CS226 Practice Problem Set 2 (Spring 2016)

Date posted: Feb 7, 2016 **Expected Solving Time: 2 hours**

- *If you need to make any assumptions, state them clearly.*
- ***These are ungraded practice questions. You are strongly encouraged to solve these on your own to ensure you understand the material being taught in class.***
- ***Mutual discussion is allowed, but copying is not. Please read the guidelines on the course webpage if you don't understand the distinction between the two.***

1. Consider a datapath block with three n-bit integers A, B, and X as inputs, and a single $n+1$ bit integer Z as output. The function implemented by this datapath block is as shown in the pseudo-code below. You are required to implement this datapath block using a single n-bit comparator, a single n-bit adder and at most two 2-to-1 n-bit multiplexers. You are not allowed to use any other logic gate or sub-circuit.

```
if $A < B$ then
  $Z = X + A$
else if $A = B$ then
  $Z = X + B$
else
  $Z = A + B$
end if
```

   Note that an n-bit comparator has two n-bit inputs In_1 and In_2, and three 1-bit outputs, C_1, C_2, and C_3. It treats all its inputs as unsigned integers. The outputs of the comparator are determined as follows: if $In_1 < In_2$, $C_1 = 1$; if $In_1 = In_2$, $C_2 = 1$; if $In_1 > In_2$, $C_3 = 1$. For any In_1 and In_2, only one of the comparator outputs can be 1 at a time; the other two outputs must be 0. An n-bit adder takes as inputs two n-bit numbers p and q, and produces an $n+1$-bit output *sum*, which is the result of adding p and q, treated as unsigned integers. A 2-to-1 n-bit multiplexer takes two n-bit inputs In_0 and In_1, and a single bit input *sel*. It has an n-bit output *out* that is set equal to In_0 if $sel = 0$; otherwise *out* is set equal to In_1.

2. Give as simple Boolean functions as you can from the following K-maps:

(a)

X→ / Y↓	**0**	**1**
0	1	1
1	0	1

(b)

X,Y→ / Z↓	**00**	**01**	**11**	**10**
0	1	1	0	1
1	1	0	0	1

(c)

X,Y→ / Z,W↓	**00**	**01**	**11**	**10**
00	1	1	1	1
01	1	1	1	1
11	1	1	0	1
10	1	1	1	1

(d)

X,Y→ / Z,W↓	**00**	**01**	**11**	**10**
00	1	1	1	1
01	0	d	d	0
11	0	1	d	1
10	0	1	1	1

where "d" denotes don't-care.

3. A long message (stream of bits) is being sent from a transmitter to a receiver over a channel. The length of the message is not known a priori. The received message is said to be *valid* if the number of 1's in it is a multiple of 3, *and* if the message ends with a sequence of four consecutive 0's. Thus, a valid message is 110101110000 (has six 1's and ends with four 0's), while 11010110000 (has five 1's and ends with four 0's) and 110101010100 (has six 1's and doesn't end with four 0's) are not valid messages. You are required to design an end-of-valid-message (EOVM) indicator circuit to be used by the receiver. The EOVM indicator has a 1-bit input In through which the bit-stream comes in, one bit per clock cyclle. It produces a 1-bit output Out in every clock cycle. The output is required to become 1 as soon as the end of a valid message is detected; otherwise it stays at 0.

   (a) Design a controller for implementing the EOVM indicator. For this subquestion, you may assume that only a single valid message is being transmitted. So, once the EOVM indicator outputs 1, you can choose to keep it at 1 until the circuit is reset.

   (b) Now modify the above controller to take care of the situation where multiple messages are being sent, one after another. In this case, we'd like the EOVM indicator to output 1 when the end of the first valid message is detected. Subsequently, it should output 1 again when the end of the second valid message is detected and so on. For example, the following table shows asequence of bits received on In and the corresponding desired sequence of bits on Out.

ClkCycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
In	1	1	1	0	0	0	0	1	1	1	0	1	1	0	1	0	0	0	0
Out	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1

   Note that the first valid message starts in cycle 1 and ends in cycle 7; the second valid message start in cycle 8 and ends in cycle 19.

   In both the sub-problems above, indicate what are the reset states of registers/flip-flops in your design.

4. (a) In digital systems, it is often necessary to have circuits that can shift the bits of a vector by one or more bit positions to the left or right. Design a circuit that can shift a four-bit vector $W = w_3w_2w_1w_0$ one bit position to the right when a control signal $Shift$ is equal to 1. Assume that the circuit outputs a four-bit vector $Y = y_3y_2y_1y_0$ and a one-bit value k, such that if $Shift = 1$ then $y_3 = 0, y_2 = w_3, y_1 = w_2, y_0 = w_1, andk = w_0$. If $Shift = 0$ then $Y = W$ and $k = 0$.

   (b) The shifter circuit in the above problem shifts the bits of an input vector by one bit position to the right. It fills the vacated bit on the left side with 0. A more versatile shifter circuit may be able to shift by more bit positions at a time. If the bits that are shifted out are placed into the vacated positions on the left, then the circuit effectively rotates the bits of the input vector by a specified number of bit positions. Such a circuit is often called a barrel shifter. Design a four-bit barrel shifter that rotates the bits by 0, 1, 2, or 3 bit positions as determined by the valuation of two control signals s_1 and s_0.

5. Consider the following pseudo-code for unsigned binary multiplication. Here, A and B are two n-bit unsigned integers that must be multiplied. The product is computed in a $2n$-bit register P.

```
$i = n - 1$;
$P = 0$;
while true do
  $P = P << 1$;
  if ($A[i] == 1$) then
    $P = P + B$;
  end if
  if ($i \neq 0$) then
```

```
        $i = i - 1$;
      else
        Stop;
      end if
    end while
```

Design a circuit (controller + datapath) to implement the above multiplication algorithm. You may assume $n = 128$ for this problem. You may assume access to a common adder/subtractor datapath block that implements addition if a special control input *doAdd* is set to 1; otherwise, it implements subtraction. You may also assume access to a shifter of the type discussed in the first part of the previous question.

Identify the number of registers (along with their size) and multiplexers required.

6. By now, you must have figured out that NAND gate is a universal gate (any Boolean logic circuit can be constructed using only NAND gates). Find out the minimum number of 2-input NAND gates required to implement:

   (a) Half adder

   (b) Full adder

   (c) (A'+B')(C+D)

7. Consider the function: F (W, X, Y, Z) = W'X'Y'Z' + W'X'Y'Z + W'X'YZ + W'XY'Z' + WX'Y'Z' + WX'Y'Z + WXYZ Implement the above function using only 8×1 multiplexers

8. An n-bit parallel binary adder is implemented using a Half Adder at the least significant bit (LSB) position. Full adders are used at all the remaining bit positions. Suppose that whenever required, a maximum of 2-level logic circuits are used. Assume also that the propogation delay of each logic gate is d. Calculate the worst-case time it takes for two n-bit integers to be added by a parallel binary adder in terms of n and d. How would you go about designing an adder circuit that minimizes this time?