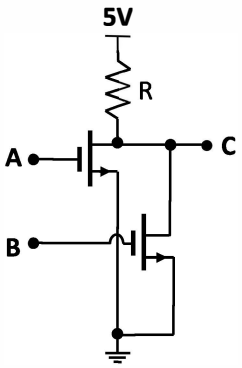
Tutorial 1 Solutions

Question 1:

Take a look at the electrical circuit described below.

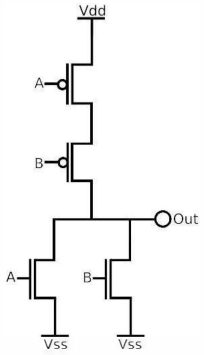


a) Identify, and explain in a couple of sentences, what happens to the output C when the inputs A and B are set to 5V or 0V, respectively.
This circuit has two N-MOS transistor. Our knowledge of the physics of N-MOS transistors tells us that for suitably high Vgs the conductive channel in the transistor opens. Signals A and B are connected to gates; both transistors have their source terminal connected to ground. When A or B are drive high Vgs, is suitably high. As the resistance across either transistor drops to a low value, the voltage of port C is pulled low.
Cover all 4 combinations of open/closed transistors in your answer.

b) Write a truth table and name the corresponding logic function.
It's a NOR gate (truth table follows easily)

c) Explain the role of resistor R in the design.
It's a pull-up resistor. It sets the value presented at output pin C as high when both transistors are nonconducting. In the absence of R, the output pin would be left floating (i.e., could not be driven high).
On the other hand, if a direct connection to the 5V rail was in place (without R), then the transistors would not be able to drive the pull the output to ground and there would be high current drain from the power supply.

d) Considering that the same circuit can be implemented as shown below, identify two points of difference which make one implementation more advantageous than the other and/or vice-versa, and explain.
The below design uses both PMOS and NMOS transistors. It uses less static power due to the fact that the circuit path Vdd-Vss always has a higher resistance than R. The initial circuit is potentially cheaper to manufacture due to the single type of MOS device.



Part 2: Combinatorial logic and hazards

Question 2:

Design a digital circuit with four inputs that will output a 1 if the majority (3 or 4) of the inputs are 1.

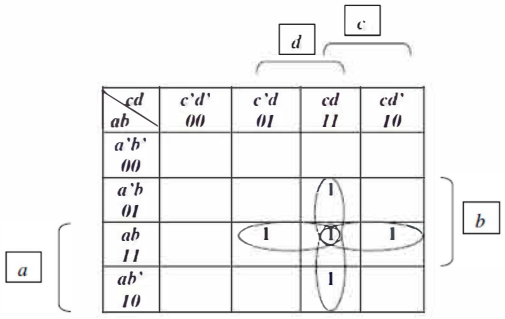
[Tips: Understand the concept of the majority circuit from the question. Then translate your concept to a truth table. From the truth table, create a Karnaugh map to produce Sum of Product (SOP) expression for the majority circuit.]

A truth table for this majority circuit is shown in table 3 below. The corresponding Karnaugh map for the output *f* is shown below. From the Karnaugh map the simplified SOP expression for *f* is derived as

f = b & c & d | a & c & d | a & b & d | a & b & c

a	*b*	*c*	*d*	*f*
0	0	0	0	0
●	●	0	1	0
0	0	1	0	●
0	0	1	1	0
●	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	●	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Table 3: Truth table for the majority circuit



Question 3:

Design a 2-bit comparator that will compare two 2-bit inputs, a[1:0] and b[1:0] and output the following three signals.

a_eq_b = 1 if a = b;
a_gt_b = 1 if a > b;
a_lt_b = 1 if a < b.

[Tips: You need to draw three Karnaugh maps to produce three different SOP expressions. Each SOP will be associated with one output only.]

Then, comment on whether any of your circuits are likely to exhibit timing hazards and if so, for what input combinations. Also comment whether said hazards can be removed through SOP redesign.

A truth table for this 2-bit comparator is shown in table 4.

b[1]	b[0]	a[1]	a[0]	a_eq_b	a_gt_b	a_lt_b
0	0	0	0	1	0	0
0	0	0	1	0	1	0
0	0	1	0	0	1	0
0	0	1	1	0	1	0
0	1	0	0	0	0	1
0	1	0	1	1	0	0
0	1	1	0	0	1	0
0	1	1	1	0	1	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	1	0	0
1	0	1	1	0	1	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	1	0	0

Table 4: Truth table for 2-bit comparator

The Karnaugh map for the output *a_eq_b* is drawn in figure 4. You can see that the four ones go down the main diagonal and so the logic equation for *a_eq_b* can not be reduced will just be the sum of minterms *m0*, *m5*, *m10*, and *m15*.

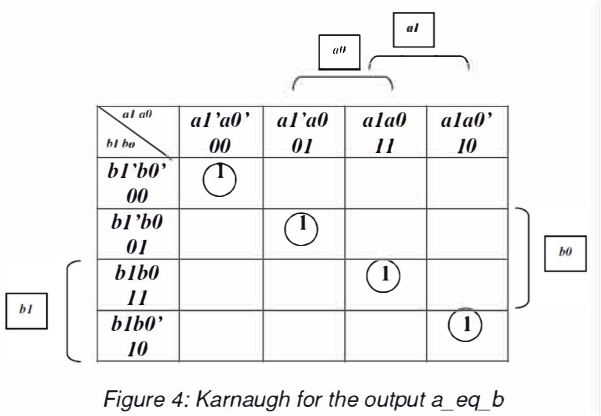


Figure 4: Karnaugh for the output a_eq_b

a_eq_b = ~b[1] & ~b[0] & ~a[1] & ~a[0] |~b[1] & b[0] & ~a[1] & a[0] | b[1] & ~b[0] & a[1] & ~a[0] | b[1] & b[0] & a[1] & a[0]

The Karnaugh map for *a_gt_b* is given in figure 5.

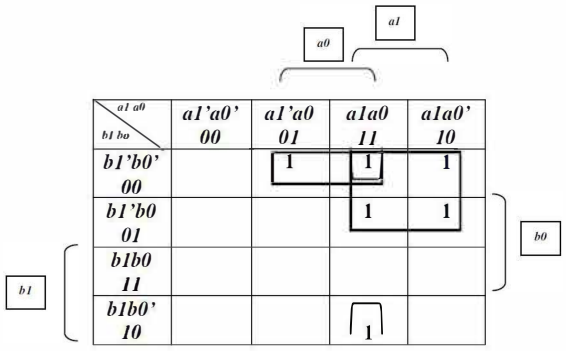


Figure 5: Karnaugh for the output a_gt_b

a_gt_b = ~b[1] & a[1] | ~b[1] & ~b[0] & a[0] | ~b[0] & a[1] & a[0]

Similarly, the SOP expression for the output *a_lt_b* can be obtained as (negating the terms above)

a_lt_b = b[1] & ~a[1] | b[1] & b[0] & ~a[0] | b[0] & ~a[1] & ~a[0]

The circuit for a_eq_b has 1-hazards as there are non-overlapping prime implicants. These may occur when shifting two bits at the same time (e.g., from 1-1 to 0-0). It is not possible to remove these with a standard SOP design.
A possible alternative solution would be to create a circuit that delivers a 1 output for the zero entries in the Karnaugh map instead of the one entries (as this can be achieved with overlapping implicants) and then placing a single NOT gate as an additional final stage of the circuit.

Part 3: Review of number systems

Question 4:

Convert the decimal number 27 to binary, and hexadecimal numbers using the most appropriate conversion method.

$(27)_{10} = (11011)_2 ; (1B)_{16}$

Question 5:

Convert binary number 101110 to hexadecimal and decimal numbers using the most appropriate conversion method

$(101110)_2 = (2E)_{16} ; (46)_{10}$

Question 6:

Convert hexadecimal number ABC to binary and decimal numbers using the most appropriate conversion method.

$(ABC)_{16} = (1010\ 1011\ 1100)_2 ; (2748)_{10}$

Question 7:

Calculate A + B, A - B, -A + B, and -A - B for binary numbers 1010101, 1010 assuming a two's complement number system and n = 8. Check your results by decimal arithmetic. Explain any unusual results.

1010101, 1010

$A = (0, 1010101)_{2cns} = +(85)_{10}$; $B = (0, 0001010)_{2cns} = +(10)_{10}$
$-A = (1, 0101011)_{2cns} = -(85)_{10}$; $-B = (1, 1110110)_{2cns} = -(10)_{10}$

$(A+B) = (0, 1011111)_{2cns} = +(95)_{10}$
$(A-B) = (0, 1001011)_{2cns} = +(75)_{10}$
$(-A+B) = (1, 0110101)_{2cns} = -(75)_{10}$
$(-A-B) = (1, 0100001)_{2cns} = -(95)_{10}$

It is important that results are also interpreted using two's complement. Otherwise negative numbers could be erroneously interpreted as large positives.
There are no unusual results here, but there can be issues with binary overflow: try to exceed 127, for example, by calculating A+A. Is the result positive or negative? Should it be?

Question 8:

In most memory structures the addresses of locations are specified in binary numbers that identify each memory circuit where a data word is stored. The number of bits that make up an address will depend on how many memory locations there are. Since the number of bits can be very large, the addresses are often specified in hex instead of binary.

a. If a microcomputer uses a 20-bit address, how many different memory locations are there?

b. How many hex digits are needed to represent the address of a memory location?

c. What is the hex address of the 256_{th} memory location? (Note: the first address is always 0)

(a) $2^{20} = 1048576$
(b) 5 *(i.e., 5 lots of 4 binary digits!)*
(c) $255_2 = FF_H$