# Solutions to Final Exam

*Most questions had two different versions.*

## Question 1 (3 marks)

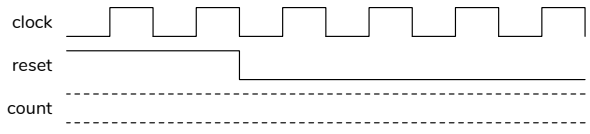
Given the following Verilog code:

```
module ex50
  ( input logic reset, clock,
    output logic [3:0] count ) ;

   logic [3:0] count_next ;

   assign count_next
     = reset ? 4'd2 :
       count + 4'd1 ;

   always_ff @(posedge clock) count = count_next ;

endmodule
```

— or —

```
module ex50
  ( input logic reset, clock,
    output logic [3:0] count ) ;

   logic [3:0] count_next ;

   assign count_next
     = reset ? 4'd1 :
       count + 4'd2 ;

   always_ff @(posedge clock) count = count_next ;

endmodule
```

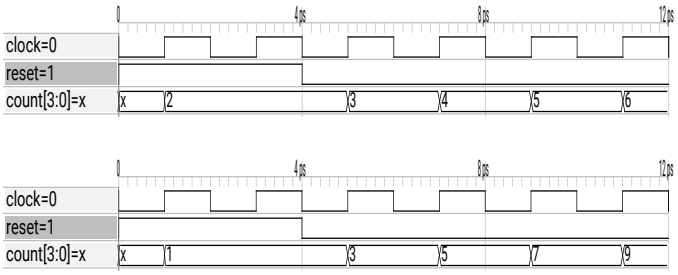
and the input signals in the diagram below, draw a vertical line between the dashed lines where the **count** output changes and write the value of **count** in-between each vertical line. Write an **X** if the value is undefined.



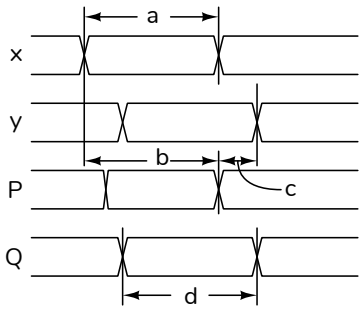

### Solution

**count_next** is set to an initial value of 2 (or 1) when **reset** is asserted or is incremented by 1 (or 2). **count** is a register whose value is set to **count_next** on the next rising edge of **clock**.

The resulting waveforms are:




## Question 2 (2 marks)




In the timing diagram above **x** and **y** are inputs (outputs), **P** and **Q** are outputs (inputs), and **a** through **d** are timing specifications. Write the letters corresponding to the timing specifications that are timing requirements and guaranteed responses in the appropriate boxes below. Each box should have zero or more of the letters **a** through **d**.

| timing requirements | guaranteed responses |
|---|---|
| | |

### Solution

Timing requirements end on inputs and guaranteed responses end on outputs. Specifications **a** and **c** end on inputs (outputs) and are thus requirements (guaranteed responses). Specifications **b** and **d** end on output (inputs) and are thus guaranteed responses (requirements).

If **x** and **y** are inputs the answer is:

| timing requirements | guaranteed responses |
|---|---|
| a, c | b, d |

If **x** and **y** are outputs the answer is:

| timing requirements | guaranteed responses |
|---|---|
| b,d | a, c |

## Question 3 (4 marks)

Given the following Verilog declarations:

```
logic [3:0] x = 4'b1010 ;
logic [7:0] y = 8'h5e ;

logic [3:0] x = 4'b0101 ;
logic [7:0] y = 8'h4e ;
```

fill in in the table below with the value of the each expression in hexadecimal and the width (number of bits) in decimal:

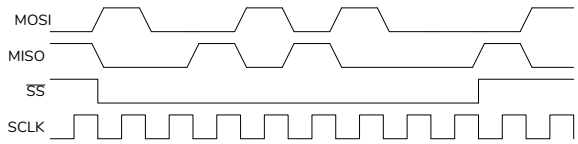
| expression | value (hexadecimal) | width (bits, decimal) |
|---|---|---|
| {x,y} | | |
| y[5:2] | | |
| x+5'b1 | | |
| y>>2 | | |

## Solution

- `{x,y}` is the concatenation of `x` (4 bits) and `y` (8 bits). The value in binary is `1010 0101 1110` (or `0101 0100 1110`) which is `0xA5E` (or `0x54E`). according to the table 11-21 in Lecture 4 the length is the sum of the lengths: 4+8 = 12 bits.
- `y[5:2]` is a "slice" `y=0101 1110` (or `0100 1110`). Bits of bits 5 "down to" 2 are `01 11` (or `00 11`) `0x7` (or `0x3`). The length of the expression is the length of the slice, 4 bits.
- `x+5'b1` is the addition of `x`, which has a 4-bit length and a value 10 (or 5) and a 5-bit value equal to 1. The value of the result is 11 (or 6) `0xB` (or `0x6`) with a length equal to the maximum of the two, 5 bits.
- `y>>2` is the logical right-shift of `y` by two bits. `y` is `y=0101 1110` (or `0100 1110`) which when shifted right by two bits is `0001 0111` (or `0001 0011`) `0x17` (or `0x13`) with a length equal to the length of `y`, 8 bits.

| expression | value (hexadecimal) | width (bits, decimal) |
|---|---|---|
| {x,y} | A5E (or 54E) | 12 |
| y[5:2] | 7 (or 3) | 4 |
| x+5'b1 | B (or 6) | 5 |
| y>>2 | 17 (or 13) | 8 |

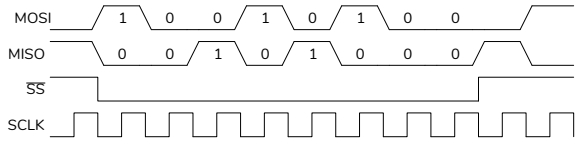
## Question 4 (2 marks)




The waveform above is measured on an SPI interface. What value was transferred from the slave to the master (master to the slave)? Give your answer as a hexadecimal number. Assume the bits are transferred most-significant-bit first.

## Solution

Data is transferred while $\overline{\texttt{SS}}$ is low and on the rising edge of the clock (since in the diagrams the transitions on $\overline{\texttt{SS}}$, `MOSI` and `MISO` are aligned with the falling edge of the clock). The bits transferred are indicated on the diagram below:



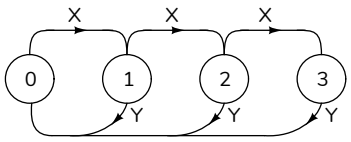

which are `10010100` = `0x94` from master to slave (on `MOSI`) and `00101000` = `0x28` from slave to master (on `MISO`).

## Question 5 (4 marks)

A module with inputs `reset` and `run`, and a clock signal, `clock`, is declared as:

```
module statemachine
( input logic reset, run, clock ) ;

logic [1:0] state, state_next ;

// your code here

endmodule
```

The module's behaviour can be described as a state machine with four states that have values 0 through 3. The state transitions are as described in the state transition diagram below:




X: reset==0 && run==1

Y: reset==1

Write Verilog statements below that, if placed after the comment, would result in the behaviour described in the state transition diagram above. Do not repeat code already given. Follow the course coding conventions.

## Solution

Three solutions are given below.

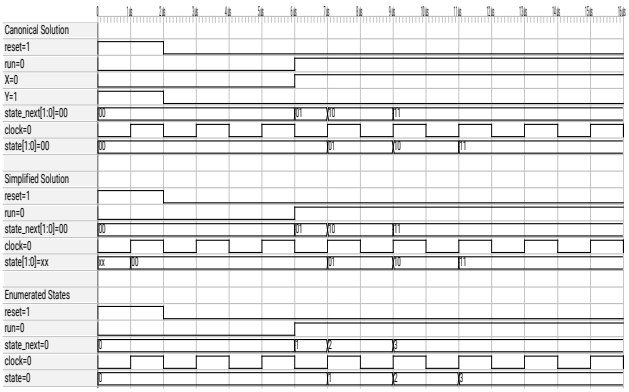
The "canonical" (simplest) solution uses one expression per transition in the state transition diagram. This expression tests for the starting state of the transition condition and the transition condition. If the expression is true then the state is set to the terminating state for that transition. Only one of the logical expression will be true and so they can be evaluated in any order.

The simplified solution arranges the order of the tests to obtain the same behaviour with less code. In this case we can make use of the fact that the state is set to 0 whenever **reset** is asserted and that the numerical value of the state is incremented when run is asserted and the state is not already at 3. Otherwise there is no change in state.

The question specifies a binary state encoding although an enumerated type, as recommended in the course coding guidelines, could be used as shown in the third solution.

```
// canonical solution

module statemachine1
  ( input logic reset, run, clock ) ;

   logic [1:0] state = 0, state_next ;
   logic X, Y ;

   assign X = reset == '0 && run == '1 ;
   assign Y = reset == '1 ;

   assign state_next
     = state == 0 && X ? 1 :
       state == 1 && X ? 2 :
       state == 2 && X ? 3 :
       state == 1 && Y ? 0 :
       state == 2 && Y ? 0 :
       state == 3 && Y ? 0 :
       state ;

   always @(posedge clock) state = state_next ;

endmodule

// simplified solution

module statemachine2
  ( input logic reset, run, clock ) ;

   logic [1:0] state, state_next ;

   assign state_next
     = reset ? '0 :
       run && state != 2'b11 ? state + 1'b1 :
       state ;

   always_ff @(posedge clock) state = state_next ;

endmodule

// solution using enumerated states

module statemachine3
  ( input logic reset, run, clock ) ;

   typedef enum int unsigned { s0, s1, s2, s3 }
        ↪ state_t ;

   state_t state, state_next ;

   assign state_next
     = reset ? s0 :
       run && state == s0 ? s1 :
       run && state == s1 ? s2 :
       run && state == s2 ? s3 :
       state ;

   always_ff @(posedge clock) state = state_next ;

endmodule
```

All three versions behave the same way as shown by the following simulation results:




## Question 6 (2 marks)

The maximum propagation delay through any combinational logic path in a CPLD is 8 (or 18) ns, the minimum setup time of its registers is 1.5 ns and the maximum clock-to-output delay is 0.5 ns. What is the

fastest clock frequency at which this CPLD can operate?

### Solution

Using the equation:

$$T_{\text{clock}} > t_{\text{CO}} + t_{\text{PD}} + t_{\text{SU}}$$

and substituting the given values we find:

$$T_{\text{clock}} > 1.5 + 8\ (\text{or } 18) + 0.5 = 10\ (\text{or } 20)\ \text{ns}$$

So the fastest clock frequency is $\frac{1}{10\ (\text{or } 20)\ \times 10^{-9}}$ = $\boxed{100\ (\text{or } 50)\ \text{MHz}}$.

## Question 7 (4 marks)

You need to design a 1 MByte memory for a 16-bit CPU using 256 k × 4 memory ICs.

(a) How many IC's will be needed?

(b) How many banks will this memory require?

(c) How many address bits does each memory IC have?

(d) If an address decoder places the first byte of this memory at address **`0x10 0000`**, what is the last address in this 1 MByte memory?

### Solution

(a) The number of ICs required can be computed from the total number of bits required (8 bits/byte × 1 Mbyte = 8 Mbits) divided by the number of bits per IC (256k × 4 = 1 Mbit) = $\boxed{8\ \text{ICs}}$.

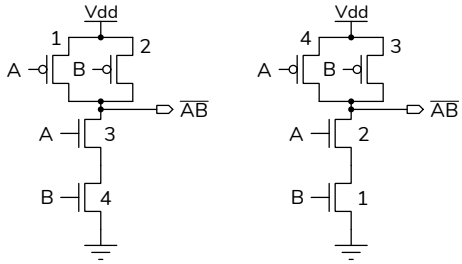
(b) The memory has 1 MByte / 2 bytes/word = 512 k words. Each bank will supply 256 k words so 521 k/256 k = $\boxed{2\ \text{banks}}$ are needed.

(c) Each memory IC has 256k words so the number of address lines required is $\log_2(256\text{ k} = \log_2(2^8 \times 2^{10}) = \boxed{18\ \text{bits}}$.

(d) The last address will be the start address plus the memory size minus one (these are all byte addresses). This can be computed as **`0x10 0000`** + 1 M − 1 = **`0x10 0000`** + $2^{20} - 1 = 1,048,576 + 1,048,576 - 1 = \boxed{2{,}097{,}151 = \texttt{0x1f ffff}}$.

## Question 8 (1 mark)

The schematic below shows a CMOS NAND gate with the four transistor numbered from 1 to 4. Which transistors are on (conducting) when the output is low? You answer should be between zero and four digits. Both **A** inputs are at the same logic level. Both **B** inputs are at the same logic level.




### Solution

For the output to be low, both transistors at the "top" of the totem-pole output must be off and both on the "bottom" must be on. In the schematic on the left transistors $\boxed{3, 4}$ must be on while in the schematic on the right transistors $\boxed{1, 2}$ must be on.

## Question 9 (5 marks)

For each term in the left column write the number of the most appropriate match in the right column. There is only one best match for each term. No marks will be deducted for wrong answers.

| | |
|---|---|
| totem pole | |
| PLD | |
| RAM | |
| DIP | |
| open-collector | |

The same set of choices was presented with different orders:

| | |
|---|---|
| (1) FPGA | (1) through-hole |
| (2) through-hole | (2) FPGA |
| (3) volatile | (3) pull-up |
| (4) pull-up | (4) CMOS |
| (5) CMOS | (5) volatile |

### Solution

Comparing the terms we can find these matches:

**totem pole** *CMOS* logic outputs use a totem pole output transistor configuration

**PLD** *FPGA*'s are Programmable Logic Devices

**RAM** memories are *volatile*

**DIP** packages are *through-hole* mounted

**open-collector** outputs require a *pull-up* resistor

The matches are shown in the table below along with the numbering in each version of the question:

| | | | |
|---|---|---|---|
| totem pole | CMOS | 5 | 4 |
| PLD | FPGA | 1 | 2 |
| RAM | volatile | 3 | 5 |
| DIP | through-hole | 2 | 1 |
| open-collector | pull-up | 4 | 3 |

## Question 10 (1 mark)

A logic signal interface has an output low logic level of $V_{OL(max)}$ of 0.7 (0.5) V and a $V_{IL(max)}$ of 1.4 (1.5) V. What is the noise margin for low logic levels?

### Solution

The noise margin is the absolute value of the difference between the guaranteed output level and the required input level. In this case:

$$|V_{IL(max)} - V_{OL(max)}| =$$
$$|1.4 - 0.7| = \boxed{0.7\text{ V}} \left(\text{or } |1.5 - 0.5| = \boxed{1\text{ V}}\right) .$$

## Question 11 (2 marks)

A CMOS digital logic circuit is operating with a supply voltage of 5 V at a clock frequency of 10 (5) MHz. You want to reduce the power consumption of this circuit by a factor of 10 (i.e. by 90%).

(i) If you were to change only the clock frequency, what frequency would you need to use?

(ii) If you were to change only the supply voltage, what voltage would you need to use?

### Solution

Power consumption in CMOS digital logic circuits is proportional to the frequency and to the square of the voltage. The power consumption at two voltages and two clock rates can thus be written as:

$$\frac{P_2}{P_1} = \frac{f_2}{f_1} \cdot \left(\frac{V_2}{V_1}\right)^2$$

(i) If only the clock frequency changes then $f_2 = f_1 \cdot \frac{P2}{P1} = 0.1 \cdot 10 \text{ (or 5)} = \boxed{1 \text{ (or 0.5) MHz}}$.

(ii) If only the supply voltage changes then $V_2 = V_1\sqrt{\frac{P2}{P1}} = 5 \cdot \sqrt{0.1} = \boxed{1.58\text{ V}}$.