Solutions - Midterm Exam (Online Section)

(Due Date: October 9th @ 11:00 am)
Clarity is very important! Show your procedure!

PROBLEM 1 (15 PTS)

- Complete the following table. Use the fewest number of bits on each case.

REPRESENTATION			
Decimal	**Sign-and-magnitude**	**1's complement**	**2's complement**
-256	1100000000	1011111111	100000000
-154	110011010	101100101	101100110
-107	11101011	10010100	10010101
135	010000111	010000111	010000111

PROBLEM 2 (15 PTS)

a) Convert the decimal number 136.6875 to i) binary, and ii) hexadecimal.
$136.6875 = (10001000.1011)_2 = 0x88.B$

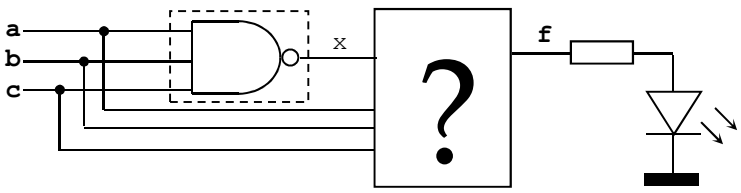
b) Represent the decimal number 136 in i) BCD, and ii) Reflective Gray Code

Decimal	Binary (unsigned)	BCD	Reflective Gray Code
136	10001000	0001 0011 0110	11001100

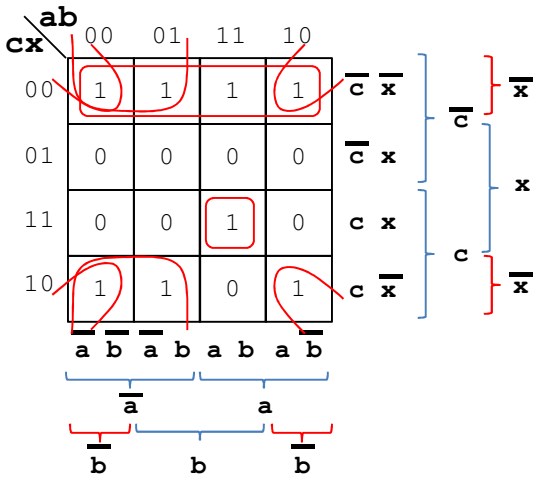
c) What is the minimum number of bits to represent numbers between 12,000 and 13,024?
Numbers between 12000 and 13024: $13024 - 12000 + 1 = 1025$
Minimum number of bits: $\lceil \log_2 1025 \rceil = 11\ bits$

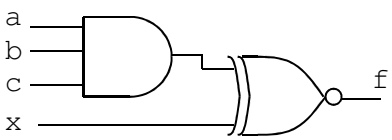
PROBLEM 3 (15 PTS)

Design a circuit (***simplify your circuit!***) that verifies the logical operation of a 3-input NAND gate. f = '1' (LED ON) if the NAND gate does NOT work properly. Assumption: when the NAND gate is not working, it generates 1's instead of 0's and vice versa.



a	b	c	x	f
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1





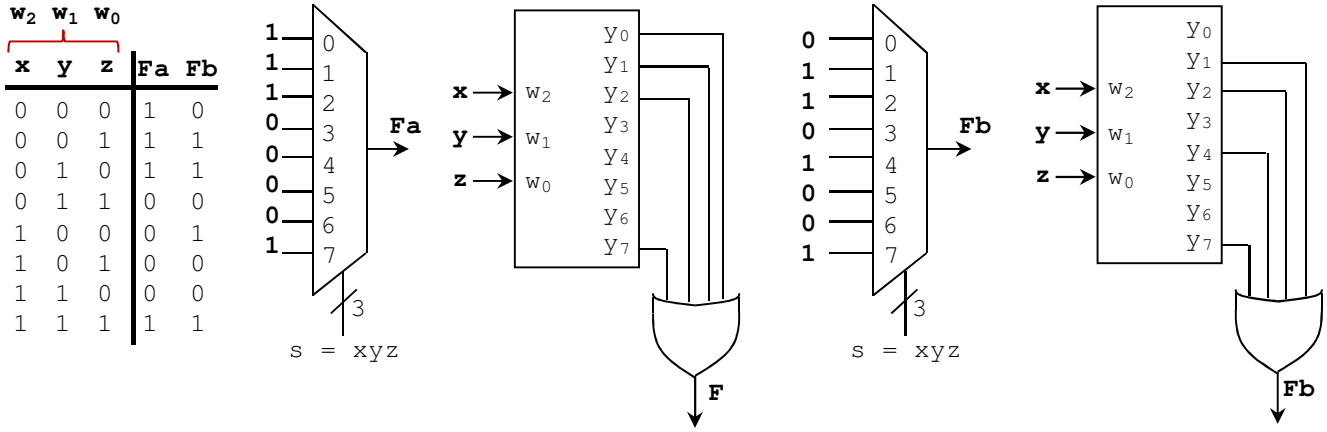
$f = \overline{b}\,\overline{x} + \overline{a}\,\overline{x} + \overline{c}\,\overline{x} + abcx$

$f = \overline{x}(\overline{abc}) + x(abc) = \overline{x \oplus (abc)}$

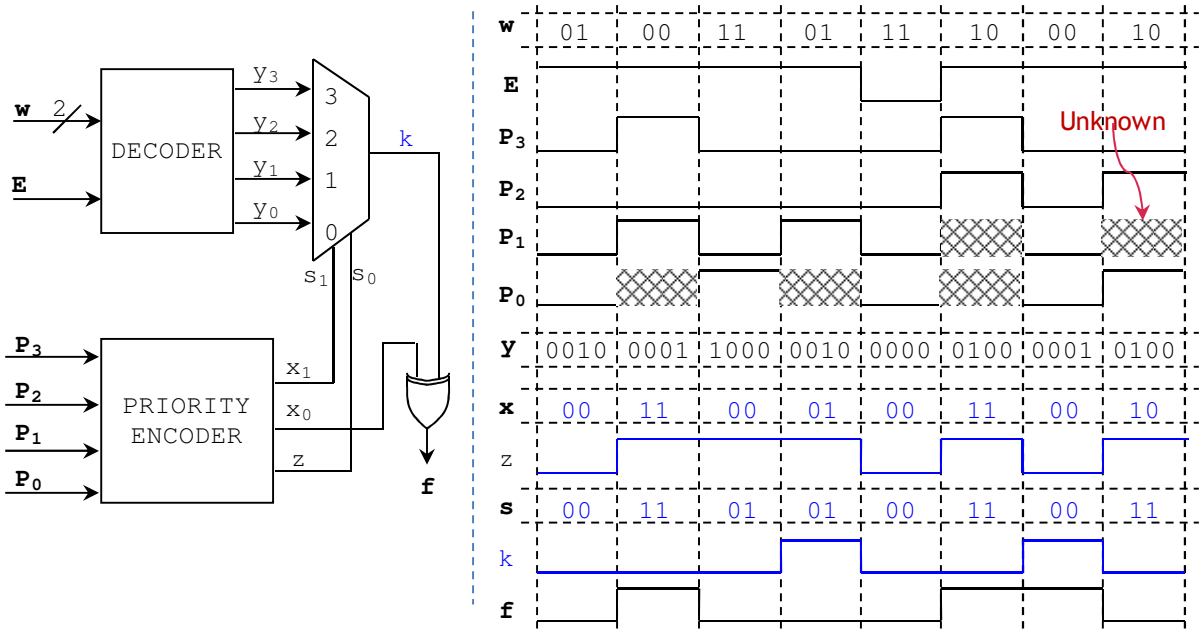
PROBLEM 4 (20 PTS)

- Implement the following function using: i) a decoder (and gates) and ii) a multiplexer:
  - ✓ $F_a(X,Y,Z) = \prod(M_3, M_4, M_5, M_6)$
  - ✓ $F_b = X \oplus Y \oplus Z$

w_2 w_1 w_0				
x	y	z	Fa	Fb
0	0	0	1	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1



- Complete the timing diagram of the circuit shown below:



PROBLEM 5 (15 PTS)

- Perform the following operations using the 2's complement arithmetic. For each case, provide the summands and the result in 2's complement representation. Use the minimum number of bits to represent the summands and the result so that overflow is avoided.

✓ −128 + 453

n = 10 bits

c_{10}=1 c_9=1

−128 = 1110000000 +
+453 = 0111000101

+325 = 0101000101

overflow = $c_{10} \oplus c_9$=0 -> no overflow

+325 ∈ $[-2^9, 2^9-1]$ -> no overflow

✓ −255 − 37

n = 9 bits

c_9=1 c_8=0

−255 = 100000001 +
−37 = 111011011

011011100

overflow = $c_9 \oplus c_8$=1 -> overflow!

−255−37=−292 ∉ $[-2^8, 2^8-1]$ -> overflow!

To avoid overflow:
n = 10 bits (sign-extension):

c_{10}=1 c_9=1

−255 = 1100000001 +
−37 = 1111011011

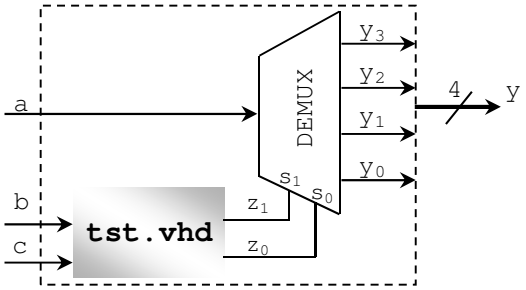
−292 = 1011011100

overflow = $c_{10} \oplus c_9$=0 -> no overflow

−292 ∈ $[-2^9, 2^9-1]$ -> no overflow

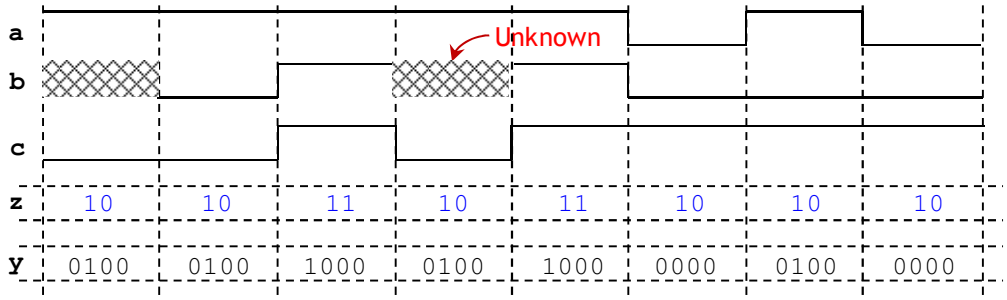
PROBLEM 6 (10 PTS)

- The following VHDL code corresponds to the shaded circuit. Complete the timing diagram:



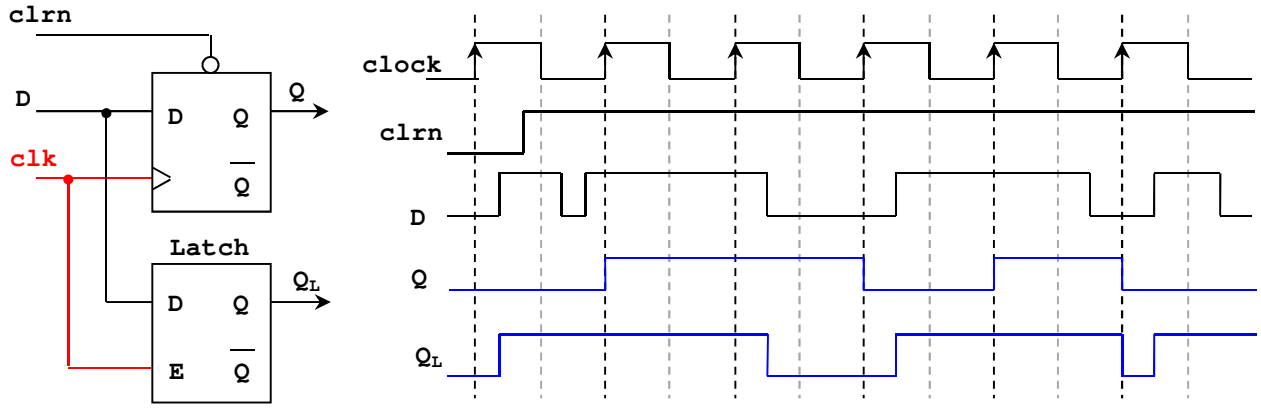

```
library ieee;
use ieee.std_logic_1164.all;

entity tst is
  port (b, c: in std_logic;
        z: out std_logic_vector(1 downto 0));
end tst;
```

```
architecture bhv of tst is
  signal x, y: std_logic;

begin

  process (b,c)
  begin
     z <= c & b;
     if c = '0' then
        z <= "10";
     end if;
  end process;

end bhv;
```



PROBLEM 7 (10 PTS)

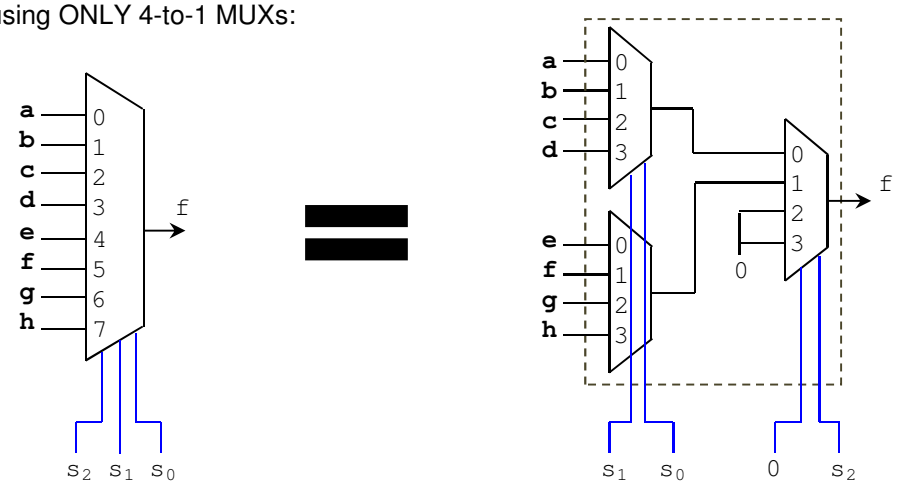
- Complete the timing diagram of the circuit shown below:



BONUS PROBLEM (+5 PTS)

- Using ONLY 4-to-1 MUXs, implement an 8-to-1 MUX.
- A microprocessor has a 24-bit address line. We connect a memory chip to the microprocessor. The memory chip addresses are assigned the range `0x800000` to `0xBFFFFF`. What is the minimum number of bits required to represent addresses in that individual memory chip?

- 8-to-1 MUX using ONLY 4-to-1 MUXs:



- If we convert the numbers to binary, we notice that the addresses in the range `0x800000` to `0xBFFFFF` require 24 bits.

```
0x800000: 1000 0000 0000 0000 0000 0000
                                    ...
                                    ...
                                    ...
0xBFFFFF: 1011 1111 1111 1111 1111 1111
```

However, we notice that all the addresses in the range `0x800000` to `0xBFFFFF` share the same first two MSBs: `10`. Therefore, if we were to address data ONLY in the individual memory chip, we do not need those two bits ⇒ we need ***22 bits***.
Another way to put it: We only need ***22 bits*** for the address line of that individual memory device.