Use a wiki as a lab book.

Wikis are excellent tools for collaborative work (i.e. where you need to efficiently share lots of information and files with multiple people).

This week is practice for wiki usage during the project.

Both lab partners will use the same wiki (i.e. you will both submit the same link to instructor and grader).

Completeness is very important … neatness should be easy.

You may use any available wiki hosting service (wm, etc …)

→ suggested provider: http://pbwiki.com/

(... with only 10 MB of space, you can link to your W&M webspace to increase the effective memory available to your wiki).
Web page lab report

- The lab report for this week’s lab will be in the form of a webpage. The webpage should be in HTML (i.e. “filename.html”). You are free to use any webpage making program you wish, but extra credit will be awarded for those reports programmed directly by you in HTML.

- The webpage should be hosted on your public H drive space (i.e. if you name your web age “index.html” and put it in the “public_html” folder of your H drive, then you can view it at:

  " http://username.people.wm.edu/ "

- You should send a link to your “lab report webpage” by e-mail to the grader and the instructor by Monday, September 29 (midnight deadline).

- Lab report should cover lab exercises 1, 2, and 3.

- There is no length limit on the lab report.

- 1 lab report per person.
Timing pulses

&

Intro to counters

(hardware & Verilog)
Timing Pulses

- Important element of laboratory electronics

- Pulses can control logical sequences with precise timing.
  - If your detector “sees” a charged particle or a photon, you might want to signal a clock to store the time at which it occurred.
  - You could use the event to generate a standard pulse so that your clock always responds in the same way.

- Alternatively, you might need to reset your electronics after the event
  - Clearly you want the reset pulse to arrive as soon as possible after the data has been processed
  - This requires a precision time delay generator
Timing Pulses

- A simple type of delay generator...
  1. A **D-type flip-flop** receives a clock edge and goes from low to high at the output.
  2. The output charges up an **RC circuit** after going high.
  3. The charged capacitor also serves as the **clear input** to the D flip-flop.
  4. **So, that after a fixed time (roughly $RC$) the flip-flop resets back to its initial state.**
  5. The net result is a single pulse that has a duration (or **pulse width**) determined by the combination of the resistor & capacitor.

- This is called a **monostable multivibrator** or **one-shot**.
One-shot: D-type flip-flop

**FUNCTION TABLE**

<table>
<thead>
<tr>
<th>INPUTS</th>
<th>OUTPUTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRE</td>
<td>CLR</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
</tr>
</tbody>
</table>

[Texas Instruments 74LS74 flip-flop datasheet]
One-shot: D-type flip-flop

S or PRE

D

Q

R or CLR

Input

clock

Pulse Input

+ 5V

Output

FUNCTION TABLE

<table>
<thead>
<tr>
<th>PRE</th>
<th>CLR</th>
<th>CLK</th>
<th>D</th>
<th>Q</th>
<th>¬Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>H</td>
<td>X</td>
<td>X</td>
<td>H</td>
<td>L</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>X</td>
<td>X</td>
<td>L</td>
<td>H</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>X</td>
<td>X</td>
<td>H↑</td>
<td>H↑</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>↑</td>
<td>H</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>↑</td>
<td>L</td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>L</td>
<td>X</td>
<td>Q₀</td>
<td>¬Q₀</td>
</tr>
</tbody>
</table>
One-shot: 74LS123

Characteristics:

- 2 clock inputs triggered by either a **rising edge** or a **falling edge**.
- 2 outputs \((Q & \overline{Q})\).
- A **reset or clear input**, instantly sets the output to a standard condition regardless of the current state or clock level.
- Can be confused a little by pulses in quick succession.

<table>
<thead>
<tr>
<th>INPUTS</th>
<th>OUTPUTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEAR</td>
<td>A</td>
</tr>
<tr>
<td>L</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>H</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
</tr>
<tr>
<td>H</td>
<td>↓</td>
</tr>
<tr>
<td>↑</td>
<td>L</td>
</tr>
</tbody>
</table>

![Waveform Diagram]
One-shot: 74LS123

Characteristics:

- 2 clock inputs triggered by either a **rising edge** or a **falling edge**.
- 2 outputs ($Q$ & $\overline{Q}$).
- A **reset or clear input**, instantly sets the output to a standard condition regardless of the current state or clock level.
- Can be confused a little by pulses in quick succession.

<table>
<thead>
<tr>
<th>INPUTS</th>
<th>OUTPUTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEAR</td>
<td>A</td>
</tr>
<tr>
<td>L</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>H</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
</tr>
<tr>
<td>H</td>
<td>$\downarrow$</td>
</tr>
<tr>
<td>$\uparrow$</td>
<td>L</td>
</tr>
</tbody>
</table>

*Pin* is the input signal, *Pout* is the output signal, and *hW* is the width of the pulse. The armed state is indicated by the *armed* output pulse.
74LS123 usage

C R
+ 5V

CLR

[ Texas Instruments 74LS123 datasheet ]
A single one-shot will produce a variable delay pulse.

2 one-shots can be combined to produce a pulse of variable duration/width produced at variable delay after a trigger.

→ Pulse Delay Generator ... very useful in a research lab.
A single one-shot will produce a variable delay pulse.

2 one-shots can be combined to produce a pulse of variable duration/width produced at variable delay after a trigger.

→ Pulse Delay Generator … very useful in a research lab.

[Image from www.thinksrs.com]
Setting the Pulse Width

\[ t_W = K R_{\text{ext}} C_{\text{ext}} \]

with

(K IS INDEPENDENT OF R)
1. One-shots are very useful in a research laboratory as pulse delay generators.

2. **One-shots should be avoided in regular circuitry, because**
   - They are useful in *asynchronous circuits* for avoiding glitches and signal races …
   - It’s very easy to put them all over your asynchronous circuit with all the pulse timing set just right. It is very hard to figure out how the circuit works just by looking at it (or even a circuit diagram).
   - The *pulse width depends on temperature* (R, C, and chip).
   - The *pulse width depends on supply voltage*. 
Variation in Output Pulse Duration

\[ \Delta t_{w(out)} - \text{Variation in Output Pulse Duration} \]

\[ T_A - \text{Free-Air Temperature - } ^\circ\text{C} \]

- \( V_{CC} = 5\, \text{V} \)
- \( C_{\text{ext}} = 60\, \text{pF} \)
- \( R_T = 10\, \text{K ohms} \)

\[ t_{w(out)} \approx 370\, \text{ns} \]

at \( T_A = 25^\circ\text{C} \)

See Note 14

Pulse Width vs. Temperature
VARIATION IN OUTPUT PULSE DURATION

vs

SUPPLY VOLTAGE

\[ \Delta t_{w(out)} - \text{Variation in Output Pulse Duration} \]

\[ \begin{align*}
C_{ext} &= 60 \text{ pF} \\
R_{ext} &= 10 \text{ K ohms} \\
T_A &= 25^\circ \text{C}
\end{align*} \]

\[ t_{w(out)} \approx 370 \text{ ns} \]

at \( V_{CC} = 5 \text{ V} \)

\[ V_{CC} - \text{Supply Voltage} - \text{V} \]
1. Frequency dividers.

2. Counters in Verilog.

3. Ripple counter (next week).

4. Synchronous counter (next week).
JK-type flip-flops are used in counters.

Logic table for clock falling edge

<table>
<thead>
<tr>
<th>J</th>
<th>K</th>
<th>Q_{n+1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Q_n</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Q_n</td>
</tr>
</tbody>
</table>
T-type flip-flops are used in counters.

**JK Logic table**

<table>
<thead>
<tr>
<th>J</th>
<th>K</th>
<th>$Q_{n+1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$Q_n$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>$\overline{Q_n}$</td>
</tr>
</tbody>
</table>

**T Logic table** (clock falling edge)

<table>
<thead>
<tr>
<th>T</th>
<th>$Q_{n+1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$Q_n$</td>
</tr>
<tr>
<td>1</td>
<td>$\overline{Q_n}$</td>
</tr>
</tbody>
</table>

T-type flip-flops are used in counters.
Counters in Verilog are easy → just use always (synchronous).

→ and a self-referential “add 1” assignment.

```verilog
module counter_v3(input1,output1);  // module for an 8-bit synchronous counter
    input input1;  // 1-bit input
    output reg [7:0] output1;  // 8-bit output register
    always @(posedge input1)  // synchronous loop, clocked on input1 rising edge
        begin
            output1 <= output1 + 1;  // self-referential add+1 assignment.
            // output1 = output1 + 8'b00000001;  could have used this instruction line instead.
        end
endmodule
```
Initializing a register

```verilog
module counter_v3(input1, output1); // module for an 8-bit synchronous counter
  input input1; // 1-bit input
  output reg [7:0] output1; // 8-bit output register

  initial // this block initializes the output register
    begin // to zero.
      output1 = 8'b00000000;
    end

  always @(posedge input1) // synchronous loop, clocked on input1 rising edge
    begin
      output1 <= output1 + 1; // self-referential add+1 assignment.
      // output1 = output1 + 8'b00000001; could have used this instruction line instead.
    end
endmodule
```
Initializing a register

This section initializes the register to zero. ("initial" is only for simulation !)
module counter_v3(input1, output1, output2); // module for an 8-bit synchronous counter
input input1; // 1-bit input
output reg [7:0] output1; // 8-bit output register
output reg [2:0] output2; // 3-bit output register

initial // this block initializes the output registers
begin // to zero.
  output1 = 8'b00000000;
  output2 = 3'b000;
end

always@(posedge input1) // synchronous loop, clocked on input1 rising edge
begin
  output1 <= output1 + 1; // self-referential add+1 assignment.
  // output1 = output1 + 8'b00000001; could have used this instruction line instead.
  if (output1 <= 8'b11100111)
    begin
      output2 = 3'b000; // output2 stays at zero for output1 <= 231.
    end
  else
    begin
      output2 = output2 + 1; // output2 starts counting for output1 > 231.
    end
endmodule
module counter_v3(input1, output1, output2);  // module for an 8-bit synchronous counter
    input input1;  // 1-bit input
    output reg [7:0] output1;  // 8-bit output register
    output reg [2:0] output2;  // 3-bit output register
    reg [1:0] temp;  // “temp” variable 2-bit register.

initial  // this block initializes the output registers
    begin  // to zero.
        output1 = 8’d00000000;
        output2 = 3’d000;
        temp = 2’d00;
    end

always@ (posedge input1)  // synchronous loop, clocked on input1 rising edge
    begin
        output1 <= output1 + 1;  // self-referential add1 assignment.
        temp <= temp + 1;  // temp is used as counter.

        if (output1 <= 8’d11100111)
            begin
                output2 = 3’d000 + temp;  // output2 counts continuously to 2 for output1 <= 231.
            end
        else
            begin
                output2 = output2 + 1;  // output2 starts counting for output1 > 231.
            end
    end
endmodule

Recommendation: check the Technology Map Viewer after compiling.
The “function” command (I)

```verbatim
// Module translates 2-bit inputs to an 8-bit output code

module Input_to_Output_converter(input_register1, input_register2, input_register3, 
                                  output_register1, output_register2, output_register3);

input [1:0] input_register1;
input [1:0] input_register2;
input [1:0] input_register3;
output reg [7:0] output_register1;
output reg [7:0] output_register2;
output reg [7:0] output_register3;

always
  begin
    // 1st output
    output_register1 = output_8bit(input_register1);
    // 2nd output
    output_register2 = output_8bit(input_register2);
    // 3rd output
    output_register3 = output_8bit(input_register3);
  end
```
The “function” command (II)

```verbatim
// This function defines the output codes given a 2-bit input
function [7:0] output_8bit;
    input [1:0] input_number_2bit;
    begin
        output_8bit = 7'b1111111;
        if (input_number_2bit == 4'b00)
            output_8bit = 7'b1111111;
        if (input_number_2bit == 4'b01)
            output_8bit = 7'b1001011;
        if (input_number_2bit == 4'b10)
            output_8bit = 7'b1000000;
        if (input_number_2bit == 4'b11)
            output_8bit = 7'b0000000;
    end
endfunction
endmodule
```