Solutions to Midterm 2

There were two versions of each question. The answers for both versions are given below. Revision 2: Corrected answer for Question 2.

Question 1

Write a Verilog module named **`xmark`** (or **`pmark`**) that has a single-bit reset input named **`reset`**, a single-bit clock input named **`clock`** and a single-bit **`logic`** output named **`mark`**.

This module should implement two counters. If **`reset`** is asserted both counters should be set to zero on the next rising clock edge.

Otherwise, the first counter should continuously count down from 30 to zero and the second counter should continuously count down from 12 to zero. The two counters operate independently except when **`reset`** is asserted.

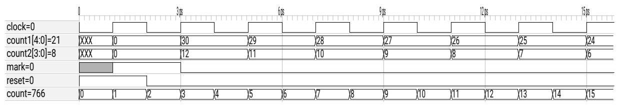
The **`mark`** output should be set high when both counters have a value of zero, and should be set low otherwise.

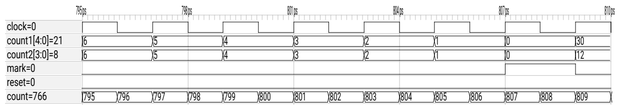
Answers

A possible solution is shown below along with a testbench and the simulation waveforms at the start and end of the simulation. The period of the **`mark`** signal is the product of the two counter periods ($31 \times 13 = 403$ clock cycles)[1].

```
module xmark
   ( input  logic reset, clock,
     output logic mark ) ;

   logic [4:0] count1 ;
   logic [3:0] count2 ;

   always_ff @(posedge clock)
     count1 <= reset ? '0 :
               !count1 ? 5'd30 : count1 - 1'b1 ;

   always_ff @(posedge clock)
     count2 <= reset ? '0 :
               !count2 ? 4'd12 : count2 - 1'b1 ;

   assign mark = !count1 && !count2 ;

endmodule

// synthesis translate_off

module mark_tb ;

   logic reset, clock, mark ;

   int count = 0 ;

   xmark xmark0 (.*) ;

   initial begin
      $dumpfile("midterm2023-2.vcd");
      $dumpvars ;
      {reset,clock} = 2'b10 ;
      #2 reset=0 ;
      #2 wait ( mark ) ;
      #4 $finish ;
   end

   always begin
      #1 clock = ~clock ;
      count++ ;
   end

endmodule
```

[1]This is because the two counter periods are relatively prime





Question 2

What are the period and duty cycle of the active-high **`logic`** signal **`x`** that is generated by the following Verilog in a testbench:

```
always begin
   #10ns x = 1'b0 ;
   #20ns x = 1'b1 ;
end
```

or

```
always begin
   #20ns x = 1'b0 ;
   #10ns x = 1'b1 ;
end
```

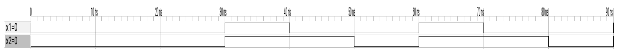
Answers

The signal **x** alternates between 0 and 1. In the first example the delay before **x** is changed to 0 (the high duration) is 10 ns and the low duration is 20 ns. Thus the period is 30 ns and since the signal is active-high the duty cycle is 10/30 or 33%.

In the second example the high and low durations are reversed so the period is still 30 ns but the duty cycle is 20/30 or 66%.

A simulation is shown below:

```
module q2 ;

   logic x1, x2 ;

   initial begin
      $dumpfile("midter2023-2-2.vcd") ;
      $dumpvars() ;
      {x1,x2} = '0 ;
      #100ns $finish ;
   end

   always begin
      #10ns x1 = 1'b0 ;
      #20ns x1 = 1'b1 ;
   end

   always begin
      #20ns x2 = 1'b0 ;
      #10ns x2 = 1'b1 ;
   end

endmodule
```



Question 3

Write System Verilog code that could be included in a testbench to print the value of the logic signal **x** each time the value of the signal **y** changes. The code that changes **y** is not given. You may use any System Verilog language features and you may print the value of **x** in any format.

Answers

The simplest solution is to use **`$display`** within an **`always`** statement with an **`@(y)`**-expression to pause execution until the value of **y** changes:

```
always @(y) $display(x);
```

For example, simulating the following:

```
module q3 ;

   logic x, y ;

   initial begin
      #10 {x,y} = 2'b00 ;
      #10 {x,y} = 2'b11 ;
      #10 {x,y} = 2'b11 ;
      #10 {x,y} = 2'b00 ;
   end

   always @(y) $display($time()," ",x);

endmodule
```

outputs:

```
#                   10 0
#                   20 1
#                   40 0
```

Question 4

You wish to ensure a logic circuit will operate reliably. What is the maximum allowed propagation delay through any combinational logic path if the clock rate is 10 (or 100) MHz, the registers have a 30 (or 3) ns setup time, and a 20 (or 2) ns clock-to-output delay? Show your work.

Answers

The equation for available setup time is:

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

In this problem we are given T_{clock}, t_{SU} (avail) - and t_{CO}(max) and we need to find when the slack is zero (corresponding to $t_{\mathrm{SU}}(\text{avail}) = t_{\mathrm{SU}}(\text{min})$). Solving for t_{PD}(max):

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

In this case $T_{\mathrm{clock}} = {}^{1}/_{10\times10^{6}\,\mathrm{MHz}} = 100\,\mathrm{ns}$ (or ${}^{1}/_{100\times10^{6}\,\mathrm{MHz}} = 10\,\mathrm{ns}$) and so:

$$t_{\mathrm{PD}}\text{ (max)} = 100\,\mathrm{ns} - 30\,\mathrm{ns} - 20\,\mathrm{ns} = 50\,\mathrm{ns}$$

or:

$$t_{\mathrm{PD}}\text{ (max)} = 10\,\mathrm{ns} - 3\,\mathrm{ns} - 2\,\mathrm{ns} = 5\,\mathrm{ns}$$

Question 5

A signal is high (or low) when a motor is running. Would you name this signal `run` or $\overline{\texttt{run}}$?

Answers

An active-low signal is true when it is low. If the motor is running when a signal is low then it is active-low. Active-low signals are denoted in various ways, including an overbar. A signal that is low when a motor is running would named $\overline{\text{run}}$.

An active-high signal is true when it is high. If the motor is running when a signal is high then it is active-high. Active-high signals do not have an overbar or other indication that they are active-low. A signal that is high when a motor is running would named run.

Question 6

Is the period of a clock input (or a propagation delay) a timing requirement or a guaranteed response? Why?

Answers

The period of a waveform is measured between any two adjacent features of a periodic signal (e.g. rising edge). The clock input is measured to a transition on an input so it must be a timing requirement.

A propagation delay is measured from a change on an input to a change on an output. Specifications ending on an output are guaranteed responses so a propagation delay is a guaranteed response.