Agenda

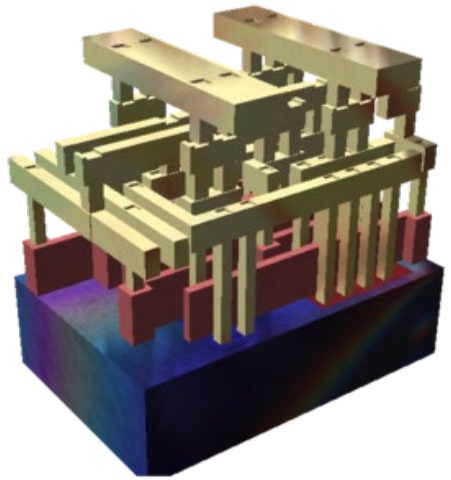
- Tutorial 3:
  - Digital Design Flow
- Tutorial 4:
  - ECE1388 Project 3 - 4-bit array multiplier block
- Case Study:
  - Why all this is useful – 2 recent design examples

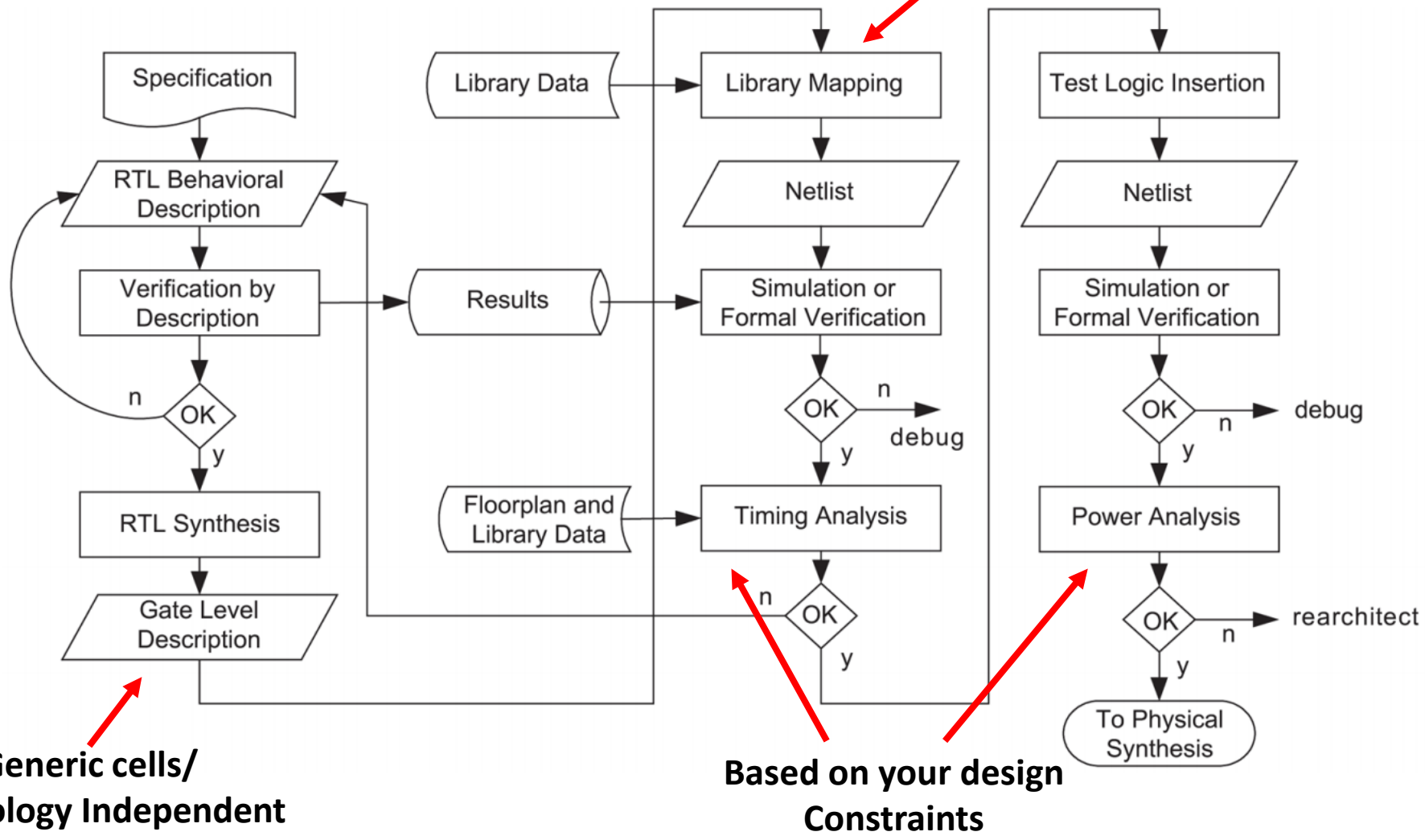
- Tutorial 5: More Advanced Design Techniques and the Future
  - Foundation Flow
  - Low-Power Design

Recap: Tutorial 3

VLSI Digital Implementation Flow

```
 1  module adder
 2    ( input [7:0] A,
 3      input [7:0] B,
 4      input clk,
 5      input rst_n,
 6      output reg [7:0] sum,
 7      output reg carry);
 8
 9    always @(posedge clk or negedge rst_n)
10      if (!rst_n)
11        {carry,sum} <= 0;
12      else
13        {carry,sum} <= A + B;
14  endmodule
```





http://www.csee.umbc.edu/~cpatel2/links/641/slides/lect01_flow.pdf

Chip Layout: Manual vs Automated

Start Here!

Manual:
GUI Based

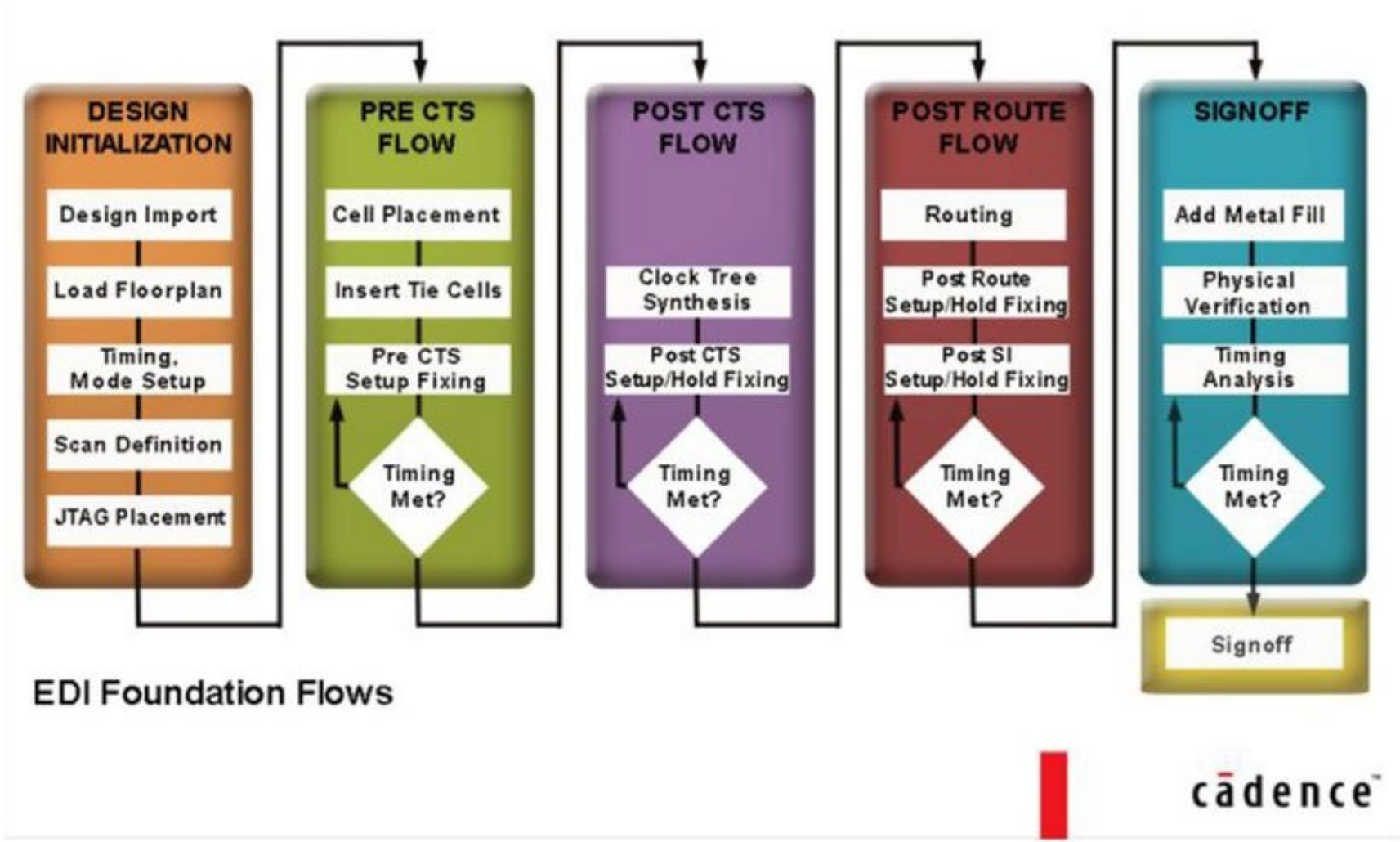
- Highly recommended for every new design!
- Control each step of the process
- Closely monitor each aspect, set options and manually tweak them as needed

Automated:
TCL Script Based

- Recommended for advanced users
- Useful for repeated layout trials with minor changes to various constraint parameters
- Enables optimizing the Layout results

Remember: You need to know the basics before you run the scripts! They will not magically work if you use them incorrectly

The Foundation Flow

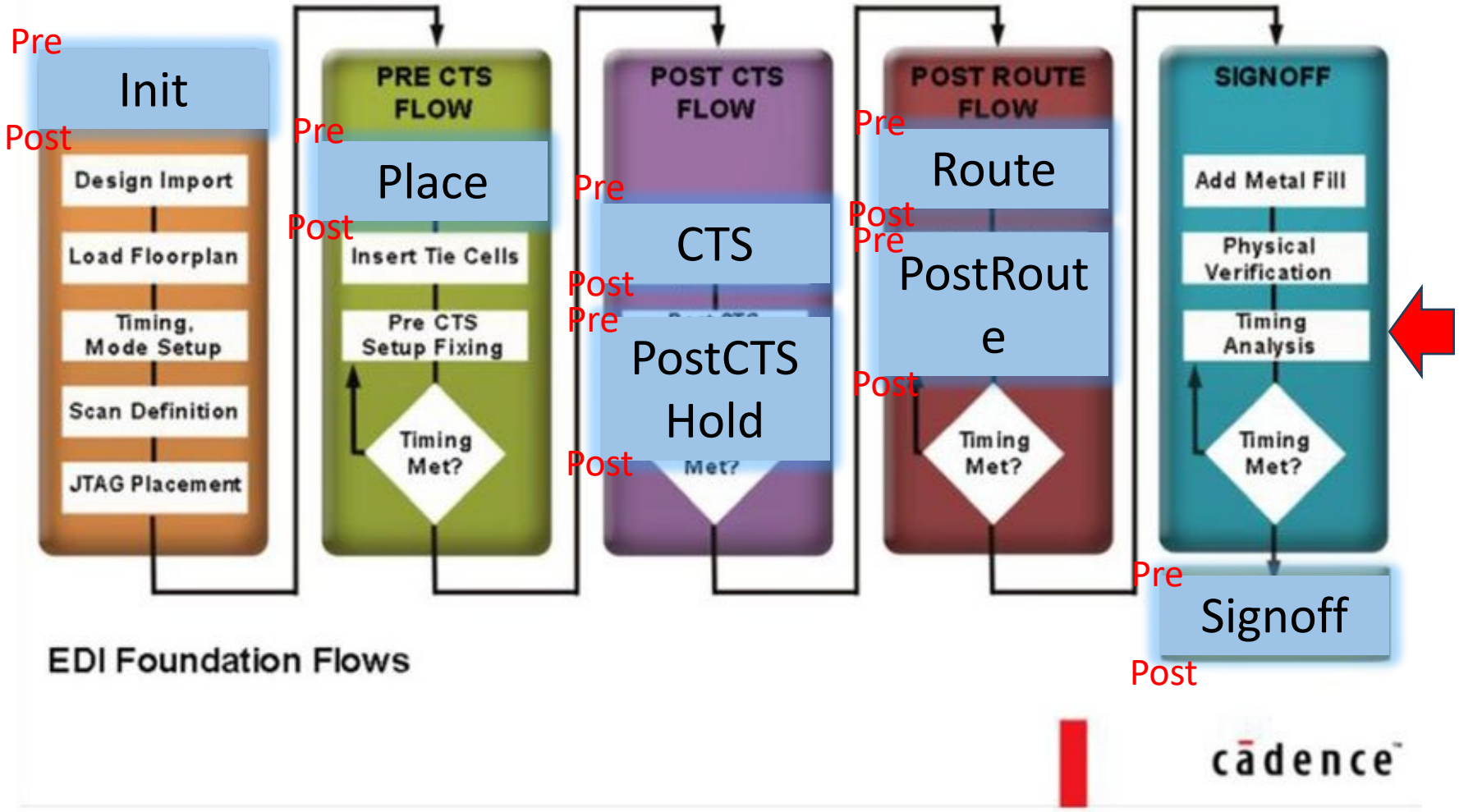


/FoundationFlow/PLUG/INNOVUS

```
always_source.tcl
post_cts.tcl
post_init.tcl
post_place.tcl
post_postcts.tcl
post_postcts_hold.tcl
post_postroute.tcl
post_postroute_hold.tcl
post_postroute_si.tcl
post_postroute_si_hold.tcl
post_prects.tcl
post_route.tcl
post_signoff.tcl
pre_cts.tcl
pre_init.tcl
pre_partition.tcl
pre_place.tcl
pre_place_checks.tcl
pre_postcts.tcl
pre_postroute.tcl
pre_postroute_hold.tcl
pre_postroute_si.tcl
pre_postroute_si_hold.tcl
pre_prects.tcl
pre_route.tcl
pre_postcts_hold.tcl
pre_signoff.tcl
```

The Foundation Flow

/FoundationFlow/PLUG/INNOVUS




```
always_source.tcl
post_cts.tcl
post_init.tcl
post_place.tcl
post_postcts.tcl
post_postcts_hold.tcl
post_postroute.tcl
post_postroute_hold.tcl
post_postroute_si.tcl
post_postroute_si_hold.tcl
post_prects.tcl
post_route.tcl
post_signoff.tcl
pre_cts.tcl
pre_init.tcl
pre_partition.tcl
pre_place.tcl
pre_place_checks.tcl
pre_postcts.tcl
pre_postroute.tcl
pre_postroute_hold.tcl
pre_postroute_si.tcl
pre_postroute_si_hold.tcl
pre_prects.tcl
pre_route.tcl
pre_postcts_hold.tcl
pre_signoff.tcl
```

General Recommendations: Learn TCL, RTM.

- Place IO buffers

```
attachIOBuffer -port  -suffix "_buf" -markFixed -out BUFFD24 -in BUFFD24 -excludeClockNet
```

- Learn TCL. E.g. Place_pins.tcl

```
editPin -side Right \
-layer $pin_layer_v \
-spreadType start \
-spreadDirection counterclockwise \
-start 0.0 $dist_from_bottom \
-pin $pins_node \
-spacing $pin_spacing \
-fixedPin 1
```

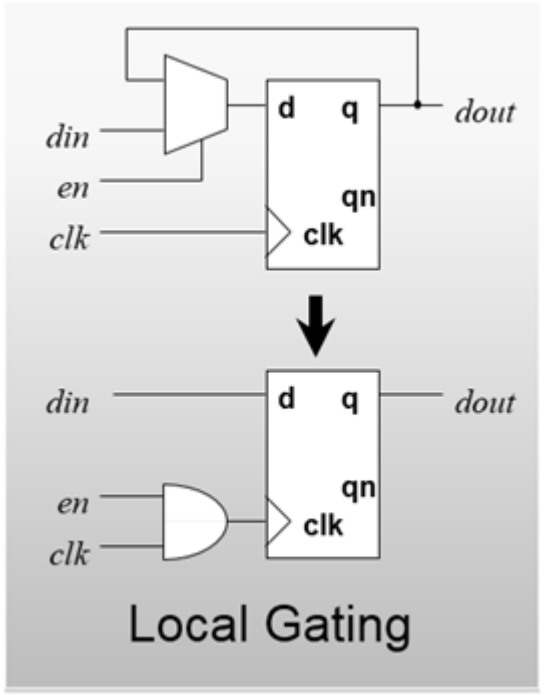
Design-Phase: Low Power Design

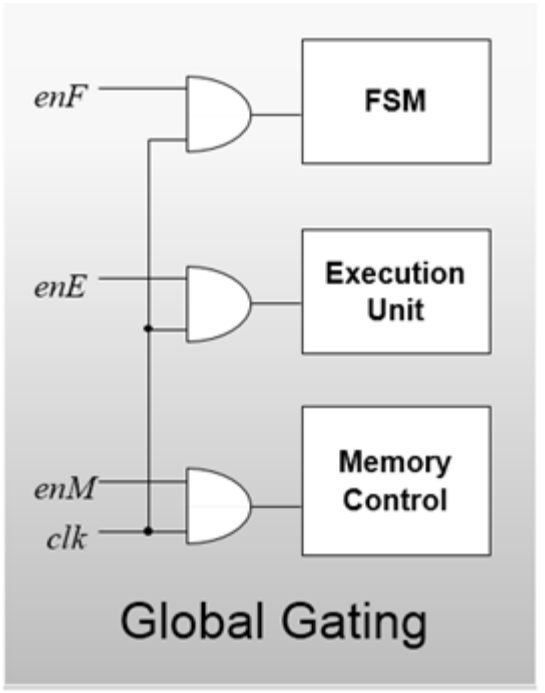
- Primary objective: minimize the effective clock frequency of the design f_{eff}
- Clock gating (useful for sequential logic)
  - Reduces / inhibits unnecessary clocking
    - Registers need not be clocked if data input hasn't changed
- Data gating (useful for combinational logic)
  - Prevents nets from toggling when results won't be used
    - Reduces wasted operations
- Power Gating / Grid Design Tips

Clock Gating

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

- 3 implementation methods
  - Logic synthesizer finds and implements local gating opportunities (use with caution!)
  - RTL code explicitly specifies clock gating
  - Clock gating cell explicitly instantiated in RTL





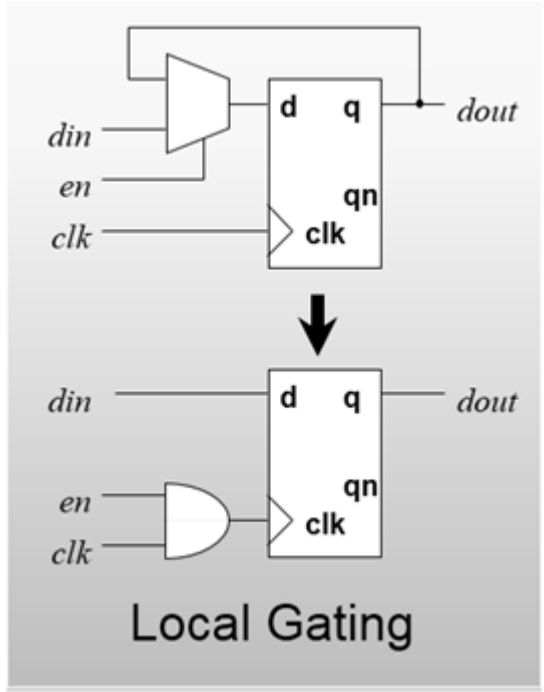
3 Clock Gating Verilog Approaches

- 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
```

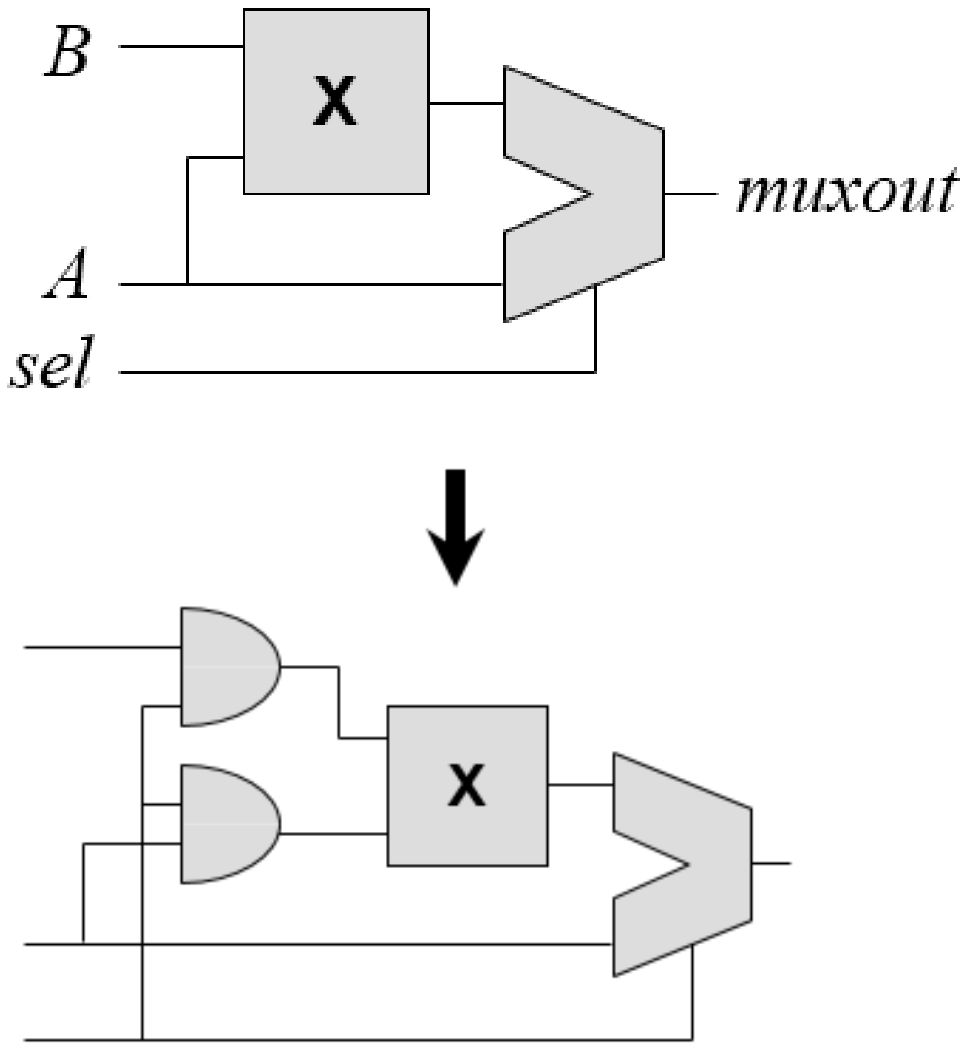


- Instantiated Clock Gating Cell (Not available in current 65nm libraries)

```
//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
```

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

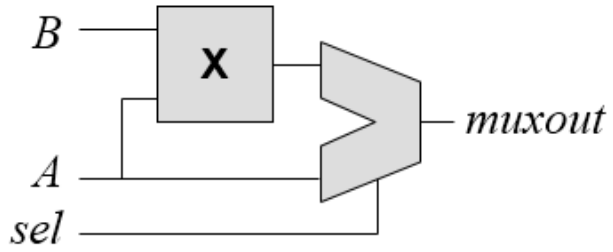


If not using multout at muxout…
why bother propagating operands?

Data Gating Verilog Code: Operand Isolation

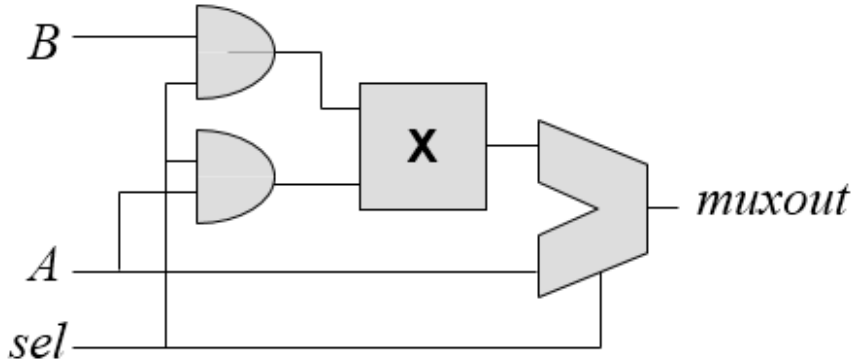
- Conventional Code

```
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 ;
```

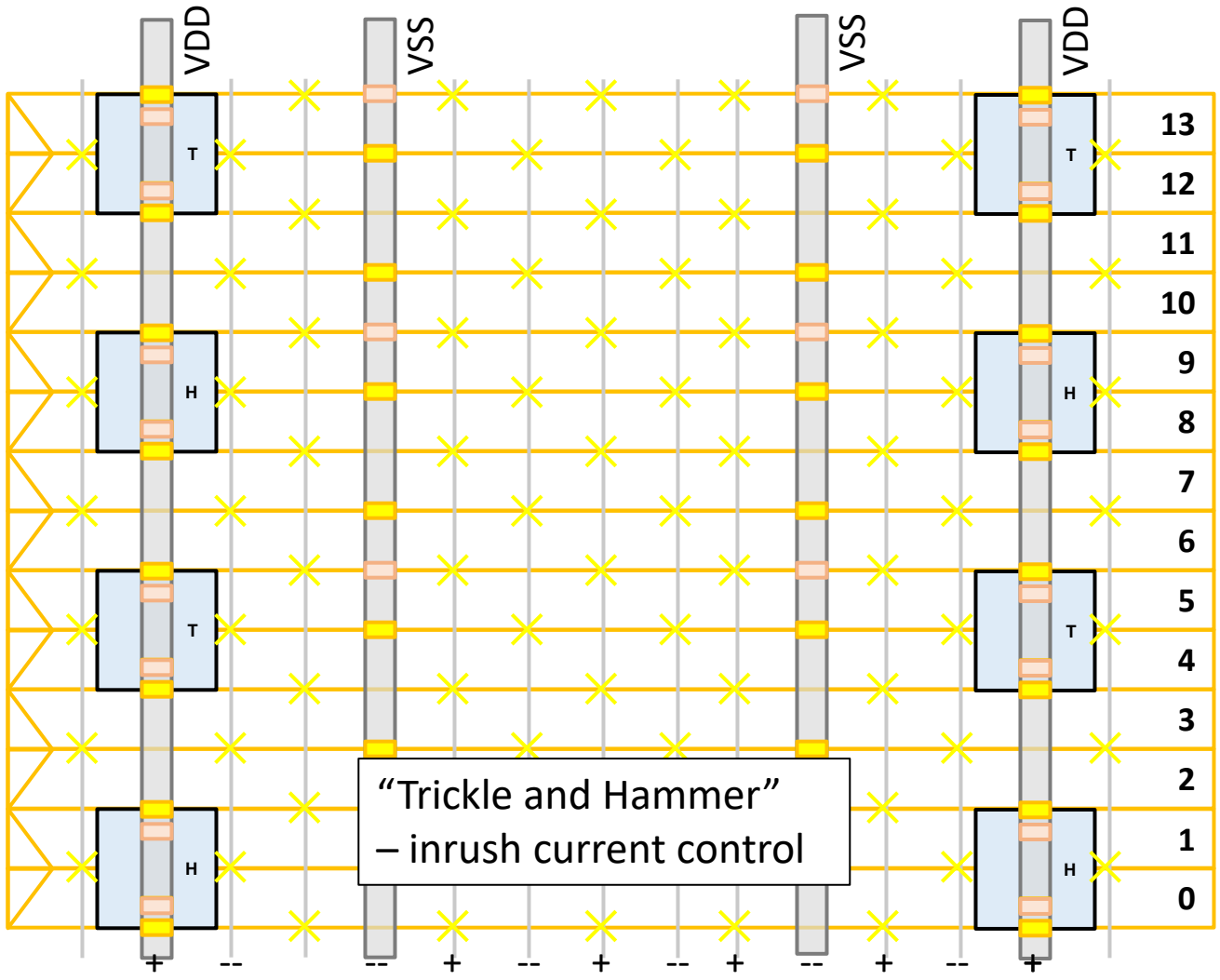


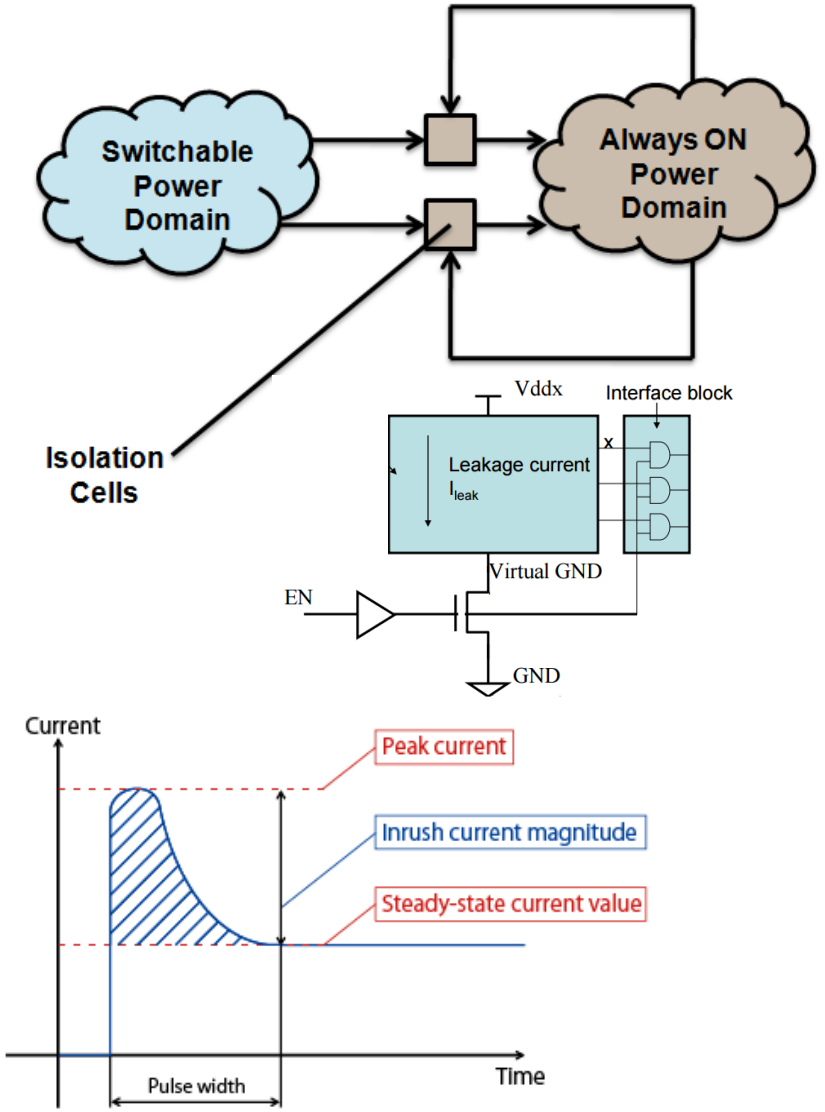
Data Gating Insertion

- 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 impacts timing!
  - Additional area due to gating cells

Multiple Power Domains – “Dark Silicon”





Resources

- **Ch 6, 7 and 8 in the VLSI User Manual**
- **“Low Power Design Essentials”**
  - Jan Rabaey