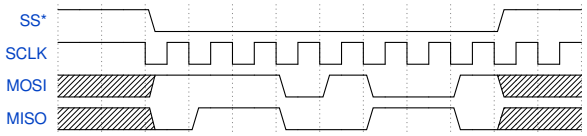
ELEX 2117 : Digital Techniques 2
2023 Fall Term

Solutions to Midterm 2

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

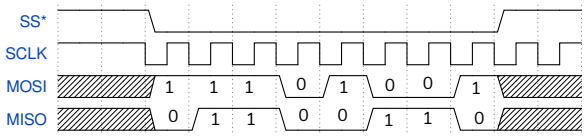
Question 1

The following waveform shows the signals on an SPI interface. What value was transmitted *from the master to the slave* (or *from the slave to the master*) assuming the bits were transferred most-significant-bit first? Give you answer as a Verilog numeric literal in hexadecimal base, including the width. Show your work.



Answers

The convention described in the lecture notes is that bits are transferred on the rising edge of SCLK when $\overline{\text{SS}}$ is asserted (low). The following diagram shows the value of the bits transferred in each direction:



From the master to the slave (MOSI) the value transferred is 8'b1110_1001 which is `8'hE9`.

From the slave to the master slave (MISO) the value transferred is 8'b0110_0110 which is `8'h66`.

Question 2

For the following testbench:

```
module mt2 ;

   integer n ;
   logic clk ;

   initial begin
      // $dumpfile("ex74a.vcd"); $dumpvars ;
      n = 1 ;
      clk = 0 ;
      wait ( n >= 5 ) $stop ;
   end

   always #4us clk = ~clk ;

   always @(posedge clk) begin
      $display("n = %d",n) ;
      n = n + 1 ;
   end

endmodule
```

or

```
module mt2 ;

   integer n ;
   logic clk ;

   initial begin
      // $dumpfile("ex74b.vcd"); $dumpvars ;
      n = 0 ;
      clk = 0 ;
      wait ( n > 5 ) $stop ;
   end

   always #2us clk = ~clk ;

   always @(posedge clk) begin
      $display("n = %d",n) ;
      n = n + 1 ;
   end

endmodule
```

(a) What is the frequency of the **clk** signal?

(b) Write the first line that is printed by this testbench:

(c) Write the last line that is printed by this testbench:

Answers

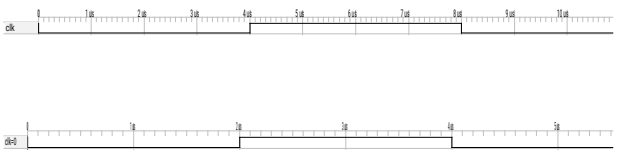
(a) The clock is inverted once every 4 µs (or 2 µs) so the period is twice this, 8 µs (or 4 µs). The frequency is twice this, 125 kHz (or 250 kHz).

(b) The first line printed by this testbench is for the initial value of **n** which is 1 (or 0).

(c) The **always** block prints the value of **n** and increments it. The simulation stops when the **wait()** condition first becomes true. This happens when **n>=5** (5) (or when **n>5** (6)). The last

line printed will be for the previous value of **n**, which is 4 (or 5).

The following waveforms show the clock period in each case:



The following listings show the simulation output for each case:

```
# n =          1
# n =          2
# n =          3
# n =          4
```

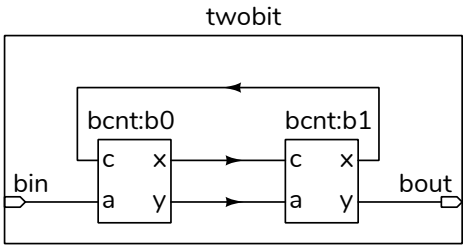
and

```
# n =          0
# n =          1
# n =          2
# n =          3
# n =          4
# n =          5
```

Question 3

The following Verilog shows the declaration of a module named **bcnt**. The diagram shows how two of these are connected within a module named **twobit**:

```
module bcnt
( input  logic c, a,
  output logic x, y ) ;
// ...
endmodule
```

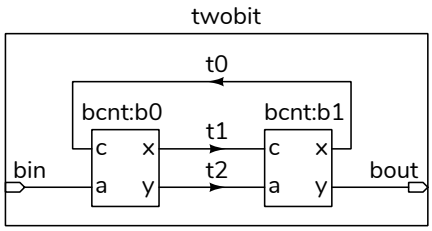


The identifiers above the boxes show the module and instance names. The identifiers above the ports show the **twobit** module's port names.

Write a System Verilog module named **twobit** that implements the diagram above. Declare any signal(s) required to implement the **twobit** module. Do not write the **bcnt** module. You may use any style to connect signals to ports. Follow the course coding guidelines but omit comments.

Answers

The following schematic shows the names of three internal signals used to connect the two **bcnt** modules:



The code using positional signal-to-port mappings is:

```
module twobit
  ( input logic bin,
    output logic bout ) ;

   logic t0, t1, t2 ;

   bcnt b0 (t0, bin, t1, t2  ) ;
   bcnt b1 (t1, t2,  t0, bout) ;

endmodule
```

The code using explicit signal-to-port mappings is:

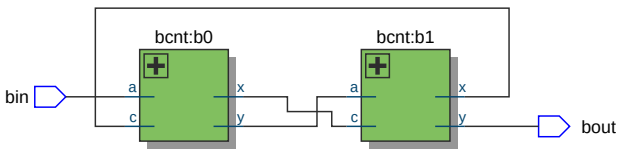
```
module twobit
  ( input logic bin,
    output logic bout ) ;

   logic t0, t1, t2 ;

   bcnt b0 (.c(t0), .a(bin), .x(t1), .y(t2)  ) ;
   bcnt b1 (.c(t1), .a(t2),  .x(t0), .y(bout)) ;

endmodule
```

The following schematic is generated by Quartus for either solution:



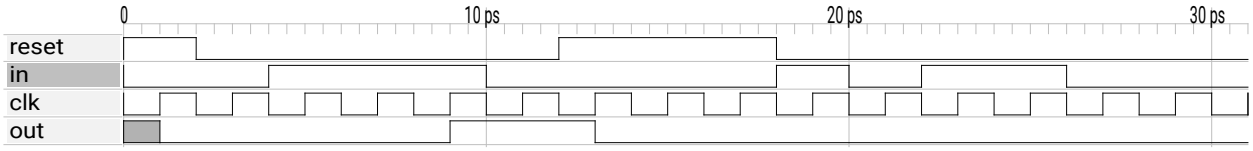


Figure 1: Simulation results for Question 4 solution.

Question 4

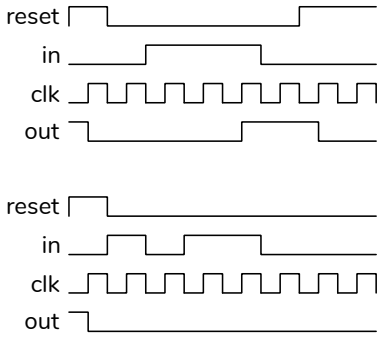
A state machine has an output named **out** and three inputs named **reset**, **clk**, and **in**.

out changes only on the rising edge of the clock, **clk**. **out** is set to 0 when **reset** is asserted. **out** is set to 1 when **reset** has been low and **in** has been high for 3 consecutive rising edges of the clock. Once **out** is set to 1, it remains set to 1 until **reset** is asserted. Example waveforms are shown.

Fill in the missing code in the following Verilog module so as to implement this state machine. You may use any state encoding. Follow the course coding guidelines but omit comments.

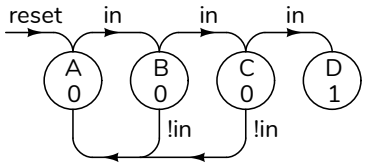
```
module pulsedetect
( input logic reset, clk, in,
  output logic out ) ;

endmodule
```



Answers

The state transition diagram below shows a possible solution. The states are labelled A through D.



Three different solutions are shown below along with a testbench. The first solution implements the state transition diagram shown above using a binary state encoding. The second solution uses a shift register to store the previous three input bits until there are three consecutive ones. The third solution uses two state machines: one counts the number of consecutive 1 bits and the second is a flip-flop that's set when the count will reach 3 and only cleared on reset.

```
module pulsedetect
  ( input logic reset, clk, in,
    output logic out ) ;

   logic [1:0] count ;

   always_ff @(posedge clk)
     count <= reset ? 2'b00 :
              count == 2'b00 &&  in ? 2'b01 :
              count == 2'b01 &&  in ? 2'b10 :
              count == 2'b10 &&  in ? 2'b11 :
              count == 2'b01 && !in ? 2'b00 :
              count == 2'b10 && !in ? 2'b00 : count ;

   assign out = count == 2'b11 ? '1 : '0 ;

endmodule

module pulsedetect
  ( input logic reset, clk, in,
    output logic out ) ;

   logic [2:0] bits;

   always_ff @(posedge clk)
     bits <= reset ? '0 : &bits ? bits : {bits,in} ;

   assign out = &bits ;

endmodule

module pulsedetect
  ( input logic reset, clk, in,
    output logic out ) ;

   logic [1:0] cnt ;

   always_ff @(posedge clk)
     cnt <= !reset && in ? cnt + 1'b1 : '0 ;

   always_ff @(posedge clk)
     out <= reset ? '0 : in && cnt == 2'b10 ? '1 : out ;

endmodule
```

```
module pulsedetect_tb ;

   logic reset, clk, in ;
   logic out ;

   pulsedetect p0 (.*) ;

   initial begin
      $dumpfile("ex76_tb.vcd");
      $dumpvars ;
      {reset,clk,in} = 3'b100 ;

      // first sample waveform
      #2 reset = '0 ;
      #2 in = '1 ;
      #6 in = '0 ;
      #2 reset = '1 ;
      #4 ;

      // second sample waveform
      #2 reset = '0 ; in = '1 ;
      #2 in = '0 ;
      #2 in = '1 ;
      #4 in = '0 ;
      #6 ;
      $stop ;
   end

   always
     #1 clk = ~clk ;

endmodule
```

Each solution produces the simulation results in Figure 1 which matches the examples.