0. Revision

Dr. Paul D. Franzon

Outline
1. Revision points
2. Digital system timing
3. Design Example
Revision Points

I do expect you to be very familiar with at least the following concepts:

- Different combinational logic structures, including gates, adders, multipliexors, coders, decoders, etc.
- Combinational logic optimization
- Flip-flops and latches, and their operation.
- Timing diagrams. How to produce one.
- Finite State Machines – purpose, operation, types, state vector encoding
- Counters – basic operation
- MOST IMPORTANTLY, I expect you to be able to design a logic function to a specification (like Q7 with relative ease)

If any of these topics are NOT familiar to you, I suggest reviewing your undergraduate logic course or logic course text. If that is not available to you, there are many suitable texts in the library. Authors include Katz, Wakerley, Mano, but there are many others.
**General Principles of Digital System Design**

Most Digital Systems are **Synchronous**

- I.e. All signals are derived off a single Master Clock fed to all registers
- In most logic implementation families, events are synchronized using edge-triggered flip-flops
  - e.g. positive-edge triggered D- or Data- flip-flop

![Flip-Flop Diagram]

After a 0->1 transition at clock, Q takes the value of D that was present just before that transition.

- Groups of flip-flops are referred to as registers
Timing Diagram

The core tool for analyzing synchronous digital designs is a timing diagram.

Don’t know (Don’t care) “x”

Glitches at input do not appear at output. F/F only samples ‘D’ at positive clock edge.

After each rising edge of the clock, Q takes on the value that was present at D just before that clock edge.
Multiple Logic Stages

Generate Timing Diagram

Clock

In

Out

© 2013 Dr. Paul D. Franzon, www.ece.ncsu.edu/erl/faculty/paulf.html
Animation

Track the logic...

In[0]  In[1]  In[2]  In[3]
       |       |       |       |
       1      0      1      0

     |       |       |       |
     0      1      0      1

Clock

In

Out
Q. If we don’t know the initial value of “Out” can we answer this question?

No!

Later, we’ll discuss how to use a global reset to initialize registers.
Digital ASIC Design

Snapshot

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**Clock**

**In**

<table>
<thead>
<tr>
<th>A</th>
<th>1</th>
<th>B</th>
<th>5</th>
</tr>
</thead>
</table>

**Out**

<table>
<thead>
<tr>
<th>4</th>
<th>4</th>
</tr>
</thead>
</table>
What is Wrong Here?

Combinational Logic Feedback does not work in synchronous design

- Must Feedback through register

What happens if the logic gets into the state shown?

Logic point * will oscillate 101010...

Given that logic delays vary with temperature and from chip to chip, what value will be valid at D input to Out[2]?

Don’t know!
Digital sub-systems are built as collections of registers and combinational logic.

- Registers store data from the end of one clock period to be available at the start of the next clock period.
- Combinational logic cannot store data. It only operates on it during each clock period.
- Muxes often used to select next values for register storage, i.e. to steer logic.
- `selB` is a control line coming from a controller.
- `zero` is a status line going to the controller.
Given that logic delay varies widely due to temperature and process dependencies, it is very hard to design logic that will have a delay of more than one clock period, but less than two. Your design must presume that all logic paths complete within one clock period.
Sample Design Problem

Accumulator:

- Design an 8-bit adder accumulator with the following properties:
- While ‘accumulate’ is high, adds the input, ‘in1’ to the current accumulated total and add the result to the contents of register with output ‘accum_out’.
  - use absolute (not 2’s complement) numbers
- When ‘clear’ is high (‘accumulate’ will be low) clear the contents of the register with output ‘accum_out’
- The ‘overflow’ flag is high is the adder overflows

Hint:

8-bit adder produces a 9-bit result:
\[ \{\text{carry\_out, sum}\} = A+B; \]
1. Determine and name registers.
2. Determine combinational logic

Sketch Design

```
Clear

accumulate
0

accum_in

in1

+ 

overflow

accum_out
```
Other Suggested Revision Points

1. Basic logic gates
   - And, Or, Nand, Nor, Xor, Xnor, Mux
2. Truth Tables
3. Karnaugh Map Optimization
   - How to produce minimum 2-level logic
3. Finite State Machines
   - Basic Operation
   - Moore and Mealy machines
Major Points

During each clock cycle:

1. On rising edge of clock, register outputs become equal to register inputs.
2. Combinational logic takes as its input, the values present on the register outputs and determines the next set of register inputs before the clock period is over.

This process repeats every clock cycle while the chip is operating, on ALL the logic in parallel.