Solutions to Final Exam

There were two versions of most questions. The questions and answers for both versions are given below.

Question 1

A state machine controls a door that opens for 6 seconds when a button is pushed. A **button** input is asserted when the button is pushed. **open** and **close** outputs control a motor. The **closed** and **opened** inputs are asserted when the door is closed or opened respectively. The **clk** clock input frequency is 100 Hz. The Verilog module implementing this state machine is declared:

```
module controller ( input logic clk, button, opened,
    ↪ closed, output logic open, close ) ;
```

Neither output should be asserted until **button** is asserted. Then **open** should be asserted until **opened** is asserted. Then there should be a 6 second delay with neither output asserted. Then **close** should be asserted until **closed** is asserted. Then the state machine should wait for the next push of **button**.

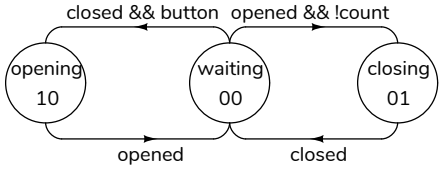
Write a **controller** module to implement this state machine. You may assume the controller starts in a valid state. Declare any additional signals required. Follow the course coding conventions but do not include comments. *Hints: Start by deciding on the states, their encodings, and the outputs in each state. You can implement the delay with a counter that is decremented in the state where the door is open and is initialized to its maximum value in other states.*

Answer

From the description there are three combinations of outputs required. For a Moore state machine each of these must be a different state:

state	open	close
opening	1	0
closing	0	1
waiting	0	0

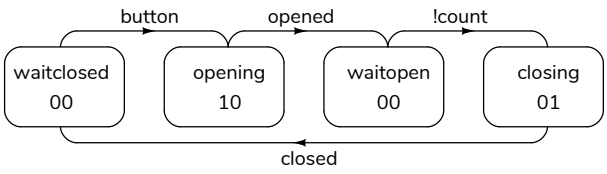
Since each state has a different output we can use an "output" state encoding in which the state variables are the **open** and **close** outputs. From the description we can derive a state transition diagram:



The delay can be implemented with a 9 (or more) bit counter that counts from 599 to zero in the waiting state and is reset to 599 in the opening state. The Verilog for this implementation is:

```
module controller
   ( input logic clk, button, opened, closed,
     output logic open, close ) ;

   logic [1:0] state ;
   logic [9:0] count ;

   assign {open,close} = state ;

   always_ff @(posedge clk) state
      <= state == 2'b00 && closed && button ? 2'b10:
         state == 2'b00 && opened && !count ? 2'b01:
         state == 2'b10 && opened ? 2'b00 :
         state == 2'b01 && closed ? 2'b00 :
         state ;

   always_ff @(posedge clk)
      count <= state == 2'b00 ? count - 1 : 599 ;

endmodule
```

We can simplify the transition conditions by using two "waiting" states where both outputs are off:



In this case we can't use an "output" encoding because there are two states with output 00 so a binary encoding is used in the following solution:

```
module controller
   ( input logic clk, button, opened, closed,
     output logic open, close ) ;

   logic [1:0] state ;
   logic [9:0] count ;

   always_ff @(posedge clk)
```

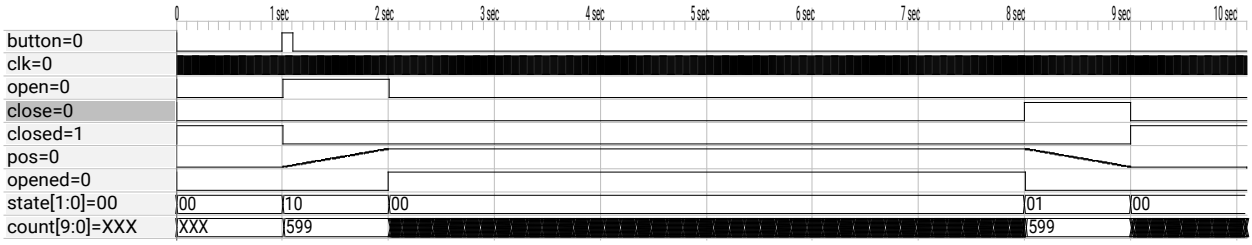


Figure 1: Simulation results for Question 1 (first implementation).

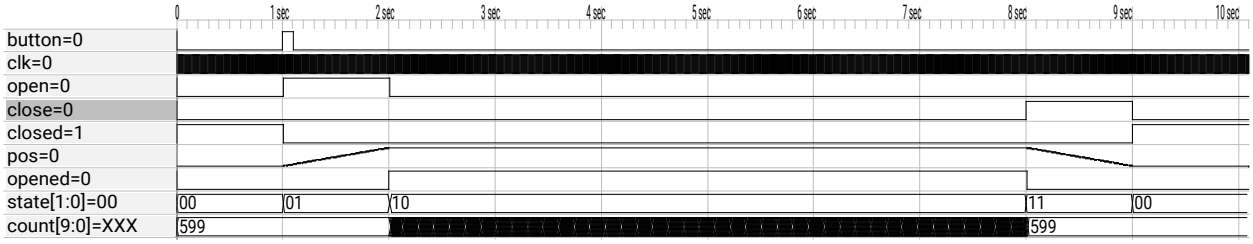


Figure 2: Simulation results for Question 1 (second implementation).

```
      state <= state == 0 && button ? 1 : // waitclosed
               state == 1 && opened ? 2 : // opening
               state == 2 && !count ? 3 : // waitopen
               state == 3 && closed ? 0 : // closing
               state ;

   always_ff @(posedge clk)
      count <= state == 2 ? count - 1 : 599 ;

   assign {open,close}
      = state == 0 ? 2'b00 :
        state == 1 ? 2'b10 :
        state == 2 ? 2'b00 :
        state == 3 ? 2'b01 : 2'b00 ;

endmodule
```

It's also possible to implement individual state machines for **open** and **close**. The first state transition diagram above can be converted into a state transition truth table[1]:

state		opened	closed	button	count	next	
0	0	x	1	1	x	1	0
0	0	1	x	x	0	0	1
1	0	0	x	x	x	1	0
0	1	x	0	x	x	0	1

from which we can write sum-of-product equations for the next-state values of **open** and **close**:

```
module controller
   ( input logic clk, button, opened, closed,
     output logic open, close ) ;

   logic [9:0] count ;

   always_ff @(posedge clk)
      open = ( !open && !close && closed && button ) ||
             ( open && !close && ! opened ) ;

   always_ff @(posedge clk)
      close = ( !open && !close && opened && !count ) ||
              ( !open && close && ! closed ) ;

   always_ff @(posedge clk)
      count <= open ? 599 : count - 1 ;

endmodule
```

This is less clear than the previous solutions but shows which variables must be included in expressions for the next values of **open** and **close**.

A testbench is shown below and simulation results are show in Figures 1 and 2.

```
module controller_tb ;

   logic clk, button, opened, closed ;
   logic open, close ;
   int pos = 0 ;      // door percent open

   controller c0 (.*) ;

   initial begin
      $dumpfile("controller.vcd") ;
      $dumpvars ;

      c0.state = '0 ;
      {c0.open,c0.close} = '0 ;
      {clk,button,opened,closed} = 4'b0001 ;
      #1s button = 1 ;
```

[1] Only rows with non-zero values for the next state are shown.

```
    #0.1s button = 0 ;
    #9s ;
    $finish ;
  end

  always #5ms clk <= !clk ;

  // emulate door opener and limit switches with 1
  // second opening and closing delays
  always begin
    #0.01s ;
    pos += pos < 100 && open  ? 1 : 0 ;
    pos -= pos >   0 && close ? 1 : 0 ;
    opened = pos == 100 ;
    closed = pos == 0 ;
  end

endmodule
```

Question 2

A logic circuit consumes 10 (or 20) mW with a clock frequency of 1 MHz and a voltage of 5 V. What clock frequency would allow this circuit to operate for one year (8760 hours) from a 2 V, 200 mA-hour battery?

Answer

The capacity of the battery is $C = I_2 T$ where I_2 is the current supplied for $T = 8760$ hours. The power the battery supplies for a year is $P_2 = V_{\text{bat}} \cdot I_2 = \frac{V_{\text{bat}} \cdot C}{T}$.

There are two possible solutions depending on your assumptions about the voltage supplied to the circuit (V_2).

Supply voltage changes to 2 V. In this case the circuit operates at the battery voltage, $V_2 = V_{\text{bat}} = 2\,\text{V}$.

Solving the power consumption equation for f_2 and substituting for P_2:

$$f_2 = f_1 \frac{P_2}{P_1} \left(\frac{V_1}{V_2}\right)^2 = f_1 \frac{V_{\text{bat}} C}{P_1 T} \left(\frac{V_1}{V_2}\right)^2$$

Substituting $f_1 = 1\,\text{MHz}$, $V_2 = V_{\text{bat}} = 2\,\text{V}$, $P_1 = 10\,\text{mW}$ (or $20\,\text{mW}$), $C = 200\,\text{mA–h}$, $T = 8760\,\text{h}$, and $V_1 = 5\,\text{V}$:

$$f_2 = 1 \times 10^6 \frac{2 \cdot 200}{10 \cdot 8760} \left(\frac{5}{2}\right)^2$$

$\boxed{f_2 = 28.5\,\text{kHz}}$ (or $\boxed{f_2 = 14.3\,\text{kHz}}$).

Supply voltage remains at 5 V. In this case a (lossless) voltage converter converts the battery's 2 V output to $V_2 = V_1 = 5\,\text{V}$.

Solving the power consumption equation for f_2 and substituting for P_2:

$$f_2 = f_1 \frac{P_2}{P_1} \left(\frac{V_1}{V_2}\right)^2 = f_1 \frac{V_{\text{bat}} C}{P_1 T} \left(\frac{V_1}{V_2}\right)^2$$

Substituting $f_1 = 1\,\text{MHz}$, $V_2 = V_1 = 5\,\text{V}$, $V_{\text{bat}} = 2\,\text{V}$, $P_1 = 10\,\text{mW}$ (or $20\,\text{mW}$), $C = 200\,\text{mA–h}$, and $T = 8760\,\text{h}$:

$$f_2 = 1 \times 10^6 \frac{2 \cdot 200}{10 \cdot 8760} \left(\frac{5}{5}\right)^2$$

$\boxed{f_2 = 4.57\,\text{kHz}}$ (or $\boxed{f_2 = 2.28\,\text{kHz}}$).

Question 3

Fill the table below with the value of each expression as a Verilog numeric literal including the correct width and the correct value in *hexadecimal* base. Assume the following declarations:

```
logic [7:0] x ;
logic [3:0] y ;
```

and that `x` has the value `8'h38` (or `8'hA6`) and that `y` has the value `4'b0101`. The first row has been filled in as an example. You need not show your work or draw another box around the answer

expression	value
`x[3:0]`	`4'h8` (or `4'h6`)
`x[1] & x`	
`x + 2 >> 1`	
`x >> 1 * 2`	
`x > {y,y} ^ x != x`	
`x[3] ? y[0] : y`	
`0 ? 1 ? 2 ? 3 : 4 : 5 : 6`	

Answer

For `x=8'hA6` (`8'd166`, `8'b1010_0110`) and `y = 4'b0101`:

expression	value
x[3:0]	4'h6
x[1] & x	8'h0
x + 2 >> 1	32'h54
x >> 1 * 2	8'h29
x > {y,y} ^ x != x	1'h1
x[3] ? y[0] : y	4'h5
x[1] ? 2 ? 3 : 4 : 5	32'h3

For `x=8'h38` (`8'd56`, `8'b0011_1000`) and `y = 4'b0101`:

expression	value
x[3:0]	4'h8
x[1] & x	8'h0
x + 2 >> 1	32'h1d
x >> 1 * 2	8'he
x > {y,y} ^ x != x	1'h0
x[3] ? y[0] : y	4'h1
x[1] ? 2 ? 3 : 4 : 5	32'h5

Question 4

Write a System Verilog testbench named **`mult_tb`** to test a **`mult`** module declared as:

```
module mult (
   input logic [31:0] a, b,
   output logic [31:0] c,
   input logic clk, valid,
   output logic ready ) ;
```

Your testbench must instantiate the module and supply it with a 10 MHz clock on **`clk`**.

The testbench must do the following, in order: set **`a`** to **`32'd11`** and **`b`** to **`32'd2`**, assert **`valid`**, wait until **`ready`** is asserted, print the string **`"error"`** if the value of **`c`** is not equal to **`32'd22`**, and terminate the simulation.

Declare all signals used in your testbench. Do not write or declare the **`mult`** module. Do **not** do anything else in your testbench such as reading or writing files (including **`.vcd`** files). Omit comments.

Answer

```
module mult
   ( input logic [31:0] a, b,
     output logic [31:0] c,
     input logic clk, valid,
     output logic ready ) ;

   logic [5:0] n = 0 ;

   assign c = a * b ; // + 1 ;

   always_ff @(posedge clk)
      n <= ready && valid ? 31 : n ? n-1 : 0 ;

   assign ready = !n ;

endmodule

module mult_tb ;

   logic [31:0] a, b, c ;
   logic clk, valid, ready ;

   mult m0 (.*) ;

   initial clk = '0 ;
   always #0.05us clk = ~clk ;

   initial begin
      a = 11 ;
      b = 2 ;
      valid = '1 ;
      wait ( ready ) ;
      // or:
      while ( ready !== 1 ) @(ready) ;
      $display(a,b,c) ;
      if ( c != 22 ) $write("error") ;
      $finish ;
   end


endmodule
```

Question 5

A logic family has $V_{IH}(\text{min}) = 0.8$ (or 0.7) V and $V_{IL}(\text{max}) = 0.3$ (or 0.4) V. What $V_{OH}(\text{min})$ and $V_{OL}(\text{max})$ levels would result in a noise margin of 0.2 V for both high and low logic levels?

Answer

Solving for $V_{OH}(\text{min})$ and $V_{OL}(\text{max})$:

$V_{OH(\text{min})} = V_{IH(\text{min})} + \text{noise margin(high)} = 0.8 + 0.2 = \boxed{1\,\text{V}}$ (or $0.7 + 0.2 = \boxed{0.9\,\text{V}}$).

and

$V_{\texttt{OL(max)}} = V_{\texttt{IL(max)}} - \text{noise margin(low)} = 0.3 - 0.2 = \boxed{0.1\,\text{V}}$ (or $0.4 - 0.2 = \boxed{0.2\,\text{V}}$).

Question 6

What is the maximum clock frequency that would result in reliable operation for a circuit whose maximum delay through any combinational logic path is 50 (or 30) ns and that uses registers with a clock-to-output delay of 2 ns and a minimum setup time requirement of 10 (or 6) ns?

Answer

At the maximum clock frequency slack is zero and

$$t_{\text{SU(required)}} = T_{\text{clock}} - t_{\text{CO(max)}} - t_{\text{PD(max)}}$$

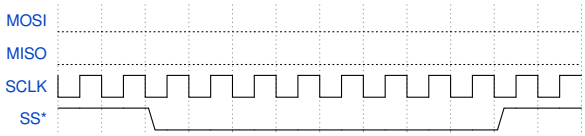
Solving for T_{clock}:

$$T_{\text{clock}} = t_{\text{SU(required)}} + t_{\text{CO(max)}} + t_{\text{PD(max)}}$$

and $T_{\text{clock}} = 10 + 2 + 50 = 62$ ns which is clock frequency of $\boxed{16.1\,\text{MHz}}$ (or $6 + 2 + 30 = 38$ ns which is a clock frequency of $\boxed{26.3\,\text{MHz}}$).

Question 7

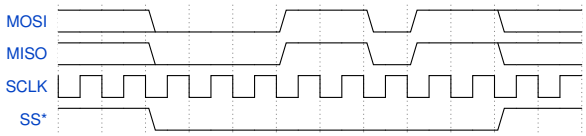
The diagram below shows the waveforms of an SPI interface that follows the conventions described in the lecture notes. Fill in *one* of the two missing waveforms so that the 8-bit value 8'h1b (or 8'h27) is transferred from the **slave to the master** (or the **master to the slave**). *Note: No marks will be awarded if you fill in both waveforms.*



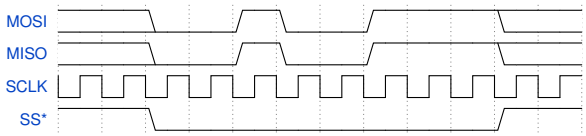
Answer

For data transfer from **slave to the master** the waveform will appear **only on MISO**. For data transfer from **master to the slave** the waveform will appear **only on MOSI**. The waveforms for the two values transferred are shown below:

The data bits are transferred most-significant-bit (msb) first and the data is captured on the rising edge of SCLK. For the transfer of 8'h1b the waveforms are:



For the transfer of 8'h27 the waveforms are:



Question 8

You wish to sample the signal from a sensor so that it can be analyzed on a computer. The signal contains frequency components up to 100 kHz (or 50 kHz). You would like the quantization noise power to be less than 0.5 (or 0.1)% of the signal power. You can assume that the equation for the quantization SNR of a sine wave applies to this signal.

(a) What sampling rate(s) should you use?

(b) What is the minimum number of bits of ADC resolution that should be used?

Answer

(a) The sampling rate must be greater than twice the highest frequency. In this case $f_s > 2 \times 100\,\text{kHz} = \boxed{200\,\text{kHz}}$ (or $2 \times 50\,\text{kHz} = \boxed{100\,\text{kHz}}$.

(b) If the noise is 0.5% of the signal power, then $N/S = 0.005$, $S/N = 200$ and $SNR(dB) = 10\log(200) \approx 23$ dB (or $10\log(1000) \approx 30$ dB).

Solving the equation for quantization SNR for the number of bits of resolution, $n = (\text{SNR(dB)} - 1.76)/6$, and $n = (23 - 1.76)/6 \approx 3.54$ so we should use $\boxed{4\text{ bits}}$ (or $n = (30 - 1.76)/6 \approx 4.71$ so we should use $\boxed{5\text{ bits}}$).

Question 9

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.

Moore's Law	
ASIC	
FPGA	
fabless	
wafer	
die	
NRE	
SoC	
LE	
LUT	

(1) programmable logic with 1000's of flip-flops
(2) 300mm diameter
(3) made for a specific application
(4) historical rate of increase
(5) often a few square mm in area
(6) a memory storing output values
(7) LUT plus a flip-flop
(8) product development costs
(9) ASIC plus CPU
(10) most semiconductor companies

or:

(1) programmable logic with 1000's of flip-flops
(2) often a few square mm in area
(3) most semiconductor companies
(4) historical rate of increase
(5) LUT plus a flip-flop
(6) a memory storing output values
(7) ASIC plus CPU
(8) product development costs
(9) 300mm diameter
(10) made for a specific application

Answer

Moore's Law	4
ASIC	3
FPGA	1
fabless	10
wafer	2
die	5
NRE	8
SoC	9
LE	7
LUT	6

or:

Moore's Law	4
ASIC	10
FPGA	1
fabless	3
wafer	9
die	2
NRE	8
SoC	7
LE	5
LUT	6