Review: The Regenerative Property

If the gain in the transient region is larger than 1, only A and B are stable operation points. C is a metastable operation point.

Bistable Circuits

• The cross-coupling of two inverters results in a bistable circuit (a circuit with two stable states).
• Have to be able to change the stored value by making A (or B) temporarily unstable by increasing the loop gain to a value larger than 1
  – done by applying a trigger pulse at \( V_{i1} \) or \( V_{i2} \)
  – the width of the trigger pulse need be only a little larger than the total propagation delay around the loop circuit (twice the delay of an inverter)
• Two approaches used
  – cutting the feedback loop (mux based latch)
  – overpowering the feedback loop (as used in SRAMs)

Review: SR Latch

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Q</th>
<th>( Q' )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Q</td>
<td>( Q' ) memory</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0 set</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1 reset</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0 disallowed</td>
</tr>
</tbody>
</table>

Review: Clocked D Latch

MUX Based Latches

• Change the stored value by cutting the feedback loop
  - Negative Latch
    \( Q = \text{clk} \& Q \) transparent when the clock is low
  - Positive Latch
    \( Q = \text{clk} \& Q \) transparent when the clock is high
**TG MUX Based Latch Implementation**

- Input sampled (transparent mode)
- Feedback (hold mode)

**PT MUX Based Latch Implementation**

- Reduced clock load, but threshold drop at output of pass transistors so reduced noise margins and performance

**Latch Race Problem**

- Two-sided clock constraint
  \[ T \geq t_{c-q} + t_{\text{logic}} + t_{\text{su}} \]
  \[ T_{\text{high}} < t_{c-q} + t_{\text{logic}} \]

- Which value of B is stored?

**Master Slave Based ET Flipflop**

- Master transparent, slave hold
- Master hold, slave transparent

**MS ET Implementation**

- Master
- Slave
**MS ET Timing Properties**

- Assume propagation delays are $t_{pd_{inv}}$ and $t_{pd_{tx}}$, that the contamination delay is 0, and that the inverter delay to derive $\overline{clk}$ is 0
- Set-up time - time before rising edge of clk that $D$ must be valid
- Propagation delay - time for $Q_M$ to reach $Q$
- Hold time - time $D$ must be stable after rising edge of clk

**MS ET Timing Properties**

- Assume propagation delays are $t_{pd_{inv}}$ and $t_{pd_{tx}}$, that the contamination delay is 0, and that the inverter delay to derive $\overline{clk}$ is 0
- Set-up time - time before rising edge of clk that $D$ must be valid
- Propagation delay - time for $Q_M$ to reach $Q$
- Hold time - time $D$ must be stable after rising edge of clk

- $t_{setup} = 0.21$ ns works correctly
- $t_{setup} = 0.20$ ns fails

**Propagation Delay Simulation**

$\tau_{qr_{LH}} = 160$ psec

$\tau_{qr_{HL}} = 180$ psec

**Reduced Load MS ET FF**

- Clock load per register is important since it directly impacts the power dissipation of the clock network.
- Can reduce the clock load (at the cost of robustness) by making the circuit ratioed
  - to switch the state of the master, $T_1$ must be sized to overpower $I_2$
  - to avoid reverse conduction, $I_4$ must be weaker than $I_1$
Non-Ideal Clocks

Ideal clocks
clk
!clk

Non-ideal clocks
clk
!clk
clock skew
1-1 overlap
0-0 overlap

Example of Clock Skew Problems

Race condition – direct path from D to Q during the short time when both clk and !clk are high (1-1 overlap)
Undefined state – both B and D are driving A when clk and !clk are both high
Dynamic storage – when clk and !clk are both low (0-0 overlap)

Pseudostatic Two-Phase ET FF

 clk1
 clk2
 master transparent
 slave hold
 dynamic storage
 clk1
 clk2
 N_non_overlap
 master hold
 slave transparent

 Two Phase Clock Generator

 clk
 clk1
 clk2
 A
 B

 Power PC Flipflop

 clk
 l!clk
 1 D
 0 1
 Q 0

 Power PC Flipflop

 clk
 l!clk
 1 D
 0 → 1
 Q 0 → 1
 master transparent
 slave hold
 master hold
 slave transparent
Ratioed CMOS Clocked SR Latch

Sizing Issues

so $W/L_{Sand1} > 3$

$W/L_{Sand4} = 1.5\mu m/0.25\mu m$

$W/L_{Sand3} = 0.5\mu m/0.25\mu m$

Transient Response

6 Transistor CMOS SR Latch

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

Finite State Machine

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 minimum 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

- **Combinational Logic**
- **Inputs**
- **Outputs**
- **Current State**
- **Next State**
- **Clock**

Timing Metrics

- **Clock**
- **In**
- **Out**
- **Data Stable**
- **Output Stable**

System Timing Constraints

- **Inputs**
- **Combinational Logic**
- **Outputs**
- **Current State**
- **Next State**
- **Clock (clock period)**
- \( t_{\text{cold}} + t_{\text{logic}} \geq t_{\text{hold}} \)
- \( T \geq t_{\text{q}} + t_{\text{logic}} + t_{\text{u}} \)

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**
    + Tiny
    + Low clock load
  - **Cons**
    - $V_t$ drop
    - Nonrestoring
    - Backdriving
    - Output noise sensitivity
    - Dynamic
    - Diffusion input
  - Used in 1970’s

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

- **Inverting buffer**
  +
  + Fixes either
  -
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
  +
  +
  +

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

**Sequencing Methods**
- Flip-flops
- 2-Phase Latches
- Pulsed Latches

**Timing Diagrams**

**Contamination and Propagation Delays**
- \( t_{pd} \): Logic Prop. Delay
- \( t_{pd} \): Logic Cont. Delay
- \( t_{pd} \): Latch/Flop Clk-Q Prop Delay
- \( t_{pd} \): Latch/Flop Clk-Q Cont. Delay
- \( t_{pd} \): Latch D-Q Prop Delay
- \( t_{pd} \): Latch D-Q Cont. Delay
- \( t_{setup} \): Latch/Flop Setup Time
- \( t_{hold} \): Latch/Flop Hold Time

**Max-Delay: Flip-Flops**
- \( t_{pd} \leq t_{c_d} \) (sequencing overhead)
- \( t_{pd} \leq t_{c_d} + t_{p_d} \) (combining overhead)

**Max Delay: 2-Phase Latches**
- \( t_{pd} = t_{c_d} + t_{p_d} + t_{c_d} \) (combining overhead)
Max Delay: 2-Phase Latches

\[ t_{\text{delay}} \leq t_0 + t_1 + t_2 + t_{\text{setup}} \]

\[ t_{\text{setup}} = t_{\text{c}} + t_{\text{pdq1}} + t_{\text{pdq2}} \]

Max Delay: Pulsed Latches

\[ t_{\text{delay}} \leq t_0 + \max \left\{ t_{\text{pd}}, t_{\text{pdc}}, t_{\text{pcq}} \right\} \]

Min-Delay: Flip-Flops

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

Min-Delay: 2-Phase Latches

\[ t_{\text{cd}} \geq t_{\text{thold}} - t_{\text{thold}} \]

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

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

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

\[ t_{cd} \geq \]

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

Loops may borrow time internally but must complete within the cycle

How Much Borrowing?

2-Phase Latches

\[ t_{borrow} \leq \frac{t_b}{2} \left( t_{setup} + t_{nonoverlap} \right) \]

Pulsed Latches

\[ t_{borrow} \leq t_{p} - t_{setup} \]
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

\[ t_{skew} \leq T_c - (t_{ccq} + t_{pdq} + t_{pcq}) \]

\[ t_{skew} \geq t_{tdpq} - t_{tpdq} + t_{tdpq} \]

\[ t_{setup} \leq t_{tdpq} + t_{tccq} + t_{tsetup} \]

\[ t_{hold} \geq t_{tdpq} + t_{tccq} + t_{thold} \]

Skew: Latches

2-Phase Latches

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

\[ t_{tsetup} \geq t_{tdpq} - t_{tpdq} + t_{tdpq} \]

\[ t_{tsetup} \leq \left( t_{ccq} + t_{pdq} + t_{pcq} \right) \]

Pulsed Latches

\[ t_{setup} \leq T_c - \max \left( \left( \frac{t_{skew}}{2} \right), \left( t_{ccq} + t_{pdq} + t_{pcq} \right) \right) \]

\[ t_{hold} \geq t_{tdpq} - t_{tpdq} + t_{tdpq} \]

\[ t_{tsetup} \leq \left( t_{ccq} + t_{pdq} + t_{pcq} \right) \]

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

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>Latch Type</th>
<th>Propagation</th>
<th>Hold Time</th>
<th>Setup Time</th>
<th>Time Borrowing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flip-Flop</td>
<td>( t_{ccq} + t_{pdq} + t_{pcq} )</td>
<td>0</td>
<td>( t_{setup} )</td>
<td></td>
</tr>
<tr>
<td>2-Phase Transparent Latch</td>
<td>( t_{ccq} + t_{pdq} + t_{pcq} )</td>
<td>( t_{thold} )</td>
<td>( t_{tdpq} )</td>
<td>( t_{tpdq} )</td>
</tr>
<tr>
<td>Pulsed Latch</td>
<td>( t_{ccq} + t_{pdq} + t_{pcq} )</td>
<td>( t_{thold} )</td>
<td>( t_{tdpq} )</td>
<td>( t_{tpdq} )</td>
</tr>
</tbody>
</table>