Review: Pseudo-nMOS

- In the old days, nMOS processes had no pMOS
  - Instead, use pull-up transistor that is always ON
- In CMOS, use a pMOS that is always ON
  - Ratio issue
  - Make pMOS about ¼ effective strength of pulldown network

Review: Pseudo-nMOS Gates

- Design for unit current on output to compare with unit inverter.
- pMOS fights nMOS

Review: Pseudo-nMOS Power

- Pseudo-nMOS draws power whenever \( Y = 0 \)
  - Called static power \( P = I V_{DD} \)
  - A few mA / gate * 1M gates would be a problem
  - This is why nMOS went extinct!
- Use pseudo-nMOS sparingly for wide NORs
- Turn off pMOS when not in use

Review: Dynamic Logic

- Dynamic gates uses a clocked pMOS pullup
- Two modes: precharge and evaluate
Review: Logical Effort

Inverter NAND2 NOR2

<table>
<thead>
<tr>
<th></th>
<th>Unfooted</th>
<th>Footed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inverter</td>
<td>$g_d = 1/3$</td>
<td>$g_d = 2/3$</td>
</tr>
<tr>
<td>$p_d$</td>
<td>$p_d = 2/3$</td>
<td>$p_d = 3/3$</td>
</tr>
</tbody>
</table>

Monotonicity

- Dynamic gates require *monotonically rising* inputs during evaluation
  - $0 \to 0$
  - $0 \to 1$
  - $1 \to 1$
  - But not $1 \to 0$

Monotonicity Woes

- But dynamic gates produce monotonically falling outputs during evaluation
- Illegal for one dynamic gate to drive another!

Domino Gates

- Follow dynamic stage with inverting static gate
  - Dynamic / static pair is called domino gate
  - Produces monotonic outputs

Domino Optimizations

- Each domino gate triggers next one, like a string of dominos toppling over
- Gates evaluate sequentially but precharge in parallel
- Thus evaluation is more critical than precharge
- HI-skewed static stages can perform logic
Dual-Rail Domino

- Domino only performs noninverting functions:
  - AND, OR but not NAND, NOR, or XOR
- Dual-rail domino solves this problem
  - Takes true and complementary inputs
  - Produces true and complementary outputs

<table>
<thead>
<tr>
<th>sig_h</th>
<th>sig_l</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Precharged</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>'0'</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>'1'</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Invalid</td>
</tr>
</tbody>
</table>

Example: AND/NAND

- Given $A_h, A_l, B_h, B_l$
- Compute $Y_h = A \cdot B$, $Y_l = \overline{(A \cdot B)}$
- Pulldown networks are conduction complements

Example: XOR/XNOR

- Sometimes possible to share transistors

Leakage

- Dynamic node floats high during evaluation
  - Transistors are leaky ($I_{OFF} \neq 0$)
  - Dynamic value will leak away over time
  - Formerly milliseconds, now nanoseconds!
- Use keeper to hold dynamic node
  - Must be weak enough not to fight evaluation

Charge Sharing

- Dynamic gates suffer from charge sharing
Charge Sharing

• Dynamic gates suffer from charge sharing

\[ V_i = V_f = \]

Charge sharing noise

Secondary Precharge

• Solution: add secondary precharge transistors
  – Typically need to precharge every other node
• Big load capacitance \( C_y \) helps as well

\[ V_i = V_f = \frac{C_i}{C_i + C_y} V_{DD} \]

Noise Sensitivity

• Dynamic gates are very sensitive to noise
  – Inputs: \( V_{IH} \approx V_{th} \)
  – Outputs: floating output susceptible to noise
• Noise sources
  – Capacitive crosstalk
  – Charge sharing
  – Power supply noise
  – Feedthrough noise
  – And more!

Domino Summary

• Domino logic is attractive for high-speed circuits
  – 1.5 – 2x faster than static CMOS
• But many challenges:
  • Monotonicity
  • Leakage
  • Charge sharing
  • Noise
• Widely used in high-performance microprocessors

NMOS Transistors in Series/Parallel

• Primary inputs drive both gate and source/drain terminals
• NMOS switch closes when the gate input is high

\[ X = Y \text{ if A and B} \]

\[ X = Y \text{ if A or B} \]

• Remember – NMOS transistors pass a strong 0 but a weak 1
PMOS Transistors in Series/Parallel

- Primary inputs drive both gate and source/drain terminals
- PMOS switch closes when the gate input is low
  \[ X = Y \text{ if } A \text{ and } B = A + B \]
  \[ X = Y \text{ if } A \text{ or } B = A \lor B \]
- Remember — PMOS transistors pass a strong 1 but a weak 0

Pass Transistor (PT) Logic

- Gate is static — a low-impedance path exists to both supply rails under all circumstances
- N transistors instead of 2N
- No static power consumption
- Ratioless
- Bidirectional (versus unidirectional)

VTC of PT AND Gate

Pure PT logic is not regenerative - the signal gradually degrades after passing through a number of PTs (can fix with static CMOS inverter insertion)

NMOS Only PT Driving an Inverter

- \( V_x \) does not pull up to \( V_{DD} \) but \( V_{DD} - V_{Tn} \)
- Threshold voltage drop causes static power consumption (\( M_2 \) may be weakly conducting forming a path from \( V_{DD} \) to GND)
- Notice \( V_{Tn} \) increases of pass transistor due to body effect \( (V_{SB}) \)

Voltage Swing of PT Driving an Inverter

- Body effect – large \( V_{SB} \) at \( x \) - when pulling high (\( B \) is tied to GND and \( S \) charged up close to \( V_{DD} \))
- So the voltage drop is even worse
  \[ V_x = V_{DD} - (V_{Tn0} + \gamma(\sqrt{|2\phi_f| + V_x} - \sqrt{|2\phi_f|})) \]

Cascaded NMOS Only PTs

- Pass transistor gates should never be cascaded as on the left
- Logic on the right suffers from static power dissipation and reduced noise margins
Solution 1: Level Restorer

- Full swing on x (due to Level Restorer) so no static power consumption by inverter
- No static backward current path through Level Restorer and PT since Restorer is only active when A is high
- For correct operation M, must be sized correctly (ratioed)

Solution 2: Multiple V_T Transistors

- Technology solution: Use (near) zero V_T devices for the NMOS PTs to eliminate most of the threshold drop (body effect still in force preventing full swing to VDD)
- Impacts static power consumption due to subthreshold currents flowing through the PTs (even if V_{GS} is below V_T)

Solution 3: Transmission Gates (TGs)

- Most widely used solution
- Full swing bidirectional switch controlled by the gate signal C, A = B if C = 1
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

\[ F = \overline{\overline{S_1 \cdot S_2 + S_1 \cdot S_2}} \]

TG Multiplexer

\[ F = \overline{\overline{S_1 \cdot S_2 + S_1 \cdot S_2}} \]

Transmission Gate XOR

\[ F = A \oplus B \]

Transmission Gate XOR

\[ F = A \oplus B \]

TG Full Adder

\[ F = A \oplus B \]

Differential TG Logic (DPL)

\[ F = A \oplus B \]
CPL

- Complementary Pass-transistor Logic
  - Dual-rail form of pass transistor logic
  - Avoids need for ratioed feedback
  - Optional cross-coupling for rail-to-rail swing

Differential PT Logic (CPL)

- 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 VDD 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

CPL Full Adder

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

Sequential Logic

- 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

Timing Metrics

**System Timing Constraints**

- \( T_{\text{delay}} + t_{\text{logic}} + t_{\text{hold}} \geq T \)
- \( T \geq t_{\text{q}} + t_{\text{logic}} + t_{\text{u}} \)
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

Latch Design

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

- **Transmission gate**
  - **Pros**
    - No $V_t$ drop
    - Requires inverted clock
Latch Design

- 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

Latch Design

- 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
**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

**Max-Delay: Flip-Flops**

\[ t_{pd} \leq T_r = \left( t_{setup} + t_{hold} \right) \]

\[ t_{pd} \leq T_r = \left( t_{setup} + t_{hold} + t_{prop} \right) \]

- Combinational Logic
- Latch/Flop Hold Time
- Latch/Flop Setup Time
- Latch D-Q Prop Delay
- Latch D-Q Cont. Delay
- Latch/Flop Clk-Q Prop Delay
- Latch/Flop Clk-Q Cont. Delay
- Logic Prop. Delay
- Logic Cont. Delay
- Combinational Logic
Max Delay: 2-Phase Latches

\[ T_d = \max \left\{ T_{pd}, T_{pq} \right\} \]

Max Delay: 2-Phase Latches

\[ T_d = \max \left\{ T_{pd}, T_{pq} \right\} \]

Max Delay: Pulsed Latches

\[ T_d \leq T = \max \left\{ \text{supporting overhead} \right\} \]

Max Delay: Pulsed Latches

\[ T_d \leq T = \max \left\{ \text{setup}, T \right\} \]

Min-Delay: Flip-Flops

\[ T_d \geq \]

Min-Delay: Flip-Flops

\[ T_d \geq \]

\[ t_{setup} \]
Min-Delay: 2-Phase Latches

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

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

\[ t_{pu} \geq t_{pu} \]

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
How Much Borrowing?

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

Pulsed Latches
\[ t_{\text{borrow}} \leq t_{\text{nonoverlap}} \]

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

\[ t_{\text{skew}} \leq T - (t_{\text{setup}} + t_{\text{skew}} + t_{\text{nonoverlap}}) \]

Skew: Latches

2-Phase Latches
\[ t_{\text{skew}} \leq T - (2t_{\text{setup}} + t_{\text{skew}} + t_{\text{nonoverlap}} + t_{\text{hold}}) \]

Pulsed Latches
\[ t_{\text{skew}} \leq T - \max\left(t_{\text{setup}}, t_{\text{skew}}, t_{\text{nonoverlap}}, t_{\text{hold}}\right) \]

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