

Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

6.111 – Introductory Digital Systems Laboratory

Problem Set 5

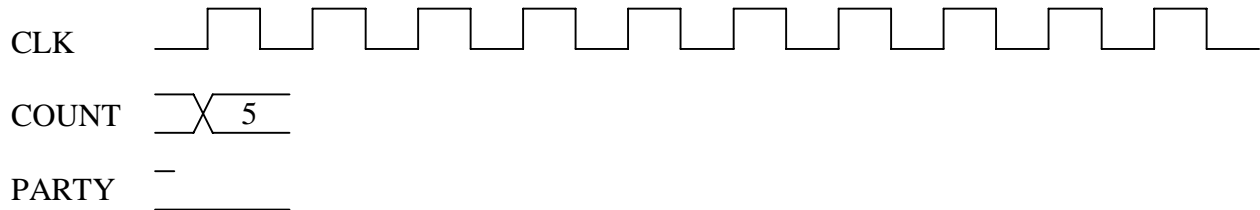
Issued: March 8, 2000

Due: March 17, 2000

Problem 1: Millennium Countdown!

Alyssa P. Hacker has big plans for the real Millennium party this coming New Year’s Eve. In order to set the mood right, she has asked four of her 6.111 friends to design a VHDL entity to countdown the Millennium and set a “party” signal high for exactly one clock period once the count reaches zero.

For each of the four approaches suggested to Alyssa, either determine that the architecture is invalid due to conflictive statements, or provide a timing diagram if the architecture is otherwise valid. Which architecture(s) sets the “party” signal high for exactly one clock period when the count reaches zero?



```
architecture approach_one of millennium is
begin
  timer: process(clk)
  begin
    if rising_edge(clk) then
      if count = "001" then
        party <= '1';
      end if;
      count <= count - 1;
    end if;
  end process timer;
end approach_one;
```

```
architecture approach_two of millennium is
begin
  timer: process(clk)
  begin
    if rising_edge(clk) then
      if count = "001" then
        party <= '1';
      else
        party <= '0';
      end if;
      count <= count - 1;
    end if;
  end process timer;
end approach_two;
```

```
architecture approach_three of millennium is
begin
  timer: process(clk)
  begin
    if rising_edge(clk) then
      party <= '0';
      if count = "001" then
        party <= '1';
      end if;
      count <= count - 1;
    end if;
  end process timer;
end approach_three;
```

```
architecture approach_four of millennium is
begin
  party <= '0';
  timer: process(clk)
  begin
    if rising_edge(clk) then
      if count = "001" then
        party <= '1';
      end if;
      count <= count - 1;
    end if;
  end process timer;
end approach_four;
```

Problem 2: The Zephyr Pager

As part of an “interactive” campus project, students are required to wear pagers at all times. The pagers have a tiny screen that delivers all kinds of messages: changes in lab due dates, LSC’s movie schedule, and even (gasp) the daily menu at Lobdell. Students are completely controlled by this messaging.

The small firm you work for has been hired to produce the next generation of Zephyr pagers. Looking over the specification sheet, you find that Zephyr pagers operate in the following (unique) fashion:

1. Wait for a Zephyr message.
2. If a message is received and the student is awake, buzz loudly until he/she presses the “acknowledge” button. After the button is pressed, the pager should display the message (perhaps in some unreadably small font) and wait for the next message.
3. If a message is received and the student is asleep, repeatedly deliver a 10,000V (low current) shock to the student until he/she presses the “acknowledge” button. After the button is pressed, display the message and wait for the next message.

Your control circuitry should be set up to handle the following inputs and outputs:

Inputs	Description
message	Specifies when a message has been received
asleep	True if the student is asleep (e.g., the pager is horizontal)
acknowledge	Activated when the student presses the button

Outputs	Description
buzz	Activates the buzzer for a short time
shock	Delivers a small shock to the student
display	Latches a received message onto the screen

Part A. Draw a state diagram for the pager.

Part B. Write a VHDL file that implements the pager as an FSM. Turn in your `zephyr.vhd` file.

Part C. Implement the zephyr pager using a micro-controller design as follows:

1. Using a ‘151, ‘163, EEPROM, 20v8 PAL, and some random logic, draw a basic circuit diagram implementing the MCU. Exact details are not required, but it should be clear how the system is intended to work. Also, provide a reset signal that clears your address counter to 0, provide an input that always returns true for your ‘151, and make the data path out of the EEPROM be 8 bits.
2. Write out a specification file for the micro-controller. Many examples are available in `/mit/6.111/prom/examples` In this case, your instruction word should be formatted as follows:

Instructions	I7	I6	I5	I4	I3	I2	I1	I0
IF <i>cond</i> THEN <i>address</i>	0	C2	C1	C0	A3	A2	A1	A0
ASSERT <i>signal</i>	1	S6	S5	S4	S3	S2	S1	S0

3. Write the assembler file for your micro-controller. Assume that asserting a signal in a loop is functionally equivalent to the signal being continually high. Compile the code using “assem”.

Turn in your `zephyr.sp`, `zephyr.as`, and `zephyr.lst` files for Part C.