

Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 - Introductory Digital Systems Laboratory

Quiz 2 Solutions

April 12, 2002

1(30)
2(40)
3(30)
TOTAL(100)

NAME

Indicate Your Section

- James Oey 12 PM
- Rajul Shah 1 PM
- Cynthia Chow 2 PM
- Jennifer Maurer 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**.

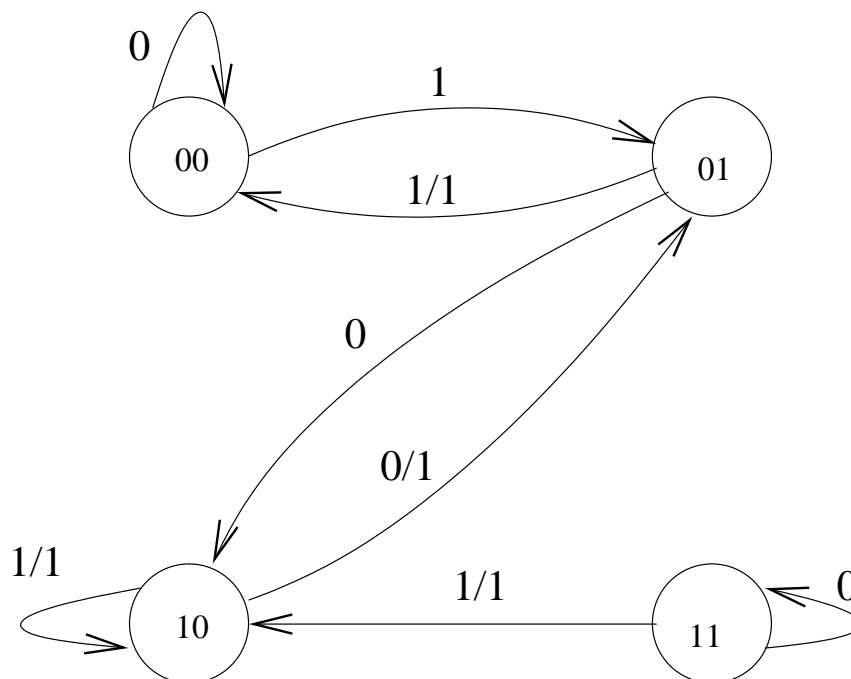
Problem 1 (30 points)

Consider the FSM specified by the following table. As usual, all transitions are initiated by the rising edge of a single clock, CLK.

State Name	Q1	Q0	x (in)	D1	D0	z (out) (should have been y!)
S0	0	0	0	0	0	0
S0	0	0	1	0	1	0
S1	0	1	0	1	0	0
S1	0	1	1	0	0	1
S2	1	0	0	0	1	1
S2	1	0	1	1	0	1
S3	1	1	0	1	1	0
S3	1	1	1	1	0	1

Problem 1a (15 points)

Draw a state transition diagram showing the states Q_1Q_0 , and the transitions between the states. Use the template shown below. Note that the states are labeled such that 01 means $Q_1 = 0$ and $Q_0 = 1$.



Problem 1b (15 points)

Correct the architecture for the following VHDL code. Syntax is not important, i.e., we will not take off for a missing semicolon, etc. However, meaning is important.

We suspect that the output is not right. If we are wrong, give us an argument as to why the output is always one. If we are right, correct the output using a concurrent Boolean assignment statement.

That must not be the only thing wrong as the file doesn't compile. Provide anything that is missing and should be there to get the file to compile correctly and implement the FSM. Hint: Nothing shown is wrong except, perhaps, for the output specification.

```

library ieee;
use ieee.std_logic_1164.all;
entity what is port (
  x, clk : in std_logic;
  y       : out std_logic);
end what;
architecture state_machine of what is
-- You decide that Q1 = p_s(1) and Q0 = p_s(0)
  signal p_s, n_s : std_logic_vector(1 downto 0);
  constant S0 : std_logic_vector(1 downto 0) := "00";
  constant S1 : std_logic_vector(1 downto 0) := "01";
  constant S2 : std_logic_vector(1 downto 0) := "10";
  constant S3 : std_logic_vector(1 downto 0) := "11";
begin
  state_clocked:process(clk) -- register
  begin
    if rising_edge(clk) then p_s <= n_s;
    end if;
  end process state_clocked;
-- combinational output specification
-- This isn't right. Please fix it>
  y <= (p_s(1) AND NOT p_s(0)) OR (p_s(0) AND x);
-- combinational next state specification
  with p_sx select          -- The problem is that p_sx hasn't been declared
  n_s <= S0 when "000", -- or specified. Perhaps the simplest solution
        S1 when "001", -- is to change p_sx to p_s & x in this statement.
        S2 when "010", -- Alternately, one could declare it along with
        S0 when "011", -- the other signals declared in the architecture
        S1 when "100", -- and then include the statement
        S2 when "101", --   p_sx <= p_s & x;
        S3 when "110", -- somewhere in the architecture
        S2 when "111", -- (but not in the processs).
        S0 when others;
end architecture state_machine;

```

Problem 2 (40 points)

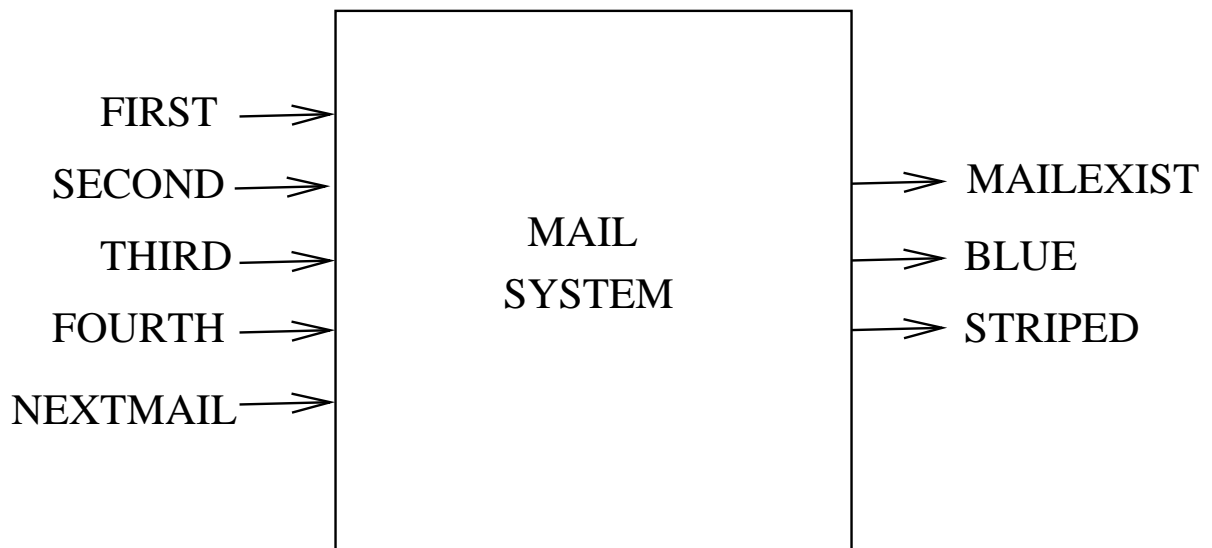
The living group where you reside has just hired you to sort mail. You quickly realize that you have a system to identify and deliver mail. Unfortunately, this system still requires human intervention to tell it where mail should be delivered. Since you don't want to spend all your time sorting mail, you design a controller for this system using an MCU.

The incoming mail has two different characteristics: color and pattern. The color is either blue or green and the pattern is either striped or solid. The mail should be delivered to separate floors as follows:

1st floor - blue striped
2nd floor - blue solid
3rd floor - green striped
4th floor - green solid

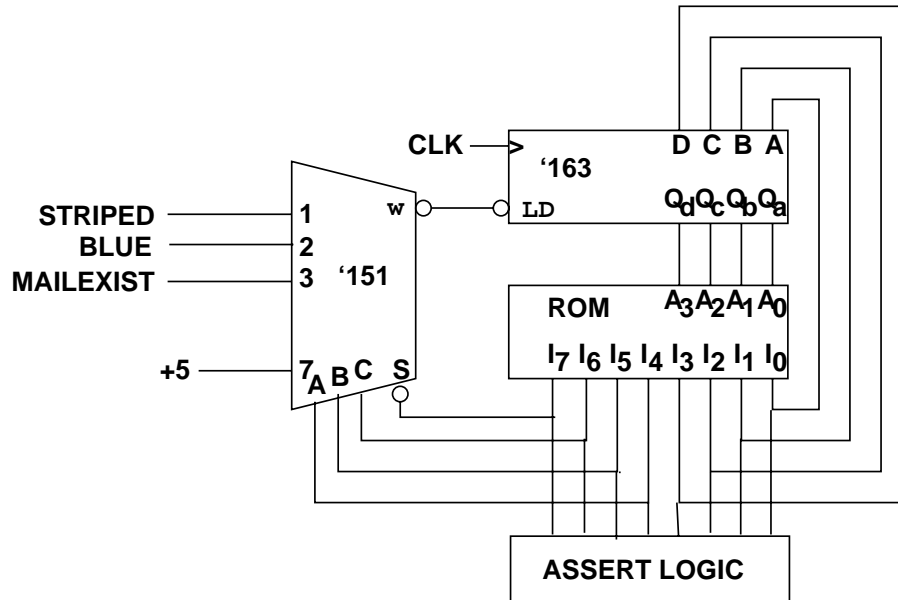
Details of your mail system (shown below) are: Your system has an input, NEXTMAIL, which starts your system. If there is mail to sort, then your system asserts MAILEXIST and is ready to deliver the mail. However, if there is no mail, then MAILEXIST is not asserted and no delivery takes place.

Once NEXTMAIL is asserted then the system outputs named BLUE and STRIPED are valid (asserted high or low depending on the mail's characteristics). When an input FIRST, SECOND, THIRD, or FOURTH is asserted then the mail currently being sorted is transported to the appropriate floor.



Problem 2 (continued) (40 points)

The MCU used to control your mail system is shown below. An extract of the data sheet for the '151 is on the next page.

**Problem 2a** (15 points)

Complete the mail.sp file for the controller.

```

op<7:0>;
address op<3:0>;

cjmp    op< 7 > = 0      ; /* answer here */
assert  op< 7 > = 1      ; /* answer here */
jmp     op<7:4 > = %b0111; /* answer here */ /* or %h7 or simply 7 */

first    op<0> = 1;
second   op<1> = 1;
third    op<2> = 1;
fourth   op<3> = 1;
nextmail op<4> = 1;

mailexist op< 6:4 > = 3 ; /* answer here */
blue      op< 6:4 > = 2 ; /* answer here */
striped   op< 6:4 > = 1 ; /* answer here */

```

Problem 2b (15 points)

Finish the mail.as file for your controller.

```
#SPEC_FILE = mail.sp;
#LIST_FILE = mail.lst;
#SET_ADDRESS = 0;
#LOAD_ADDRESS = 0;

NEXT:   ASSERT NEXTMAIL;           /* get next mail */
        CJMP MAILEXIST MAIL;       /* is there actual mail? */
        JMP NEXT;                  /* if not, wait for mail */
MAIL:   CJMP BLUE BLUEMAIL;        /* start checks for mail types */
        CJMP STRIPED STRIPEDMAIL;
        ASSERT FOURTH;             /* neither */
        JMP NEXT;
BLUEMAIL: CJMP STRIPED BLUESTRIPEDMAIL;
        ASSERT SECOND;             /* BLUE SOLID */
        JMP NEXT;
STRIPEDMAIL: ASSERT THIRD          /* GREEN STRIPED */
        JMP NEXT;
BLUESTRIPEDMAIL: ASSERT FIRST     /* BLUE STRIPED */
        JMP NEXT;

/* addresses BLUEMAIL, STRIPEDMAIL. and BLUESTRIPEDMAIL can appear in
   any order */
```

Problem 2c (10 points)

Hand assemble the first three instructions (provide the ROM encoding in binary). Assume that the first line is at address 0x00.

```
NEXT:   ASSERT NEXTMAIL;          1 0 0 1 0 0 0 0
        CJMP MAILEXIST MAIL;      0 0 1 1 0 0 1 1
        JMP NEXT;                 0 1 1 1 0 0 0 0

MAIL:   CJMP BLUE BLUEMAIL;
```

Problem 3 (30 points)

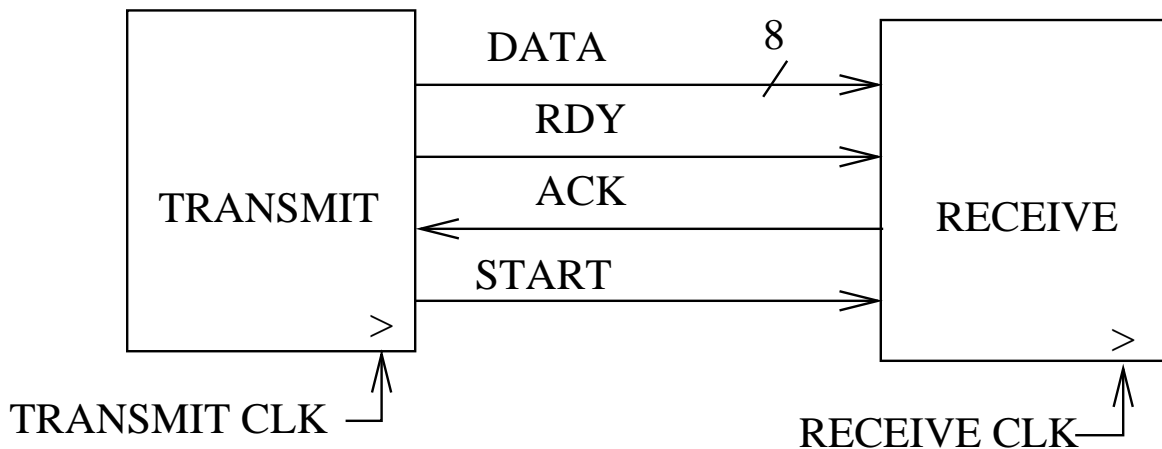
You have been working hard on your project. Hooray! Your part works. Now you just have to ship the data to your partner's kit and she can display the data. You now realize that it would have been a good idea to talk to her about this before hand, but you didn't.

Luckily you only have to ship 512 8-bit bytes of data every minute.

Eventually you are to design both the transmit side and the receive side of a system to transfer 8-bit bytes of data from your kit to her kit. For now, you will concentrate on the interface between the two sides and on the design of the receive side so your partner can get to work on it. Hopefully she will have room to implement the receive circuit. But, will she know when you start a block of data?

Problem 3a (15 points)

Label all the signals that need to go between the kits on the block diagram below. Write an English description of each signal in the space below the figure. You are NOT allowed to ship her a clock signal that she must use instead of her local clock.

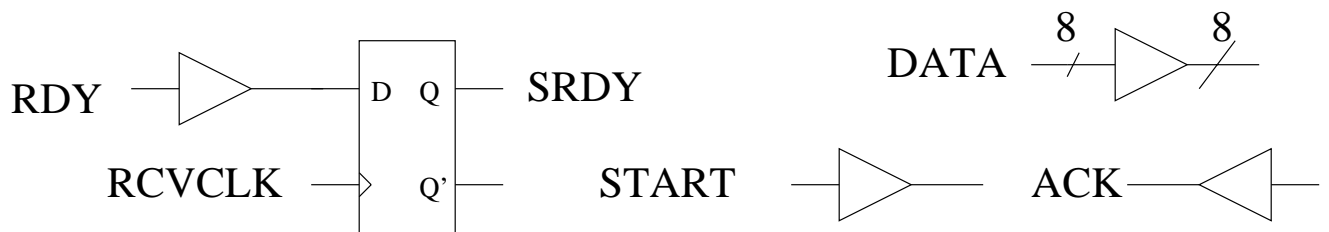


- DATA – Eight DATA bits
- RDY – DATA is ready to sample.
- ACK – I took the DATA already.
- START – The next byte is byte zero.

Problem 3b (15 points)

Provide a design for the receive block. You may use FSMs, flip-flops, registers, any combinational logic you want. However, your design has to be clear to us. After all, it has to be clear to your project partner.

Who knows what your partner will do with each data point as it is received. The FSM waits until her “used” signal is asserted and then gets the next byte of data.



The buffers are optional, although a good idea so that noise is not coupled into a signal that one uses on ones kit.

