COL215 Digital Logic and System Design

06.10.2017 **Minor Test 2** **Max Marks: 10**

1. 2's complement of an n-bit binary number $B = b_{n-1}\, b_{n-2} \dots b_1\, b_0$ is given by

   (a) $b'_{n-1}\, b'_{n-2} \dots b'_{k+1}\, b_k\, b_{k-1} \dots b_1\, b_0$, where b_k is the rightmost 0 in B.
   (b) $b'_{n-1}\, b'_{n-2} \dots b'_{k+1}\, b_k\, b_{k-1} \dots b_1\, b_0$, where b_k is the rightmost 1 in B.
   (c) $b_{n-1}\, b_{n-2} \dots b_{k+1}\, b'_k\, b'_{k-1} \dots b'_1\, b'_0$, where b_k is the rightmost 0 in B.
   (d) $b_{n-1}\, b_{n-2} \dots b_{k+1}\, b'_k\, b'_{k-1} \dots b'_1\, b'_0$, where b_k is the rightmost 1 in B.

   Which one of the above is correct? Give a proof. Here b_0 is the least significant bit.

   [2]

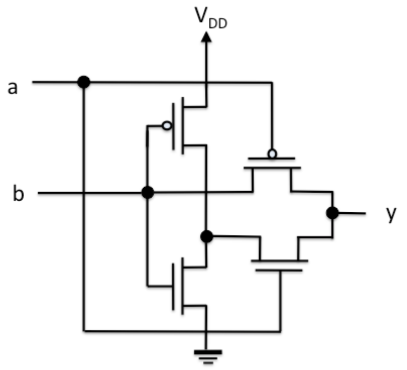
2. Consider the algorithm given below for dividing 16 bit unsigned *Dividend* by 16 bit unsigned *Divisor*. The *Quotient* and *Remainder* are also 16 bit. R and D are 32 bit integers, R is signed and D is unsigned. Give a VHDL implementation of this algorithm such that it takes 18 clock cycles – one cycle for each iteration of the loop and one cycle each for the initialization step and the final step. Do not use variables, LOOP, WAIT and AFTER.

```
Initialization step:    R = Dividend; Q = 0; D = Divisor x 2^(n-1)
Loop:  for i = 0 to 15 do {
           if (R < 0) R = R + D else R = R - D
           Q = 2 x Q + s        where s is the sign bit of adder/subtractor output
           D = D / 2
       }
Final step:    if (R < 0) Remainder = R + B else Remainder = R
               Quotient = Q
```

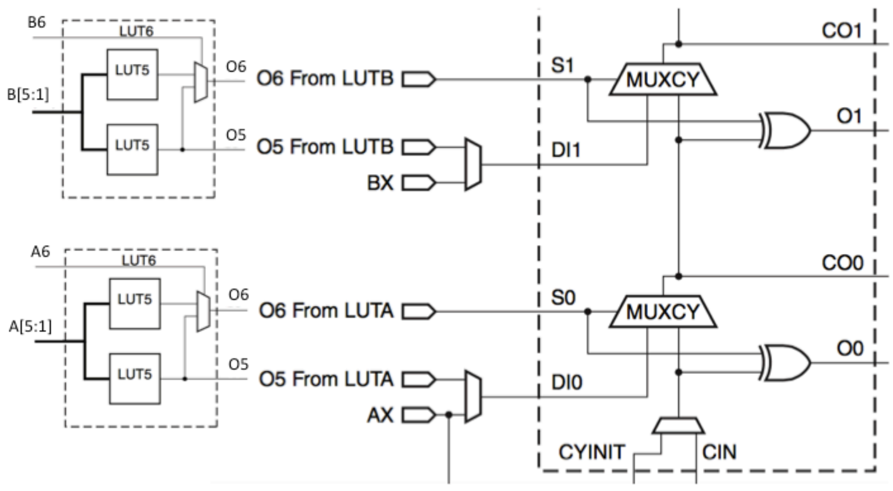
   [3]

3. What logic gate is realized by the circuit shown? Does the circuit suffer from any major drawbacks? Give reasons for your answer.

   [3]



4. The figure shows a part of a slice of Spartan 6 series FPGAs. On the right side, a portion of the carry chain circuit (2 bits out of 4) is shown and on the left side, two of the 6-input LUTs are shown. Note that each 6-input LUT consists of two 5-input LUTs. What logic functions should be implemented by the LUTs shown to make a 2-bit portion of a fast carry propagate adder?



   [2]

Solutions

Q1:
Answer (b) is correct.

Proof:

Here b_k is the rightmost 1 in B. This means that bits towards right of b_k are 0. Let us now find 2's complement of B by first finding its 1's complement and then adding 1. While doing so, we keep in view the bit pattern of b_k to b_0, as shown below.

bit index =>	n-1	n-2		k+1	k	k-1		1	0
The number B =	b_{n-1}	b_{n-2}	. . .	b_{k+1}	b_k	b_{k-1}	. . .	b_1	b_0
Putting values of b_k to b_0, B =	b_{n-1}	b_{n-2}	. . .	b_{k+1}	1	0	. . .	0	0
1's complement of B =	b'_{n-1}	b'_{n-2}	. . .	b'_{k+1}	0	1	. . .	1	1
Adding 1 to above, 2's compl of B =	b'_{n-1}	b'_{n-2}	. . .	b'_{k+1}	1	0	. . .	0	0
Putting back b_k to b_0, 2's compl of B =	b'_{n-1}	b'_{n-2}	. . .	b'_{k+1}	b_k	b_{k-1}	. . .	b_1	b_0

This proves the result.

Marking scheme:
[1] mark for choosing (b) as the correct answer
[1] mark for proving it (proof need not be very formal)

Q2:

```
entity divider is
   port (
      dividend, divisor : input bit_vector (15 downto 0);
      quotient, remainder : output bit_vector (15 downto 0);
      reset, clock, start : input bit;
      done : output bit
   );
end divider;
architecture sequential of divider is
   type state_type is (init, main, final);
   signal state : state_type;
   signal R, D, ALU : std_logic_vector (31 downto 0);
   signal op1, op2 : signed (31 downto 0);
   signal Q : std_logic_vector (15 downto 0);
   signal i : unsigned (3 downto 0);
   constant zero16 : std_logic_vector (15 downto 0) = "0000000000000000";
begin
   ALU <= std_logic_vector (signed (R) + op1) when state = main else
              std_logic_vector (signed (R) + op2);
   op1 <= signed (D) when R (31) = '1' else - signed (D);
   op2 <= signed (zero16 & divisor) when R (31) = '1' else signed (zero16 & zero16);
   div_proc: process (reset, clock)
   begin
      if reset = '1' then state <= init;
      elsif (clock'event and clock = '1') then
         case state is
            when init =>
               done <= '0';
               if start = '1' then
                  R <= zero16 & dividend;
                  Q <= zero16;
                  D <= '0' & divisor & zero16 (14 downto 0);
                  i <= 15;
                  state <= main;
               end if;
            when main =>
               R <= ALU;
               Q <= Q (14 downto 0) & not ALU (31);
               D <= '0' & D (31 downto 1);
               if i > 0 then i <= i – 1; else state <= final;
               end if;
            when final =>
               remainder <= ALU (15 downto 0);
               quotient <= Q;
               done <= 1;
               state <= init;
         end case;
      end if;
   end process;
end sequential;
```

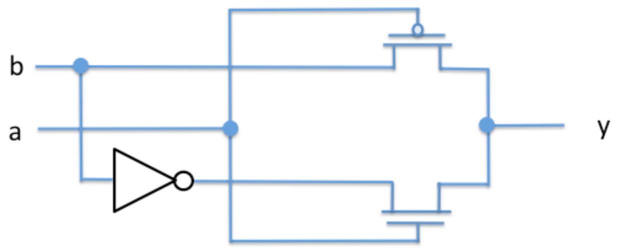
Marking scheme:
1 mark for defining states and ensuring 18 cycles (no credit for using LOOP)

1 mark for separation of add/subtract operation as a combinational circuit (no credit for doing add/subtract and sign checking in sequence)

1 mark for overall correctness, no deduction for errors related to syntax and type conversions

Q3:

The given circuit has a CMOS inverter with b as the input. Therefore we can redraw the circuit as follows.



The PMOS transistor tries to pass b to the output when a = ‘0’.
The NMOS transistor tries to pass b’ to the output when a = ‘1’.
Therefore, if we assume the two pass transistors as ideal, the circuit implements the function
y = a’ b + a b’ {exclusive OR}

The PMOS pass transistor does not pass ‘0’ properly because as the output voltage starts falling below V_T, it makes V_{SG} below V_T and turns the transistor off.
The NMOS pass transistor does not pass ‘1’ properly because as the output voltage starts going above V_{DD}-V_T, it makes V_{GS} below V_T and turns the transistor off.

Therefore, the drawbacks or the problems with this circuit are as follows.
* when a = ‘0’ and b = ‘1’, the output does not get a proper ‘1’ level.
* when a = ‘1’ and b = ‘1’, the output does not get a proper ‘0’ level.

Marking scheme:
1 mark for the answer y = a’ b + a b’ or exclusive OR
½ mark for correct reason / explanation
1 mark for correctly stating the problems or drawbacks in the circuit.
½ mark for correct reason / explanation

Q4:

For each bit, the two LUT5's need to implement *carry generate* and *carry propagate* functions and feed these to DI_i and S_i so that the multiplexer MUXCY produces correct carry for the next stage at CO_i, but it needs to be done in such a way that the correct sum is produced by the XOR gate on O_i.

The functions implemented by multiplexer MUXCY and the XOR gate are as follows.

multiplexer MUXCY: $CO_i = S_i \,.\, CO_{i-1} + S_i' \,.\, DI_i$ (1)

XOR gate: $O_i = S_i \text{ XOR } CO_{i-1}$ (2)

The usual expression for *carry generate*, *carry propagate*, *carry out* and *sum* are as follows.

$g_i = a_i \,.\, b_i$

$p_i = a_i + b_i$

$c_{i+1} = g_i + p_i \,.\, c_i$

$s_i = a_i \text{ XOR } b_i \text{ XOR } c_i$

mapping input carry c_i to CO_{i-1}, output carry c_{i+1} to CO_i and sum s_i to O_i, our requirement is to implement the following.

$CO_i = g_i + p_i \,.\, CO_{i-1}$ (3)

$O_i = a_i \text{ XOR } b_i \text{ XOR } CO_{i-1}$ (4)

In order to generate correct sum, we define S_i as follows.

$S_i = a_i \text{ XOR } b_i$

Then requirement (4) becomes the following.

$O_i = S_i \text{ XOR } CO_{i-1}$ (5)

This is same as (2).

We note that $S_i + g_i = (a_i \text{ XOR } b_i) + a_i \,.\, b_i = a_i + b_i = p_i$

Therefore, we can replace p_i by $S_i + g_i$ in (3) to get the following.

$CO_i = g_i + (S_i + g_i) \,.\, CO_{i-1}$

$= g_i + S_i \,.\, CO_{i-1} + g_i \,.\, CO_{i-1}$

$= g_i + S_i \,.\, CO_{i-1}$ (6)

We note that $g_i = g_i \,.\, S_i' + g_i \,.\, S_i = g_i \,.\, S_i' + a_i \,.\, b_i \,.\, (a_i \text{ XOR } b_i) = g_i \,.\, S_i'$

Therefore, we can replace g_i by $g_i \,.\, S_i'$ in (6) to get the following.

$CO_i = g_i \,.\, S_i' + S_i \,.\, CO_{i-1}$ (7)

Making $DI_i = g_i = a_i \,.\, b_i$, we can replace g_i by DI_i in (7) to get the following.

$CO_{i+1} = DI_i \,.\, S_i' + S_i \,.\, CO_{i-1}$ (8)

This is same as (1).

Therefore, the two LUT5's for each bit need to implement the following functions.

$\mathbf{DI_i = a_i \,.\, b_i}$

$\mathbf{S_i = a_i \text{ XOR } b_i}$

Marking scheme:

½ mark each for expressions of DI_i and S_i

1 mark for correct reasoning/explanation (it need not be as formal as above)