</td>
<td>0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>trb[20]</td>
<td>0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>trc[7:0]</td>
<td>sl</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>clk</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>binary[3:Mod</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>resetout</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pulse[3:1]</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>inbits[15:0]</td>
<td>q1035</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pathMatch</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>restart</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>repcount[3:1]</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>insym[3:0]</td>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>valid</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>state</td>
<td>n1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pnum[3:0]</td>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>patcount[7:4]</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>outMod signals</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>top[15]</td>
<td>b, a, c, 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bot[15]</td>
<td>f, , , ,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Based on the simulation output, what would you expect to see on the LCD display of the prototype board, if the same test data was input to the prototype board? What does the displayed information tell you about the test data?

The top row should show the string “daadabbdcdcd” since this is the value that appears in top. The bottom row should show “02” since this is what bot shows. The top row of the output shows the last 14 symbols that were input to the patternMatcher and the current inSym value. So b is the current inSym value and “cdb” are the last three input symbols. The bottom row shows the final value of patCount.
What pattern is matched during this time period? How can you tell? What is the value on the LEDs at the time the pattern match is first recognized? What is the significance of this value?

The pattern abbbd is recognized during this time period. We can tell this by observing the sequence of inSym values at the moments when the valid signal goes high (01113). The LEDs show the value 33 at the time the d is input. The first 3 is the value of inSym and the second 3 is the value of repCount.
Paste a screenshot below, with an even closer-up view of the simulation showing a place where the valid signal goes high for a clock tick and also where the inBits signal from binaryInMod changes. Make sure that the button and knob signals are clearly readable.

Explain how changes to the inputs of the top module cause valid to go high. To answer this question, you may need to study the code for binaryInMod.

Valid is controlled by btn(2). We can see that btn(2) goes high at 53.1 microseconds. The debouncer inside binaryInMod delays this signal (for four clock ticks when simulating) and this causes the signal pulse(2) to go high at 53.2 microseconds. Valid is connected directly to pulse(2).

Now, observe the changes to the knob input to top. Explain how these changes affect the inputs to patternMatcher.

The first few changes to knob involve knob(0). This is the signal that is activated when you press down on the knob and release it. So, we see two presses on the knob. This causes knob rotations to affect the portion of inBits starting at bit 8. The next few changes to knob involve knob(2) and knob(1). These are the signals that change when you turn the knob. The knob rotation causes the value of inBits to change from 030b to 020b and since the repCount input of patternMatcher is connected to bits(11..8), its value changes from 3 to 2.
Part G. (10 points) Proceed to this part only after you have successfully completed part F and have carefully checked the simulation results. Add the file protoBoard.ucf to your project (by selecting Add Source from the Project menu). Generate a bit file for your completed circuit. Make sure that the operationMode constant in commonDef.vhd is set to 1. Then download the bit file to a prototype board using the Impact tool. The prototype boards are kept in a locker in Bryan Hall, Room 316 and can be signed out as needed. Stuart Cranor (who can usually be found in 316 or 308) can sign out a board for you to use. If he is not available, any of the TAs can also sign out a board for you. Boards should normally be kept for no more than two hours at a time, but you may check them out overnight. In this case, you should return them first thing the next morning (by 9:00). Note that you are responsible for these boards when they are checked out to you. They cost about $150 each, so please handle them with care.

You may do your testing using any of the PCs in the Bryan 316 lab or in any of the Urbauer labs. They all have the necessary software. However, if there is a class that is using the Bryan 316 lab, please do your testing in Urbauer. 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 patternMatcher 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):