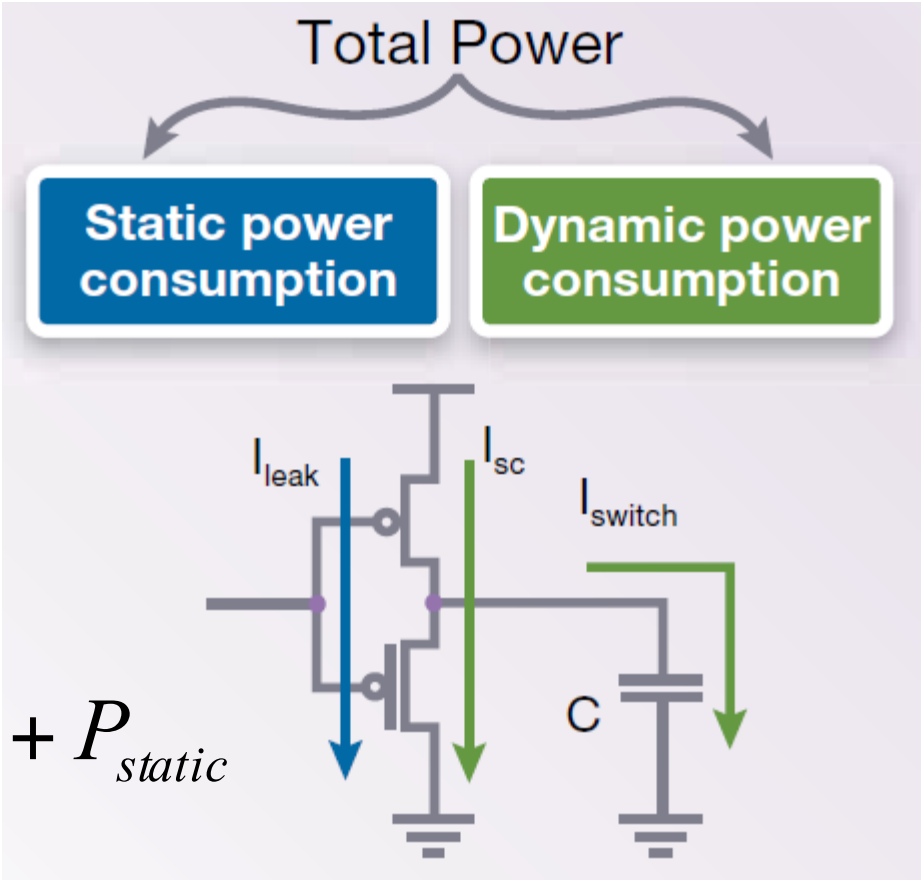
Lecture 21
Power Optimization (Part 2)

Xuan ‘Silvia’ Zhang
Washington University in St. Louis

http://classes.engineering.wustl.edu/ese461/

Power Dissipation

- Dynamic power consumption
  - switching current
- Static power consumption
  - short-circuit current
  - leakage current



$$P_{avg} = P_{dyn} + P_{short} + P_{lkg} + P_{static}$$

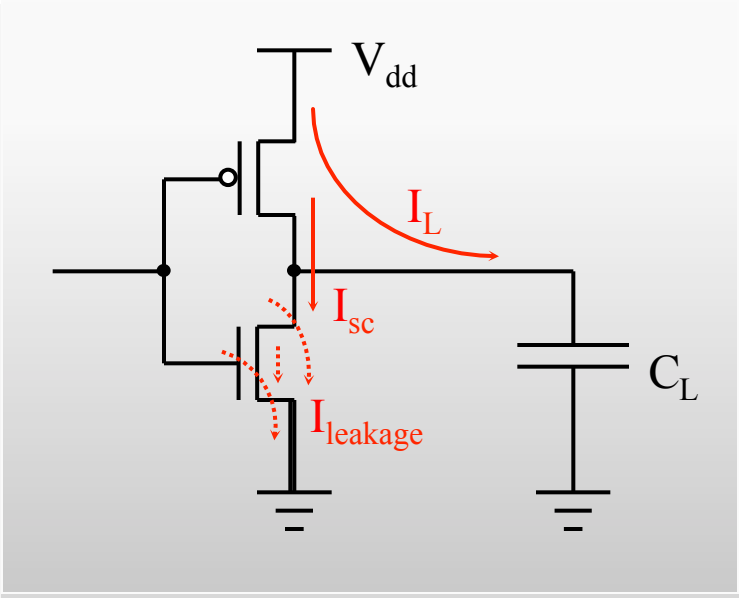
Low Power Design Methodologies

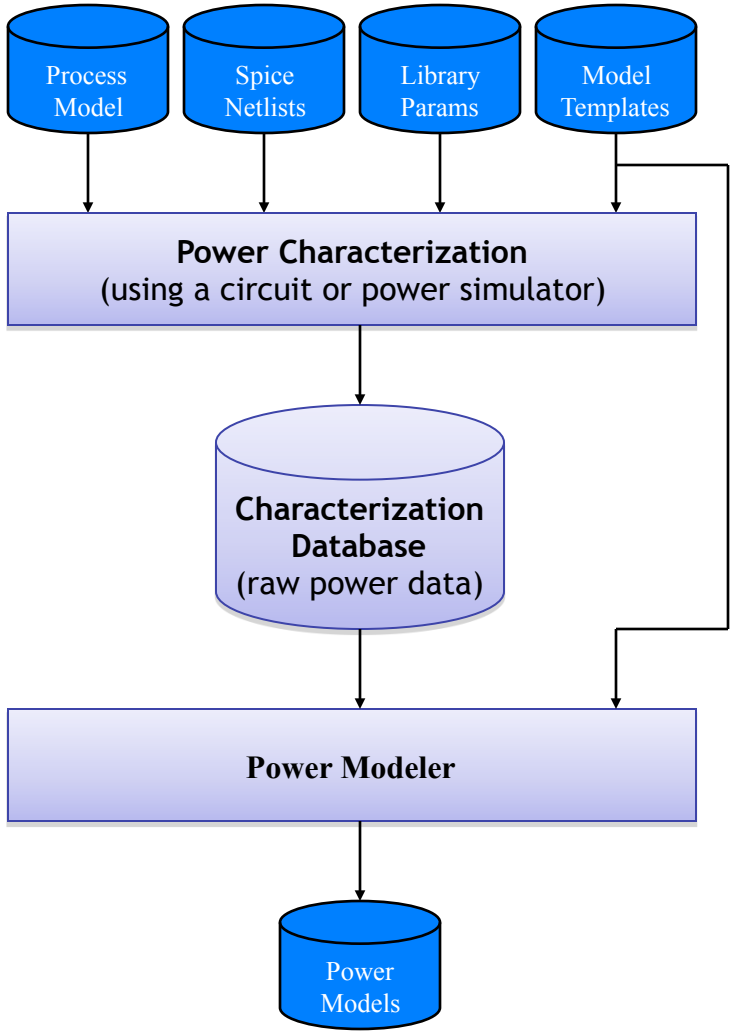
- Adapt process technology
  - reduce capacitance
  - reduce leakage current
  - reduce supply voltage
- Reduce switch activity
  - minimize glitches
  - minimize number of operations
  - low power bus encoding
  - scheduling and binding optimization
- Power down modes
  - clock gating
  - memory partitioning
  - power gating
- Voltage optimization and scaling

Design Flow Integration

- **Power Characterization and Modeling**
  - How to generate macro-model power data?
  - Model accuracy
- **Power Analysis**
  - When to analyze?
  - Which modes to analyze?
  - How to use the data?
- **Power Reduction**
  - Logical modes of operation
    - For which modes should power be reduced?
  - Dynamic power versus leakage power
  - Physical design implications
  - Functional and timing verification
  - Return on Investment
    - How much power is reduced for the extra effort? Extra logic? Extra area?
- **Power Integrity**
  - Peak instantaneous power
  - Electromigration
  - Impact on timing

Power Characterization and Modeling





[source: J. Frenkil, Kluwer'02]

Generalized Low-Power Design Flow

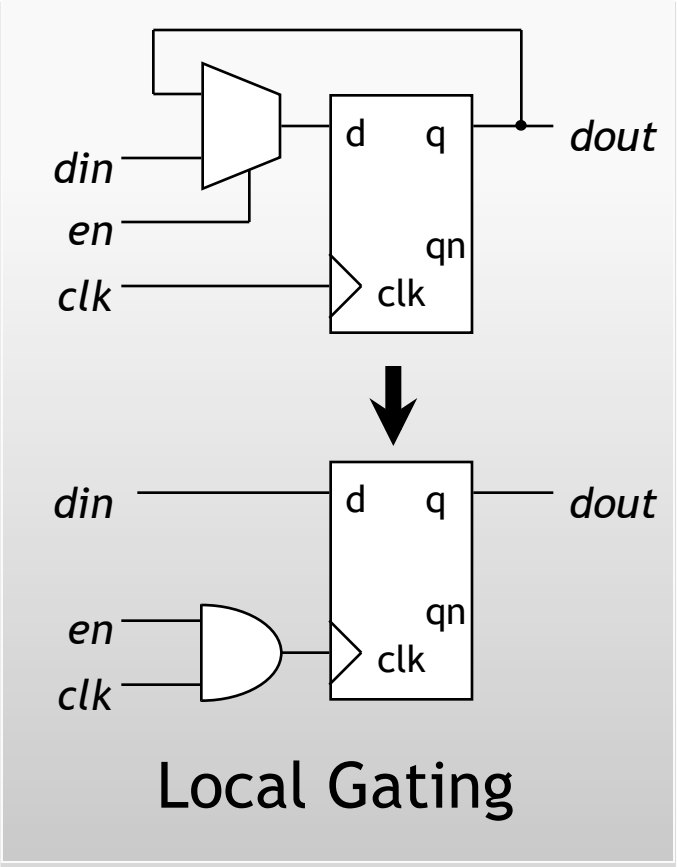
Design Phase	Low Power Design Activities
System-Level Design	• Explore architectures and algorithms for power efficiency • Map functions to sw and/or hw blocks for power efficiency • Choose voltages and frequencies • Evaluate power consumption for different operational modes • Generate budgets for power, performance, area
RTL Design	• Generate RTL to match system-level model • Select IP blocks • Analyze and optimize power at module level and chip level • Analyze power implications of test features • Check power against budget for various modes
Implementation	• Synthesize RTL to gates using power optimizations • Floorplan, place and route design • Optimize dynamic and leakage power • Verify power budgets and power delivery

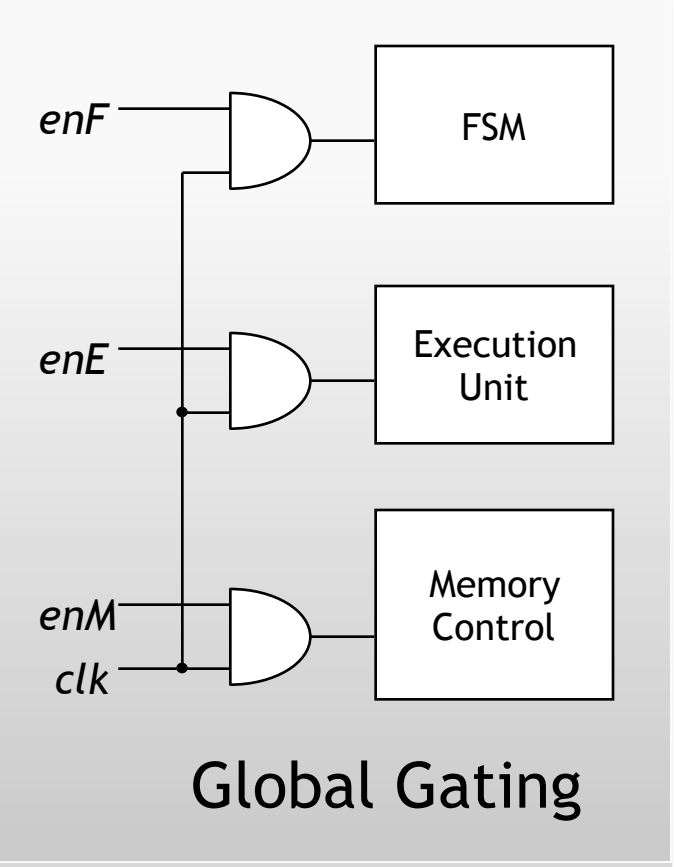
Design-Phase Low Power Design

- Primary objective: minimize f_{eff}
- Clock gating
  - Reduces / inhibits unnecessary clocking
    - Registers need not be clocked if data input hasn't changed
- Data gating
  - Prevents nets from toggling when results won't be used
    - Reduces wasted operations
- Memory system design
  - Reduces the activity internal to a memory
    - Cost (power) of each access is minimized

Clock Gating

- Power is reduced by two mechanisms
  - Clock net toggles less frequently, reducing f_{eff}
  - Registers' internal clock buffering switches less often





Clock Gating Insertion

- Local clock gating: 3 methods
  - Logic synthesizer finds and implements local gating opportunities
  - RTL code explicitly specifies clock gating
  - Clock gating cell explicitly instantiated in RTL

- Global clock gating: 2 methods
  - RTL code explicitly specifies clock gating
  - Clock gating cell explicitly instantiated in RTL

Clock Gating Verilog Code

- Conventional RTL Code

```
//always clock the register
always @ (posedge clk) begin   // form the flip-flop
    if (enable) q = din;
end
```

- Low Power Clock Gated RTL Code

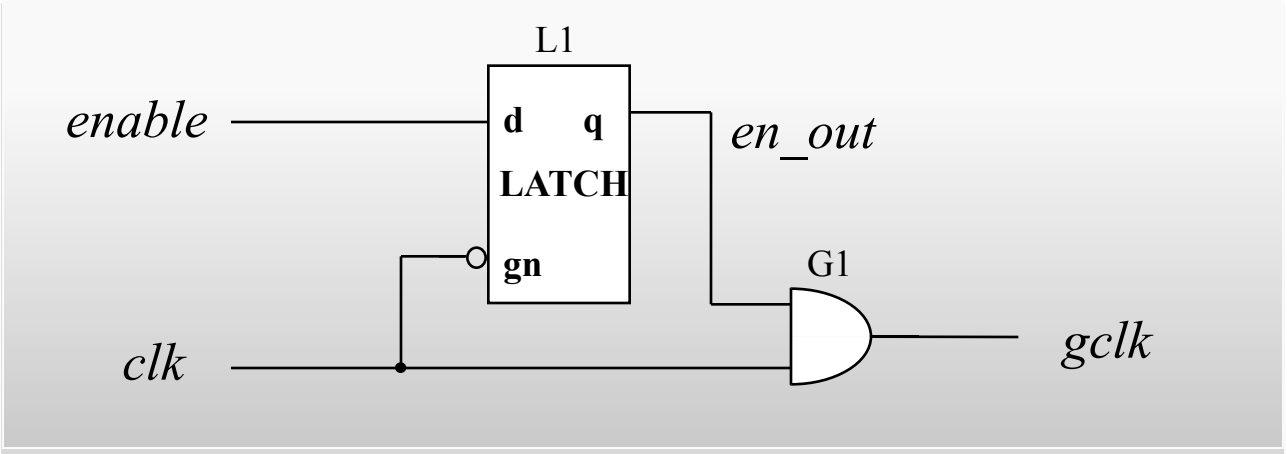
```
//only clock the register when enable is true
assign gclk = enable && clk;   // gate the clock
always @ (posedge gclk) begin  // form the flip-flop
    q = din;
end
```

- Instantiated Clock Gating Cell

```
//instantiate a clock gating cell from the target library
clkgx1 i1 .en(enable), .cp(clk), .gclk_out(gclk);
always @ (posedge gclk) begin   // form the flip-flop
    q = din;
end
```

Clock Gating: Glitch Free Verilog

- Add a Latch to Prevent Clock Glitching

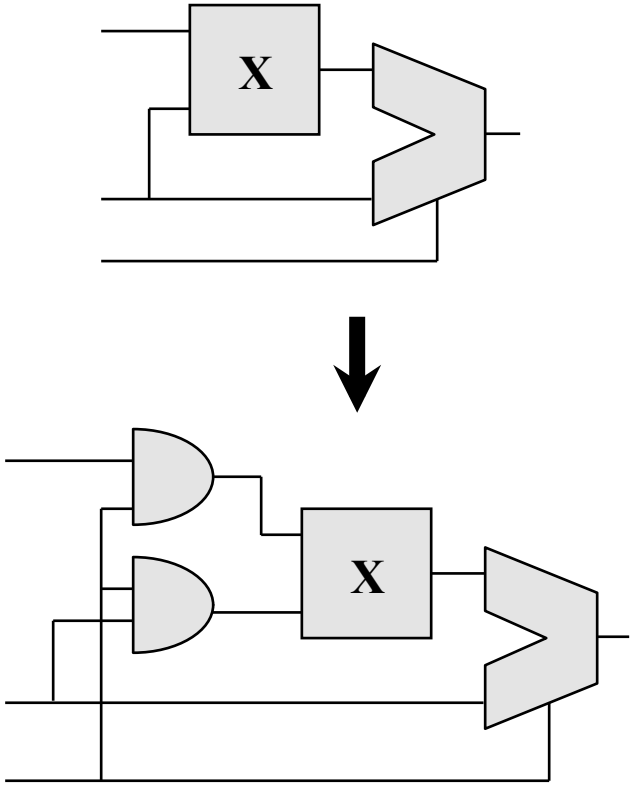


- Clock Gating Code with Glitch Prevention Latch

```
always @ (enable or clk) begin
    if !clk then en_out = enable // build latch
end
assign gclk = en_out && clk;     // gate the clock
```

Data Gating

- Objective
  - Reduce wasted operations => reduce f_{eff}
- Example
  - Multiplier whose inputs change every cycle, whose output conditionally feeds an ALU
- Low Power Version
  - Inputs are prevented from rippling through multiplier if multiplier output is not selected



Data Gating Insertion

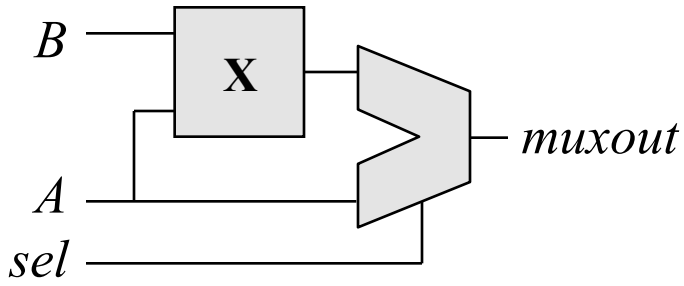
- Two insertion methods
  - Logic synthesizer finds and implements data gating opportunities
  - RTL code explicitly specifies data gating
    - Some opportunities cannot be found by synthesizers

- Issues
  - Extra logic in data path slows timing
  - Additional area due to gating cells

Data Gating Verilog Code: Operand Isolation

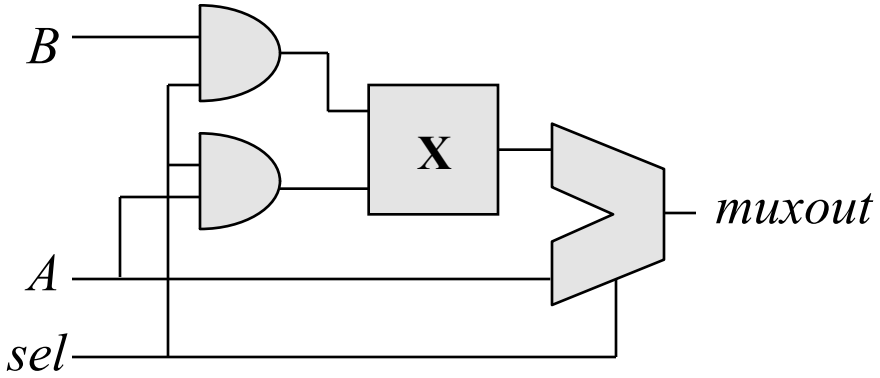
- Conventional Code

```
assign muxout = sel ? A : A*B ;  // build mux
```



- Low Power Code

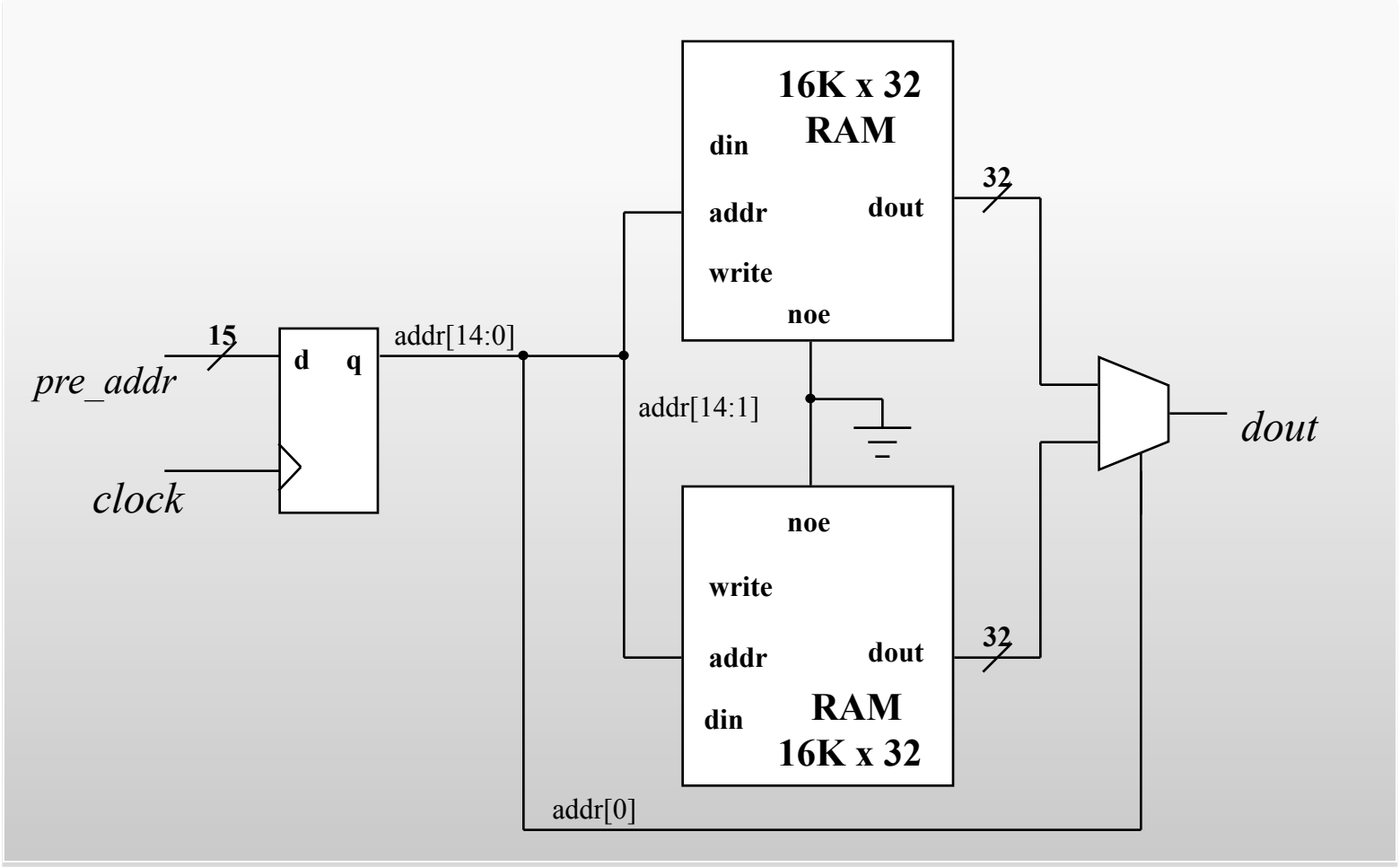
```
assign multinA = sel & A ;  // build and gate
assign multinB = sel & B ;  // build and gate
assign muxout =  sel ? A : multinA*multinB ;
```



Memory System Design

- Primary objectives: minimize f_{eff} and C_{eff}
  - Reduce number of accesses or (power) cost of an access
- Power Reduction Methods
  - Memory banking / splitting
  - Minimization of number of memory accesses
- Challenges and Tradeoffs
  - Dependency upon access patterns
  - Placement and routing

Split Memory Access



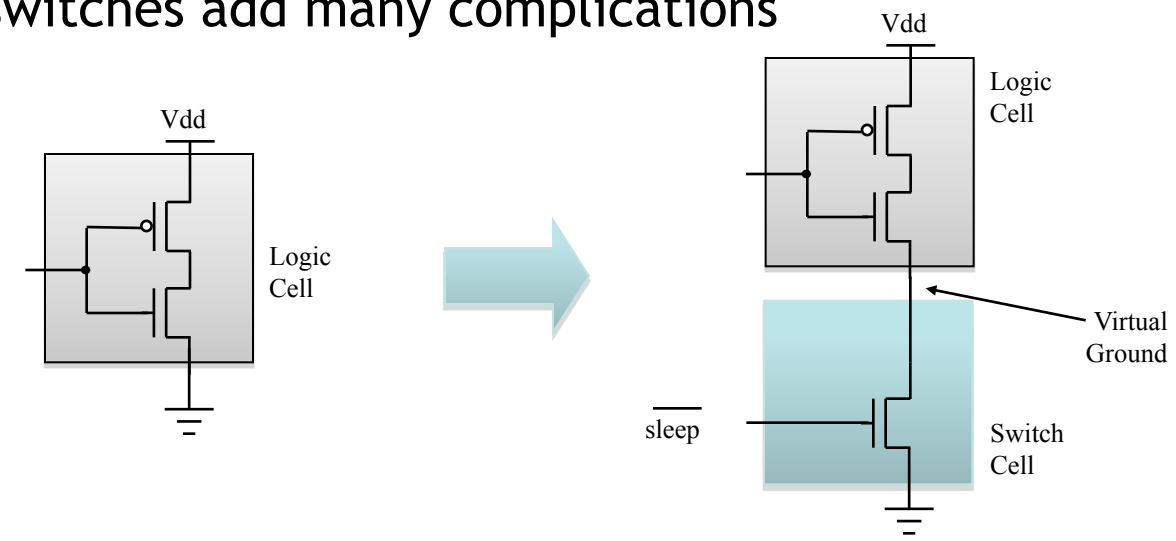
Implementation Phase Low Power Design

Primary objective: minimize power consumed by individual instances

- Low power synthesis
  - Dynamic power reduction via local clock gating insertion, pin-swapping
- Slack redistribution
  - Reduces dynamic and/or leakage power
- Power gating
  - Largest reductions in leakage power
- Multiple supply voltages
  - The implementation of earlier choices
- Power integrity design
  - Ensures adequate and reliable power delivery to logic

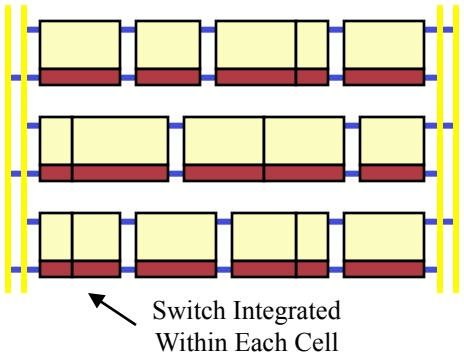
Power Gating

- Objective
  - Reduce leakage currents by inserting a switch transistor (usually high V_{TH}) into the logic stack (usually low V_{TH})
    - Switch transistors change the bias points (V_{SB}) of the logic transistors
- Most effective for systems with standby operational modes
  - 1 to 3 orders of magnitude leakage reduction possible
  - But switches add many complications

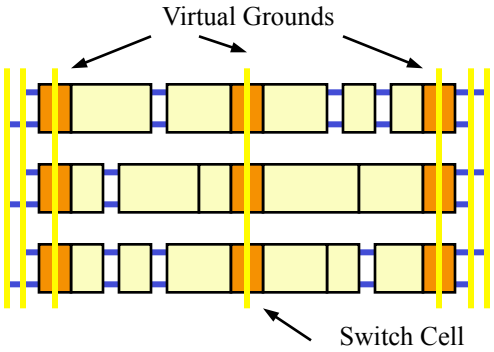


Power-Gating Physical Design

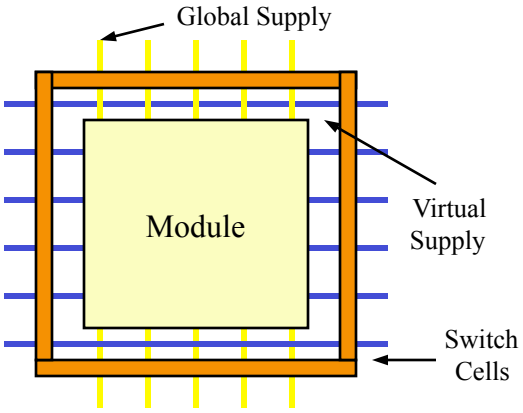
- Switch placement
  - In each cell?
    - Very large area overhead, but placement and routing is easy
  - Grid of switches?
    - Area efficient, but a third global rail must be routed
  - Ring of switches?
    - Useful for hard layout blocks, but area overhead can be significant



Switch-in-cell



Grid of Switches

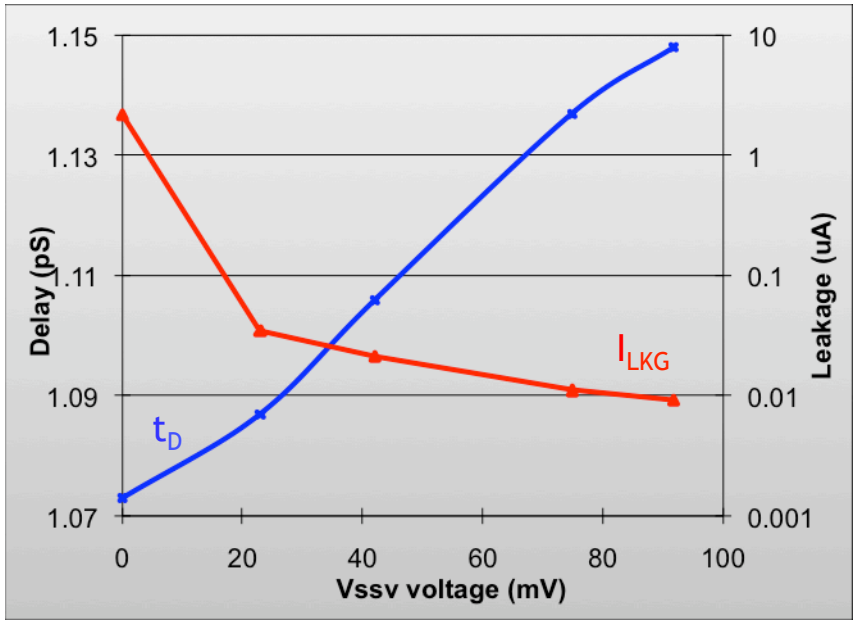


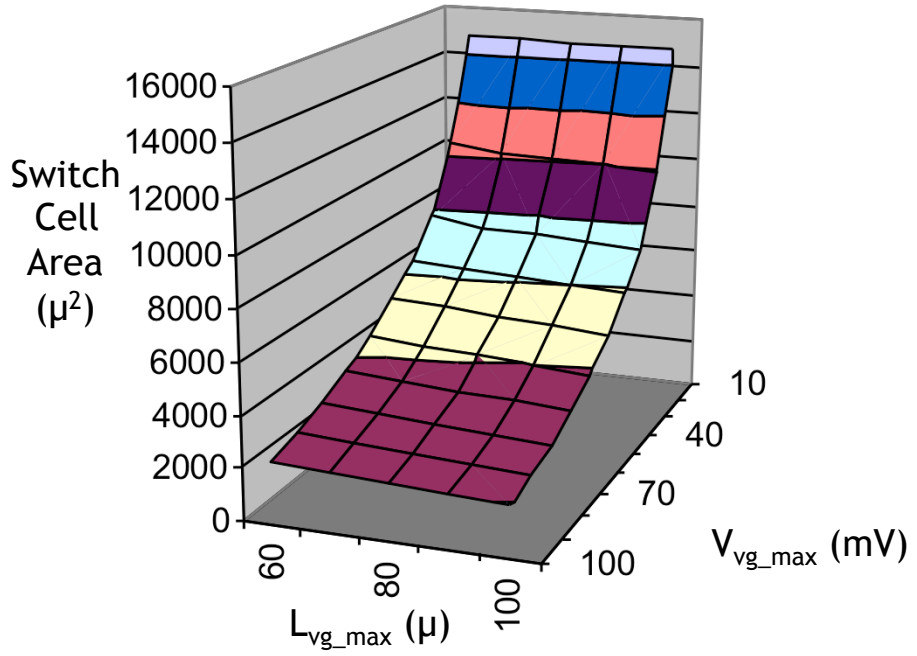
Ring of Switches

[source: S. Kosonocky, ISLPED'01]

Power Gating Switch Sizing

- Tradeoff between area, performance, leakage
  - Larger switches => less voltage drop, larger leakage, more area
  - Smaller switches => larger voltage drop, less leakage, less area



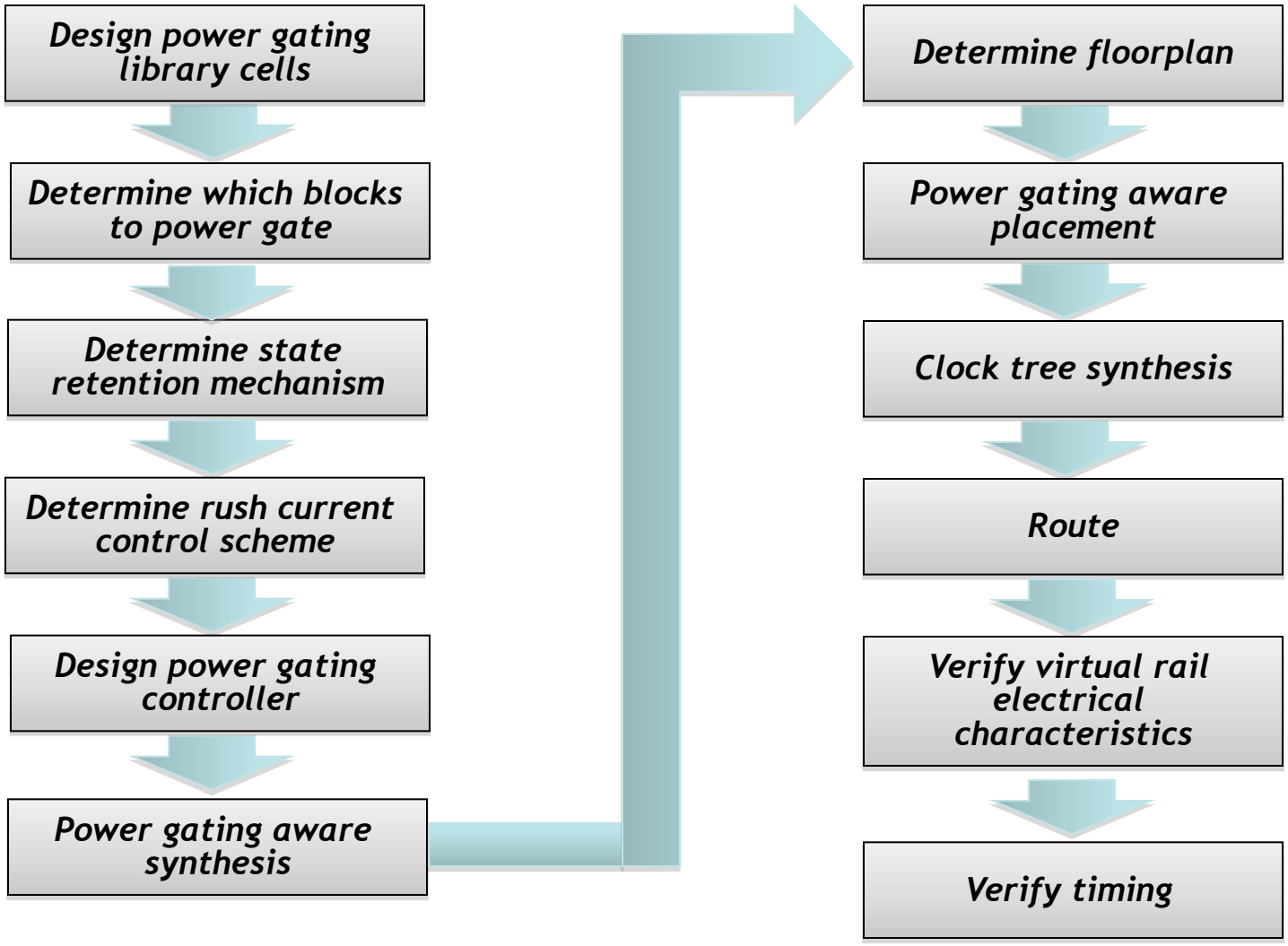


[source: J. Frenkil, Springer'07]

Power Gating: Additional Issues

- Library design: special cells are needed
  - Switches, isolation cells, state retention flip-flops (SRFFs)
- Headers or Footers?
  - Headers better for gate leakage reduction, but ~ 2X larger
- Which modules, and how many, to power gate?
  - Sleep control signal must be available, or must be created
- State retention: which registers must retain state?
  - Large area overhead for using SRFFs
- Floating signal prevention
  - Power-gate outputs that drive always-on blocks must not float
- Rush currents and wakeup time
  - Rush currents must settle quickly and not disrupt circuit operation
- Delay effects and timing verification
  - Switches affect source voltages which affect delays
- Power-up & power-down sequencing
  - Controller must be designed and sequencing verified

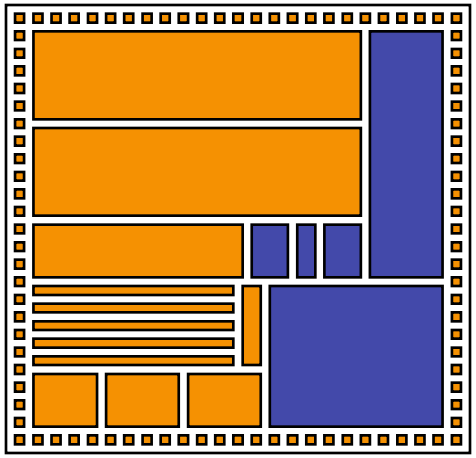
Power Gating Flow



Multi-VDD

- Objective
  - Reduce dynamic power by reducing the V_{DD}^2 term
    - Higher supply voltage used for speed-critical logic
    - Lower supply voltage used for non speed-critical logic

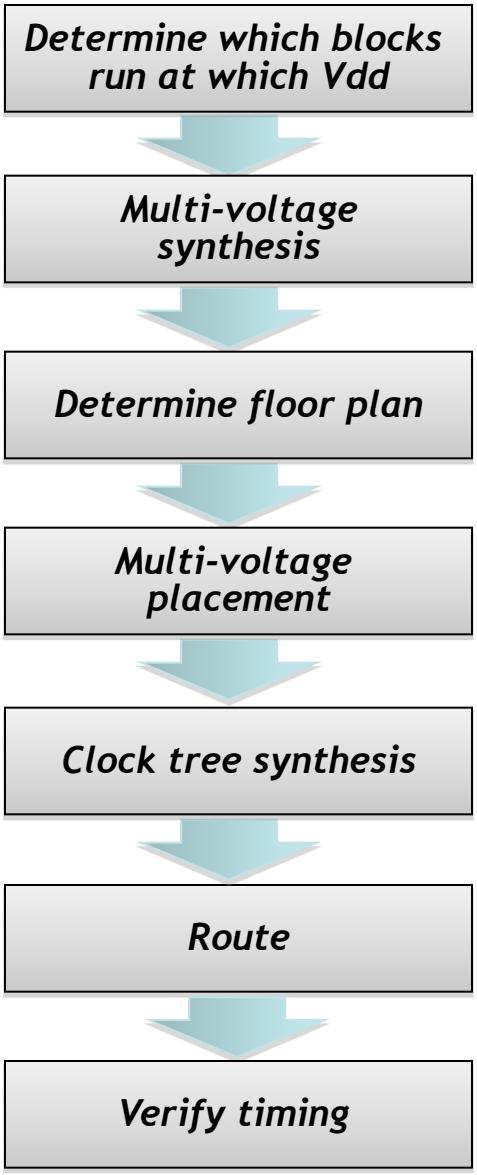
- Example
  - Memory V_{DD} = 1.2 V
  - Logic V_{DD} = *1*.0 V
  - Logic dynamic power savings = 30%



Multi-VDD Issues

- Partitioning
  - Which blocks and modules should use with voltages?
  - Physical and logical hierarchies should match as much as possible
- Voltages
  - Voltages should be as low as possible to minimize $CV_{DD}^2 f$
  - Voltages must be high enough to meet timing specs
- Level shifters
  - Needed (generally) to buffer signals crossing islands
    - May be omitted if voltage differences are small, ~ 100mV
  - Added delays must be considered
- Physical design
  - Multiple V_{DD} rails must be considered during floorplanning
- Timing verification
  - Signoff timing verification must be performed for all corner cases across voltage islands.
  - For example, for 2 voltage islands V_{hi}, V_{lo}
    - Number of timing verification corners doubles

Multi-VDD Flow



Questions?

Comments?

Discussion?