###### *CSE 260 – Introduction to Digital Logic and Computer Design Jon Turner*

Lab 2 Report

##### *Your name* Likai Yan *2/18/2013*

***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.

*paste your diagram here*

***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.

paste your code here

----------------------------------------------------------------------------

-- Pattern Matcher

--

-- by Likai Yan 02/2014

--

-- This circuit implemented the method to recognize regex pattern a({b}|c)d from input, and

-- count the number of times this pattern appears. It also have the ability to

-- restart the circuit such that the num of repeated pattern is cleared.

--

-- In the regex. the num of

-- repeated 'b' is input via repCount, and the number of repeated pattern is

-- output by patCoutn.

----------------------------------------------------------------------------

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 (start,blank,Astate,Bstate,Cstate,Dstate);

signal state: stateType;

--signal bp: std\_logic;

signal bcnt:std\_logic\_vector(3 downto 0);

signal pcnt: byte;

signal inc: nibble;

begin

process(clk)begin

if rising\_edge(clk) then

if restart='1' then

state<=start; inc<=repCount; bcnt<=x"0"; pcnt<=x"00";

else

case state is

when start=>

pcnt<=x"00";

-- bp<=0;

state<=blank;

when blank=>

bcnt<=x"0";

-- bp<=0;

if valid='1' then

if inSym=x"0" then state<=Astate;--double quote for hex. don't consider bcnt, bp as they are 0.

else state<=blank;

end if;

end if;

when Astate=>

if valid='1' then

if inSym=x"0" then state<=Astate;

elsif inSym=x"1" then state<=Bstate; bcnt<=bcnt+1;

--end if; -- I think use elsif is better

elsif inSym=x"2" then state<=Cstate;

else state<=blank;

end if;

end if;

when Bstate=>

if valid='1' then

if inSym=x"1" then bcnt<=bcnt+1; --implies retaining its stage

elsif (inSym=x"3" and bcnt=inc) then state<=Dstate;

else state<=blank;

end if;

end if;

when Cstate=>

if valid='1' then

if inSym=x"3" then state<=Dstate;

else state<=blank;

end if;

end if;

when Dstate=>

pcnt<=pcnt+1;

state<=blank;

when others=>

end case;

end if;

end if;

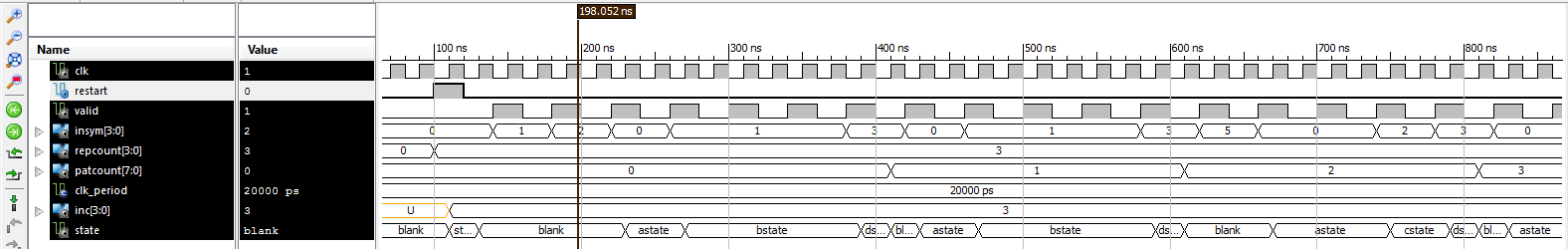
end process;

patCount<=pcnt;

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.

(p1)



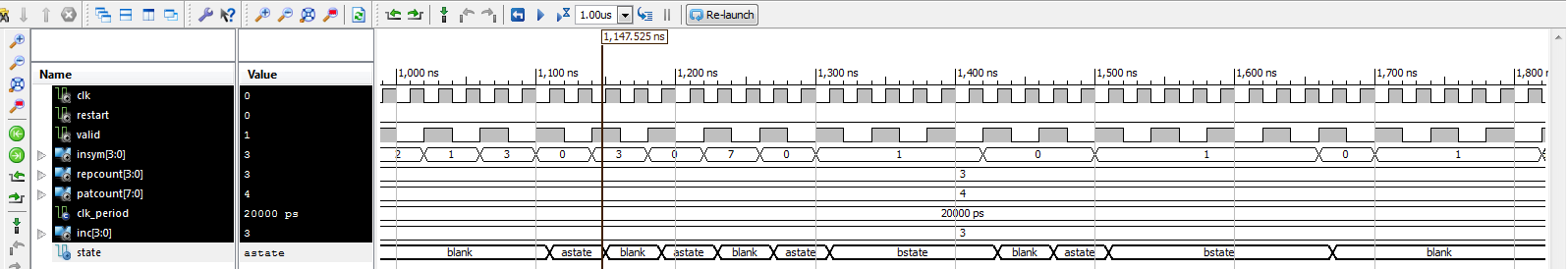
How many patterns are matched during this period? Exactly what patterns are matched?

2 patterns are matched. They are:

a(b3|c)d: abbbc, abbbcf

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

(p2)

**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?

0 matched. 4 failed matches attempted. Failed attempts (assume not including those failed to match the first symbol) are detected at 1150ns, 1230ns, 1430ns, 1670ns respectively. Partial patterns: c, b, d, ad, ah, abbba, abbb, in chronical order.

***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

--

--

-- by Likai Yan 02/2014

--

-- This circuit uses buttons and siwtches with LEDs to take in the input,

-- passing the information to the patternMatcher and outMode,

-- and use LED to display certain information.

--

-- It connects the patterMatcher circuit with the binaryInMod and outMod circuits.

---------------------------------------------------------------------

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 patternMatcher port (

clk, restart, valid: in std\_logic;

inSym: in nibble;

repCount: in nibble;

patCount: out byte);

end component;

component binaryInMod is 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; -- input symbol to matcher

valid: in std\_logic; -- valid symbol signal

patCount: in byte; -- number of matches since restart

-- signals for controlling LCD display

lcd: out lcdSigs);

end component outMod;

signal inSym, repCount : nibble;

signal patCount: byte;

signal restart, valid, resetOut, reset: std\_logic;

signal dBtn, pulse: std\_logic\_vector(3 downto 1);

signal inBits: word;

--do we need to deal with signal buttons?

-- TODO - your code here. Do we need to care about clk?

begin

imod: binaryInMod port map(clk,btn,knob,resetOut,dBtn,pulse,inBits);

pm: patternMatcher port map(clk,restart,valid,insym,repcount,patcount);

omod: OutMod port map(clk,reset,insym,valid,patCount,lcd);

restart<=pulse(1);

valid<=pulse(2);

inSym<=inBits(3 downto 0);

repCount<=inBits(11 downto 8);

led(3 downto 0)<=inSym;

led(7 downto 4)<=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.

--------------------------------------------------------------------------------

-- Test Pattern Matcher from top

-- Jon Turner, 12/2013

-- Modified by Likai Yan 02/2014

-- Modification: I have added a few test as instructed in the comment. The circuit

-- can be tested for: 1. three repeated b and all pass, 2.recognize pattern a, but failed,

-- 3. recognize pattern ab{n}, but failed to proceed. 4. recognize pattern ac, but failed to proceed.

-- 5. a few other tests with different num of a and b

--------------------------------------------------------------------------------

LIBRARY ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

use ieee.std\_logic\_unsigned.all;

use work.commonDefs.all;

entity testTop is end testTop;

architecture behavior of testTop is

-- Component Declaration for the Unit Under Test (UUT)

COMPONENT top

PORT(

clk : IN std\_logic;

btn : IN std\_logic\_vector(3 downto 0);

knob : IN std\_logic\_vector(2 downto 0);

swt : IN std\_logic\_vector(3 downto 0);

led : OUT std\_logic\_vector(7 downto 0);

lcd : OUT lcdSigs

);

END COMPONENT;

--Inputs

signal clk : std\_logic := '0';

signal btn : std\_logic\_vector(3 downto 0) := (others => '0');

signal knob : std\_logic\_vector(2 downto 0) := (others => '0');

signal swt : std\_logic\_vector(3 downto 0) := (others => '0');

--Outputs

signal led : std\_logic\_vector(7 downto 0);

signal lcd : lcdSigs;

-- Clock period definitions

constant clk\_period : time := 20 ns;

constant pause : time := 5\*clk\_period;

signal rot: std\_logic\_vector(1 downto 0) := "00";

signal press: std\_logic := '0';

-- These signals are used in the procedures defined below.

signal inSym, repCount : nibble := x"0";

BEGIN

knob(0) <= press; knob(2 downto 1) <= rot;

-- Instantiate the Unit Under Test (UUT)

uut: top PORT MAP (clk, btn, knob, swt, led, lcd);

-- Clock process definitions

clk\_process :process

begin

clk <= '0'; wait for clk\_period/2;

clk <= '1'; wait for clk\_period/2;

end process;

-- Stimulus process

stim\_proc: process

-- The procedures below are designed to make it easier to specify

-- your test input. Take a few minutes to make sure you understand

-- how they work. Ask questions if you're not sure. Then, use them

-- to specify your tests.

-- rotate the knob to the right cnt times

procedure rrot(cnt: in integer) is

begin

for i in 1 to cnt loop

rot <= "10"; wait for pause; rot<= "11"; wait for pause;

rot <= "01"; wait for pause; rot<= "00"; wait for pause;

end loop;

end;

-- rotate the knob to the left cnt times

procedure lrot(cnt: in integer) is

begin

for i in 1 to cnt loop

rot <= "01"; wait for pause; rot<= "11"; wait for pause;

rot <= "10"; wait for pause; rot<= "00"; wait for pause;

end loop;

end;

-- press down on the knob cnt times

procedure bump(cnt: in integer) is

begin

for i in 1 to cnt loop

press<='1'; wait for pause; press<='0'; wait for pause;

end loop;

end;

-- push the reset button

procedure reset is

begin

btn(0) <= '1'; wait for pause; btn(0) <= '0'; wait for pause;

end;

-- Start a round of tests using a specified repeat count

procedure restart(count: in nibble) is begin

bump(2);

if count > repCount then rrot(int(count - repCount));

elsif count < repCount then lrot(int(repCount - count));

end if;

repCount <= count;

bump(2);

btn(1) <= '1'; wait for pause; btn(1) <= '0'; wait for pause;

end;

-- Input one symbol to the circuit, where x"0" corresponds to 'a',

-- x"1 to b and so forth

procedure nextSym(sym: in nibble) is begin

if sym > inSym then rrot(int(sym - inSym));

elsif sym < inSym then lrot(int(inSym - sym));

end if;

inSym <= sym;

btn(2) <= '1'; wait for pause; btn(2) <= '0'; wait for pause;

end;

-- Input a string of up to 9 hex digits. The input vector holds

-- exactly 10 and f is interpreted as a termination character.

-- So for example, use nextSymVec(x"0132ffffff") to input the

-- symbols x0, 1, 3 and 2.

procedure nextSymVec(ss: in std\_logic\_vector(0 to 39)) is begin

for i in 0 to 10 loop

if ss(4\*i to 4\*i+3) = x"f" then exit; end if;

nextSym(ss(4\*i to 4\*i+3));

end loop;

end;

begin

wait for 100 ns;

reset; -- reset circuit using reset procedure above

-- 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 to use all the state-machine transitions that

-- lead to successful matche

restart(x"3");

nextSymVec(x"01113fffff");-- a->bbb->d

nextSymVec(x"023fffffff");-- a->c->d

nextSymVec(x"0023213fff");-- some other test that also pass with aa

-- next add cases that fail after matching one or more initial

-- 'a' characters

nextSymVec(x"03ffffffff");

nextSymVec(x"07ffffffff");--fail after matching a.

-- now cases that fail after matching ab

nextSymVec(x"01110fffff"); --abbba

nextSymVec(x"01111fffff"); --abbbb

nextSymVec(x"01112fffff"); --abbbc

nextSymVec(x"01115fffff"); --abbbf

nextSymVec(x"0110ffffff"); --fail after matching ab.

-- and finally, cases that fail after matching ac

nextSymVec(x"020fffffff"); --aca

nextSymVec(x"021fffffff"); --acb

nextSymVec(x"022fffffff"); --acc

-- now, a few more tests using a repeat count of 1

restart(x"1");

nextSymVec(x"013fffffff");--abd(p)

nextSymVec(x"003fffffff");--aad(f)

nextSymVec(x"0113ffffff");--abbd(f)

nextSymVec(x"023fffffff");--acd(p)

nextSymVec(x"0223ffffff");--accd(f)

nextSymVec(x"231fffffff");--cdb(f)

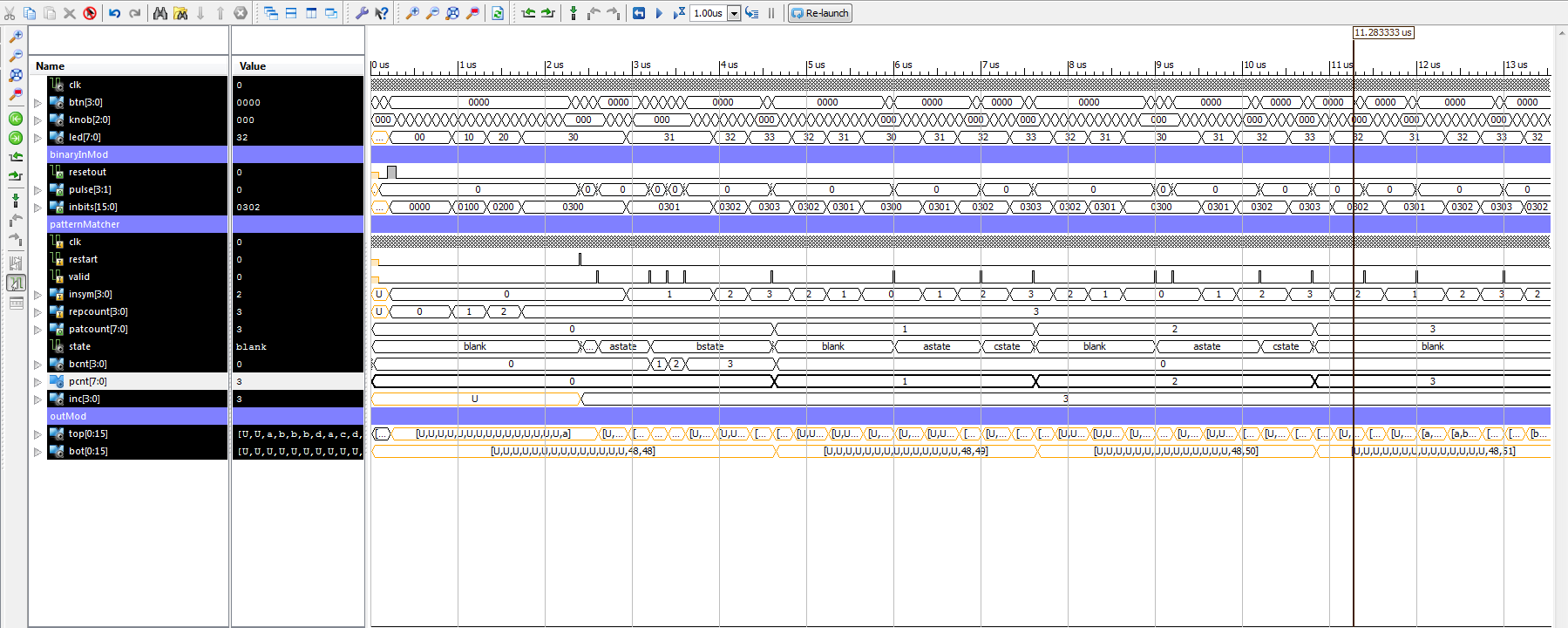
wait for 50 ns;--wait for the graph!

assert (false) report "normal termination" severity failure;

end process;

END;

***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. (p3)

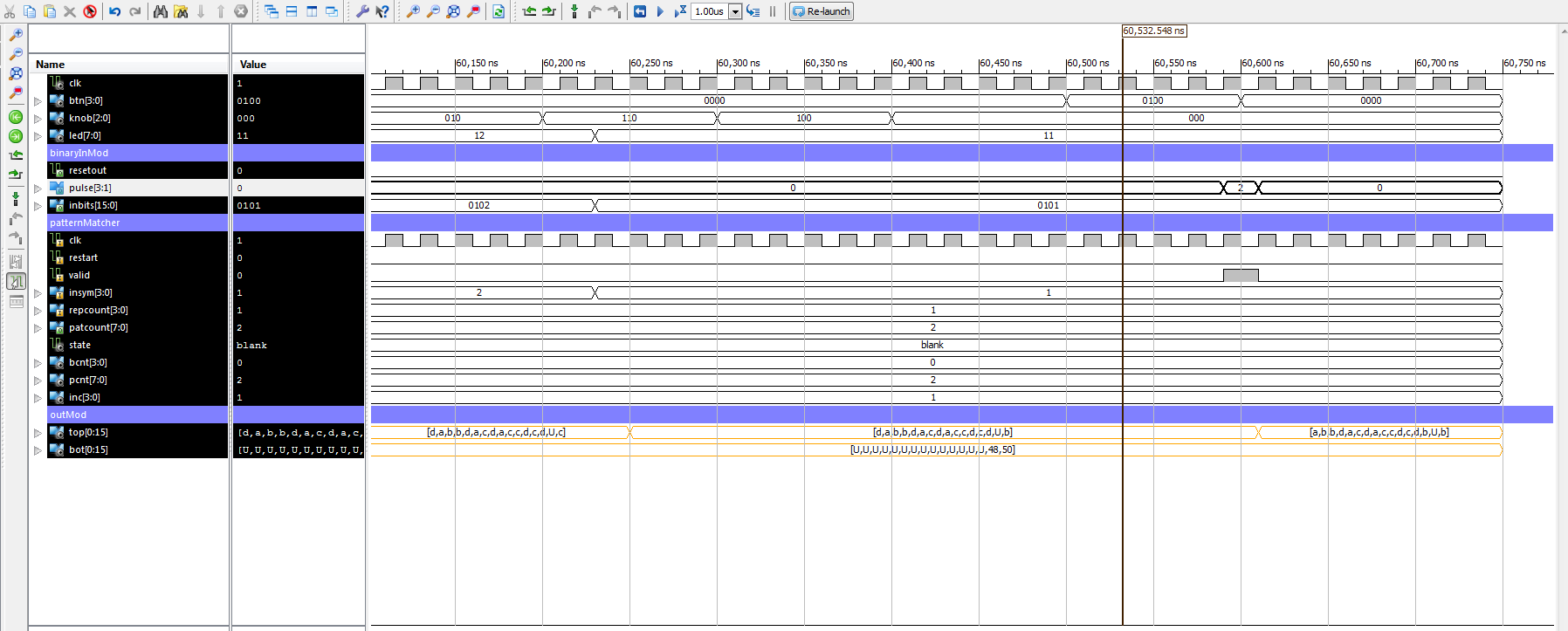
What are the first three patterns that are matched during this part of the simulation? At what times does the pattern matcher recognize that a match has occurred?

abbbd, acd, aacd

Each at 4.63 µs, 7.63 µs, 10.83 µs

Paste a screenshot below showing the final portion of your simulation, including the final values of the *top* and *bot* signals from *outMod*.

*paste your screenshot here*(p4)

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?

I will see in first row: abbdacdaccdcdb b

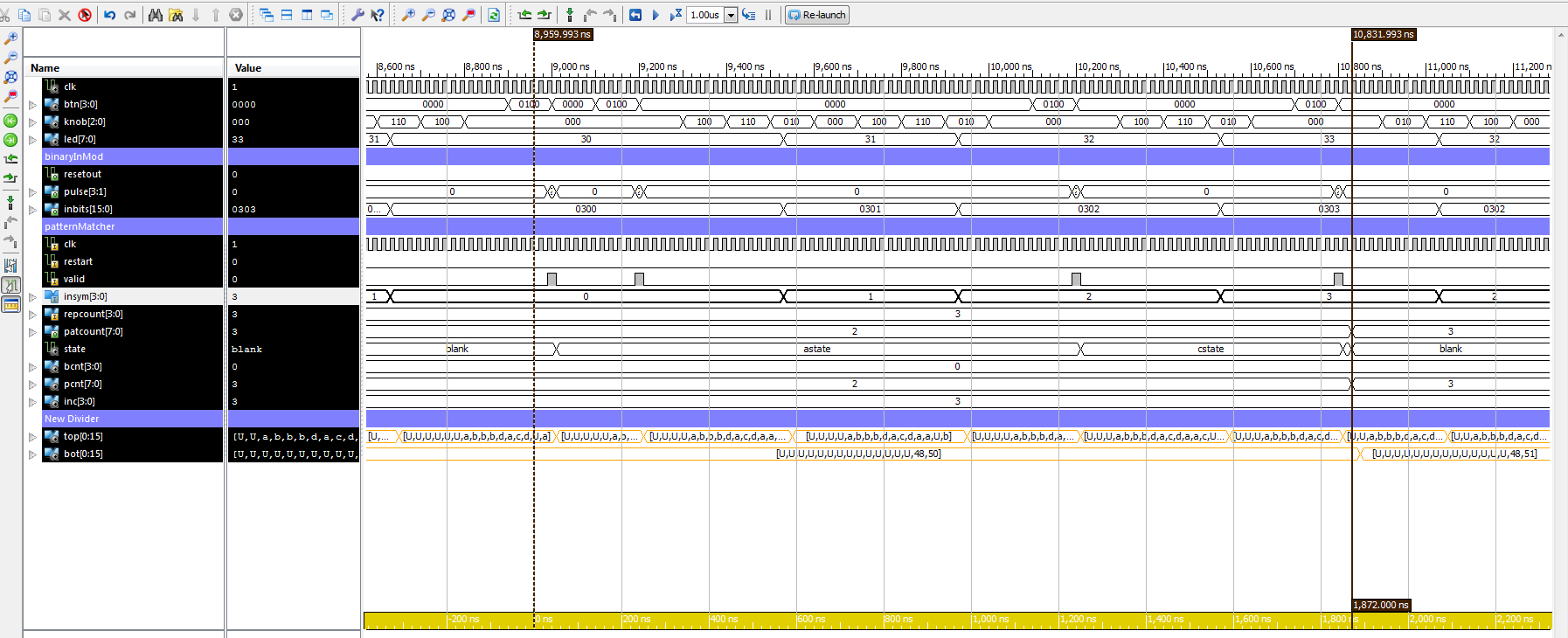
Second row: 48, 50

The first row means the historical inputs that fits in the first row.

The second row is the ASCII code for 0 and 2, which combines to 02, the number of counts of the patterns appeared since last restart.

Paste a screenshot below, showing a close-up view of a single pattern being matched.

*paste your screenshot here* (p5)



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?

Matched pattern: acd

Start from blank state, and follow the valid signal, we can get the current reading inSym. If the pattern count increases before switch back to blank state, then the pattern is matched. Value of LED: 3 in decimal.

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.

*paste your screenshot here*

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*.

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

***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 *commonDefs.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):