2. Timing Design in Digital Systems

Dr. Paul D. Franzon

Outline

1. Timing design in Synchronous (clocked) Logic
   - Min/Max timing with flip-flops
   - Latch-based design

3. Timing Issues in CMOS circuits

4. Timing verification Flow

5. Techniques to Improve Performance
Course “Mantras”

- One clock, one edge, Flip-flops only (per module)
- Design BEFORE coding
- Behavior implies function
- Clearly separate control and datapath
Module Objectives:

- Describe how a clock-synchronized design behaves at a clock-cycle to cycle level. Know how to draw a timing diagram for a design.
- Describe how clocks are distributed. Define clock skew and jitter.
- Describe the basic behavior of flip-flops and latches.
- Define setup and hold times.
- Derive the timing equations for flip-flop based designs.
- Describe the basic behavior of latches, including set-up and hold times.
- Derive the timing equations for latch based designs.
- Understand the application of cycle stealing in latches.
- Describe timing closure in the CAD tool flow.
- Understand the relationship between design and clock frequency
Motivation

- Synchronous design is one cornerstone of modern digital design
  - Events are synchronized by a master clock using flip-flops and latches

- The logic design strongly impacts the achievable clock frequency and thus IC performance
  - The synthesis tools help achieve a specific clock target but can not fix a poorly structured design

- The designer needs to understand
  - How the design and the CAD tools achieve a clock frequency that satisfies set-up requirements
  - How the CAD tools try to fix hold requirements
    - Any why this is much harder when latches are used
References

Smith and Franzon,
- Section 11.2 (timing relationships)
- Chapter 13 (CAD flow)

Ciletti,
- Sections 3.1, 3.2.1 (flip-flops, latches)
- Section 11.2.1 (Static timing analysis)
Detailed Outline

1. Timing “mantra”
2. General timing design concepts.
3. Clock level timing and timing diagrams.
4. How clocks are created.
5. Timing for flip-flop based designs.
6. Timing for latch-based designs.
7. Example
8. Delay and CMOS process impact
9. Impact of design on achievable clock period
10. Managing multiple clock domains
Detailed Outline

1. Timing “mantra”
2. General timing design concepts.
3. Clock level timing and timing diagrams.
4. How clocks are created.
5. Timing for flip-flop based designs.
6. Timing for latch-based designs.
7. Example
8. Delay and CMOS process impact
9. Impact of design on achievable clock period
10. Managing multiple clock domains
Mantra #1

One clock, one edge; Flip-flops only

- For your design (at least for each module) use one clock source and only one edge of that clock
- Only use edge-triggered flip-flops

Why?
- Moving data between different clock domains requires careful timing design and synthesis “scripting”

If you need multiple clocks in your design
- Make them related by a powers of 2 if possible
  - E.g 50, 100 and 200 MHz
- Use one clock per module if at all possible
- You might need resynchronizing using flip-flops between clock domains
General Approach to Timing Design

- In general, all signals start and end in registers every clock period
- This is how the clock synchronizes logic events

![Diagram of D-Flip-flops and Combinational Logic]
Critical Path

- Thus, the clock speed is determined by the slowest feasible path between registers in the design
  - *Often referred to as “the critical path”*

Critical path is longer with increased logic depth (# gates in series)
Synchronous Clock Distribution

- The goal of clock tree is for the clock to arrive at every leaf node at the same time:
- Usually designed after synthesis: Matched buffers; matched capacitance loads
- Common design method:
  - “H tree”
  - Clock tree done after place & route
  - Balance RC delays
  - Balance buffer delays
Clock skew and jitter

- Clock skew = systematic clock edge variation between sites
  - Mainly caused by delay variations introduced by Process (manufacturing) variations, Voltage variations and Temperature variations (PVT) between individual buffers
  - Random variation

- Clock jitter = variation in clock edge timing between clock cycles
  - Mainly caused by noise
  - Usually much smaller than clock skew

Equations below lump jitter and skew into a single “skew” number.

Skew + jitter called “Uncertainty” in Synopsys.
Comments on Clock Skew

- ASIC design relies on (semi-) automatic clock tree synthesis
  - Works to guarantee a global skew target
- Custom clock distribution can be used to add the following features to a clock:
  - Smaller skews
  - Local skews < Global skew
  - Multiple non-overlapping clock phases
  - Deliberate non-random skew at flip-flop/latch level
  - Future automatic clock tree synthesis tools might include features like this
Summary

- The critical path is the slowest path in the design and determines the clock period.
- The clock skew between any two flip-flops is designed to be zero but achieves a non-zero value due to random process variations.
- By far the easiest way to integrate clocks into your design is to use only one clock and one edge of that clock within a module.
  - Inter-clock transitions will be discussed later.

After completing the quiz for this sub-module, please proceed to the next sub-module.
Detailed Outline

1. Timing “mantra”
2. General timing design concepts.
3. Clock level timing and timing diagrams.
4. How clocks are created.
5. Timing for flip-flop based designs.
6. Timing for latch-based designs.
7. Example
8. Delay and CMOS process impact
9. Impact of design on achievable clock period
10. Managing multiple clock domains
**Flip-Flop based design**

**Edge triggered D-flip-flop**

Q becomes D after clock edge

**Set-up time:**

Data can not change no later than this point before the clock edge.

**Hold time:**

Data can not change during this time after the clock edge.

\[ t_{\text{clock-Q}} \]

Delay on output (Q) changing from positive clock edge

**In general skew is bounded but you do not know whether it is positive or negative at any particular point**
What can happen with a timing violation?

D changes outside setup and hold $\Rightarrow$ tclock-Q is correct

D changes during setup and hold $\Rightarrow$ tclock-Q longer than specified, or Q does not transition correctly

$\Rightarrow$ Incorrect timing in next logic stage

$\Rightarrow$ Logic failure
What can happen with a timing violation?

D changes during setup and hold

$Ck$ changes in WRONG clock cycle
- Racethrough
Deriving Timing Equations

- Situation:
Preventing Set-Up Violations

Set-up violation:
Logic is too slow for the correct logic value to arrive at the inputs to the register on the right before one set-up time before the clock edge.

Constraint to prevent this:

\[
 t_{\text{clock}} \geq t_{\text{clock}-Q-\text{max}} + t_{\text{logic}-\text{max}} + t_{\text{set-up}} + t_{\text{skew}}
\]

- The amount of time required to turn ‘>‘ into ‘=‘ is referred to as **timing slack**
Preventing hold violations

Hold violations occur when race-through is possible

Constraint to prevent hold violations:

\[ t_{hold} + t_{skew} \leq t_{clock−Q−min} + t_{logic−min} \]

- sometimes have to insert additional logic to prevent hold violations
Preventing hold violations

Hold violations occur when race-through is possible

Constraint to prevent hold violations:

\[ t_{hold} + t_{skew} \leq t_{clock-Q-min} + t_{logic-min} \]

- sometimes have to insert additional logic to prevent hold violations
Summary

- In a D-flip-flop, the input D can not change during the setup and hold around the clock edge
- Setup violations can happen as a result of slow conditions (slow process, high temperature) leading to signals arriving too late in the clock period
- Hold violations can happen as a result of fast conditions (fast process, low temperature) leading to signals arriving too early in the clock period
- Since skew is random the worst case skew condition must be used

Do the internal quiz, then proceed to the next sub-module
Detailed Outline

1. Timing “mantra”
2. General timing design concepts.
3. Clock level timing and timing diagrams.
4. How clocks are created.
5. Timing for flip-flop based designs.
6. Timing for latch-based designs.
7. Example
8. Delay and CMOS process impact
9. Impact of design on achievable clock period
10. Managing multiple clock domains


**Latch Based Design**

**D-latch**

- Q follows D while clock is high ("transparent")
- Value on D when clock goes low is stored on Q

**Set-up and hold times:**

D can not change close to the falling ('latching') clock edge.

\[ t_{clock-Q} \]

Delay from clock going high to Q changing

\[ t_{D-Q} \]

Delay from D changing to Q changing while clock high (assumed = \( t_{ck-Q} \) here)
**Latch Timing Constraints**

Set-up constraints under nominal design same as for flip-flop

\[ t_{\text{clock}} \geq t_{\text{clock-}Q-\text{max}} + t_{\text{log}ic-\text{max}} + t_{\text{set-up}} + t_{\text{skew}} \]

“Transparency” of latch can be used to improve flexibility of timing = “cycle stealing”

**Example:** If critical path is in logic block 1:

- \( t_1 \) longer than for FF – exploit transparent latch

---

©2013 Dr. Paul D. Franzon, www.ece.ncsu.edu/erl/faculty/paulf.html
**Latch timing constraints WITH cycle stealing**

To prevent set-up violations:

\[
t_{clock} + t_{clock-high-min} \geq t_{clock-Q-max} + t_{logic-max} + t_{set-up} + t_{skew}
\]

**Notes:**
- The percentage of time the clock is high is referred to as the *duty-cycle*
- In Synopsys DesignCompiler the default is that cycle stealing is not enabled
**Latches … Cycle Stealing**

- Can use up to a total of $t_{\text{clock\_high}}$ within a pipeline structure, to help in timing closure
  - Example:

```
t1 \hspace{1cm} t2 \hspace{1cm} t3
```

\[ t1, t2 > t_{\text{clock}} \text{ (cycle stealing)} \]

What can $t3$ be?

$T3 = t_{\text{clock}}$
...**Latch timing constraints**

To prevent hold violations:

\[
t_{\text{clock-high-max}} + t_{\text{hold}} + t_{\text{skew}} \leq t_{\text{clock-Q-min}} + t_{\text{logic-min}}
\]

**Note:**

- Hold violations are harder to prevent in latch-based designs
Summary

- Latches are transparent while $t_{ck}$ is high. This leads to
  - Some increased flexibility to reduce $t_{ck}$ if cycle stealing is enabled
  - Greater susceptibility to hold violations

Please do the sub-module quiz and move to the next sub-module
Detailed Outline

1. Timing “mantra”
2. General timing design concepts.
3. Clock level timing and timing diagrams.
4. How clocks are created.
5. Timing for flip-flop based designs.
6. Timing for latch-based designs.
7. Example
8. Delay and CMOS process impact
9. Impact of design on achievable clock period
10. Managing multiple clock domains
Example:

\[ D\text{-flip-flop} \]

\[
\begin{align*}
\text{T}_{\text{clock-Q}} & = 3 : 4 : 5 \\
\text{TNOR} & = 1 : 2 : 3 \\
\text{T}_{\text{su max}} & = 1 \\
\text{Thold max} & = 2 \\
\text{T}_{\text{skew}} & = 1 \text{ ns}
\end{align*}
\]

If this is the critical path, what is the fastest clock frequency? Is there potential for a hold violation?

Setup:

\[
t_{\text{clock}} \geq t_{\text{clock-Q-max}} + t_{\text{logic-max}} + t_{\text{set-up}} + t_{\text{skew}}
\]

\[
= 5 + 6 + 1 + 1 = 13 \text{ ns}
\]

I.e. 76 MHz

Hold:

\[
\begin{align*}
\text{Is } t_{\text{hold}} + t_{\text{skew}} & \leq t_{\text{clock-Q-min}} + t_{\text{logic-min}} \\
2 + 1 & \quad 3 + 1 \\
< \quad & \text{OK. No hold violation}
\end{align*}
\]
Detailed Outline

1. Timing “mantra”
2. General timing design concepts.
3. Clock level timing and timing diagrams.
4. How clocks are created.
5. Timing for flip-flop based designs.
6. Timing for latch-based designs.
7. Example
8. Delay and CMOS process impact
9. Impact of design on achievable clock period
10. Managing multiple clock domains
**CMOS Drive Strength**

Revision: CMOS transistors operating in the linear region:

\[ I_{ds} = \beta((V_{gs} - V_t)V_{ds} - V_{ds}^2) / 2 \]

where \( \beta = (\mu \varepsilon / t_{ox})(W / L) \)

where \( W \) is the transistor width, and \( L \) is the channel length

i.e. To a first approximation,

\[ I_{ds} \approx V_{ds} / R_{on} \]

\[ R_{on} \approx 1/ \beta(V_{GS}-V_T) \]

\[ \tau = R_{on}C_{load} \]

Thus, delay in CMOS circuits depends largely on \( W/L \) of the drive transistor and the capacitance of the load it is driving.

- Different drive cell sizes can be selected by synthesis tool (x1, x2, x3, etc.)
- That capacitance consists of:
  - Input gates of cells being driven, and Capacitance of wiring
Detailed Outline

1. Timing “mantra”
2. General timing design concepts.
3. Clock level timing and timing diagrams.
4. How clocks are created.
5. Timing for flip-flop based designs.
6. Timing for latch-based designs.
7. Example
8. Delay and CMOS process impact
9. Impact of design on achievable clock period
10. Managing multiple clock domains
Estimating and Improving Performance

- With a focus on timing:
- Topics:
  - Metrics: FO-4
  - Pipelining and Parallelism
**Delay Metric**

- **Usual Metric for delay:**
  - Fanout of 4 inverter delay: FO4

  ![Inverter Diagram]

- **Often used to characterize performance of a process**

- **Estimating FO4**
  - Typical $\sim 360 \times L_{\text{gate}}$ (ps)
  - Worst Case $\sim 600 \times L_{\text{gate}}$ (ps)
  - E.g. In a 28 nm process, FO-4 $\leq 16.8$ ps
    (This is actually a large under-estimate)
Timing Driven Design

- Using FO-4 and the following delays in the examples on the next few pages
- Exemplar delays (not meant to be accurate):
  - Inverter = FO-4
  - 1-bit adder = 4 FO4
  - Flip-flop t_cp-Q = 2 FO4
  - 2-input NAND gate = 2 FO4
  - 2-input Multiplexer = 4 FO4
  - Flip-flop t_su / t_h = FO4
  - Clock skew = FO4
Examples of Improving Timing Performance

- Example 1: Benefits of Pipelining and Parallelism
- Example:

If $t_{comparator} = 10$ FO4, what is the clock period? (Use values on previous page)

$$T_{cp} = t_{ck-Q} + t_{logic} + t_{su} + t_{skew}$$
$$= 2 + 10 \times 3 + 1 + 1 = 34 \text{ FO4}$$
Pipelining

- Replace with a pipelined version
- Operation:
  - Adding pipeline stages reduces the clock period
  - We are doing several overlapping computations at the same time
  - Example: finding max of 4 numbers over 4 clock cycles

```
5  5
7  7
9  9
0  0
```

```
5
7
9
0
```

```
5
7
9
0
```

```
5
7
9
0
```

```
5
7
9
0
```

```
5
7
9
0
```
Pipelining

- What would happen if the registers that are just storage were not there?

```
A  A  A
1  1  2
C  C
D  D
```
Pipelining

- Replace with:

\[ T_{cp} = t_{ck-Q} + t_{logic} + t_{su} + t_{skew} \]
\[ = 2 + 10 + 1 + 1 = 14 \text{ FO4} \]

- What is the delay improvement? 2.4x (note: NOT 3x)

- What is the drawback? Increased area AND power
  BUT performance/area improved
Logic Level Parallelism

- Replace with:

  ![Diagram]

  - Clock Period = 24 FO-4
  - No increase in area

To see how to code these three structures in Verilog, please refer to the end of the Verilog1 notes.
**Retiming**

- Impact of critical paths can often be reduced by retiming or rebalancing a design:

- Example:
  - Before:
    
    ![Diagram Before](image1)
    
    $$T_{cp} = 2 + 20 + 5 + 1 + 1 = 29 \text{ FO4}$$

  - After:
    
    ![Diagram After](image2)
    
    $$T_{cp} = 2 + 20 + 1 + 1 = 24 \text{ FO4}$$

- Note: Clock level logic sequence has been changed
Summary

- FO-4 = delay of an inverter with a fanout of 4
  - Characterizes process performance
  - Can be used for delay estimates
- Exploiting micro-parallelism, adding pipeline stages and retiming are all effective techniques for reducing $t_{ck}$

Please proceed to end of sub-module quiz and next sub-module
Detailed Outline

1. Timing “mantra”
2. General timing design concepts.
3. Clock level timing and timing diagrams.
4. How clocks are created.
5. Timing for flip-flop based designs.
6. Timing for latch-based designs.
7. Example
8. Delay and CMOS process impact
9. Impact of design on achievable clock period
10. Timing in CAD Flow
11. Managing multiple clock domains
Timing in CAD Flow

- **During synthesis**
  - Tool calculates path delays under worst-case delay conditions
  - Determines critical path
  - Moves logic to a faster path if setup violation predicted
  - Some tools do a preliminary placement while doing logic synthesis so that wiring delay can be properly estimated
  - Logic is added, if needed to prevent hold violations

- **After synthesis:**
  - Perform Static Timing Analysis
  - Determine no setup violations exist under worst case conditions
  - Determine no hold violations exist under best case conditions

- **After place and route:**
  - Perform Timing Analysis
  - i.e. Run timing verification tools on netlist with actual delays
  - Back-annotate actual delays to netlist from later tools
Delay Estimation Sub-Flow

- Primetime: Gate-level static timing analysis tool
  - Report timing for critical path
- Back-annotate wire parasitics for more accuracy
  - SPEF file from place and route tool
  - i.e. Actual RC delays of wires as they appear in the layout

Start

Load, link, and constrain design in Primetime

Verilog structural description of design (post clock tree synthesis)

Back-annotate wiring parasitics in Primetime

Post-route SPEF file from SoC Encounter

Report timing in Primetime

Finish
Digital ASIC Design

Detailed Outline

1. Timing “mantra”
2. General timing design concepts.
3. Clock level timing and timing diagrams.
4. How clocks are created.
5. Timing for flip-flop based designs.
6. Timing for latch-based designs.
7. Example
8. Delay and CMOS process impact
9. Impact of design on achievable clock period
10. Timing in CAD Flow
11. Managing multiple clock domains
Synchronization across clock domains

- When clocks have are related by powers of 2:

  Logic design must be aware that Q2 only changes every other clock cycle

  Logic design must ensure that Q3 only changes every other clock cycle

Put in different Verilog modules OR using multicycle path commands in synthesis
**Synchronization across clock domains**

- When clocks are NOT related by powers of two – going from slower clock to faster clock
  - Need to build a resynchronizer
    - Concept:
      - Q2 will often have extra delay or take an extra cycle due to metastability
      - With no logic to Q3 and Q4, metastability eventually dissapears
    - Three flip-flops is a very safe design
      - Depending on the flip-flop style, two (or sometimes even one) might suffice
    - Might want a transition detector on Q4 to detect when value changes
Synchronization between clock domains

- Going from a faster clock to a slower clock

![Diagram showing synchronization between clock domains]
Synchronization Across Clock Domains

- A very general method is to build a First In First Out (FIFO) buffer
  
  ![Diagram of FIFO buffer]

- Array of registers organized as a circular queue
- Register “write” points at next location to write to
- Register “read” points at next location to read from
- Flags: \( \text{Empty} = (\text{read} == \text{write}) \) \( \text{Full} = ((\text{write}+1) == \text{read}) \)
- Use gray code for pointer arithmetic
  - Only one bit changes at a time
  - Ensures address calculations are using old or new ones, preventing hazards

Gray code

<table>
<thead>
<tr>
<th>Gray code</th>
<th>0 000</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 001</td>
<td></td>
</tr>
<tr>
<td>2 011</td>
<td></td>
</tr>
</tbody>
</table>
Module Summary

- It is best to use flip-flops, and a single clock per module, and only one clock edge
  - Flip-flops are least susceptible to hold violations
  - One clock, one edge, greatly simplifies synthesis
- Synthesis tries to minimize the critical path to prevent setup violations
  - But the critical path is largely determined by longest logic delay implicit to your design
- Synthesis adds logic to eliminate hold violations
- The most general approach to communicating between clock domains is to use a FIFO with Gray scale counting pointers

Proceed to final (for credit) quiz and homework 2