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.
Sequential Logic

Inputs — Combinational Logic — Outputs

Current State — State Registers — Next State

clock

Timing Metrics

In — D — Q — Out

clock

t_{su} t_{hold} t_{b-q}

data stable

In

Out

output stable

output stable
System Timing Constraints

\[ t_{cdreg} + t_{cdlogic} \geq t_{hold} \]
\[ T \geq t_{c-q} + t_{logic} + t_{su} \]

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
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

Latch Design

- **Pass Transistor Latch**
- **Pros**
  - +
  - +
- **Cons**
  - _
  - _
  - _
  - _
  - _
  - _
Latch Design

- Pass Transistor Latch

Pros
- Tiny
- Low clock load

Cons
- $V_t$ drop
- nonrestoring
- backdriving
- output noise sensitivity
- dynamic
- diffusion input

Used in 1970’s

Latch Design

- Transmission gate

+ $\phi$

- $D \quad Q$

$\phi$
Latch Design

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

Latch Design

- Inverting buffer
  - Fixes either
    - 
    - 

11/1/2006  VLSI Design I;  A. Milenkovic
Latch Design

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

Latch Design

- Tristate feedback
  -
    -
Latch Design

- Tristate feedback
  + Static
    - Backdriving risk

- Static latches are now essential
Latch Design

- Buffered input
  - Fixes diffusion input
  - Noninverting

Latch Design

- Buffered output
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
  +
  -
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

### Reset

- Force output low when reset asserted
- Synchronous vs. asynchronous
**Set / Reset**

- Set forces output high when enabled
- Flip-flop with asynchronous set and reset

```plaintext
<table>
<thead>
<tr>
<th>D</th>
<th>set</th>
<th>reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>φ</td>
<td>φ</td>
<td>φ</td>
</tr>
<tr>
<td></td>
<td></td>
<td>set</td>
</tr>
<tr>
<td></td>
<td></td>
<td>φ</td>
</tr>
<tr>
<td></td>
<td></td>
<td>φ</td>
</tr>
</tbody>
</table>
```

**Sequencing Methods**

- Flip-flops
- 2-Phase Latches
- Pulsed Latches

```plaintext
<table>
<thead>
<tr>
<th>Flip Flops</th>
<th>2-Phase Transparent Latches</th>
<th>Pulsed Latches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tc</td>
<td>Tc/2</td>
<td></td>
</tr>
<tr>
<td>tnonoverlap</td>
<td>tnonoverlap</td>
<td></td>
</tr>
<tr>
<td>tnonoverlap</td>
<td>tnonoverlap</td>
<td></td>
</tr>
<tr>
<td>tpw</td>
<td>tpw</td>
<td></td>
</tr>
</tbody>
</table>
```

11/1/2006 VLSI Design I; A. Milenkovic
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_{poq}</td>
<td>Latch/Flop Clk-Q Prop Delay</td>
</tr>
<tr>
<td>t_{ccoq}</td>
<td>Latch/Flop Clk-Q Cont. Delay</td>
</tr>
<tr>
<td>t_{pq}</td>
<td>Latch D-Q Prop Delay</td>
</tr>
<tr>
<td>t_{cqc}</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>

Max-Delay: Flip-Flops

\[ t_{pd} \leq T_c - \left( \frac{\text{sequencing overhead}}{2} \right) \]
Max-Delay: Flip-Flops

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

\( t_{pd} \) is the propagation delay, \( T_c \) is the cycle time, \( t_{setup} \) is the setup time, and \( t_{pcq} \) is the propagation delay of the combinational logic. The sequencing overhead includes the propagation delay of the combinational logic.

Max Delay: 2-Phase Latches

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

\( t_{pd} \) is the propagation delay, \( T_c \) is the cycle time, \( t_{setup} \) is the setup time, and \( t_{pcq} \) is the propagation delay of the combinational logic. The sequencing overhead includes the propagation delay of the combinational logic.
Max Delay: 2-Phase Latches

\[ t_{pd} = t_{pd1} + t_{pd2} \leq T_c - \left( \frac{2t_{pd0}}{\text{sequencing overhead}} \right) \]

Max Delay: Pulsed Latches

\[ t_{pd} \leq T_c - \max \left( \frac{t_{pd0}}{\text{sequencing overhead}} \right) \]
Max Delay: Pulsed Latches

\[ t_{pd} \leq T_c - \max\left( t_{pdq} + t_{pdc} + t_{setup} - t_{pde} \right) \]

Combinational Logic

Min-Delay: Flip-Flops

\[ t_{cf} \geq \]
**Min-Delay: Flip-Flops**

\[ t_{cd} \geq t_{\text{hold}} - t_{\text{req}} \]

**Min-Delay: 2-Phase Latches**

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

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: 2-Phase Latches

\[ t_{cd1} + t_{cd2} \geq t_{hold} - t_{cq} - t_{nonoverlap} \]

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

\[ t_{cd} \geq t_{nonoverlap} \]

Hold time increased by pulse width
Min-Delay: Pulsed Latches

\[ t_{req} \geq t_{\text{hold}} - t_{\text{req}} + t_{pu} \]

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 → Combinational Logic
   Borrowing time across half-cycle boundary

(b) Latch → Combinational Logic → Latch → Combinational Logic
   Borrowing time across pipeline stage boundary
   Loops may borrow time internally but must complete within the cycle

How Much Borrowing?

2-Phase Latches
\[ t_{\text{borrow}} \leq \frac{T_c}{2} - \left( t_{\text{setup}} + t_{\text{nonoverlap}} \right) \]

Pulsed Latches
\[ t_{\text{borrow}} \leq t_{\text{pu}} - t_{\text{setup}} \]
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

\[
T_{pd} \leq T_{c} - (t_{pog} + t_{slap} + t_{skew}) \quad \text{sequencing overhead}
\]

\[
T_{cd} \geq t_{hold} - t_{ceq} + t_{skew}
\]
Skew: Latches

2-Phase Latches

\[ t_{pd} \leq T_c - \left( \frac{2t_{pd}}{2} \right) \]

sequencing overhead

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

\[ t_{tressus} \leq \frac{T_c}{2} - \left( t_{setup} + t_{nonoverlap} + t_{skew} \right) \]

Pulsed Latches

\[ t_{pd} \leq T_c - \max \left( t_{pd1}, t_{pd2} + t_{setup} - t_{pre} + t_{skew} \right) \]

sequencing overhead

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

\[ t_{tressus} \leq t_{pre} - \left( t_{setup} + t_{skew} \right) \]

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

\[
\begin{align*}
D & \quad \phi_1 \quad X \quad \phi_2 \quad Q \\
\phi_1 & \quad \phi_2 \quad \phi_1 \quad \phi_2
\end{align*}
\]

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</th>
<th>Minimum logic delay</th>
<th>Time borrowing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flip-Flops</td>
<td>( t_{pl} + t_{compl} + t_{sho} )</td>
<td>( t_{hold} - t_{pl} + t_{sho} )</td>
<td>0</td>
</tr>
<tr>
<td>Two-Phase Transparent Latches</td>
<td>( 2t_{pl} )</td>
<td>( t_{hold} - t_{pl} - t_{compl} + t_{sho} ) in each half-cycle</td>
<td>( \frac{T}{2} ) ( (t_{compl} + t_{compl}) + t_{sho} )</td>
</tr>
<tr>
<td>Pulsed Latches</td>
<td>max( (t_{pl} + t_{compl} ) ( t_{pl} + t_{sho} )</td>
<td>( t_{hold} - t_{pl} + t_{sho} )</td>
<td>( t_{pl} - (t_{compl} + t_{sho}) )</td>
</tr>
</tbody>
</table>