# SET A

| Left neighbour | Your Name | Your Entry No. | Right neighbour |
|---|---|---|---|
| | | | |

**COL215 Digital Logic and System Design  Quiz 2  20.09.2017**

**Q 1.** Show all the steps of binary unsigned division of 11011010 by 00001011 using the method in which dividend and quotient shift together and the divisor does not shift.

**Solution:**

Dividend bits are shown in black, quotient bits are shown in red, vertical bar separates the two.

| | | |
|---|---|---|
| STEP 0: | Put dividend in 16 bit register | 0 0 0 0  0 0 0 0  1 1 0 1  1 0 1 0 |
| STEP 1: | Shift dividend left | 0 0 0 0  0 0 0 1  1 0 1 1  0 1 0\|0 |
| | Compare and subtract divisor | - 0 0 0 0  1 0 1 1 |
| | | 0 0 0 0  0 0 0 1  1 0 1 1  0 1 0\|0 |
| | Put quotient bit 0 in register | 0 0 0 0  0 0 0 1  1 0 1 1  0 1 0\|0 |
| STEP 2: | Shift dividend\|quotient left | 0 0 0 0  0 0 1 1  0 1 1 0  1 0\|0 0 |
| | Compare and subtract divisor | - 0 0 0 0  1 0 1 1 |
| | | 0 0 0 0  0 0 1 1  0 1 1 0  1 0\|0 0 |
| | Put quotient bit 0 in register | 0 0 0 0  0 0 1 1  0 1 1 0  1 0\|0 0 |
| STEP 3: | Shift dividend\|quotient left | 0 0 0 0  0 1 1 0  1 1 0 1  0\|0 0 0 |
| | Compare and subtract divisor | - 0 0 0 0  1 0 1 1 |
| | | 0 0 0 0  0 1 1 0  1 1 0 1  0\|0 0 0 |
| | Put quotient bit 0 in register | 0 0 0 0  0 1 1 0  1 1 0 1  0\|0 0 0 |
| STEP 4: | Shift dividend\|quotient left | 0 0 0 0  1 1 0 1  1 0 1 0 \| 0 0 0 0 |
| | Compare and subtract divisor | - 0 0 0 0  1 0 1 1 |
| | | 0 0 0 0  0 0 1 0  1 0 1 0 \| 0 0 0 0 |
| | Put quotient bit 1 in register | 0 0 0 0  0 0 1 0  1 0 1 0 \| 0 0 0 1 |
| STEP 5: | Shift dividend\|quotient left | 0 0 0 0  0 1 0 1  0 1 0\|0  0 0 1 0 |
| | Compare and subtract divisor | - 0 0 0 0  1 0 1 1 |
| | | 0 0 0 0  0 1 0 1  0 1 0\|0  0 0 1 0 |
| | Put quotient bit 0 in register | 0 0 0 0  0 1 0 1  0 1 0\|0  0 0 1 0 |
| STEP 6: | Shift dividend\|quotient left | 0 0 0 0  1 0 1 0  1 0\|0 0  0 1 0 0 |
| | Compare and subtract divisor | - 0 0 0 0  1 0 1 1 |
| | | 0 0 0 0  1 0 1 0  1 0\|0 0  0 1 0 0 |
| | Put quotient bit 0 in register | 0 0 0 0  1 0 1 0  1 0\|0 0  0 1 0 0 |
| STEP 7: | Shift dividend\|quotient left | 0 0 0 1  0 1 0 1  0\|0 0 0  1 0 0 0 |
| | Compare and subtract divisor | - 0 0 0 0  1 0 1 1 |
| | | 0 0 0 0  1 0 1 0  0\|0 0 0  1 0 0 0 |
| | Put quotient bit 1 in register | 0 0 0 0  1 0 1 0  0\|0 0 0  1 0 0 1 |
| STEP 8: | Shift dividend\|quotient left | 0 0 0 1  0 1 0 0 \| 0 0 0 1  0 0 1 0 |
| | Compare and subtract divisor | - 0 0 0 0  1 0 1 1 |
| | | 0 0 0 0  1 0 0 1 \| 0 0 0 1  0 0 1 0 |
| | Put quotient bit 1 in register | 0 0 0 0  1 0 0 1 \| 0 0 0 1  0 0 1 1 |

REMAINDER = 0000 1001 QUOTIENT = 0001 0011

Marking: 3 marks for correct answer, deduct 1 mark if there are minor mistakes.

**Q 2.** While describing a circuit in VHDL, how would you **ensure** that (a) there are no combinational circuit loops and (b) there are no inferred latches? Give your answer with illustrations.

**Solution:**
(a) An assignment statement creates paths from the signals that appear in the expression on the right hand side or the conditions under which this assignment is executed. The assignment statements in a process, individually or collectively, may form cyclic paths representing combinational circuit loops. We need to avoid formation of such paths.

For example, the following statements individually imply combinational circuit loops.

```
x <= (C and D) or (not C and x);
```

```
if x = y then x <= z; else x <= not w; end if;
```

The following statements together form a combinational circuit loop involving signals x, y and v.

```
x <= y and not z;
y <= u xor v;
v <= a or not x;
```

(b) If a signal is assigned a value in a process, then it needs to be ensured that it is assigned some value under all conditions. If this is not ensured, the synthesizer infers a latch to hold the present value of the signal under the conditions when it is not assigned any value.

For example, in the following if-statement, there is no else clause. Therefore, if there is no other statement in the process that assigns a value to x when u = y is false, the value of x needs to be held under that condition. Therefore, the synthesizer will infer a latch for x.

```
if u = y then x <= z; end if;
```

A similar situation arises using case statement when clauses of this statement leave some condition uncovered and there is no *otherwise* clause. For example, see the following statement.

```
case s is
  when “00” =>    x <= a;
  when “01” =>    x <= b;
  when “10” =>    x <= c;
end case;
```

Another situation that leads to inference of latch (es) is omission of certain signals from the sensitivity list of the process. If some signal(s) that appear in right hand side expressions or conditions in the process are omitted from the sensitivity list, latches are inferred. An example is given below.

```
process (a, b)
begin
  d <= a + b + c;
end;
```

Here c should have been included in the sensitivity list to avoid inference of a latch.
Marking: 1 mark for each part, consisting of ½ mark for explanation and ½ mark for examples.

# SET B

| Left neighbor | Your Name | Your Entry No. | Right neighbour |
|---|---|---|---|
| | | | |

**COL215 Digital Logic and System Design  Quiz 2  20.09.2017**

**Q 1.** Show all the steps of binary unsigned division of 11001110 by 00001101 using the method in which dividend and quotient shift together and the divisor does not shift.

**Solution:**

Dividend bits are shown in black, quotient bits are shown in red, vertical bar separates the two.

| | | |
|---|---|---|
| STEP 0: | Put dividend in 16 bit register | 0 0 0 0  0 0 0 0  1 1 0 0  1 1 1 0 |
| STEP 1: | Shift dividend left | 0 0 0 0  0 0 0 1  1 0 0 1  1 1 0\|0 |
| | Compare and subtract divisor | - 0 0 0 0  1 1 0 1 |
| | | 0 0 0 0  0 0 0 1  1 0 0 1  1 1 0\|0 |
| | Put quotient bit 0 in register | 0 0 0 0  0 0 0 1  1 0 0 1  1 1 0\|0 |
| STEP 2: | Shift dividend\|quotient left | 0 0 0 0  0 0 1 1  0 0 1 1  1 0\|0 0 |
| | Compare and subtract divisor | - 0 0 0 0  1 1 0 1 |
| | | 0 0 0 0  0 0 1 1  0 0 1 1  1 0\|0 0 |
| | Put quotient bit 0 in register | 0 0 0 0  0 0 1 1  0 0 1 1  1 0\|0 0 |
| STEP 3: | Shift dividend\|quotient left | 0 0 0 0  0 1 1 0  0 1 1 1  0\|0 0 0 |
| | Compare and subtract divisor | - 0 0 0 0  1 1 0 1 |
| | | 0 0 0 0  0 1 1 0  0 1 1 1  0\|0 0 0 |
| | Put quotient bit 0 in register | 0 0 0 0  0 1 1 0  0 1 1 1  0\|0 0 0 |
| STEP 4: | Shift dividend\|quotient left | 0 0 0 0  1 1 0 0  1 1 1 0 \| 0 0 0 0 |
| | Compare and subtract divisor | - 0 0 0 0  1 1 0 1 |
| | | 0 0 0 0  1 1 0 0  1 1 1 0 \| 0 0 0 0 |
| | Put quotient bit 0 in register | 0 0 0 0  1 1 0 0  1 1 1 0 \| 0 0 0 0 |
| STEP 5: | Shift dividend\|quotient left | 0 0 0 1  1 0 0 1  1 1 0\|0  0 0 0 0 |
| | Compare and subtract divisor | - 0 0 0 0  1 1 0 1 |
| | | 0 0 0 0  1 1 0 0  1 1 0\|0  0 0 0 0 |
| | Put quotient bit 1 in register | 0 0 0 0  1 1 0 0  1 1 0\|0  0 0 0 1 |
| STEP 6: | Shift dividend\|quotient left | 0 0 0 1  1 0 0 1  1 0\|0 0  0 0 1 0 |
| | Compare and subtract divisor | - 0 0 0 0  1 1 0 1 |
| | | 0 0 0 0  1 1 0 0  1 0\|0 0  0 0 1 0 |
| | Put quotient bit 1 in register | 0 0 0 0  1 1 0 0  1 0\|0 0  0 0 1 1 |
| STEP 7: | Shift dividend\|quotient left | 0 0 0 1  1 0 0 1  0\|0 0 0  0 1 1 0 |
| | Compare and subtract divisor | - 0 0 0 0  1 1 0 1 |
| | | 0 0 0 0  1 1 0 0  0\|0 0 0  0 1 1 0 |
| | Put quotient bit 1 in register | 0 0 0 0  1 1 0 0  0\|0 0 0  0 1 1 1 |
| STEP 8: | Shift dividend\|quotient left | 0 0 0 1  1 0 0 0 \| 0 0 0 0  1 1 1 0 |
| | Compare and subtract divisor | - 0 0 0 0  1 1 0 1 |
| | | 0 0 0 0  1 0 1 1 \| 0 0 0 0  1 1 1 0 |
| | Put quotient bit 1 in register | 0 0 0 0  1 0 1 1 \| 0 0 0 0  1 1 1 1 |

REMAINDER = 0000 1011  QUOTIENT = 0000 1111

Marking: 3 marks for correct answer, deduct 1 mark if there are minor mistakes.

**Q 2.** While describing a sequential circuit using a process in VHDL, how would you **ensure** that it is fully synchronous except for initialization? Give your answer with illustrations.

**Solution:**

In a fully synchronous process, all assignments are made inside one or more if- statements that check for a clock edge. For example,

```
if (clock'event and clock = '1') then
  x <= expression1;
  y <= expression2;
end if;
```

Inside then-clause, other if-statements or case-statements may be nested in any manner.

Any assignments for asynchronous initialization of signals may be done before checking for clock edge, as shown below. Note that the level of reset signal is checked here, not the edge.

```
if reset = '1' then
  x <= init_x;
  y <= init_y;
elsif (clock'event and clock = '1') then
  x <= expression1;
  y <= expression2;
end if;
```

The sensitivity list of the process should consist of the clock signal and the signal(s) causing initialization. In the above example, this will be done as shown below.

```
process (clock, reset)
begin
. . .
end;
```

Marking: 1 mark for sensitivity list and 1 mark for checking clock edge and initialization signal levels. Each of these marks is divided into ½ mark for explanation and ½ mark for example(s).