

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

Quiz 2

Friday, April 13, 2001

1 (20)
 2 (20)
 3 (30)
 4 (30)
 TOTAL (100)

NAME

Indicate Your Section

- Danny Seth 12 PM
 Peter Agboh 1 PM
 Todd Hiers 2 PM
 Brian Perrin 3 PM

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

Put your name on this page, all loose 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 (20 points)

What is this?

Here is a VHDL file. Sketch the circuit that the code implements. Note, you are NOT to sketch an optimized equivalent circuit to the one described by the VHDL code. Note also that this problem is not intended to be tricky.

```

library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;  -- needed for integer + signal
entity alu is
  port(cin      : in std_logic;
        a, b    : in std_logic_vector(1 downto 0);
        alu_ctl : in std_logic_vector(1 downto 0);
        c       : out std_logic_vector(2 downto 0));
end alu;

architecture what_is_this of alu is
  signal a_int, b_int, c_int : std_logic_vector(3 downto 0);
  constant add  : std_logic_vector(1 downto 0) := "00";
  constant sub  : std_logic_vector(1 downto 0) := "01";
begin
  a_int <= '0' & a & cin;
  b_int <= '0' & b & cin;
  small_alu: process(a_int, b_int, cin, alu_ctl)
  begin
    case alu_ctl is
      when add    => c_int <= a_int + b_int;
      when sub    => c_int <= a_int - b_int;
      when others => c_int <= (others => '-');  -- '-' is a don't care
    end case;
  end process small_alu;
  c <= c_int(3 downto 1);
end architecture what_is_this;

```

Space for your answer to problem 1

Problem 2 (20 points)

I wish I had the whole spec file!

Here is a partially complete specification file, p1.sp:

```

op          <7:0>;
address op <2:0>;
outa  op <   > =   ; /* answer */
outb  op <   > =   ; /* answer */
outc  op <   > =   ; /* answer */
inpx  op <   > =   ; /* answer */
assert op <7:6> = 2;
jsr   op <7:6> = 3;
rts   op <7:6> = 0;
goto  op <7:6> = 1  op <5:3> = %b111;
if    op <7:6> = 1;
jmp   nop;

```

And here is an assembler file, p1.as, associated with the above p1.sp file:

```

#SPEC_FILE      = p1.sp;
#LOAD_ADDRESS   = 0;
#SET_ADDRESS    = 0;

start: assert outb outc;
        jsr sub1;
        assert outb;
        goto start;
sub1:   assert outa;
loop:  if inpx jmp loop;
        rts;

```

And finally, here are the 7 bytes of output produced by compiling the above assembler file, p1.as, except they are *****OUT OF ORDER!*****

```
00000000  10000001  01010101  10000100  01111100  11000100  10000011
```

Given all of the information, please fill in the blanks in p1.sp

Problem 3 (30 points)

Odd Counter Implementation

It's the day before your final project is due. Almost everything is working. All you need is one more 4-bit adder, and you'll be done.

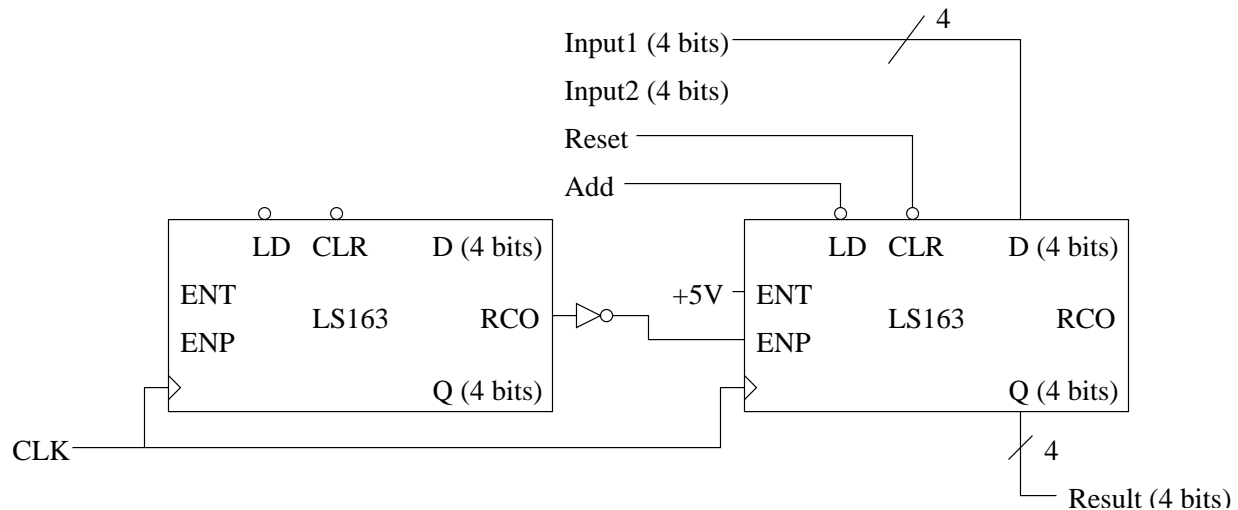
Unfortunately, you're out of adder chips. And the equipment desk refuses to give you any more. In fact, all you have are two LS163s, and some combinational logic.

You've come up with a way to create the adder from counters. One of the counters loads in the first number, and then counts once for each clock pulse the second counter hasn't maxed out. The other counter loads in the second number and counts down that number of clock pulses.

We've wired up the first counter (from the description above) for you. Wire up the second counter. You can use any simple combinational logic you need to (AND, OR, NOT, NOR, NAND). Reset is the clear signal, asserted low. Add tells the counters to begin adding, using the values at the two inputs, and is asserted low. Input1 and Input2 are the two numbers to add.

Hint: An up counter (like the LS163) can count down if you invert the inputs.

N.B.: Make sure the second counter (the one you're wiring) will not allow the first counter to count once it's done adding. The result may not be taken immediately, so the result should be held indefinitely.



In the worst case, how many clock ticks after Add goes high do you need to wait for this adder to produce a correct result?

Problem 4 (30 points)

Stupid MCU Tricks

You wish to make a system using an MCU that can do some actions based on the timing of signals. Specifically, you want an MCU that takes two synchronized inputs, A and B. If A or B goes high and the other follows within at most 3 clock cycles, the MCU should assert toggle. If A goes high and B does not within 3 cycles, the MCU should assert reset. If B goes high and A does not within 3 cycles, the MCU should assert set. Once A and B are both '0' again, another set, reset, or toggle can occur.

You have an MCU made up of a sequencer, an input multiplexer, a ROM, and an assertion pal. You do not, however, have any timer circuitry. You must implement this behavior using only a standard 6.111 MCU.

Your code should automatically re-cycle (i.e. after a set, reset, or toggle, be prepared to catch additional edges) You may assume A and B stay in the low or high state for many MCU clock cycles. You may also assume that A and B will both return to '0' before the next set, reset, or toggle.

```

/* mcu.sp - PROVIDED FOR REFERENCE - THIS CODE IS COMPLETE */
/* specification file of the MCU */
op <7:0>;
address op <3:0>;
/* Instruction codes for the MCU:
* ASSERT indicates that signals should be asserted
* CJMP indicates a jump to an address if the condition is true
* JMP indicates a jump to an address regardless of any conditions */
ASSERT op<7>=%b1;
CJMP op<7>=%b0;
JMP op<7:4>=%b0111;
/* Assertion signals for the ASSERT statements. */
SET op<2>=%b1;
RESET op<1>=%b1;
TOGGLE op<0>=%b1;
/* Conditional signals for use in a CJMP statement. */
A op<6:4>=0;
B op<6:4>=1;
TRUE op<6:4>=7;
/* These are meaningless signals added for readability. */
IF nop;
THEN nop;
/* end of mcu.sp */

```

Finish the assembly code started below.

```
/* mcu.as */  
/* assembly file of the MCU */
```

```
#SPEC_FILE = mcu.sp;  
#LIST_FILE = mcu.lst;  
#MASK_COUNT = 8;  
#SET_ADDRESS = 0;  
#LOAD_ADDRESS = 0;
```

```
/* FILL IN CODE HERE */
```

```
/* end of mcu.as */
```