Your Name: ___________________________________	netid:________________	Group #:
Name: _______________________________________	netid:________________	
Name: _______________________________________	netid:________________	
Name: _______________________________________	netid:________________	

ECE 120 Worksheet 6: Bit-sliced design

Many operations on groups of bits can be broken down into operations on individual bits, sometimes with a small amount of information from operations on other bits. For these operations, we can build a circuit that performs the required operation on individual operand bits and then make as many copies of this circuit as needed to perform the operation on the group of bits. We call this approach a bit-sliced design.

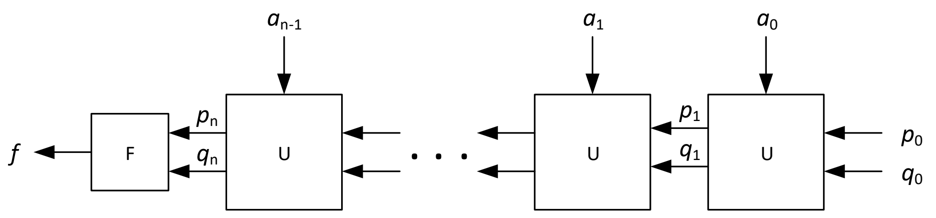
Let's consider some examples. Calculating one bit of the output for a bitwise logic operation such as AND depends only on one bit from each of the input operands. An AND gate suffices for each such calculation, and no information need flow from one AND gate to the next, since the results are independent of one another. In a bit-sliced circuit for addition, such as a ripple carry adder, a given bit of the sum depends not only on the corresponding bits of the two operands, but also on the carry out of the operation on all less significant bits. We thus need one bit of information (the carry) to pass from each full adder to the next. Each full adder adds three one-bit numbers, producing both the sum and the carry out. One can prove by induction that such a design produces the correct sum.

Think about the information that flows from one copy of a bit-sliced design to the next. In an adder, we have a natural representation: we set the carry value to 1 if a carry occurs, and to 0 otherwise. But we could choose to use the other representation. As you know, any time we encode information in a computer, we need to choose a representation. The representation used to pass information between bit slices is completely up to the designer, and makes a difference in the amount of logic needed to implement the bit slice. Think about that fact as you work on these problems.

In today's discussion, you will apply the bit-sliced design principle to implement a power-of-two checker circuit.

1. Power-of-two checker

Design a *bit-sliced circuit* U that checks whether an unsigned integer $A = a_{n-1}a_{n-2}\ldots a_1a_0$ is a power of 2, starting with the least significant bit:



The U cell should be designed so that the *n*-bit network shown above correctly checks if the unsigned integer number A is a power of two. The final output f should be 1 iff A is a power of two.

1. If an unsigned number is a power of two, how many 1s are in its binary representation? Give an example of an 8-bit unsigned number that is a power of two and an example of one that is not a power of two.

Exactly one bit must be set to 1 and all other bits must be set to 0s.

Power of two: $01000000_2 = 64_{10}$

Not a power of two: $00011000_2 = 24_{10}$

2. List the possible 'answers' (or information) that your bit slice U may need to communicate to the next bit slice (and receive from the previous bit slice). One such answer is already provided for you to get started. For each possible answer, assign a bit pattern for the information passed between your bit slices.

p_i	q_i	**Meaning**
0	0	Have not seen any 1s so far
0	**1**	**Have seen only one '1' so far**
1	**0**	**Have seen more than one '1' so far**
1	**1**	**Unused**

3. What values should be used for p_0 and q_0?

p_0 = _**0**_ q_0 = _**0**_

4. Based on your answer to **Part 2**, draw K-maps for p_{i+1} and q_{i+1}.

p_{i+1}:

a_i \ p_iq_i	00	01	11	10
0	0	0	X	1
1	0	1	X	1

q_{i+1}:

a_i \ p_iq_i	00	01	11	10
0	0	1	X	0
1	1	0	X	0

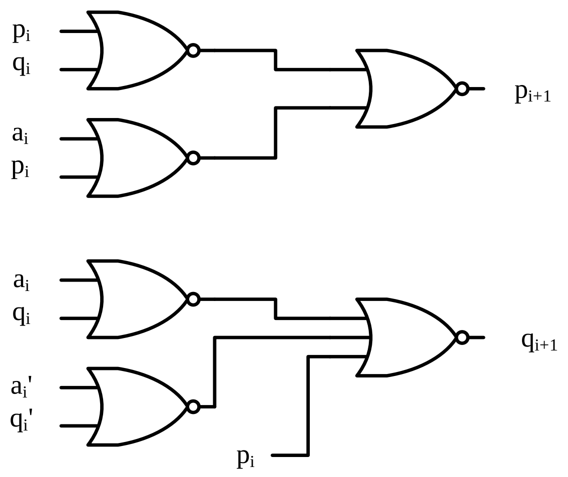
5. Give minimal POS expressions for p_{i+1} and q_{i+1}.

p_{i+1} = __$(p_i+q_i)(a_i+p_i) = (p_i+q_i)(a_i+q_i')$______ q_{i+1} = _____$p_i'(a_i+q_i)(a_i'+q_i')$______

6. Write a Boolean expression for the decision circuit F that maps from the outputs of the most-significant-bit bit-slice p_nq_n to the answer $f=1$ *if A is a power of two, or* $f=0$ *otherwise*.

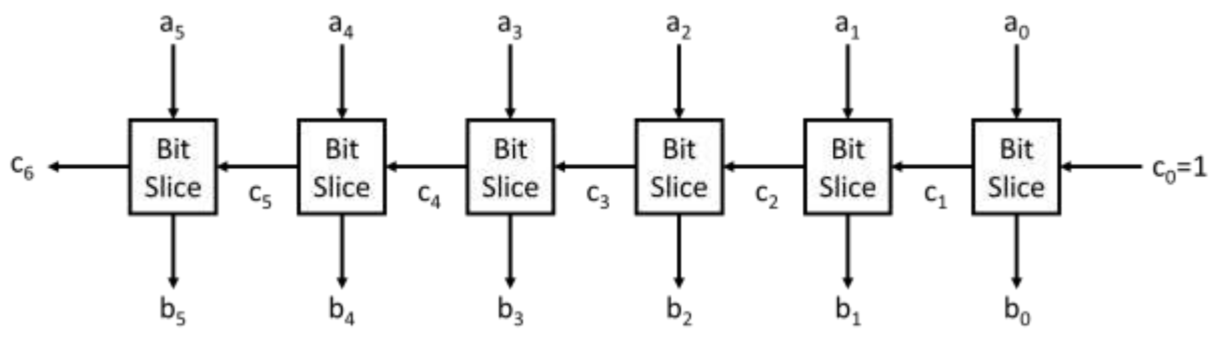
$f(p_n, q_n)$ = __$p_n'q_n$_____

7. Implement circuit U using NOR gates only. You may assume that all literals (including complements) are available to you.



2. 2's complement

The circuit shown below is intended to negate a 6-bit 2's complement number. For example, if the input $a_5a_4a_3a_2a_1a_0$ = 110100, then the output $b_5b_4b_3b_2b_1b_0$ = 001100. Note that the initial carry-in bit c_0 is set to 1 in this design.



1. Each bit slice is identical. Complete the truth table for a single bit slice and write the minimal SOP expressions.

c_i	a_i	c_{i+1}	b_i
0	0	0	1
0	1	0	0
1	0	1	0
1	1	0	1

Minimal SOP: **c_{i+1}=**_c_ia_i'

Minimal SOP: **b_i=**_$c_i'a_i' + c_ia_i$

2. Assume that the b_i and c_{i+1} outputs of each bit slice are implemented using 2-level AND-OR networks based on their minimal SOP expressions. **In this question, count AND and OR gates, but do not count NOT gates.**

How many gates are required in the entire circuit (all bit slices)? __________24____________

How many gates are there along one of the longest paths from a_0 to b_5? ________7__________