Review: Pass Transistor Circuits

- Use pass transistors like switches to do logic
- Inputs drive diffusion terminals as well as gates

- CMOS + Transmission Gates:
  - 2-input multiplexer
  - Gates should be restoring
TG Multiplexer

\[ F = \neg (I_{in1} \cdot S + I_{in2} \cdot \overline{S}) \]

Transmission Gate XOR

\[ A \oplus B \]
Transmission Gate XOR

A \oplus B = A \cdot \overline{B} \oplus \overline{A} \cdot B

off

A \cdot \overline{B}

weak 0 if \overline{A}

\overline{A} \cdot B

weak 1 if A

an inverter

TG Full Adder

C_{in}

B

A

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B

\overline{A} \cdot \overline{B}

\overline{A} \cdot B
### Differential TG Logic (DPL)

**AND/NAND**

\[ F = A \oplus B \]

**XOR/XNOR**

\[ F = \overline{A} \oplus \overline{B} \]

### CPL

- Complementary **P**ass-transistor **L**ogic
  - Dual-rail form of pass transistor logic
  - Avoids need for ratioed feedback
  - Optional cross-coupling for rail-to-rail swing
### CPL Properties

- **Differential** so complementary data inputs and outputs are always available (so don’t need extra inverters)
- Still static, since the output defining nodes are always tied to $V_{DD}$ or GND through a low resistance path
- Design is **modular**; all gates use the same topology, only the inputs are permuted.
- Simple XOR makes it attractive for structures like **adders**
- Fast (assuming number of transistors in series is small)
- Additional routing overhead for complementary signals
- Still have static power dissipation problems
Sequential Circuits

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

Timing Metrics
System Timing Constraints

Combinational Logic

Outputs

Current State

Next State

State Registers

Clock

T (clock period)

\[ t_{\text{cdreg}} + t_{\text{cdlogic}} \geq t_{\text{hold}} \]

\[ T \geq t_{\text{c-q}} + t_{\text{plogic}} + t_{\text{su}} \]

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
  +
  -
Latch Design

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

Latch Design

- Inverting buffer
  +
  +
  + Fixes either
    -
    -
Latch Design

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

- Tristate feedback
  -
Latch Design

- Tristate feedback
  + Static
    - Backdriving risk

- Static latches are now essential

Latch Design

- Buffered input
  +
    +
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

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_{pq} )</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**

\[
\text{Max-Delay} = T_e - \left( \frac{t_{pd} + \text{sequencing overhead}}{T_e} \right)
\]
Max-Delay: Flip-Flops

\[ t_{pd} \leq T_e - \left( t_{\text{setup}} + t_{\text{pq}} \right) \]

Max Delay: 2-Phase Latches

\[ t_{\text{pd}} = t_{\text{pd1}} + t_{\text{pd2}} \leq T_e - \left( t_{\text{setup}} + t_{\text{pq}} \right) \]
Max Delay: 2-Phase Latches

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

Max Delay: Pulsed Latches

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

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

Max Delay: Pulsed Latches

Min-Delay: Flip-Flops

\[ t_{of} \geq \]

Min-Delay: Flip-Flops
Min-Delay: Flip-Flops

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

Min-Delay: 2-Phase Latches

\[ t_{\text{setup1}}, t_{\text{setup2}} \geq \]

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

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

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

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

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

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

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

Loops may borrow time internally but must complete within the cycle.

(b) Latch → Combinational Logic → Latch

**How Much Borrowing?**

2-Phase Latches

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

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

\[
\begin{align*}
I_{pd} & \leq T_c - \left( I_{pq} + I_{atap} + I_{skew} \right) \\
I_{cd} & \geq I_{hold} - I_{pq} + I_{skew}
\end{align*}
\]
Skew: Latches

2-Phase Latches
\[ t_{pd} \leq \frac{T_c}{2} - \left( \frac{2r_{pd}}{c_{pd}} \right) \]
sequencing overhead
\[ t_{cd1}\cdot t_{cd2} \geq t_{hold} - t_{ccq} - t_{nonoverlap} + t_{skew} \]
\[ t_{hmax} \leq \frac{T_c}{2} - \left( t_{setup} + t_{nonoverlap} + t_{skew} \right) \]

Pulsed Latches
\[ t_{pd} \leq \frac{T_c}{2} - \max \left( t_{pd1}\cdot t_{pd2} + t_{setup} - t_{pre} + t_{skew} \right) \]
sequencing overhead
\[ t_{cd} \geq t_{hold} + t_{pre} - t_{ccq} + t_{skew} \]
\[ t_{hmax} \leq t_{pre} - \left( t_{setup} + t_{skew} \right) \]

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

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

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 ( t_{\text{seq}} )</th>
<th>Minimum logic delay ( t_{\text{ld}} )</th>
<th>Time borrowing ( t_{\text{borrow}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flip-Flops</td>
<td>( f_{\text{seq}} + t_{\text{comp}} + t_{\text{delay}} )</td>
<td>( t_{\text{hold}} - t_{\text{setup}} - t_{\text{delay}} )</td>
<td>0</td>
</tr>
<tr>
<td>Two-Phase Transparent Latches</td>
<td>( 2t_{\text{seq}} )</td>
<td>( t_{\text{hold}} - t_{\text{setup}} - t_{\text{delay}} ) in each half-cycle</td>
<td>( t_{\text{seq}} - (t_{\text{comp}} + t_{\text{setup}} + t_{\text{delay}}) )</td>
</tr>
<tr>
<td>Pulsed Latches</td>
<td>( \max(f_{\text{seq}} + f_{\text{comp}} - f_{\text{setup}} + t_{\text{delay}}) )</td>
<td>( t_{\text{hold}} - f_{\text{setup}} + t_{\text{delay}} )</td>
<td>( f_{\text{seq}} - (t_{\text{comp}} + t_{\text{delay}}) )</td>
</tr>
</tbody>
</table>