11. Sequential Elements

- Last module:
  - Adder circuits
  - Simple adders
  - Fast addition

- This module
  - Sequential circuit design
  - Clock skew
Sequencing

- **Combinational logic**
  - output depends on current inputs

- **Sequential logic**
  - output depends on current and previous inputs
  - Requires separating previous, current, future
  - Called *state* or *tokens*
  - Ex: FSM, pipeline
Sequencing Cont.

- If tokens moved through pipeline at constant speed, no sequencing elements would be necessary
- Ex: fiber-optic cable
  - Light pulses (tokens) are sent down cable
  - Next pulse sent before first reaches end of cable
  - No need for hardware to separate pulses
  - But dispersion sets min time between pulses
- This is called wave pipelining in circuits
- In most circuits, dispersion is high
  - Delay fast tokens so they don’t catch slow ones.
Sequencing Overhead

- Use flip-flops to delay fast tokens so they move through exactly one stage each cycle
- Inevitably adds some delay to the slow tokens
- Makes circuit slower than just the logic delay
  - Called sequencing overhead
- Some people call this clocking overhead
  - But it applies to asynchronous circuits too
  - Inevitable side effect of maintaining sequence
Sequencing Elements

• **Latch**: Level sensitive
  – a.k.a. transparent latch, D latch

• **Flip-flop**: edge triggered
  – A.k.a. master-slave flip-flop, D flip-flop, D register

• **Timing Diagrams**
  – Transparent
  – Opaque
  – Edge-trigger

![Timing Diagrams](image-url)
Sequencing Elements

- **Latch**: Level sensitive
  - a.k.a. transparent latch, D latch

- **Flip-flop**: edge triggered
  - A.k.a. master-slave flip-flop, D flip-flop, D register

- **Timing Diagrams**
  - Transparent
  - Opaque
  - Edge-triggered

![Timing Diagrams](image-url)
Latch Design

• Pass Transistor Latch
  • Pros
    + Tiny
    + Low clock load
  • Cons
    – $V_t$ drop
    – nonrestoring
    – backdriving
    – output noise sensitivity
    – dynamic
    – diffusion input

Used in 1970s
Latch Design

- Transmission gate
  - No $V_t$ drop
  - Requires inverted clock

- Inverting buffer
  - Restoring
  - No backdriving
  - Fixes either
    - Output noise sensitivity
    - Or diffusion input
  - Inverted output
Latch Design

• Tristate feedback
  + Static
    – Backdriving risk
• Static latches are now essential

• Buffered input
  + Fixes diffusion input
  + Noninverting
Latch Design

- Buffered output
  + No backdriving

- Widely used in standard cells
  + Very robust (most important)
  - Rather large
  - Rather slow (1.5 – 2 FO4 delays)
  - High clock loading
Latch Design

- Datapath latch
  + Smaller, faster
  - unbuffered input
Flip-Flop Design

- Flip-flop is built as pair of back-to-back latches
Enable

- Enable: ignore clock when en = 0
  - Mux: increase latch D-Q delay
  - Clock Gating: increase en setup time, skew

![Symbol](image1)

- ![Multiplexer Design](image2)
- ![Clock Gating Design](image3)
Reset

- Force output low when reset asserted
- Synchronous vs. asynchronous

![Reset Diagram]

Synchronous Reset

Asynchronous Reset

Flop

Latch

Symbol

D Q

Reset

D Q

Reset

Reset

Reset

Symbol

Flop

Latch

D Q

Reset

reset

reset

reset

reset

reset

reset

reset
Set / Reset

- Set forces output high when enabled

- Flip-flop with asynchronous set and reset
Sequencing Methods

- Flip-flops
- 2-Phase Latches
- Pulsed Latches
## Timing Diagrams

### Contamination and Propagation Delays

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_{pd}$</td>
<td>Logic Prop. Delay</td>
</tr>
<tr>
<td>$t_{cd}$</td>
<td>Logic Cont. Delay</td>
</tr>
<tr>
<td>$t_{pcq}$</td>
<td>Latch/Flop Clk-Q Prop Delay</td>
</tr>
<tr>
<td>$t_{ccq}$</td>
<td>Latch/Flop Clk-Q Cont. Delay</td>
</tr>
<tr>
<td>$t_{pdq}$</td>
<td>Latch D-Q Prop Delay</td>
</tr>
<tr>
<td>$t_{cdq}$</td>
<td>Latch D-Q Cont. Delay</td>
</tr>
<tr>
<td>$t_{setup}$</td>
<td>Latch/Flop Setup Time</td>
</tr>
<tr>
<td>$t_{hold}$</td>
<td>Latch/Flop Hold Time</td>
</tr>
</tbody>
</table>

![Timing Diagrams](image)
Master-Slave Flip-Flop

Illustration of delays

\[ t_{\text{setup}} = \]
\[ t_{\text{pcq}} = \]
\[ t_{\text{hold}} = \]

© Digital Integrated Circuits 2nd

Karu Sankaralingam
Max-Delay: Flip-Flops

\[ t_{pd} \leq T_c - (t_{setup} + t_{pcq}) \]

sequencing overhead
Max Delay: 2-Phase Latches

\[ t_{pd} = t_{pd1} + t_{pd2} \leq T_c - (2t_{pdq}) \]

sequencing overhead

\[ T_c = t_{pd1} + t_{pd2} \cdot T_c - (2t_{pdq}) \]
Max Delay: Pulsed Latches

\[ t_{pd} \leq T_c - \max(t_{pdq}, t_{pcq} + t_{setup} - t_{pw}) \]

(a) \( t_{pw} > t_{setup} \)

(b) \( t_{pw} < t_{setup} \)

sequencing overhead
Min-Delay: Flip-Flops

$t_{cd} \geq \text{Min-Delay}$
Min-Delay: 2-Phase Latches

\[ t_{cd1}, t_{cd2} \geq - - - \]

Hold time reduced by nonoverlap

Paradox: hold applies twice each cycle, vs. only once for flops.

But a flop is made of two latches!
Min-Delay: Pulsed Latches

\[ t_{cd} \geq \ldots - \ldots + \ldots \]

Hold time increased by pulse width
Time Borrowing

• In a flop-based system:
  – Data launches on one rising edge
  – Must setup before next rising edge
  – If it arrives late, system fails
  – If it arrives early, time is wasted
  – Flops have hard edges

• In a latch-based system
  – Data can pass through latch while transparent
  – Long cycle of logic can borrow time into next
  – As long as each loop completes in one cycle
Time Borrowing Example

(a) Latch → Combinational Logic → Latch  
Borrowing time across half-cycle boundary

(b) Latch → Combinational Logic → Latch  
Borrowing time across pipeline stage boundary

Loops may borrow time internally but must complete within the cycle
How Much Borrowing?

2-Phase Latches

\[ t_{\text{borrow}} \leq \frac{T}{2} - t_{\text{nonoverlap}} + t_{\text{setup}} \]

Pulsed Latches

\[ t_{\text{borrow}} \leq \frac{T}{2} - t_{\text{nonoverlap}} \]
Clock Skew

• We have assumed zero clock skew
• Clocks really have uncertainty in arrival time
  – Decreases maximum propagation delay
  – Increases minimum contamination delay
  – Decreases time borrowing
Skew: Flip-Flops

\[ t_{pd} \leq T_c - (t_{pcq} + t_{	ext{setup}} + t_{	ext{skew}}) \]

sequencing overhead

\[ t_{cd} \geq t_{	ext{hold}} - t_{	ext{ccq}} + t_{	ext{skew}} \]
Skew: Latches

2-Phase Latches

\[ t_{pd} \leq T_c - (2t_{pdq}) \]

sequencing overhead

\[ t_{cd1}, t_{cd2} \geq t_{hold} - t_{ccq} - t_{nonoverlap} + t_{skew} \]

Pulsed Latches

\[ t_{pd} \leq T_c - \max(t_{pdq}, t_{pcq} + t_{setup} - t_{pw} + t_{skew}) \]

sequencing overhead

\[ t_{cd} \geq t_{hold} + t_{pw} - t_{ccq} + t_{skew} \]

\[ t_{borrow} \leq t_{pw} - (t_{setup} + t_{skew}) \]
Two-Phase Clocking

- If setup times are violated, reduce clock speed
- If hold times are violated, chip fails at any speed
- In this class, working chips are most important
  - No tools to analyze clock skew
- An easy way to guarantee hold times is to use 2-phase latches with big nonoverlap times
- Call these clocks $\phi_1$, $\phi_2$ (ph1, ph2)
Safe Flip-Flop

• In class, use flip-flop with nonoverlapping clocks
  – Very slow – nonoverlap adds to setup time
  – But no hold times

• In industry, use a better timing analyzer
  – Add buffers to slow signals if hold time is at risk
Summary

• Flip-Flops:
  – Very easy to use, supported by all tools

• 2-Phase Transparent Latches:
  – Lots of skew tolerance and time borrowing

• Pulsed Latches:
  – Fast, some skew tol. & borrow, hold time risk

<table>
<thead>
<tr>
<th></th>
<th>Sequencing overhead ((T_c - t_{pd}))</th>
<th>Minimum logic delay (t_{cd})</th>
<th>Time borrowing (t_{borrow})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flip-Flops</td>
<td>(t_{pcq} + t_{setup} + t_{skew})</td>
<td>(t_{hold} - t_{eqc} + t_{skew})</td>
<td>0</td>
</tr>
<tr>
<td>Two-Phase Transparent</td>
<td>(2t_{pdq})</td>
<td>(t_{hold} - t_{eqc} - t_{nonoverlap} + t_{skew}) in each half-cycle</td>
<td>(\frac{T_c}{2}) - (t_{setup} + t_{nonoverlap} + t_{skew})</td>
</tr>
<tr>
<td>Pulsed Latches</td>
<td>(\max(t_{pdq}, t_{pcq} + t_{setup} - t_{pw} + t_{skew}))</td>
<td>(t_{hold} - t_{eqc} + t_{pw} + t_{skew})</td>
<td>(t_{pw} - (t_{setup} + t_{skew}))</td>
</tr>
</tbody>
</table>
Floorplan

• How do you estimate block areas?
  – Begin with block diagram
  – Each block has
    • Inputs
    • Outputs
    • Function (draw schematic)
    • Type: array, datapath, random logic

• Estimation depends on type of logic
MIPS Floorplan

10 I/O pads

5000 λ

3500 λ

2700 λ

10 I/O pads

10 I/O pads

10 I/O pads

10 I/O pads

1690 λ

1500 λ x 400 λ

(0.6 Mλ²)

control

200 λ x 100 λ

(20 kλ²)

alucontrol

2700 λ x 1050 λ

(2.8 Mλ²)

datapath

2700 λ x 100 λ

(2.8 Mλ²)

bitslice 2700 λ x 100 λ

wiring channel: 30 tracks = 240 λ.
Area Estimation

• Arrays:
  – Layout basic cell
  – Calculate core area from # of cells
  – Allow area for decoders, column circuitry

• Datapaths
  – Sketch slice plan
  – Count area of cells from cell library
  – Ensure wiring is possible

• Random logic
  – Compare complexity do a design you have done
MIPS Slice Plan
Typical Layout Densities

- Typical numbers of high-quality layout
- Derate by 2 for class projects to allow routing and some sloppy layout.
- Allocate space for big wiring channels

<table>
<thead>
<tr>
<th>Element</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random logic (2 metal layers)</td>
<td>1000-1500 $\lambda^2$ / transistor</td>
</tr>
<tr>
<td>Datapath</td>
<td>250 – 750 $\lambda^2$ / transistor</td>
</tr>
<tr>
<td></td>
<td>Or 6 WL + 360 $\lambda^2$ / transistor</td>
</tr>
<tr>
<td>SRAM</td>
<td>1000 $\lambda^2$ / bit</td>
</tr>
<tr>
<td>DRAM</td>
<td>100 $\lambda^2$ / bit</td>
</tr>
<tr>
<td>ROM</td>
<td>100 $\lambda^2$ / bit</td>
</tr>
</tbody>
</table>