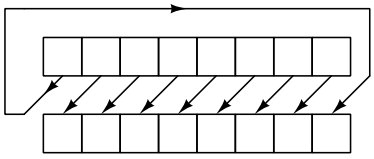
Solutions to Midterm 2

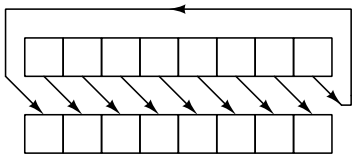
There were two versions of each question. The values and the answers for both versions are given below.

Question 1

Write *one* System Verilog statement that implements an 8-bit register named **rr** that *rotates* its contents left (or right) when a **rot** signal is asserted (true). Rotate means that the contents of the register are shifted left (or right) and the leftmost (or rightmost) bit is shifted in on the right (or left) as shown in the following diagram:



or:



Do not include a module definition. You can assume that **rr** has been declared as **logic [7:0] rr**, that **rot** has been declared as **logic rot**, and that there is a clock signal named **clk**. Follow the course coding guidelines but omit comments.

Answers

The register can be rotated by concatenating the bit that would be shifted out and the remaining bits, in the correct order. The boxes in the simulation below show the rotate-left and rotate-right statements.

```
module midterm2q1 ;

   logic [7:0] rr = 8'hf0 ;
   logic clk = '0, rot = '0 ;
```

```
   // rotate left
   always_ff @(posedge clk)
     rr <=  rot ? { rr[6:0], rr[7] } : rr ;
```

```
   // rotate right
   always_ff @(posedge clk)
     rr <=  rot ? { rr[0], rr[7:1] } : rr ;
```

```
   always @(negedge clk) $write("%8b\n",rr) ;
   always   #1 clk = ~clk ;
   initial  #4 rot = '1 ;
   initial #16 $finish ;

endmodule
```

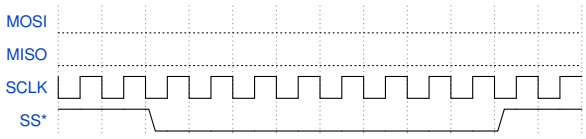
The simulation results for rotate-left and rotate-right are shown below. The first two rows are for **rot** not asserted.

```
# run -all
# 11110000
# 11110000
# 11100001
# 11000011
# 10000111
# 00001111
# 00011110
# 00111100
# 01111000
# 11110000
```

```
# run -all
# 11110000
# 11110000
# 01111000
# 00111100
# 00011110
# 00001111
# 10000111
# 11000011
# 11100001
# 11110000
```

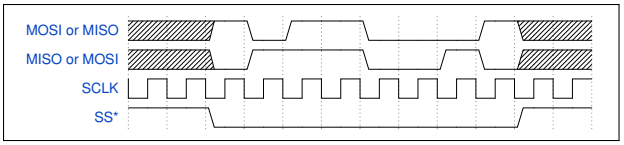
Question 2

The diagram below shows the waveforms of an SPI interface that follows the conventions described in the lecture notes. Fill in *one* of the two missing waveforms so that the 8-bit value 8'hb1 (or 8'h72) is transferred from the **slave to the master** (or the **master to the slave**). *Note: No marks will be awarded if you fill in both waveforms.*



Answers

Bits are transferred most-significant-bit (msb)-first on rising clock edges where **SS*** is asserted (low). **MOSI** is used in the master-to-slave direction and **MISO** in the slave-to-master direction. 8'hb1 is **8'b1011_0001** and 8'h72 is **8'b0111_0010**. Thus the waveforms would be:



Question 3

Write System Verilog statements to be placed at the top level of a testbench module to wait until the variable **clkcnt** is equal to 5000 (or 8000) and then print the string **Simulation Complete** and end the simulation.

Write only the statements required to accomplish this, not the complete testbench. You may assume **clkcnt** has been declared as **integer clkcnt**.

Hint: "top level" means not *inside an* ***initial*** *or* ***always*** *block.*

Answers

The code should be part of an **initial** or **always** statement if it is to be placed at the top level of the testbench. A **wait** statement with the condition **clkcnt == 5000** (or **clkcnt == 8000**) can block (pause execution of subsequent statements) until the condition is true. A **$display** or **$write** system task can be used to print the message and **$finish** can terminate the simulation (**$stop** will pause the simulation; this was also marked correct). The listing below shows a testbench; the answer is in the box.

```
module midterm2q2 ;

   integer clkcnt = 0 ;

   always #1 clkcnt += 1 ;

   initial begin
      wait ( clkcnt == 5000 ) ;
      $display ( "Simulation Complete" ) ;
      $finish ;
   end

endmodule
```

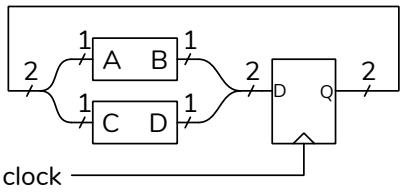
Alternatively, an unconditional (**for**, **while**, **do** or **forever**) loop could be used with an **if** statement:

```
   initial begin
      while ( 1 ) begin
         @(clkcnt) if ( clkcnt == 5000 ) begin
            $display ( "Simulation Complete" ) ;
            $finish ;
         end
      end
   end
```

@(clkcnt) blocks until the value of **clkcnt** changes; otherwise the loop would execute continuously and prevent execution of other statements.

Question 4

The digital circuit below has a 2-bit register with a minimum setup time requirement of 5 (or 10) ns. The combinational logic path from A to B has a (maximum) delay of 10 ns. The combinational logic path from C to D has a (maximum) delay of 20 ns. The register's clock-to-output delay is 5 ns (maximum). What is the maximum clock *frequency* at which this circuit will operate reliably?

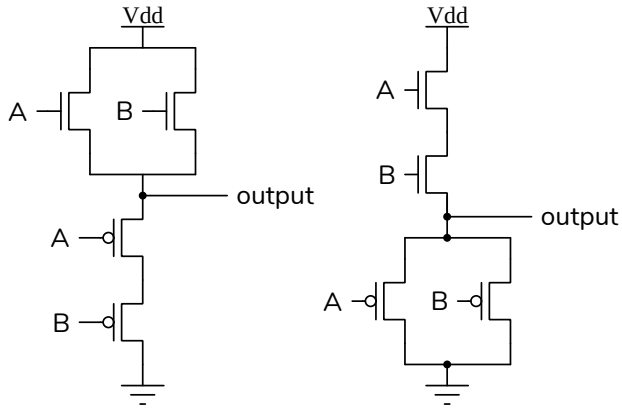


Answers

The question gives t_{SU}=5 (or 10) ns, t_{CO}=5 ns and the maximum propagation delay through any combinational logic path as t_{PD}=20 ns. The minimum clock period (maximum frequency) is achieved when the slack is zero which implies $t_{clock}=t_{CO}+t_{PD}+t_{SU}$ = 5+5 (or 10)+20 = 30 (or 35) ns. The corresponding maximum frequency is $1/30\,\text{ns}$ = $\boxed{33.3\,\text{MHz}}$ (or $1/35\,\text{ns}$ = $\boxed{28.6\,\text{MHz}}$).

Question 5

What logic function of the inputs A and B does the following schematic implement? *Briefly* explain your reasoning or show how you obtained your answer. *Hint1: The arrangement of the N- and P-channel MOSFETs is reversed from the NAND and NOR gates in the lecture notes. Hint2: Work out which transistors are on and which are off for all four possible input combinations.*



Answers

The following table shows the state of the top- and bottom-halves of the totem-pole output:

first figure				second figure			
A	B	top	bottom	A	B	top	bottom
L	L	off	on	L	L	off	on
L	H	on	off	L	H	off	on
H	L	on	off	H	L	off	on
H	H	on	off	H	H	on	off

The top half of the totem-pole output in the first figure conducts if A OR B is high. The bottom half does *not* conduct if A OR B is high. Thus the output is the OR function of A and B.

The top half of the totem-pole output in the second figure conducts if A AND B are high. The bottom half does *not* conduct if A AND B are high. Thus the output is the AND function of A and B.