2012 Fall CSE140L

Digital Systems Laboratory

Lecture #4

by

Dr. Choon Kim

CSE Department, UCSD
chk034@cs.ucsd.edu
Important Timing issues

- Tpd
- Slack
- Required Time
- Actual time
LAB#2 Demo & Explanation
Question: What is it?

How to get a clock on DE1 board?
D Flip-Flop

Asynchronous Clear

Inputs               Output

<table>
<thead>
<tr>
<th>CLR</th>
<th>CE</th>
<th>D</th>
<th>CK</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>Q</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(No change)</td>
</tr>
</tbody>
</table>

CLK = 0

CLK = 1

Lecture #4
T Flip-Flop

- **Inputs**
  - CLR
  - CE
  - D
  - C

- **Output**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLR</td>
<td>CE</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- **Asynchronous Clear**
- **Clock Enable**
D-FF Timing

CLK

CLK

D

Q

~Q

D - FF Timing

$t_{setup}$ $t_{hold}$

$t_{cq}$

Lecture #4
Input Timing Constraints

- **Setup time**: $t_{\text{setup}} = \text{time before the clock edge that data must be stable (i.e. not changing)}$
- **Hold time**: $t_{\text{hold}} = \text{time after the clock edge that data must be stable}$
- **Aperture time**: $t_a = \text{time around clock edge that data must be stable} \ (t_a = t_{\text{setup}} + t_{\text{hold}})$
Timing/Frequency of Sequential Circuit

- The **minimum** delay from register R1 through the combinational logic to R2 determines the **maximum** frequency.

\[ T_c \geq t_{pcq} + t_{pd} + t_{setup} \]

**Fmax** = \( \frac{1}{Tc} \)

*E.g. if Tc \( \geq 6.7 \) ns, Fmax = \( \frac{1}{6.7\text{ns}} = 149.25\text{MHz} \)*
Shift Register

Signal A is given as input.

<table>
<thead>
<tr>
<th>Time Steps</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Counter using T-FF (Down counting)

\[ \text{Reset } A(0) = B(0) = C(0) = 0 \]

<table>
<thead>
<tr>
<th>Time</th>
<th>C</th>
<th>B</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Lecture #4
Up Counter

• Homework#8:
  How to make an up counter using TFF?
  000 -> 001 -> 010 -> ...... -> 111 -> 000

  Hint:
  Use ~Q instead of Q to connect to the clk port of next stage TFF
Behavioral modeling

- We have designed a counter using **structural-level modeling** (i.e., schematics of TFF chain, gates, etc).

- Can we design the same counter using **behavioral-level modeling** (i.e., similar to SW language, e.g.,
  
  ```
  counter_out = counter_out + 1;
  ```

**Hint:** Yes, we can do it with a HDL, like Verilog.

In fact, Verilog allows user to design circuit in any level modeling.
Verilog in one slide!
By Choon Kim

module module_name ( port specification... );

local wires, variables declaration;  
task, function declaration;  

continuous assignments ;  // for combinational circuit
procedural blocks;  // for sequential circuit
instantiation of modules ;  // for hierarchical design
instantiation of primitives /UDP;  // for built-in primitives

endmodule
Introduction to Verilog HDL

Home work (Reading materials)

Warning:
Some information contained in the following reading materials (or any material in general) may be incorrect and not working. Only way to verify the correctness of the information is to test it by simulating and synthesizing using our Altera Quartus SW & HW kit.

1. [Quartus II example with Verilog]: Note: You can do this without understanding of Verilog. Follow the instructions described in tut_quartus_intro_verilog_de1. This is the same homework in Lecture#1 except Verilog is used as a design entry instead of Schematic diagram.

2. [Tutorial for beginner]: Follow Verilog short tutorial to make yourself familiar with this new design entry methodology.

3. [Manual]: intro_verilog_manual

4. [Short Reference Card]: Two-page Card Multiple-page Card

5. In addition, there are numerous Verilog information & tutorials available on the web in doc, book, & video format. The Verilog on wiki is an excellent place to learn the background. This Verilog Tutorial is one of tutorial sites. You can use whatever you feel comfortable.