Massachusetts Institute of Technology

Department of Electrical Engineering and Computer Science

6.111 - Introductory Digital Systems Laboratory

Quiz 2 November 1, 2002

1	............(20)
2	............(20)
3	............(20)
4	............(40)
TOTAL	............(100)

NAME ....

Indicate Your Section

( )	James	12 PM
( )	Jennifer	1 PM
( )	Neira	3 PM

This quiz is **Closed Book:** Two handwritten "crib" sheets are allowed.

Put your name on all sheets and indicate your section on this page.

Write all your answers directly on the quiz.

Show all of your work.

You are not required to use a logic template, but you must **make sure your answers are legible**.

Introduction

Problems 2, 3 and 4 of this quiz are based on three vhdl programs which are attached to the back of the quiz. You may remove those pages from the quiz for ease in referring to them. They need not be turned in with the quiz.

Problem 1: Finger Exercises

Show, on Figure 1 how a shift-and-add multiplication of -3×5 would be carried out. In particular, you should show what the contents of the accumulator register would be after each step in the shift-and-add process. Assume that the multiplicand is in a left-shift register. Start with five bit wide representations of -3 and 5.

−3											
x 5							**0**	**0**	**1**	**0**	**1**
=											
+											
=											
+											
=											
+											
=											
+											
=											

Figure 1: Shift-and-add table for Problem 1

Problem 2: Timing

Refer to the program `easy.vhd`, the first of three. This takes a clock and two input signals a and b. It produces three output signals: x, y and z. A rudimentary timing diagram is shown in Figure 2, in which the input signals and clock are shown. Show the output signals. Remember: you don't know anything about the input signals before the start of the timing diagram.

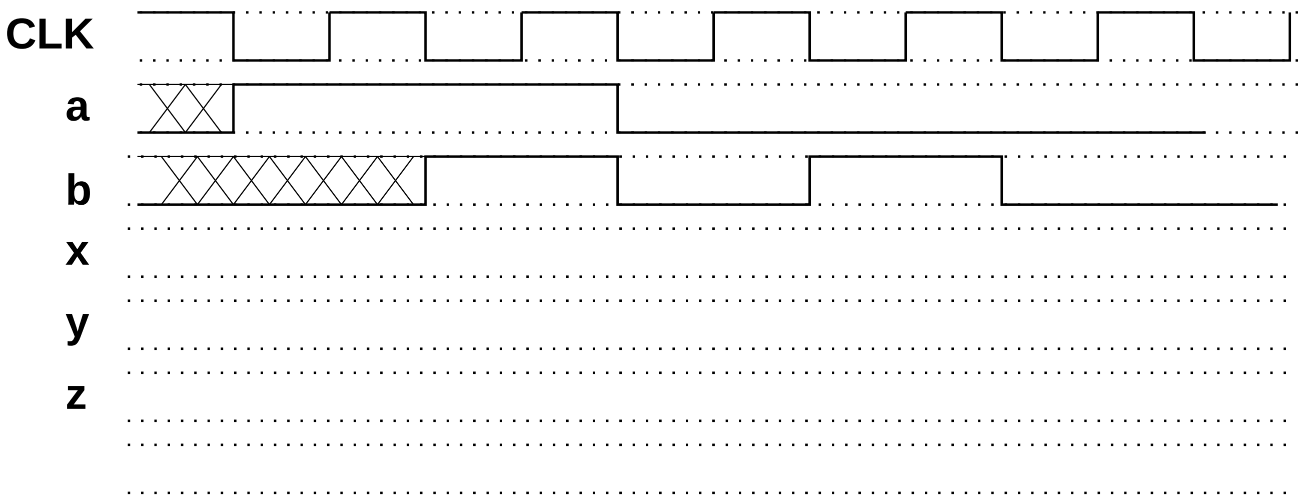

Figure 2: Timing Diagram for Problem 2

Problem 3: Unknown Machine

The second program attached to the back, `odd.vhd`, does something which you should be able to figure out.

1. What does this thing do? State its functioning in words.

2. Draw a block diagram that describes the structure of what this thing does. Show the important elements as MUXes, Registers, Tri-States, etc.

Problem 4: Problematic Lock

The third program describes a (probably faulty) digital lock. The design was inspired by a desire to discourage lock pickers. In a good example of perhaps faulty logic it was decided that, if the wrong combination was entered the lock should first warn the operator (beep) and then, if correction is not made immediately, it should blow up, thus making the would be lock picker very, very sorry. The lock is intended to be connected to an input consisting of an eight-bit number and an 'enter' key. A combination is hard-coded into the thing, but could obviously be changed just before compile time. To unlock, one enters the first number and 'enter', the second number and 'enter', the third number and 'enter' and the device should unlock. It has a provision that if a wrong number is entered it goes to an alarm state and causes an alarm to beep. At this point it must be re-set by entering all zeros. If this is not done, the next state causes a massive explosion (which is a good reason to be careful with this lock).

1. What is the combination?

2. In the handy-dandy form shown in Figure 3, draw the state transition diagram for this machine, showing all inputs and outputs.

3. A number of engineers have worked with this, but none have been able to give us a description of what happens. All we know is that each of their experiments have resulted in explosions. We suspect it may be related to the relatively high clock frequency, relative to the ability of the engineers to push buttons. What is happening?

4. You can fix the premature explosion problem by modifying the code. Write the new code here and indicate where it goes in the VHDL program.

5. Once the explosion issue is settled, it is noted that on occasion the thing unlocks even when a wrong number is entered. This is due to a 'glitch' arising from the crude way in which this thing was written. What is wrong?

6. Propose a 'fix' for this problem. Include any additional code that might be required and/or code that should be removed.

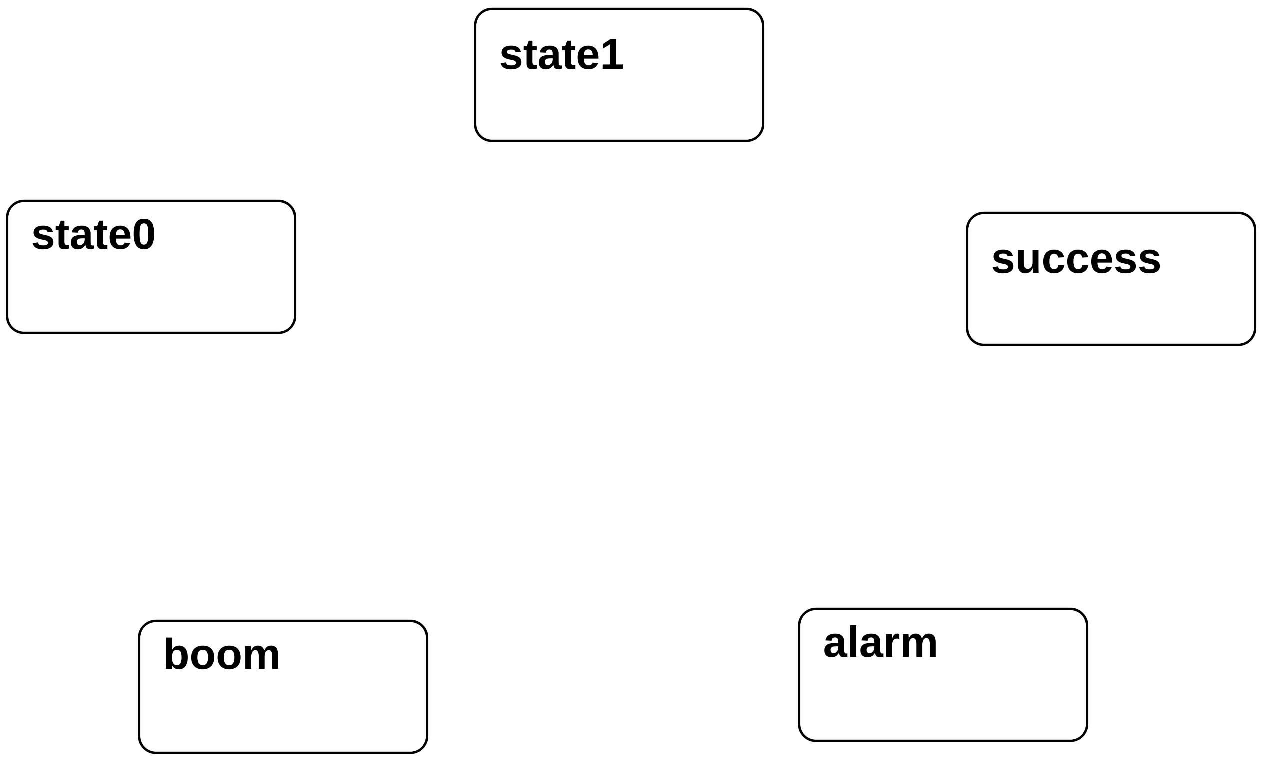

Figure 3: State Transition Diagram for Problem 3

This is lock.vhd Keep this with your quiz and turn it in: please use it to indicate where code modifications must be placed.

```
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
entity diglock is port
                (
                 innum                  : in std_logic_vector(7 downto 0);
                 enter, clk             : in std_logic;
                 beep, boom, unlok      : out std_logic);
end diglock;

architecture state_machine of diglock is
        type StateType is (state0, state1, success, alarm, boom);
        signal p_s, n_s : StateType;
        signal first    : std_logic_vector(7 downto 0) :='00000001';
        signal second   : std_logic_vector(7 downto 0) :='00000101';
        signal third    : std_logic_vector(7 downto 0) :='00001010';
        signal zeros : std_logic_vector(7 downto 0) := others => '0';
        attribute enum_encoding of StateType is : "000 001 011 010 100";
begin
        unlok <= n_s(0) and n_s(1);     -- detect 011 (don't need bit 2)
        beep <= (not n_s(0)) and n_s(1);-- detect 010
        fsm: process (p_s, innum)
        begin
                case p_s is
                        when state0 =>
                                if innum = first then
                                        n_s <= state1;
                                else
                                        n_s <= alarm;
                                end if;
                        when state1 =>
                                if innum = second then
                                        n_s <= success;
                                else
                                        n_s <= alarm;
                                end if;
                        when success =>
                                        n_s <= state0;
                        when alarm =>
                                if innum = zeros then
                                        n_s <= state0;
                                else
```

```
                                        n_s <= boom;
                                end if;
                        when boom =>
                          boom = '1';   -- don't care about next state!
                        when others => n_s <= state0; -- avoid trap states
                end case;
        end process fsm;
        state_clocked   : process (clk)
                begin
                        if rising_edge(clk) then
                          if enter = '1' then
                            p_s <= n_s;
                          end if;
                        end if;
                end process state_clocked;
end architecture state_machine;
```

This is easy.vhd

```
library ieee;
use ieee.std_logic_1164.all;
entity reg is

  port (
    a, b, clk : in std_logic;
    x, y, z   : out std_logic);

end reg;
architecture top of reg is
signal c : std_logic;
begin  -- top
y <= x and b;
z <= c and b;
reg2:  process (clk)
begin  -- process
  if rising_edge(clk) then
    c <= a;
    if b='1' then
      x <= a;
    end if;
  end if;
end process;


end top;
```

This is odd.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
entity unknown is

  port (
    clk, we        : in  std_logic;
    rdlsel, wdlsel : in  std_logic_vector(1 downto 0);
    wdata          : in  std_logic_vector(7 downto 0);
    rdlout         : out std_logic_vector(7 downto 0));
end unknown;

architecture whatisthis of unknown is
  signal zero, one, two, three : std_logic_vector(7 downto 0);
begin  -- whatisthis
  clk_process: process (clk)
  begin  -- process
    if clk'event and clk = '1' then
      if we = '1' then
        if wdlsel = "00" then
          zero <= wdata;
        elsif wdlsel = "01" then
          one <= wdata;
        elsif wdlsel = "10" then
          two <= wdata;
        else
          three <= wdata;
        end if;
      end if;
    end if;
    case rdlsel is
      when "00"=> rdlout <= zero ;
      when "01" => rdlout <= one;
      when "10" => rdlout <= two;
      when "11" => rdlout <= three;
      when others => rdlout <= "00000000";
    end case;
  end process clk_process;
end whatisthis;
```

This is lock.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
entity diglock is port
                (
                  innum                 : in std_logic_vector(7 downto 0);
                  enter, clk            : in std_logic;
                  beep, boom, unlok     : out std_logic);
end diglock;

architecture state_machine of diglock is
        type StateType is (state0, state1, success, alarm, boom);
        signal p_s, n_s : StateType;
        signal first    : std_logic_vector(7 downto 0) :='00000001';
        signal second   : std_logic_vector(7 downto 0) :='00000101';
        signal third    : std_logic_vector(7 downto 0) :='00001010';
        signal zeros : std_logic_vector(7 downto 0) := others => '0';
        attribute enum_encoding of StateType is : "000 001 011 010 100";
begin
        unlok <= n_s(0) and n_s(1);     -- detect 011 (don't need bit 2)
        beep <= (not n_s(0)) and n_s(1);-- detect 010
        fsm: process (p_s, innum)
        begin
                case p_s is
                        when state0 =>
                                if innum = first then
                                        n_s <= state1;
                                else
                                        n_s <= alarm;
                                end if;
                        when state1 =>
                                if innum = second then
                                        n_s <= success;
                                else
                                        n_s <= alarm;
                                end if;
                        when success =>
                                        n_s <= state0;
                        when alarm =>
                                if innum = zeros then
                                        n_s <= state0;
                                else
                                        n_s <= boom;
                                end if;
                        when boom =>
                          boom = '1';   -- don't care about next state!
                        when others => n_s <= state0; -- avoid trap states
                end case;
        end process fsm;
        state_clocked   : process (clk)
                begin
                        if rising_edge(clk) then
                          if enter = '1' then
                            p_s <= n_s;
                          end if;
                        end if;
                end process state_clocked;
end architecture state_machine;
```