Low Power Design

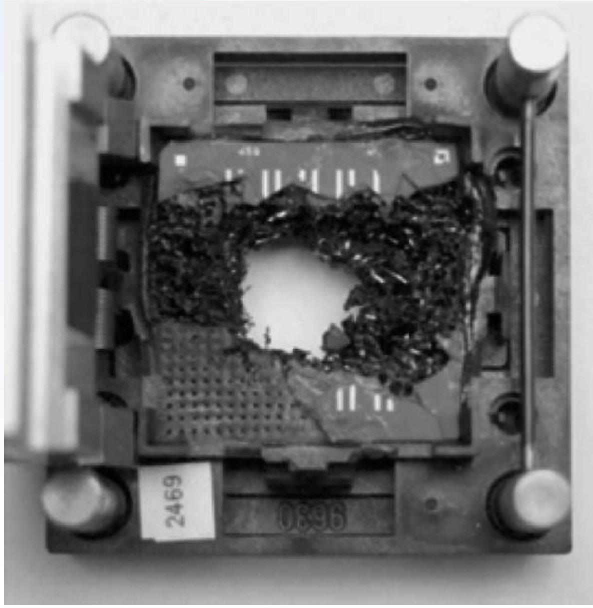
NYCU-EE IC LAB Fall-2024

NCTU OASIS Lab
founded in 2003

Lecturer: Yen-Ning Tung

Why Low Power?

- Power↑ ➔ Temperature↑ ➔ Safety↓
- May lead to permanent damage to the chip
- Portable device require low power



Outline

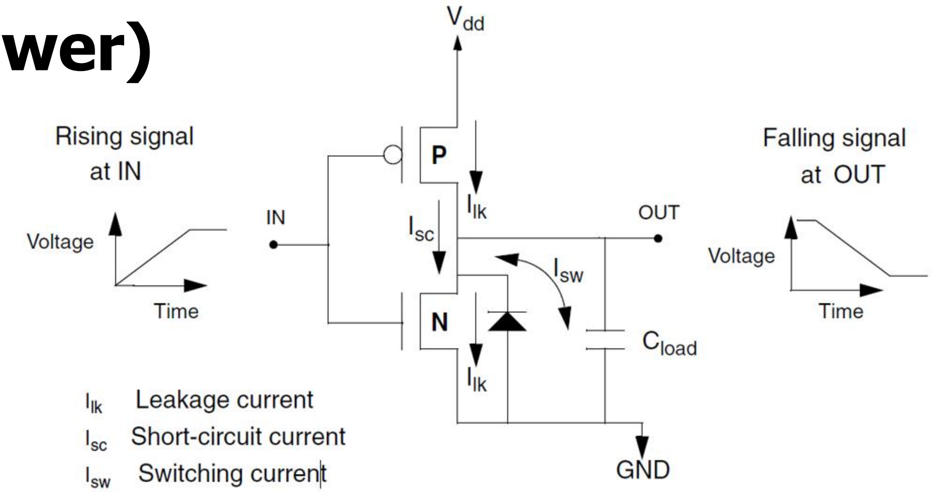
- ✓ **Power Dissipation**
  - Static Power Dissipation
  - Dynamic Power Dissipation
- ✓ **Low Power Design Introduction**
- ✓ **Static Power Reduction**
  - Multi-Threshold Voltage
- ✓ **Dynamic Power Reduction**
  - Multi-Voltage
  - Power Gating
  - RTL and Architecture Design Techniques
  - Clock gating
- ✓ **Reference**

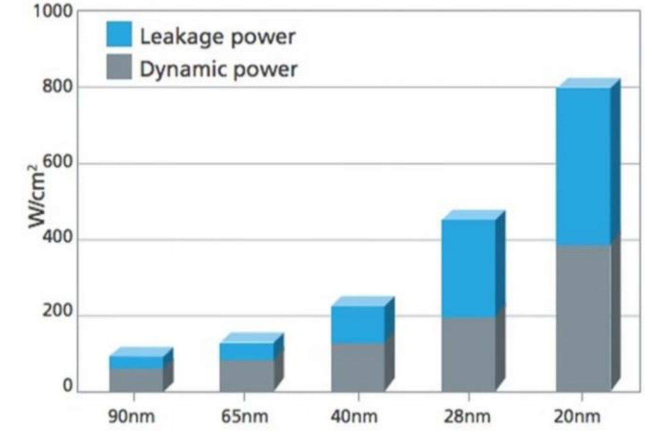
Outline

- ✓ **Power Dissipation**
  - Static Power Dissipation
  - Dynamic Power Dissipation
- ✓ Low Power Design Introduction
- ✓ Static Power Reduction
  - Multi-Threshold Voltage
- ✓ Dynamic Power Reduction
  - Multi-Voltage
  - Power Gating
  - RTL and Architecture Design Techniques
  - Clock gating
- ✓ Reference

Power Dissipation

- ✓ $P_{total} = P_{static} + P_{dynamic}$
- ✓ Static power (leakage power)
  - $P_{static} = I_{leakage} * Vdd$
    - $I_{leakage}$: leakage current
  - Sub-threshold current
  - Reverse leakage current
  - Gate Leakage current
- ✓ Dynamic power
  - $P_{dynamic} = p_t * (P_{sw} + P_{sc})$
    - p_t: Switching Probability of one clock cycle
  - Switching power (P_{sw})
    - Charging and discharging parasitic capacitance
  - Short circuit power (P_{sc})
    - Direct path between Vdd and GND when switching

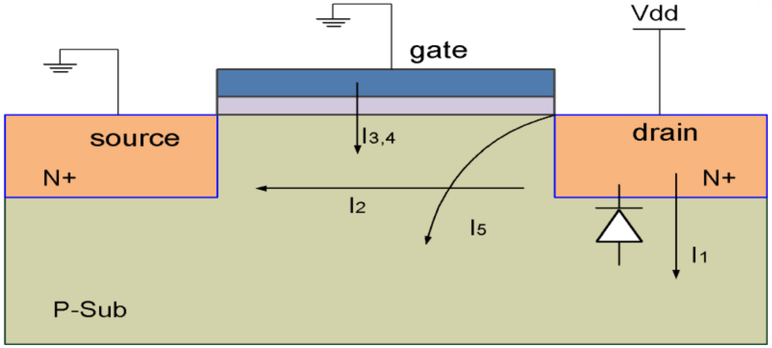




Static Power Dissipation

✓ **Caused by leakage current**

- **Sub-threshold current (I_2)**
  - Dominate the leakage current
  - Increase with temperature
- **Reverse leakage current (I_1)**
  - Happen when device is reverse biased
  - Increase with temperature
- **Gate Leakage current (I_3)**
  - Increase with Vdd



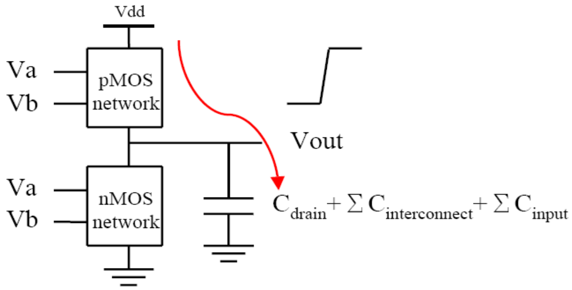
I_1-Reverse Bias p-n Junction Leakage
I_2-Sub Threshold/ Weak Inversion Current
I_3-Gate Leakage, Tunneling Current through Oxide
I_4-Gate Current due to hot carrier Injection
I_5-Gate Induced Drain Leakage(GIDL)

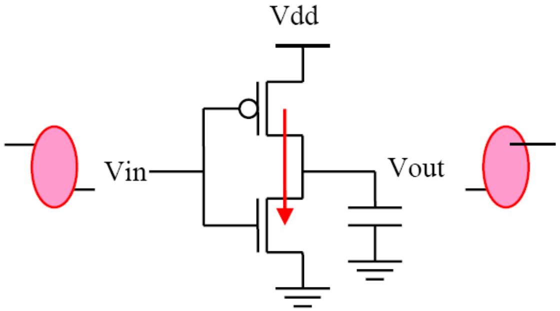
✓ **How to minimize static power?**

- Reduce voltage supply ***(Process dependent)***
- In general: the leakage current can’t be changed if process and cells are decided

Ref: https://allthingsvlsi.wordpress.com/2013/05/01/leakage-currents-in-mosfets/

Dynamic Power Dissipation





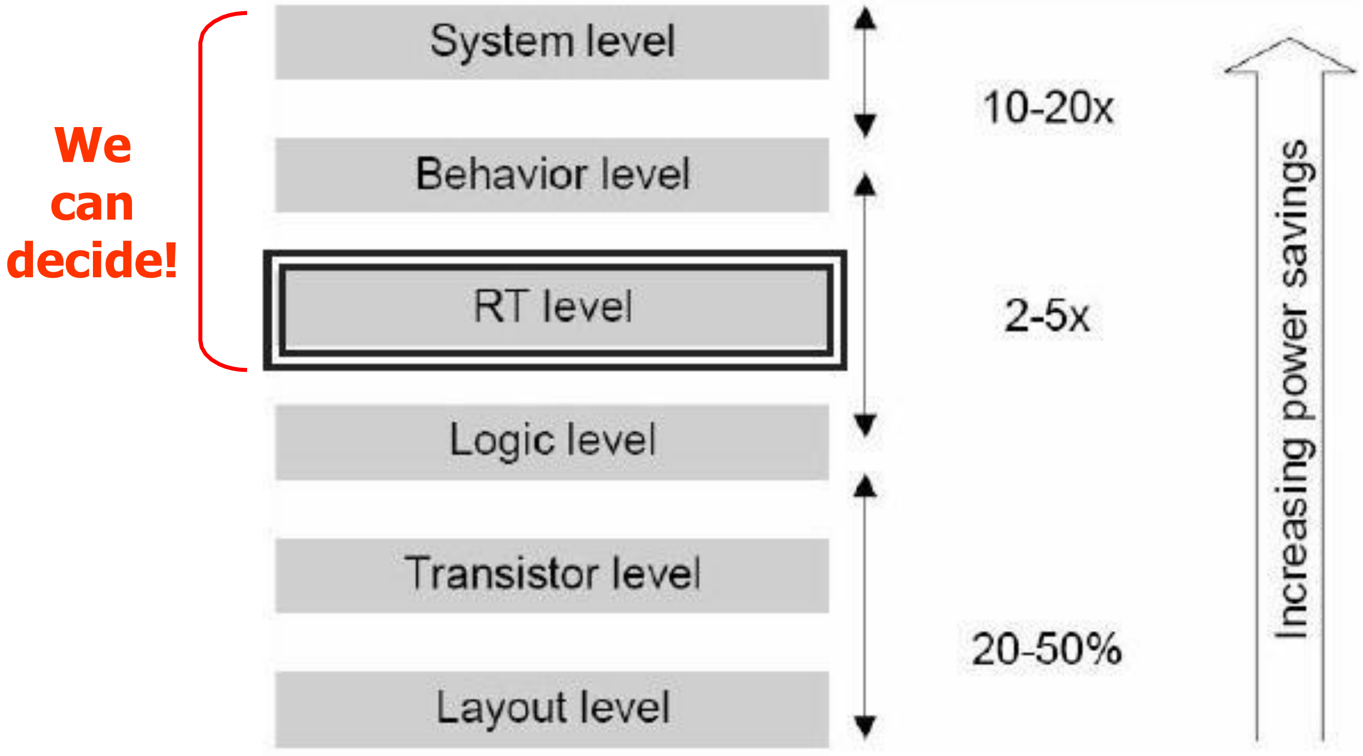
- ✓ $P_{dynamic} = p_t * (P_{sw} + P_{sc})$
  - p_t: Switching Probability
  - P_{sw}: Switching Power
    - $P_{sw} = C_L * Vdd^2 * f_p$
    - Charge and discharge the loading capacitive
  - P_{sc}: Short circuit power
    - Both NMOS and PMOS are turned on
- ✓ **How to minimize dynamic power?**
  - Reduce Vdd *(process dependent)* → P_{sw} & P_{sc}
  - Reduce parasitic capacitance → P_{sw}
  - Reduce the overlap time of PMOS and NMOS turn-on time, i.e., keep the input signal rise/fall time the same → P_{sc}
  - Reduce unnecessary switching activities → p_t
    - Gated clock (Turn off unused circuits)
    - Register retiming

Outline

- ✓ Power Dissipation
  - Static Power Dissipation
  - Dynamic Power Dissipation
- ✓ **Low Power Design Introduction**
- ✓ Static Power Reduction
  - Multi-Threshold Voltage
- ✓ Dynamic Power Reduction
  - Multi-Voltage
  - Power Gating
  - RTL and Architecture Design Techniques
  - Clock gating
- ✓ Reference

Low Power Design Mehodologies

✓ **Where to reduce?**



REF: H. Mizas, D. Soudris, S. Theoharis, G. Theodoridis, A. Thanailakis, and C.E. Goutis, "Structure of the Low-Power Design Flow, " 25/6/1998.

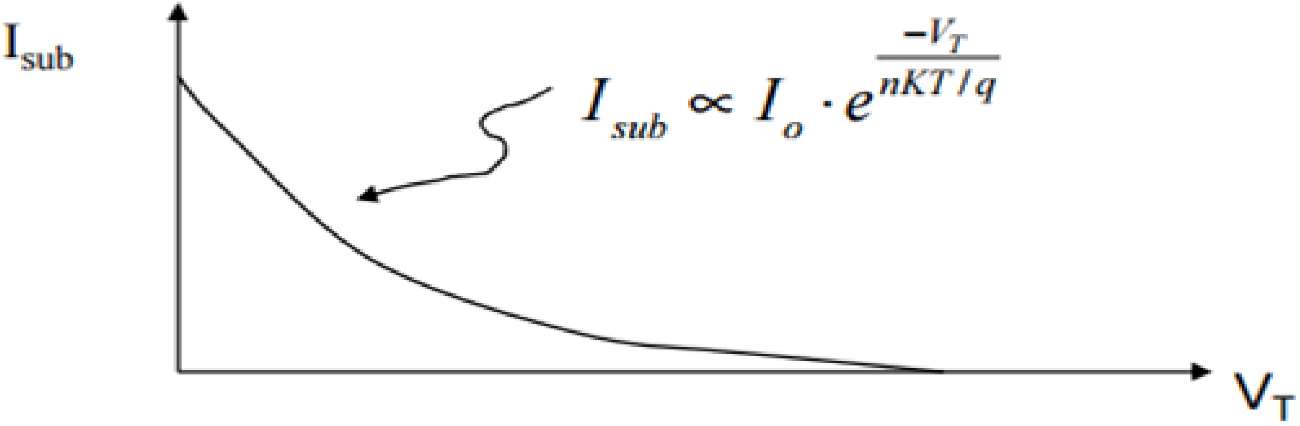
Outline

- ✓ Power Dissipation
  - Static Power Dissipation
  - Dynamic Power Dissipation
- ✓ Low Power Design Introduction
- ✓ **Static Power Reduction**
  - Multi-Threshold Voltage
- ✓ Dynamic Power Reduction
  - Multi-Voltage
  - Power Gating
  - RTL and Architecture Design Techniques
  - Clock gating
- ✓ Reference

Multi-Threshold Voltage (Multi-Vt)

✓ **Current of MOS**

$$I_{sub} = I_s \cdot e^{\frac{q(V_{GS} - V_T - V_{offset})}{nKT}} (1 - e^{\frac{-qV_{DS}}{KT}}) \qquad P_{static} \approx I_{sub} V_{DD}$$

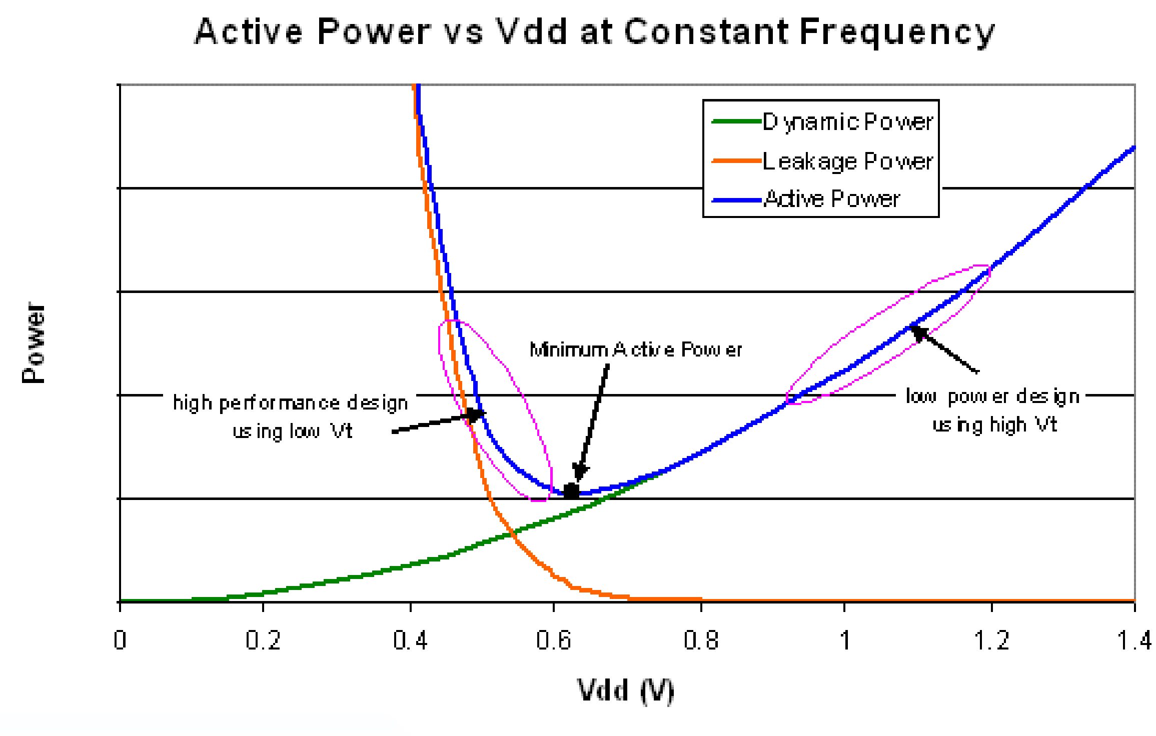


$$I_{sub} \propto I_o \cdot e^{\frac{-V_T}{nKT/q}}$$

V_T ↓, I_{sub} ↑, I_{ds} ↑, speed ↑, delay ↓

Vt Scaling

- ✓ **As Vt decrease, sub-threshold leakage increases**
  - Approaching 10% in 0.18 um technology
  - V_t ↓, I_{DS_sub} ↑, $I_{D(SAT)}$ ↑



The x-axis represents the supply voltage required maintain the device at a fixed frequency, while allowing the threshold voltage to vary to meet the fixed frequency specification.

Ref: http://www.design-reuse.com/articles/20296/power-management-leakage-control-process-compensation.html

Vt Scaling

- ✓ When **Power** is a concern: **High Vt** logic gate
- ✓ When **Delay** is a concern: **Low Vt** logic gate

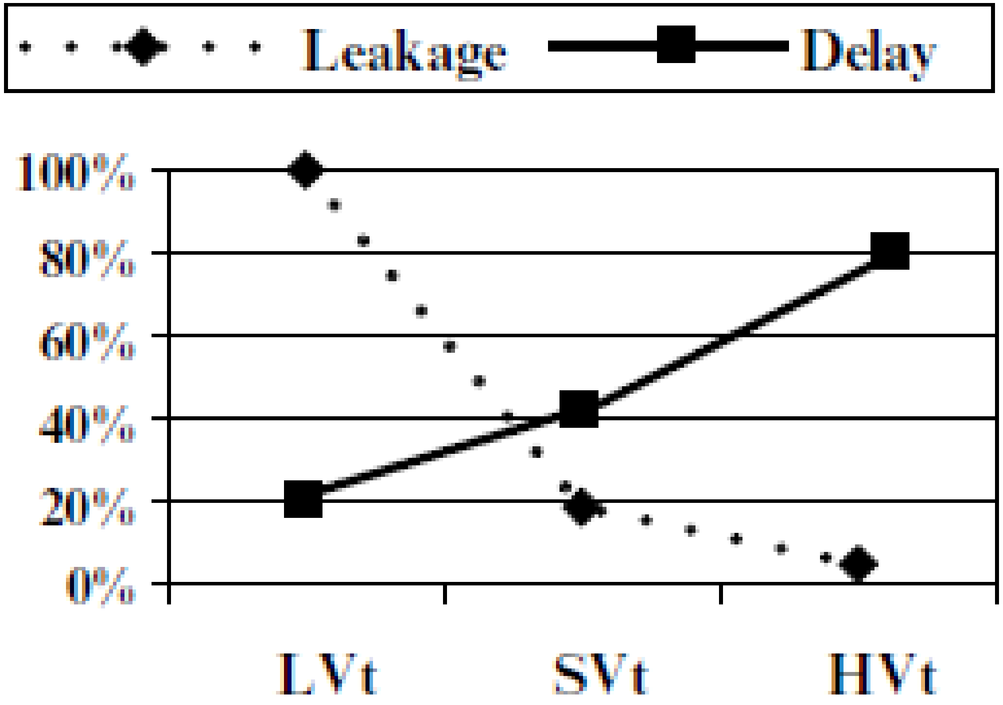


Figure 2-5 Leakage vs. Delay for a 90nm Library

Outline

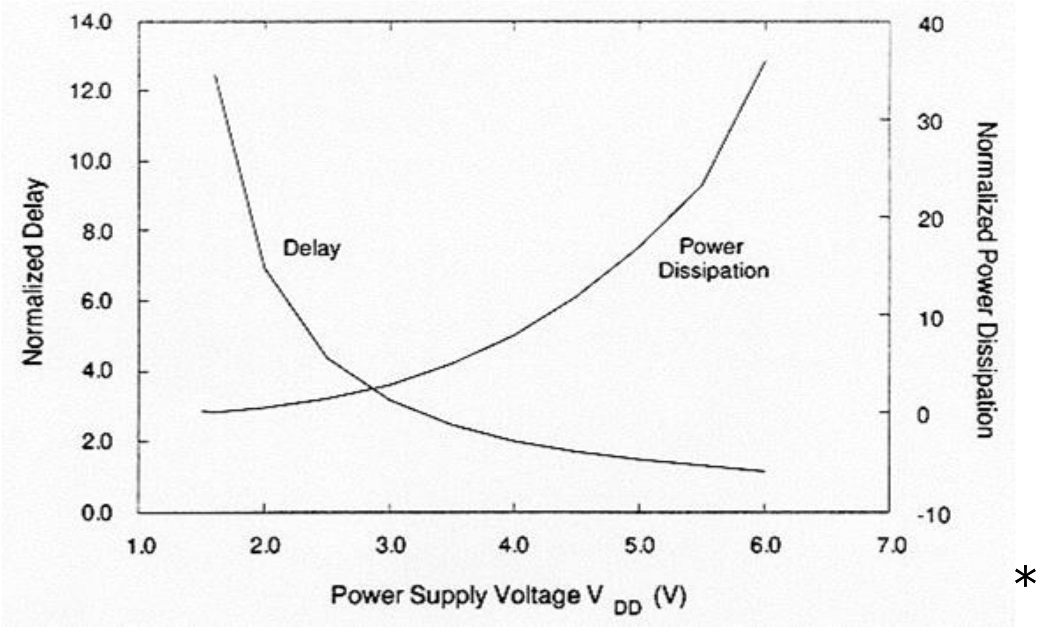
- ✓ Power Dissipation
  - Static Power Dissipation
  - Dynamic Power Dissipation
- ✓ Low Power Design Introduction
- ✓ Static Power Reduction
  - Multi-Threshold Voltage
- ✓ Dynamic Power Reduction
  - Multi-Voltage
  - Power Gating
  - RTL and Architecture Design Techniques
  - Clock gating
- ✓ Reference

Multi-Voltage

✓ Dynamic Power

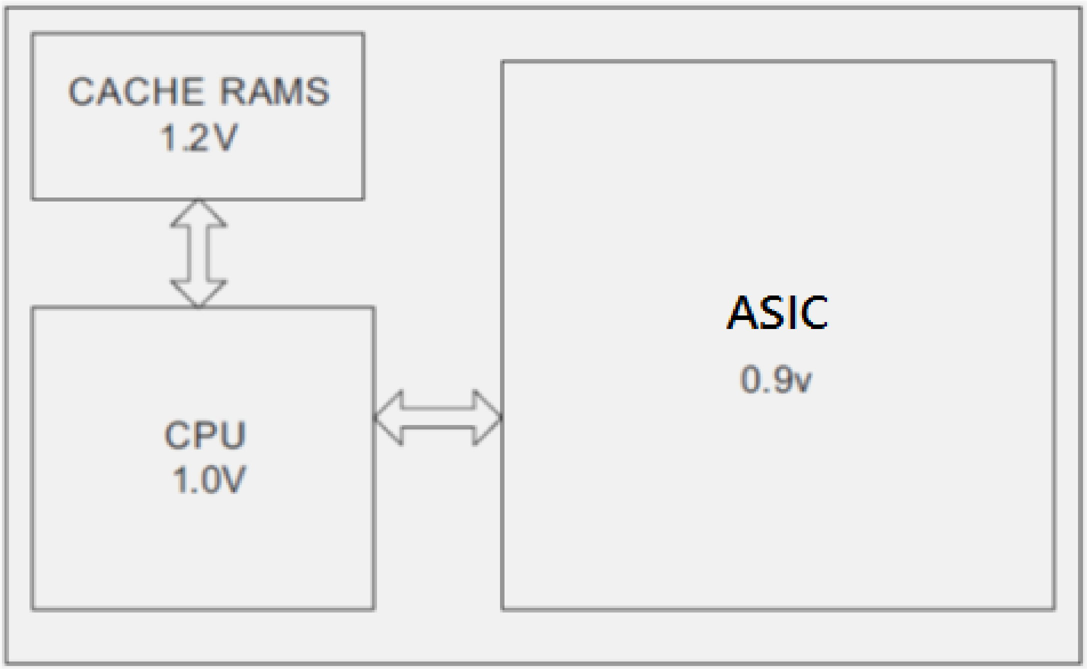
$$P_{dyn} = \underbrace{\left(C_{eff} \bullet V_{dd}^{2} \bullet f_{clock}\right)}_{\text{Switching power}} + \underbrace{\left(t_{sc} \bullet V_{dd} \bullet I_{peak} \bullet f_{clock}\right)}_{\text{Short circuit power}}$$

- Reducing the Vdd is the most straight forward way to reduce the power
- However, decreasing Vdd would cause the circuit delay increasing



*CMOS example

Multi-Voltage Example



- The cache RAMS are run at the highest voltage because they are on the critical timing path.
- The CPU is run at a high voltage because its performance determines system performance.
- The rest of the chip can run at a lower voltage still without impacting overall system performance.

Multi-Voltage Strategies

- ✓ **Static Voltage Scaling (SVS)**
  - Different blocks or subsystems are given different, fixed supply voltages

- ✓ **Multi-level Voltage Scaling (MVS)**
  - A block or subsystem is switched between tow or more voltage levels
  - Only a few, fixed, discrete level are supported for different operating modes

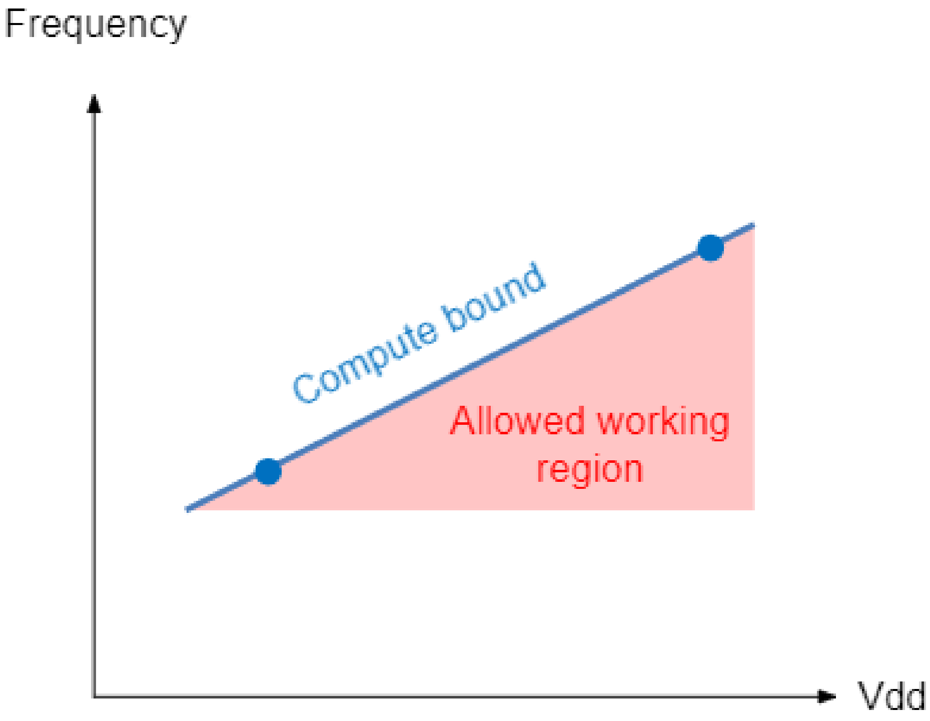
- ✓ **Dynamic Voltage and Frequency Scaling (DVFS)**
  - An extension of MVS where a larger number of voltages levels are dynamically switched to follow changing workloads

Ref: https://www.bbsmax.com/A/x9J2Rq1WJ6/

Multi-Voltage Strategies

✓ **Dynamic Voltage and Frequency Scaling (DVFS)**

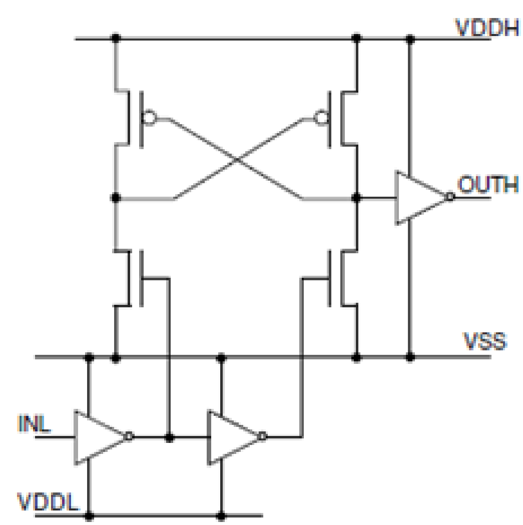
- **Compute bound in DVFS**
- Chips can only work at the region below compute bound
- DVFS must adjust Vdd first before raising frequency, and drop frequency first before turning down Vdd

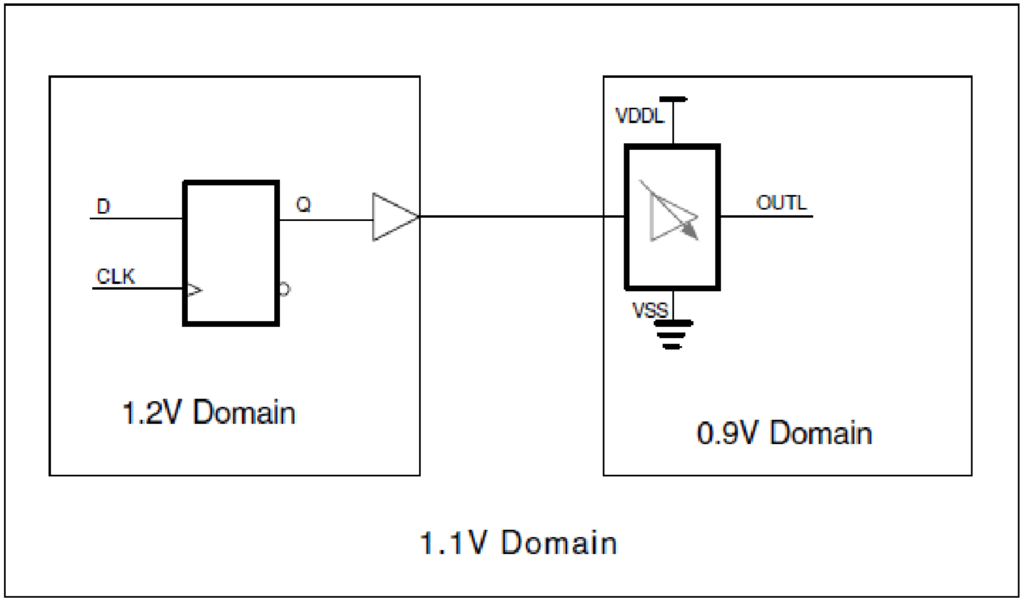


Ref: https://www.bbsmax.com/A/x9J2Rq1WJ6/

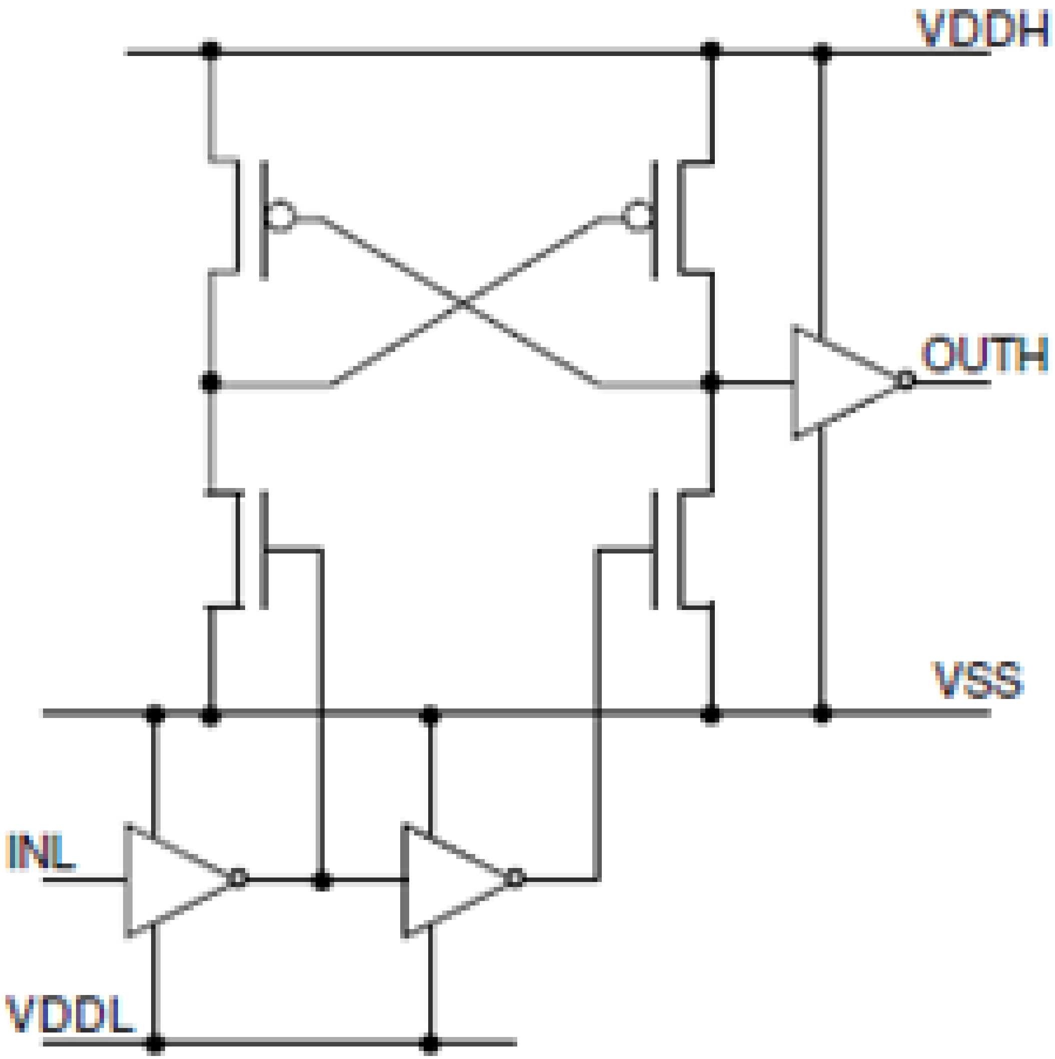
Level Shifter

- ✓ **Connecting 2 power domains at different voltage levels can cause design issues**
  - Timing inaccuracy
  - Signals are not propagated
- ✓ **A level shifter is required**
  - High to low
  - Low to High

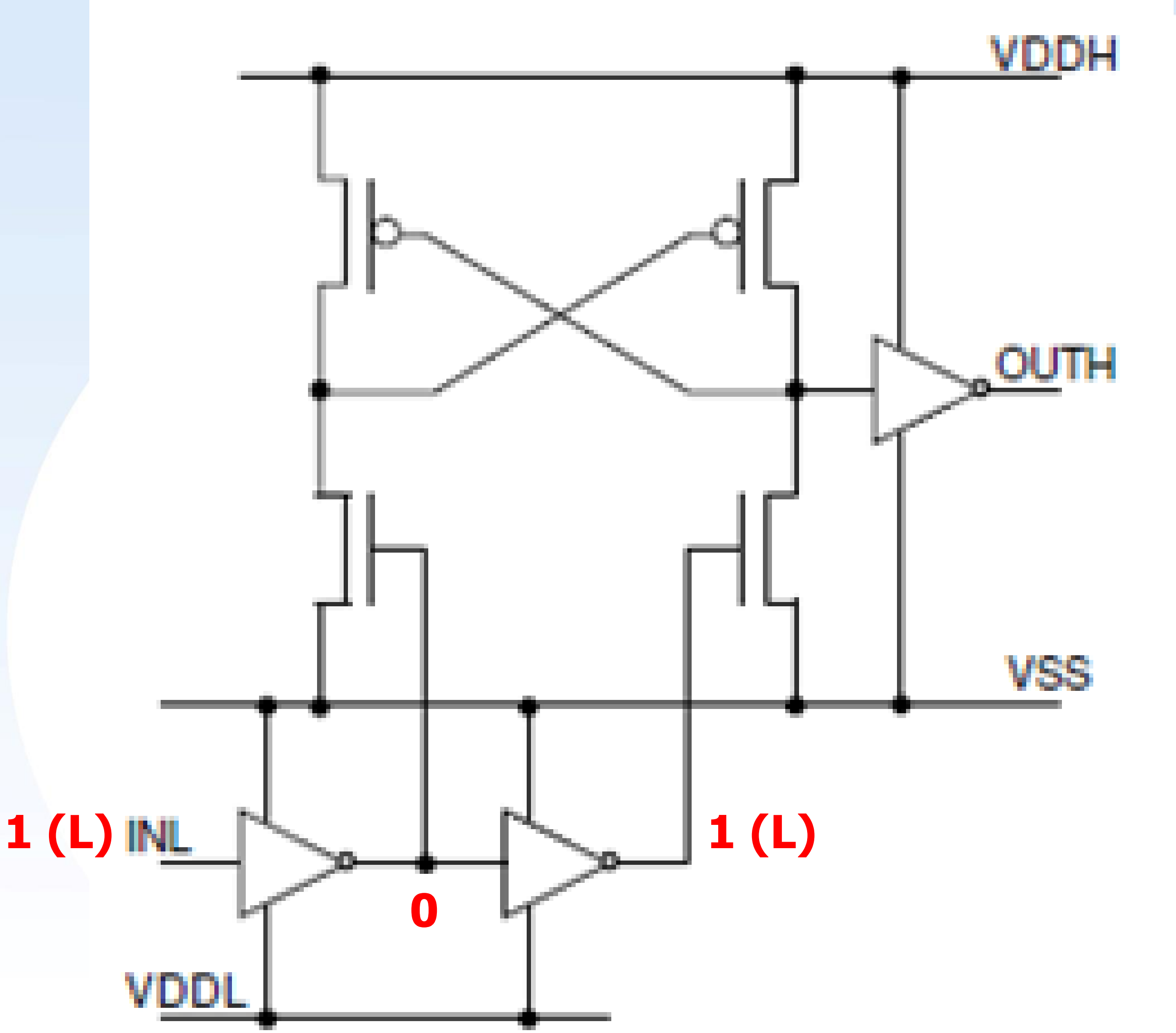




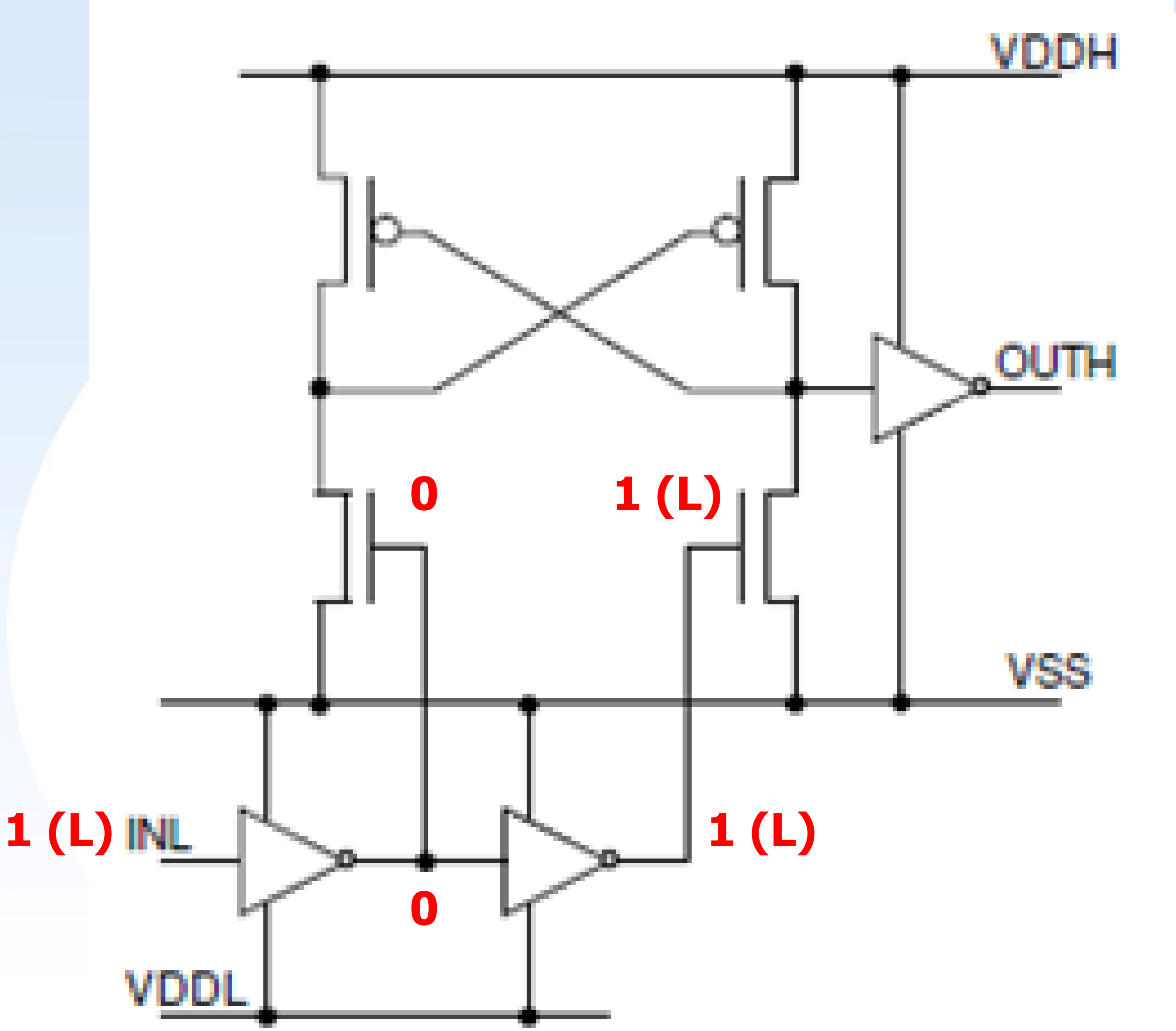
Level Shifter



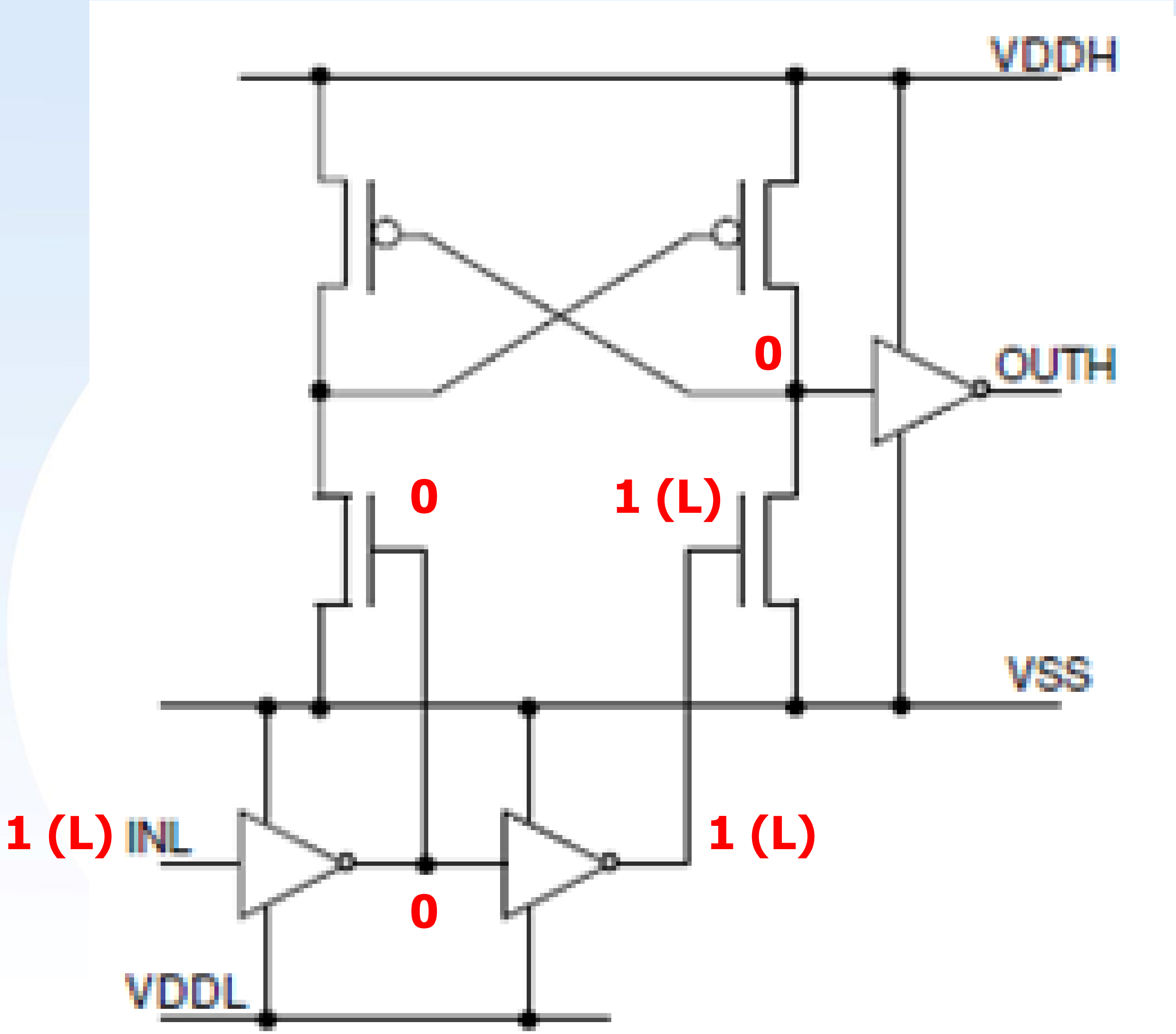
Level Shifter



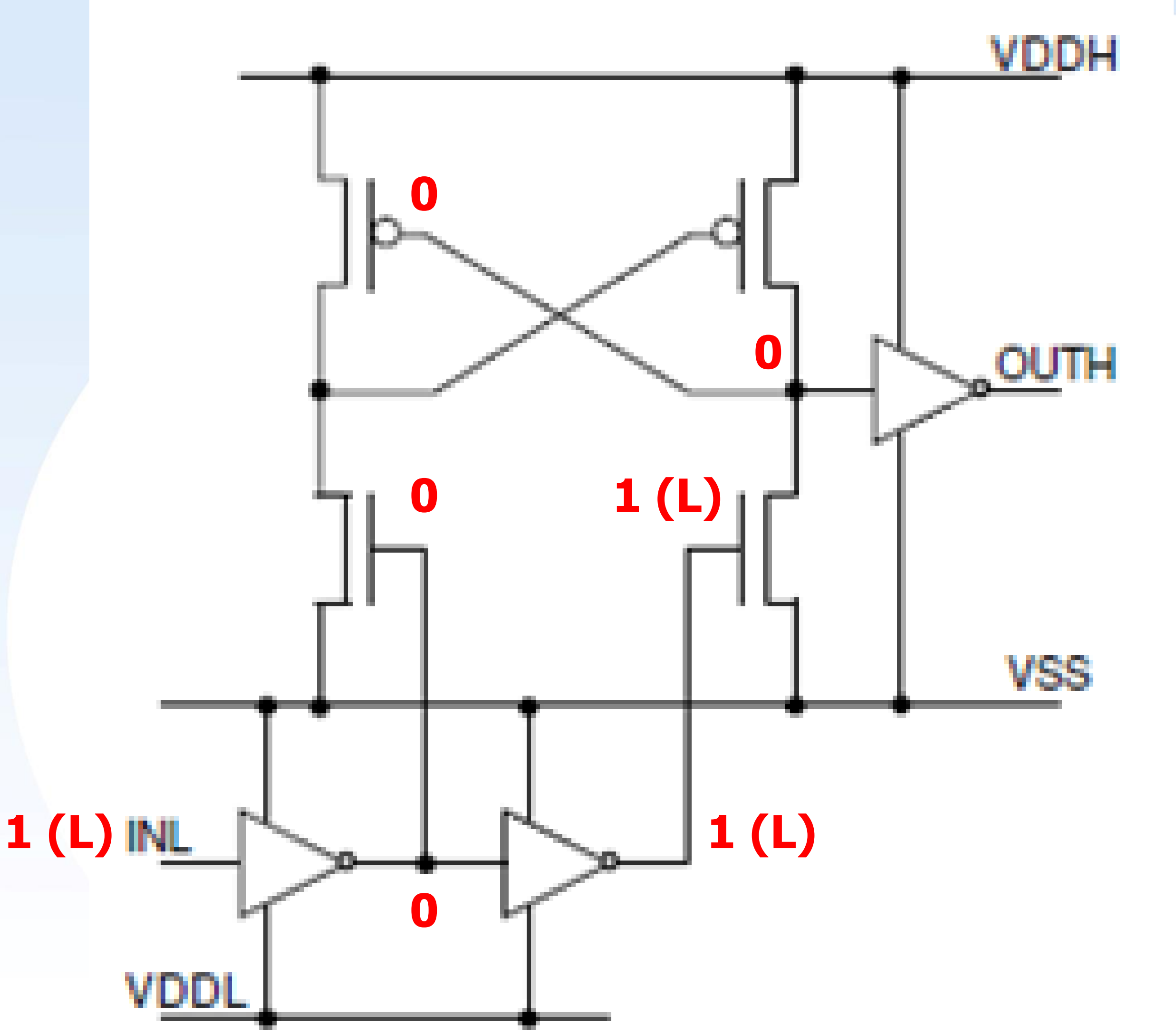
Level Shifter



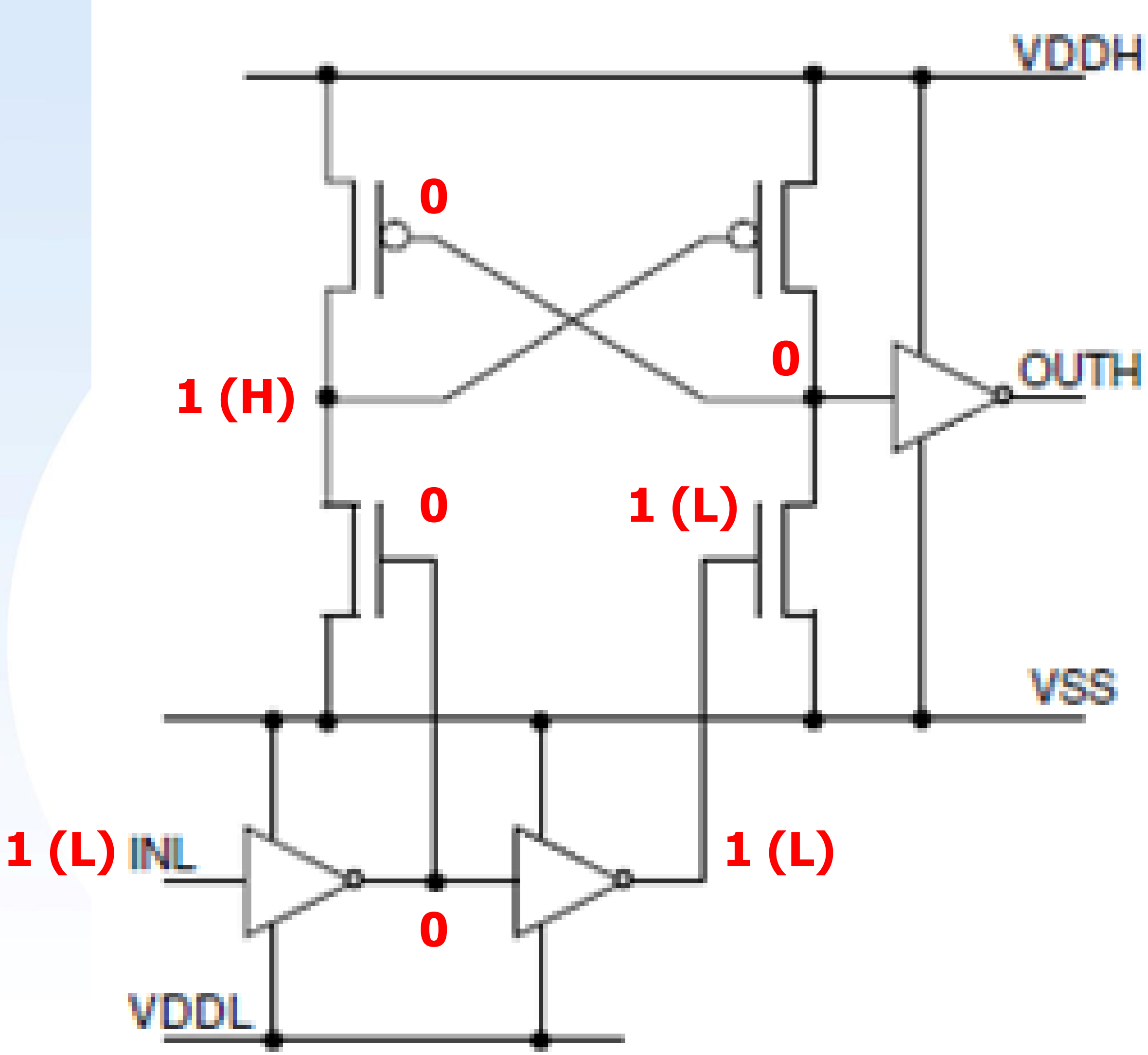
Level Shifter



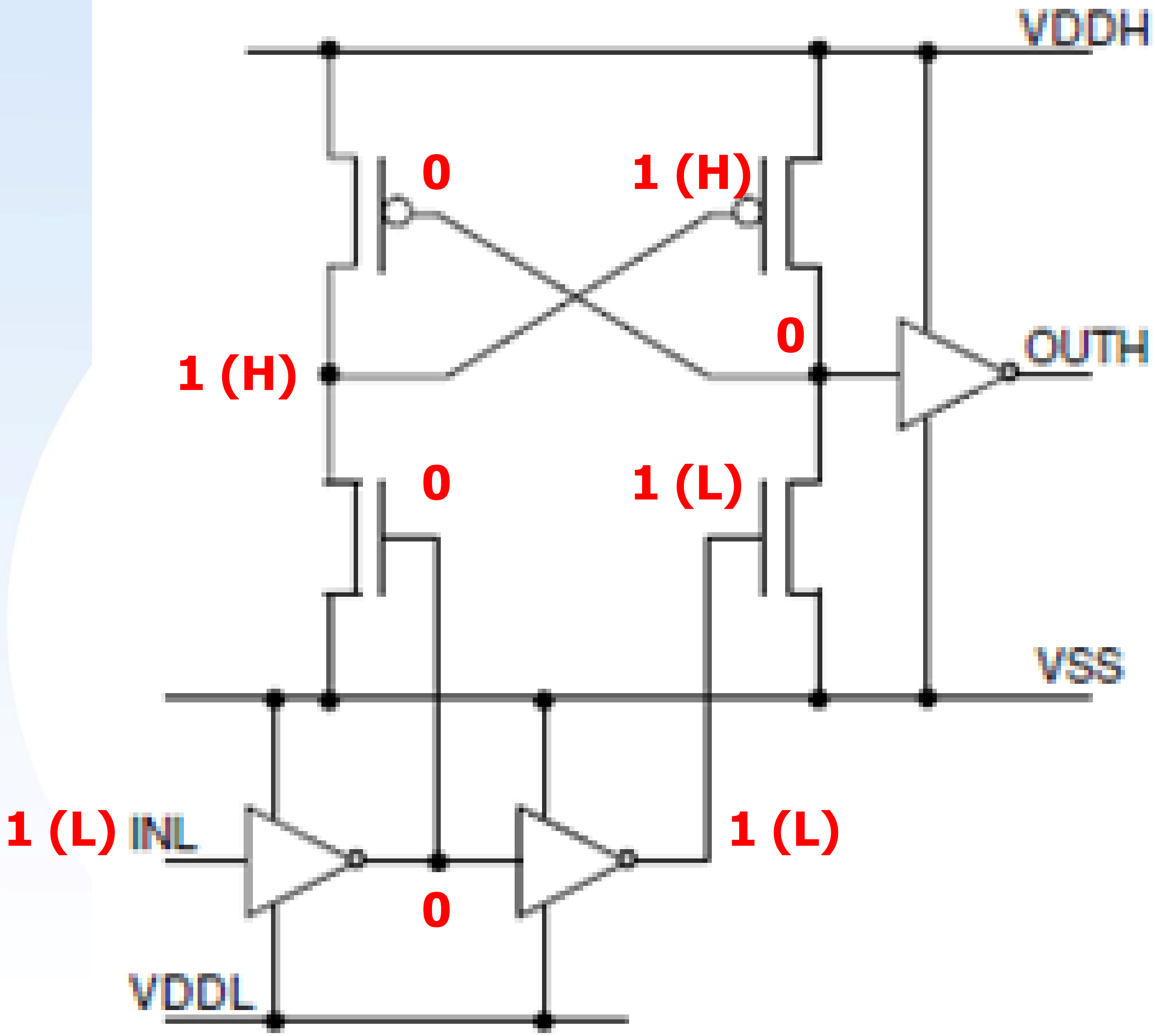
Level Shifter



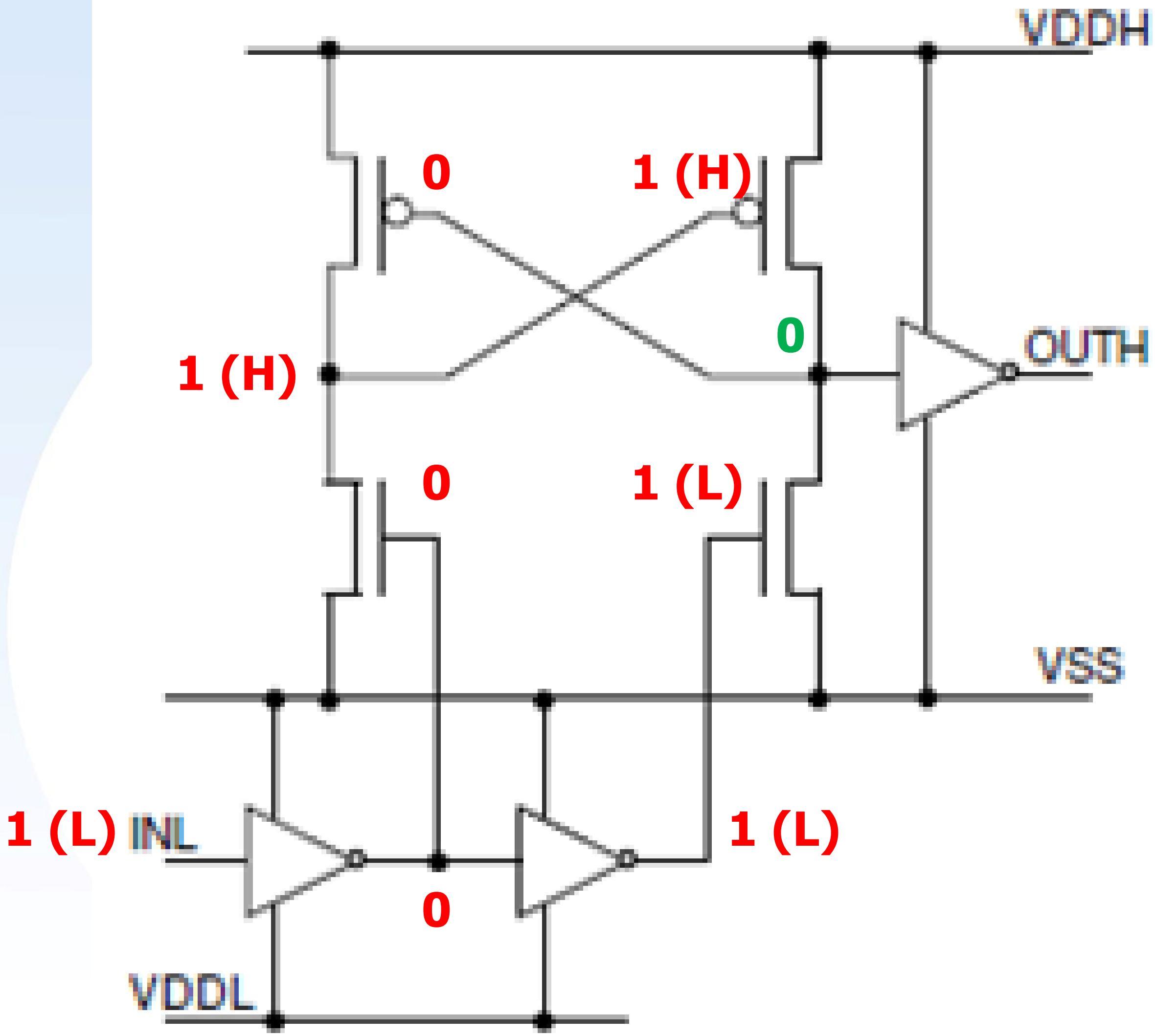
Level Shifter



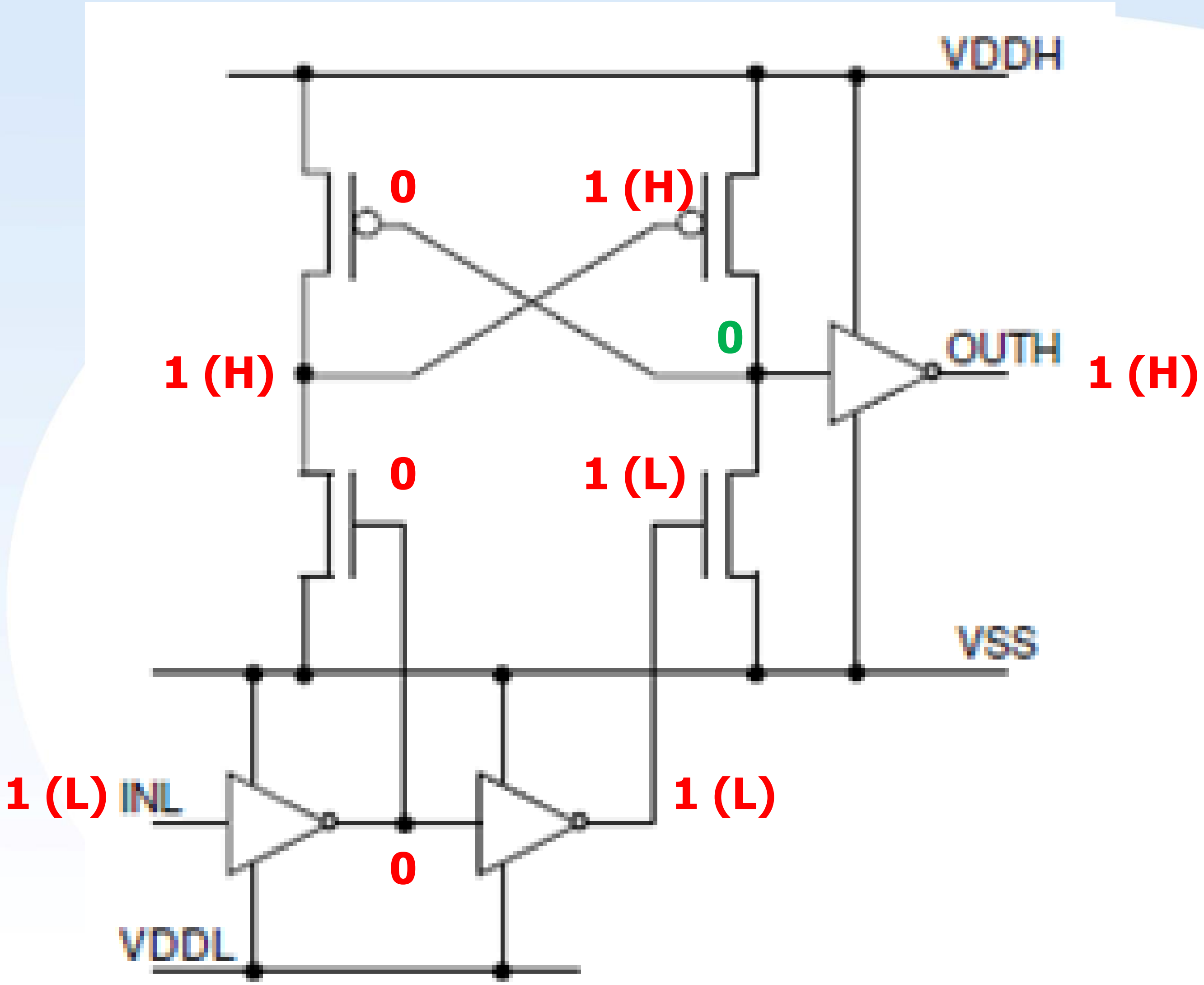
Level Shifter



Level Shifter



Level Shifter



Outline

- ✓ Power Dissipation
  - Static Power Dissipation
  - Dynamic Power Dissipation
- ✓ Low Power Design Introduction
- ✓ Static Power Reduction
  - Multi-Threshold Voltage
- ✓ **Dynamic Power Reduction**
  - Multi-Voltage
  - Power Gating
  - RTL and Architecture Design Techniques
  - Clock gating
- ✓ Reference

Power Gating

- ✓ The basic idea of power gating is to provide two power mode:
  - A low power mode and an active mode
- ✓ The goal is to switch between these mode to maximize power savings while minimizing the impact to performance
- ✓ Sleep mode (low power mode): shut down power to block of logic

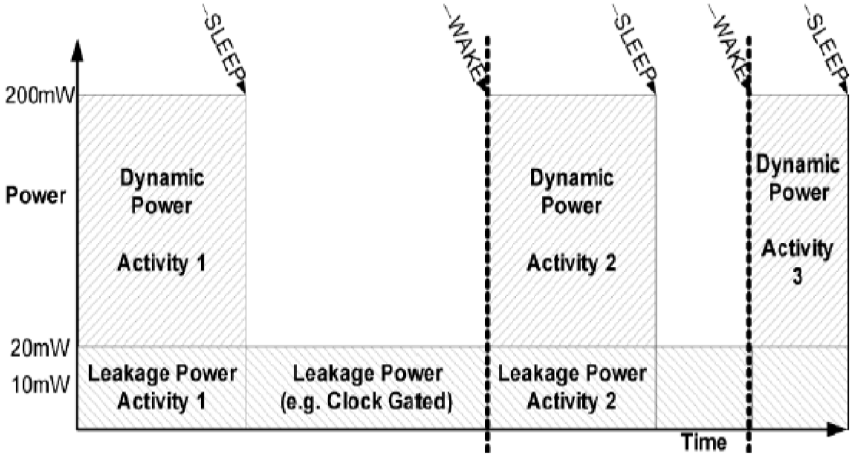


Figure 4-1 Activity Profile with No Power Gating

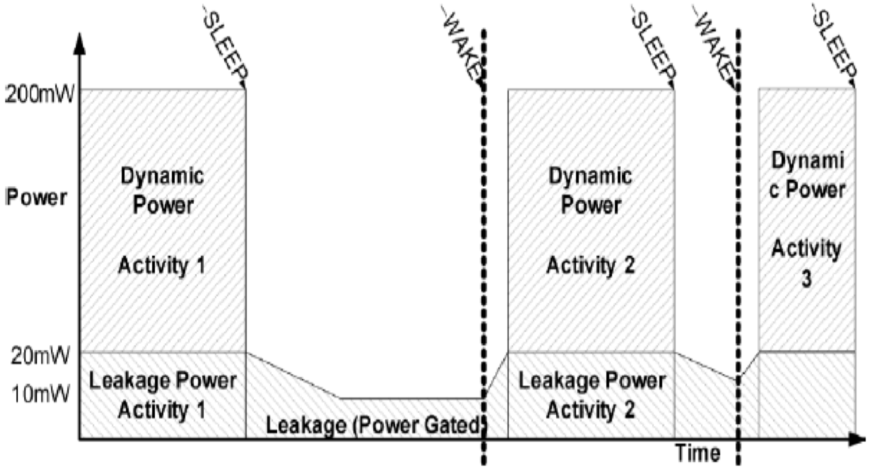
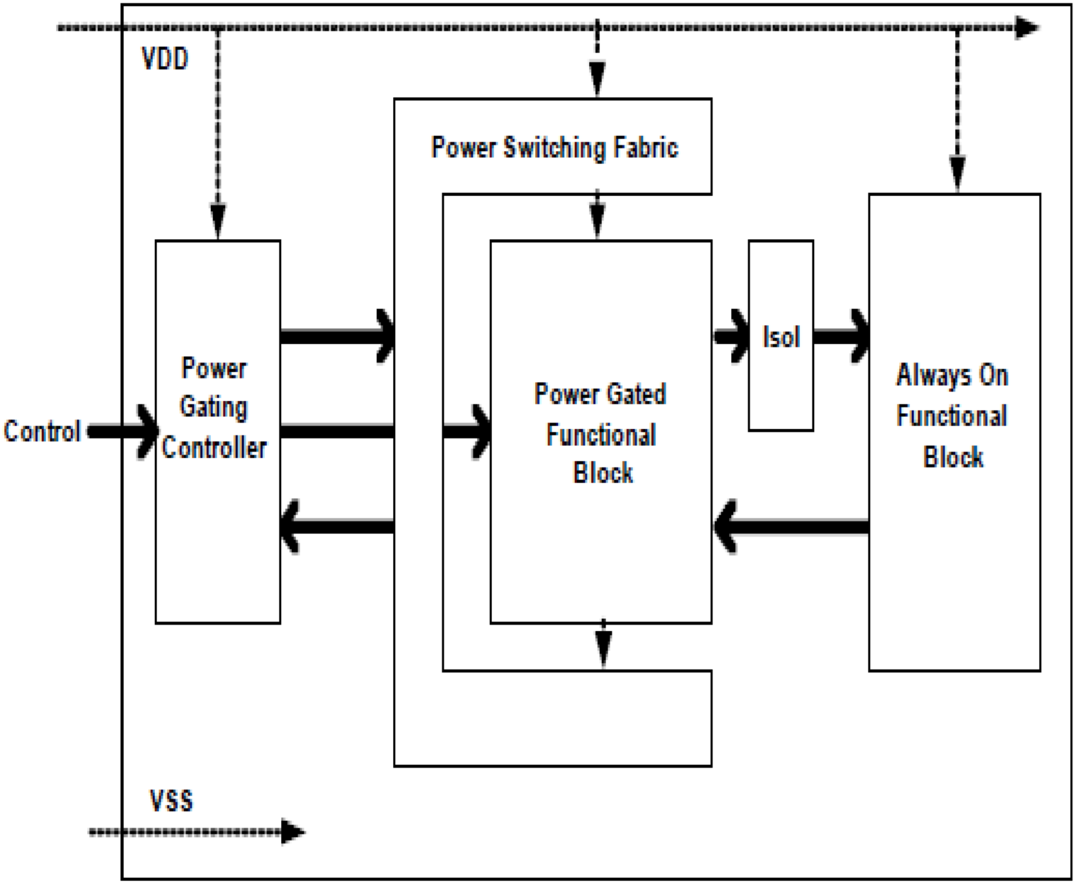
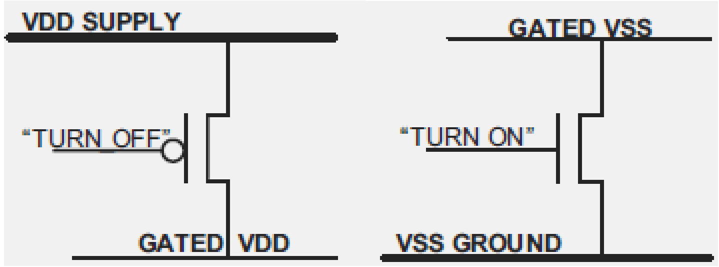


Figure 4-3 Realistic Profile with Power Gating

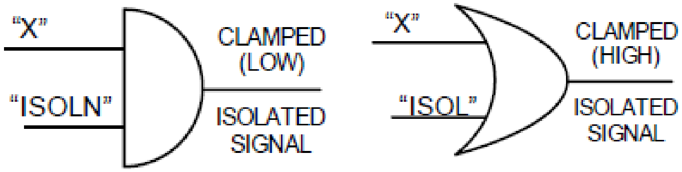
Power Gating Example



Power switching fabric



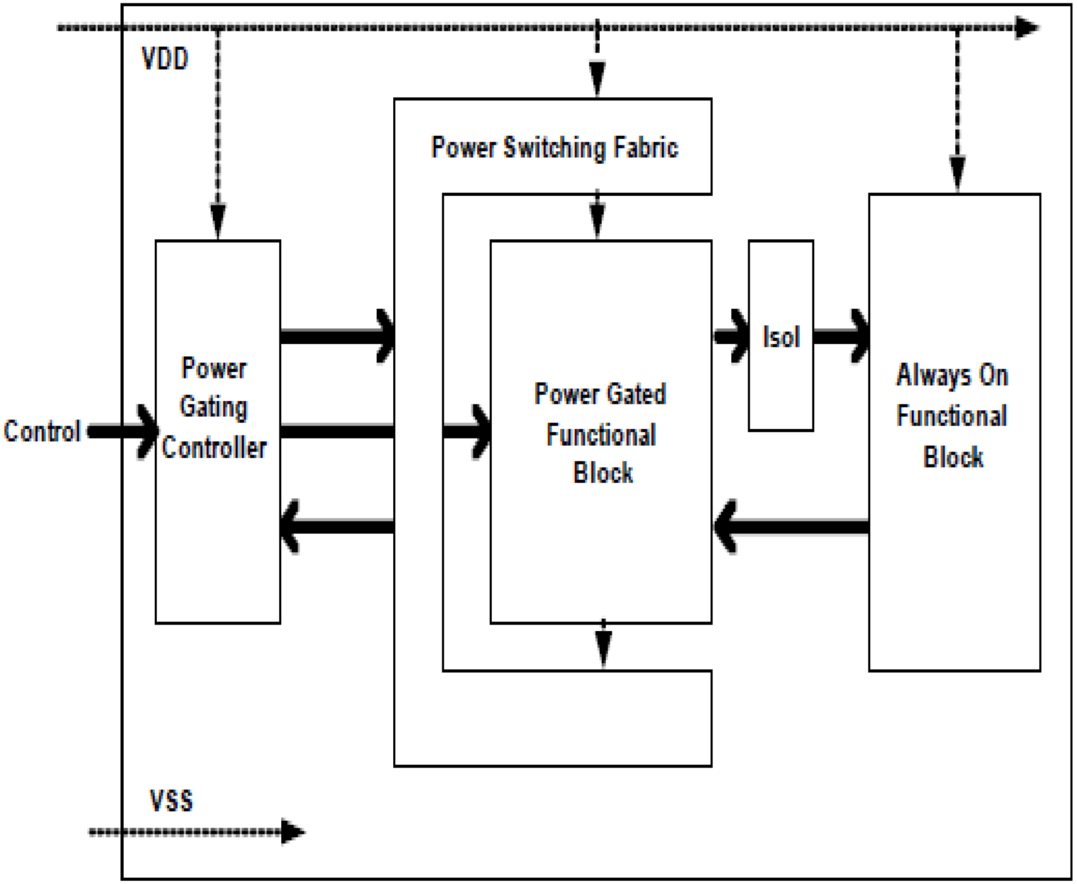
Isolation



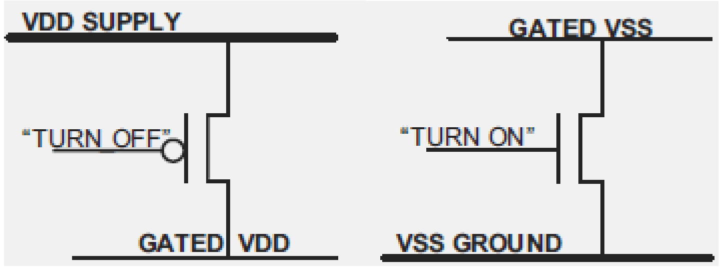
	0	1
0	0	0
1	0	1

	0	1
0	0	1
1	1	1

Power Gating Example

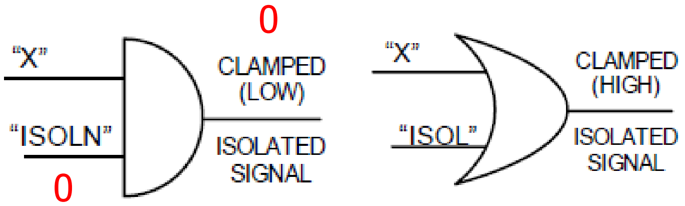


Power switching fabric



Isolation

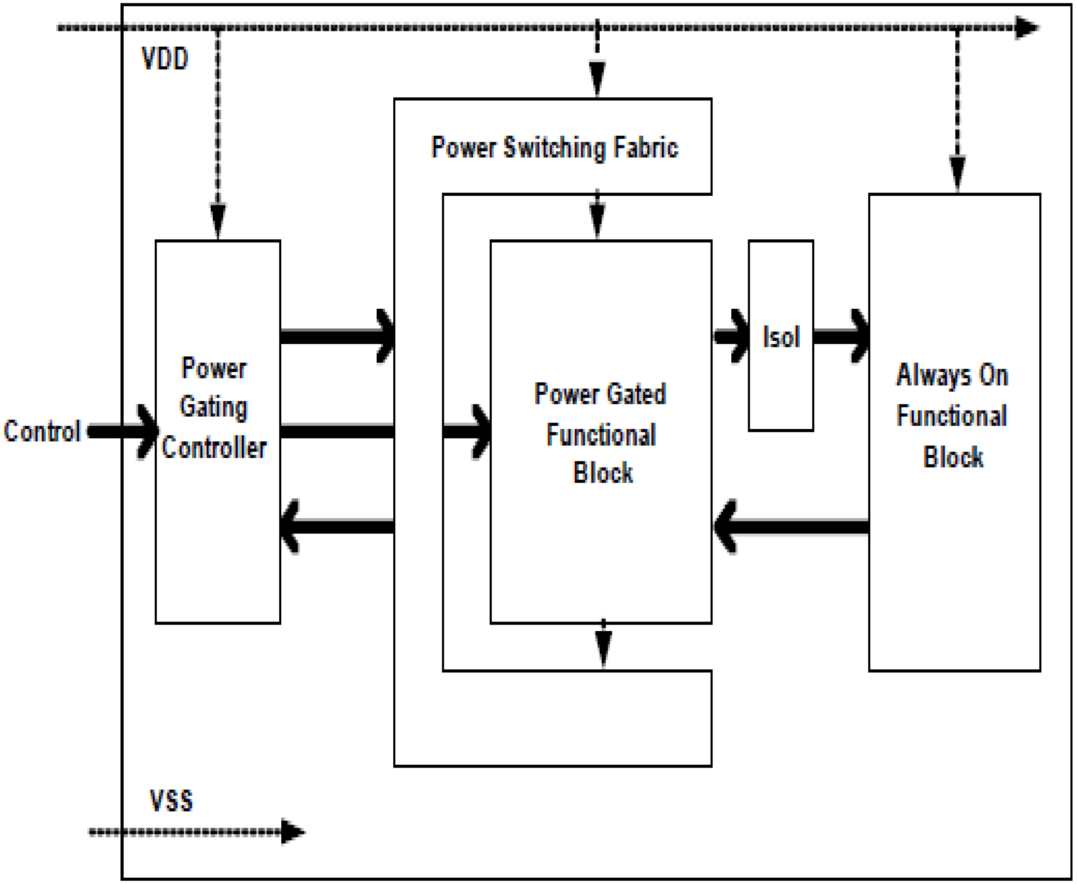
Low Clamped Isolated Signal



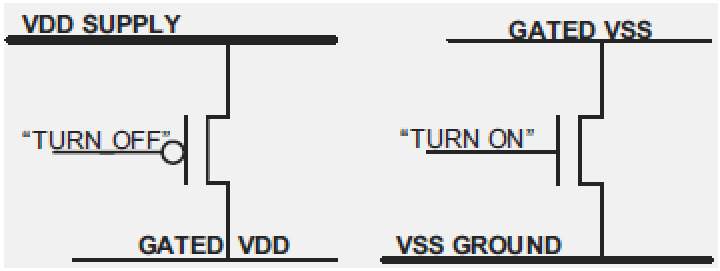
	0	1
0	0	0
1	0	1

	0	1
0	0	1
1	1	1

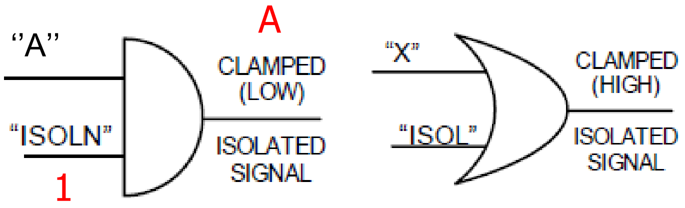
Power Gating Example



Power switching fabric



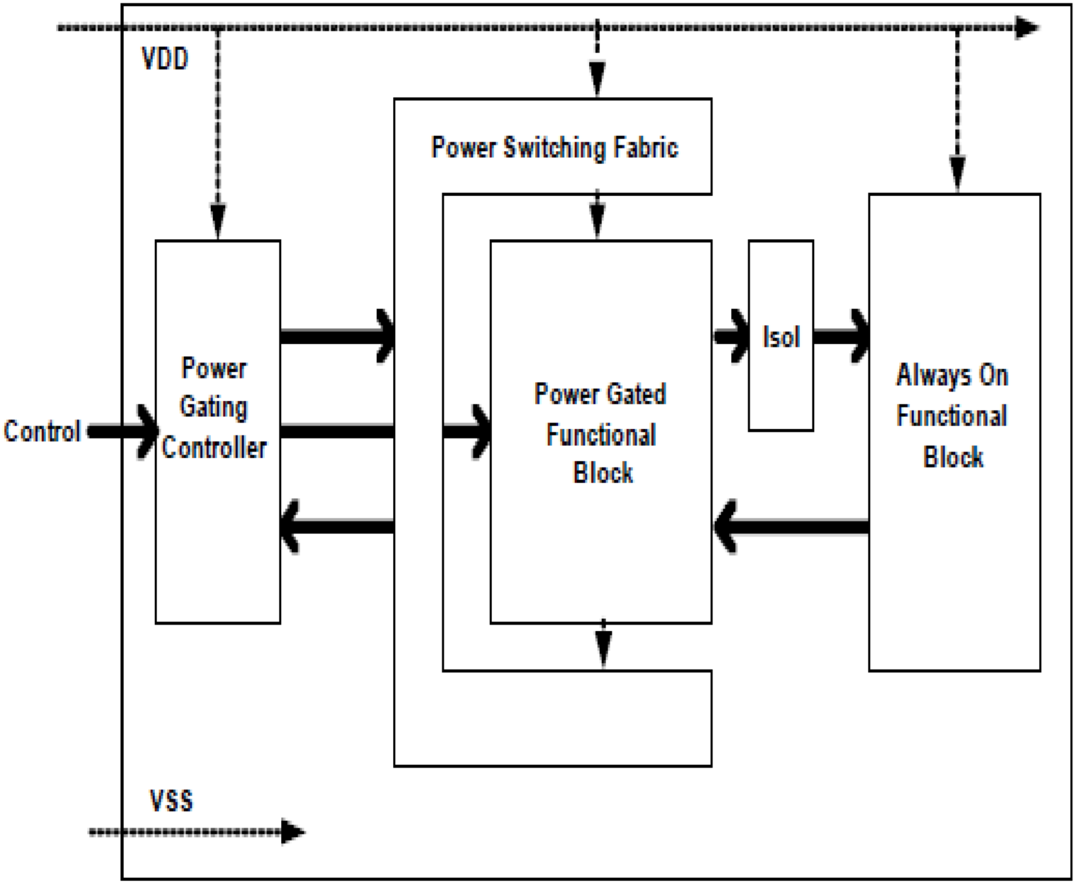
Isolation



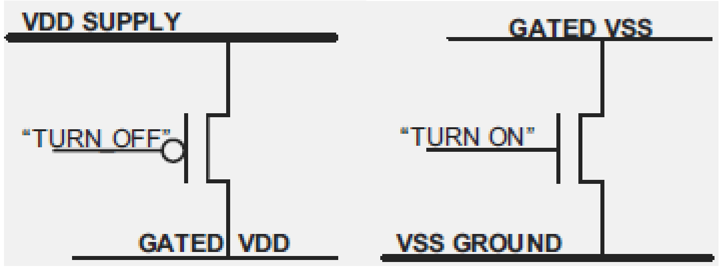
	0	1
0	0	0
1	0	1

	0	1
0	0	1
1	1	1

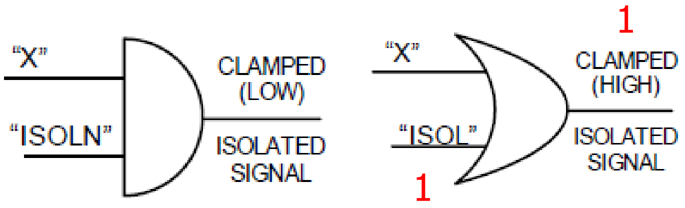
Power Gating Example



Power switching fabric



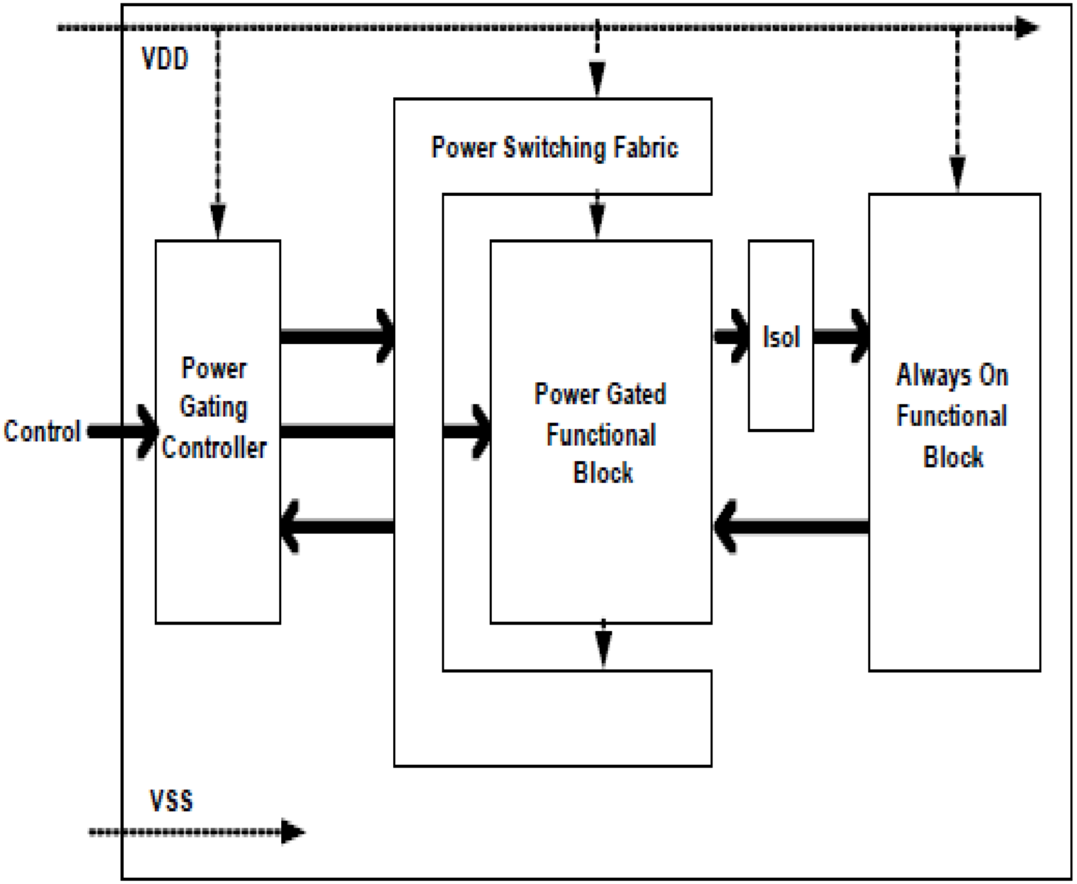
Isolation



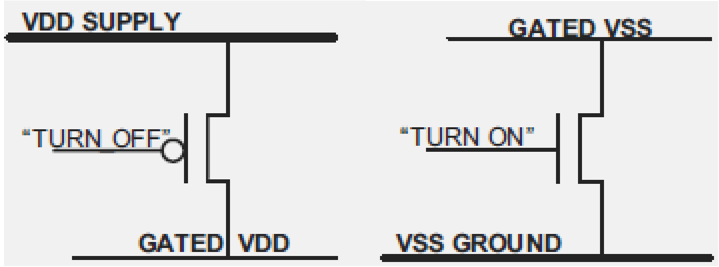
	0	1
0	0	0
1	0	1

	0	1
0	0	1
1	1	1

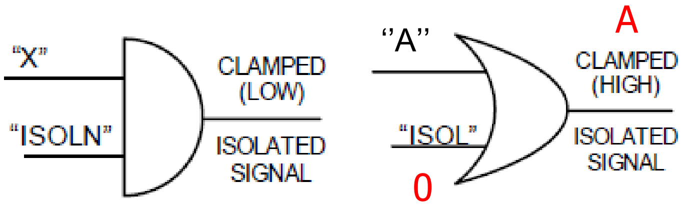
Power Gating Example



Power switching fabric



Isolation

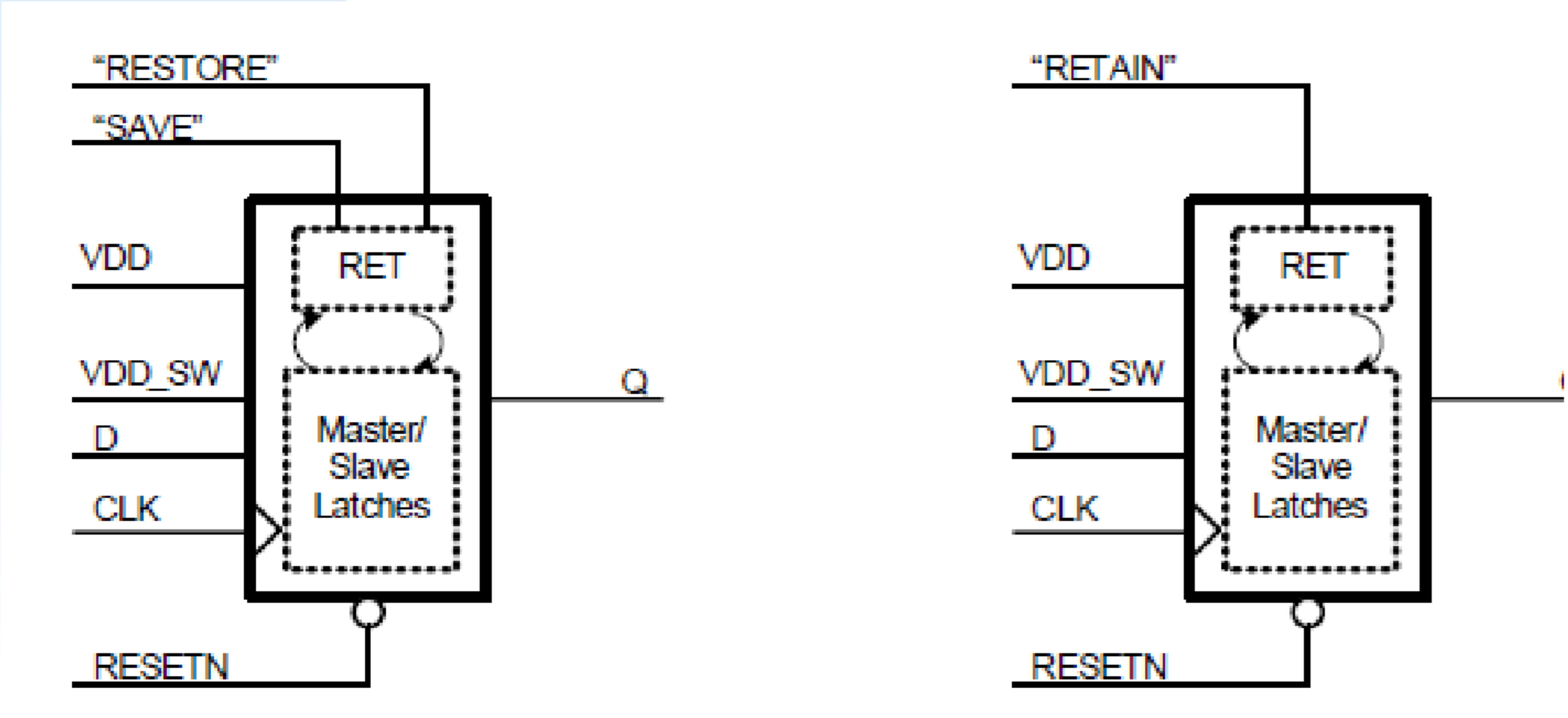


	0	1
0	0	0
1	0	1

	0	1
0	0	1
1	1	1

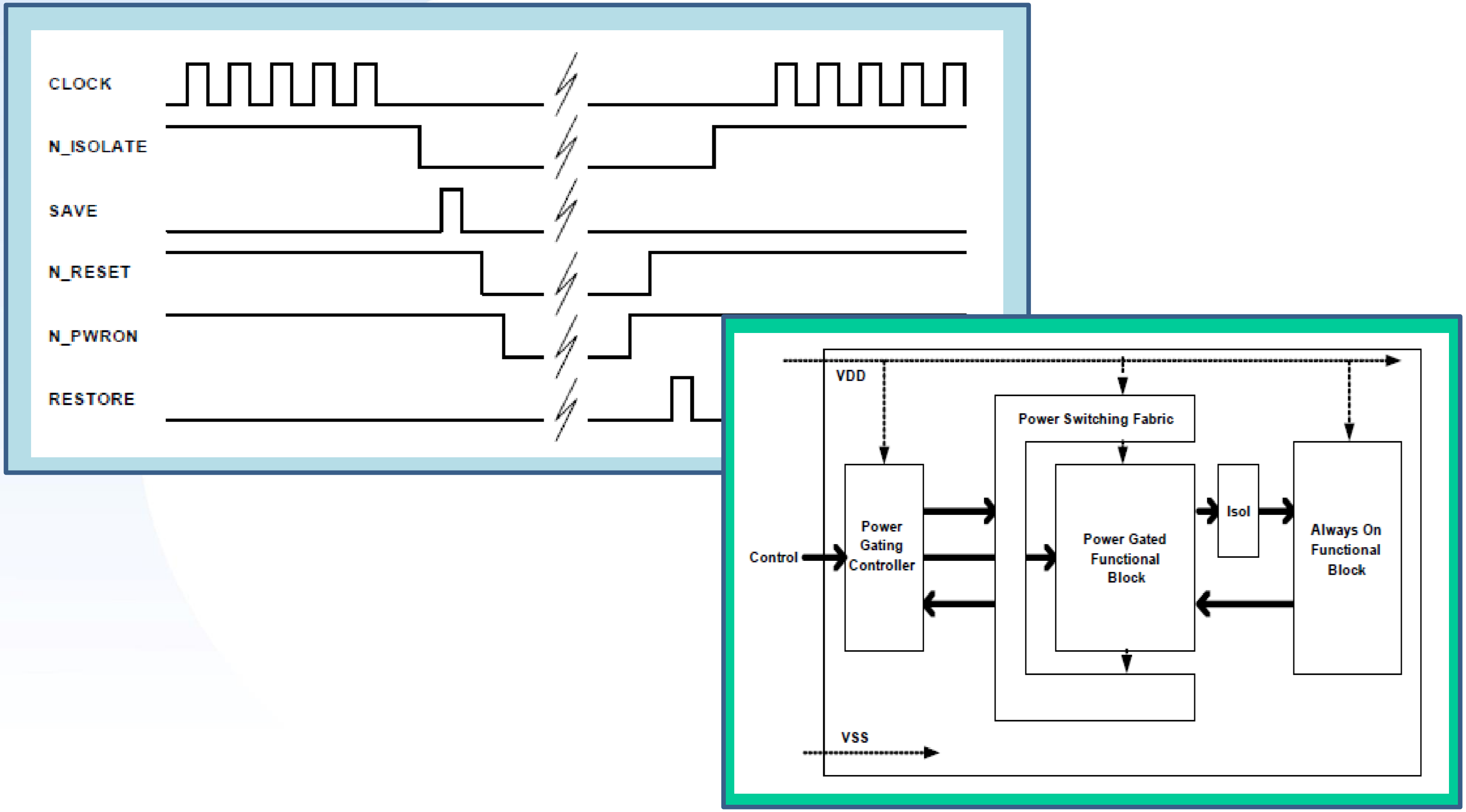
Retention registers

- ✓ States of some registers in shut-down mode need to be preserved
- ✓ Retention registers can **store data while in shut-down mode**



Power Control Sequencing

- ✓ Power gating needs a control circuit to schedule the whole procedure



Outline

- ✓ Power Dissipation
  - Static Power Dissipation
  - Dynamic Power Dissipation
- ✓ Low Power Design Introduction
- ✓ Static Power Reduction
  - Multi-Threshold Voltage
- ✓ **Dynamic Power Reduction**
  - Multi-Voltage
  - Power Gating
  - RTL and Architecture Design Techniques
  - Clock gating
- ✓ Reference

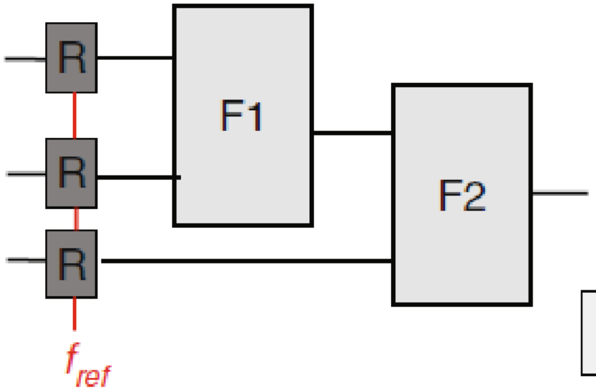
Architecture Trade-offs: Power / Frequency / Area

✓ **Dynamic Power**

$$P_{dyn} = \underbrace{\left(C_{eff} \bullet V_{dd}^2 \bullet f_{clock}\right)}_{\text{Switching power}} + \underbrace{\left(t_{sc} \bullet V_{dd} \bullet I_{peak} \bullet f_{clock}\right)}_{\text{Short circuit power}}$$

✓ **Trade-off clock frequency for area to reduce power**

Consider the following reference design



R: register,
F1, F2: combinational logic blocks
(adders, ALUs, etc.)

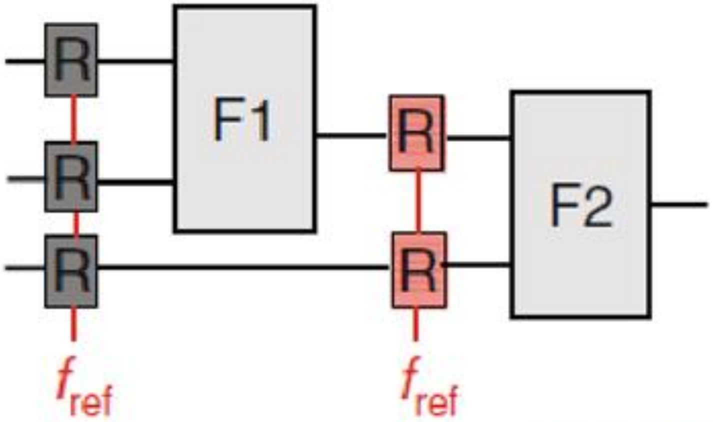
$$P_{ref} = C_{ref} \cdot V^2_{DDref} \cdot f_{ref}$$

C_{ref}: average switching capacitance

[A. Chandrakasan, JSSC'92]

Pipeline

- ✓ **Area ↑, Vdd↓**



$$f_{pipe} = f_{ref}$$
$$V_{DDpipe} = \varepsilon_{pipe} \cdot V_{DD\ ref}$$
$$C_{pipe} = (1+ov_{pipe}) \cdot C_{ref}$$

$$P_{pipe} = \varepsilon^2_{pipe} \cdot (1+ov_{pipe}) \cdot P_{ref}$$

Shallower logic reduces required supply voltage

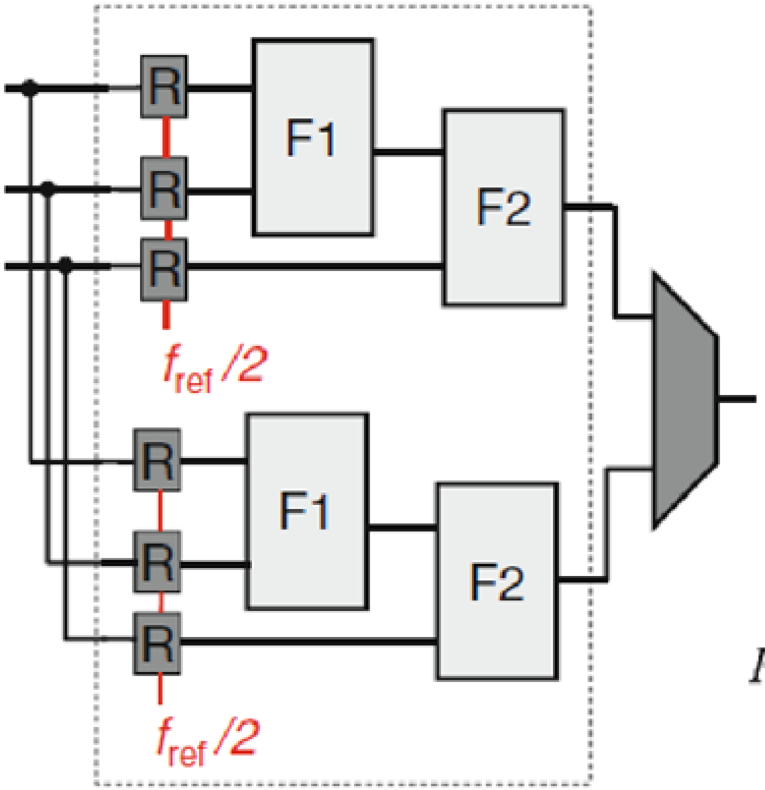
(this example assumes equal V_{DD} for par / pipe designs)

Assuming $ov_{pipe} = 10\%$

$$P_{pipe} = 0.66^2 \cdot 1.1 \cdot P_{ref} = 0.48P_{ref}$$

Parallel

- ✓ **Area↑, Frequency↑, Vdd↓**



$$f_{par} = f_{ref}/2$$
$$V_{DD\ par} = \epsilon_{par} \cdot V_{DD\ ref}$$
$$C_{par} = (2 + ov_{par}) \cdot C_{ref}$$

Almost cancels

$$P_{par} = \epsilon^2_{par} \cdot \left(\frac{2 + ov_{par}}{2}\right) \cdot P_{ref}$$

Assuming
$ov_{par} = 15\%$

$$P_{par} = 0.66^2 \cdot \frac{2.15}{2} \cdot P_{ref} = 0.47 P_{ref}$$

Outline

- ✓ Power Dissipation
  - Static Power Dissipation
  - Dynamic Power Dissipation
- ✓ Low Power Design Introduction
- ✓ Static Power Reduction
  - Multi-Threshold Voltage
- ✓ **Dynamic Power Reduction**
  - Multi-Voltage
  - Power Gating
  - RTL and Architecture Design Techniques
  - Clock gating
- ✓ Reference

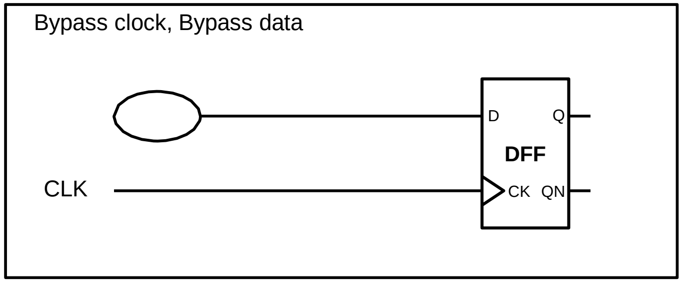
Dynamic Clock Gating

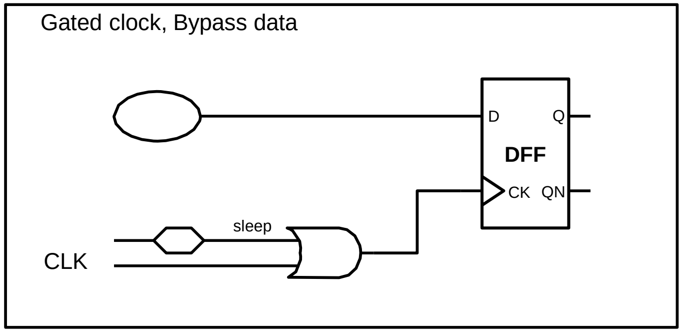
✓ Why do we proceed clock gating

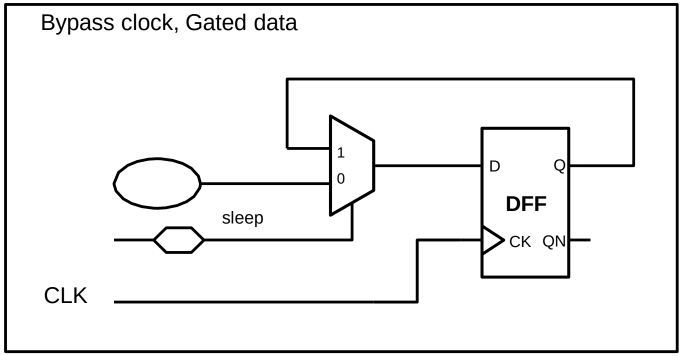
- Reduce the power consumption of registers by **turn off the un-used registers**
- Reduce the clock switching power

✓ Methods

- Gated clock, Bypass data
- Bypass clock, Gated data


Bypass clock, Bypass data


Gated clock, Bypass data

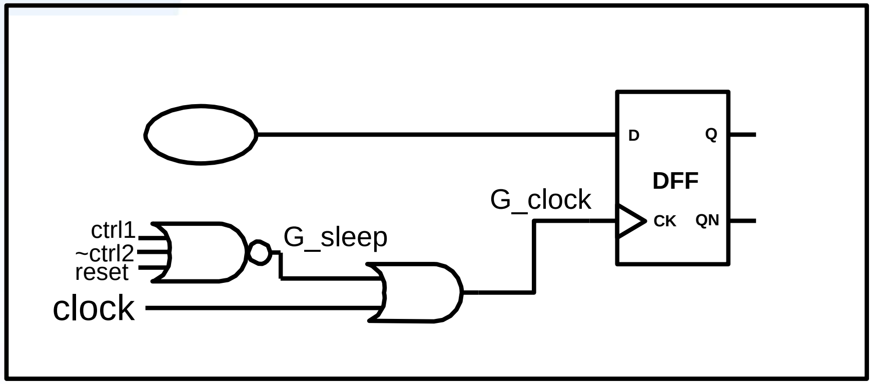

Bypass clock, Gated data

Dynamic Clock Gating

✓ Gated clock, Bypass data

- For a design on ASIC, we do clock gating to reduce the power consumption
- Notice that the gated condition should be chosen carefully

Gated clock, Bypass data



```
`ifdef SUPPORT_POWER_DOWN
   wire G_clock;
   wire G_sleep = ~( reset | ctrl1 | (~ctrl2) );
   GATED_OR GATED_G (
      .CLOCK( clock ),
      .SLEEP_CTRL( G_sleep ), // gated clock
      .CLOCK_GATED( G_clock )
   );
`else
   wire G_clock = clock;
`endif


always@(posedge G_clock) begin
          // bypass data
      if (reset) begin
         …… end
      else if (ctrl1) begin
         …… end
      else if (~ctrl2) begin
         …… end
```

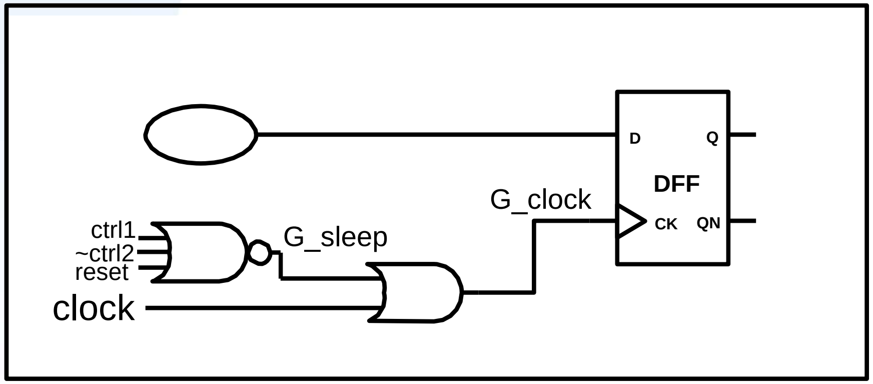
Dynamic Clock Gating

✓ Gated clock, Bypass data

- For a design on ASIC, we do clock gating to reduce the power consumption
- Notice that the gated condition should be chosen carefully

```
`ifdef SUPPORT_POWER_DOWN
  wire G_clock;
  wire G_sleep = ~( reset | ctrl1 | (~ctrl2) );
  GATED_OR GATED_G (
     .CLOCK( clock ),
     .SLEEP_CTRL( G_sleep ), // gated clock
     .CLOCK_GATED( G_clock )
  );
`else
  wire G_clock = clock;                 G_sleep = 1
`endif                                  G_clock = 1
                                        Clock gated !!

always@(posedge G_clock ) begin
          // bypass data
     if (reset) begin
       …… end
     else if (ctrl1) begin
       …… end
     else if (~ctrl2) begin
       …… end
```

Gated clock, Bypass data

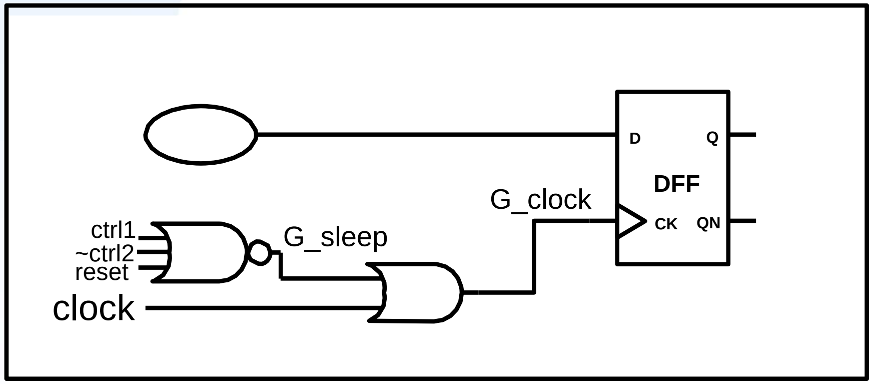


Dynamic Clock Gating

✓ Gated clock, Bypass data

- For a design on ASIC, we do clock gating to reduce the power consumption
- Notice that the gated condition should be chosen carefully

Gated clock, Bypass data



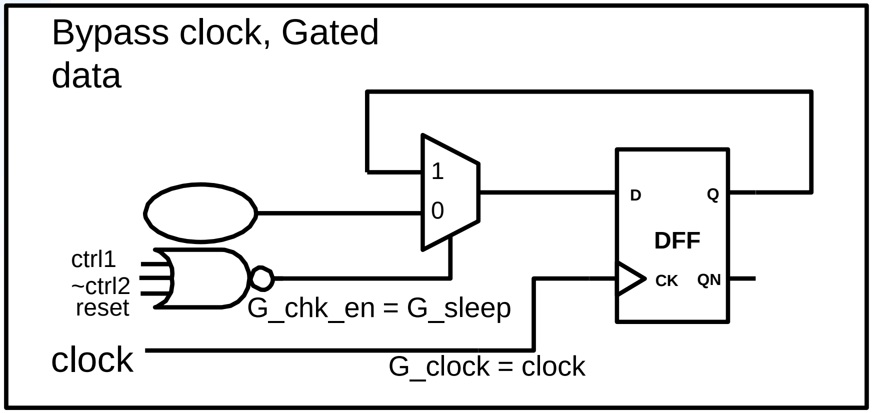
```
`ifdef SUPPORT_POWER_DOWN
   wire G_clock;
   wire G_sleep = ~( reset | ctrl1 | (~ctrl2) );
   GATED_OR GATED_G (
      .CLOCK( clock ),
      .SLEEP_CTRL( G_sleep ), // gated clock
      .CLOCK_GATED( G_clock )
   );
`else
   wire G_clock = clock;
`endif

always@(posedge G_clock ) begin
         // bypass data
     if (reset) begin
        …… end
     else if (ctrl1) begin
        …… end
     else if (~ctrl2) begin
        …… end
```

G_sleep = 0
G_clock = clock
Clock bypass !!

Dynamic Clock Gating

✓ Bypass clock, Gated data

- For a design on FPGA, the clock tree is pre-generated. It is difficult to implement dynamic gated clock scheme. Thus we do data gating to minimize signal switching.
- In order to logic equivalent with dynamic gated clock design, the data gating condition should be same with dynamic gated clock condition.



```
`ifdef SUPPORT_POWER_DOWN
   wire G_clock;
   wire G_sleep = ~( reset | ctrl1 | (~ctrl2)
   );
   wire G_chk_en = G_sleep;
   GATED_OR GATED_G (
      .CLOCK( clock ),
      .SLEEP_CTRL( 1'b0 ) // bypass clock
      .CLOCK_GATED(G_clock)
   );

`else
   wire G_clock = clock;
`endif

always@(posedge G_clock) begin // gated data
   if ( !G_chk_en) begin
      if (reset) begin
         …… end
      else if (ctrl1) begin
         …… end
      else if (~ctrl2) begin
         …… end
   end
end
```

Dynamic Clock Gating

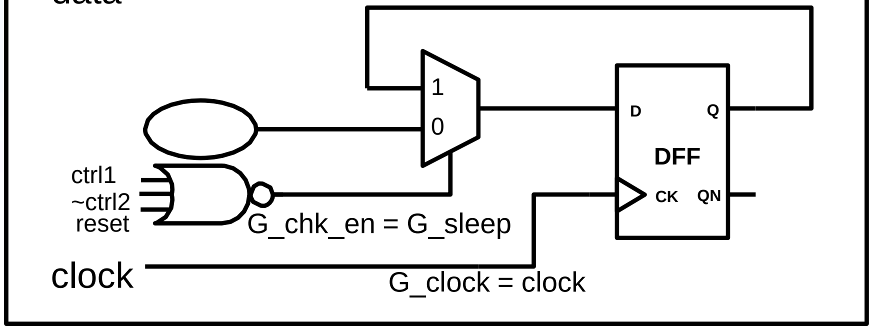
✓ Bypass clock, Gated data

- For a design on FPGA, the clock tree is pre-generated. It is difficult to implement dynamic gated clock scheme. Thus we do data gating to minimize signal switching.
- In order to logic equivalent with dynamic gated clock design, the data gating condition should be same with dynamic gated clock condition.

```
`ifdef SUPPORT_POWER_DOWN
   wire G_clock;
   wire G_sleep = ~( reset | ctrl1 | (~ctrl2)
   );
   wire G_chk_en = G_sleep;
   GATED_OR GATED_G (
      .CLOCK( clock ),
      .SLEEP_CTRL( 1'b0 ) // bypass clock
      .CLOCK_GATED(G_clock)
   );

`else
   wire G_clock = clock;
`endif

always@(posedge G_clock) begin // gated data

   if ( !G_chk_en) begin
     if (reset) begin
       …… end
     else if (ctrl1) begin
       …… end
     else if (~ctrl2) begin
       …… end
   end
end
```

G_sleep = 1
G_chg_en = 1
Data gated !!

Bypass clock, Gated data



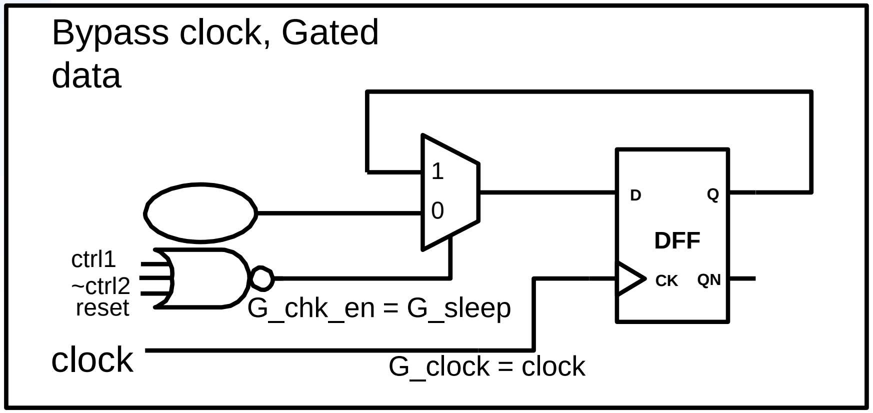
Dynamic Clock Gating

✓ Bypass clock, Gated data

- For a design on FPGA, the clock tree is pre-generated. It is difficult to implement dynamic gated clock scheme. Thus we do data gating to minimize signal switching.
- In order to logic equivalent with dynamic gated clock design, the data gating condition should be same with dynamic gated clock condition.

```
`ifdef SUPPORT_POWER_DOWN
   wire G_clock;
   wire G_sleep = ~( reset | ctrl1 | (~ctrl2)
   );
   wire G_chk_en = G_sleep;
   GATED_OR GATED_G (
      .CLOCK( clock ),
      .SLEEP_CTRL( 1'b0 ) // bypass clock
      .CLOCK_GATED(G_clock)
   );

`else
   wire G_clock = clock;
`endif

always@(posedge G_clock) begin // gated data

  if ( !G_chk_en) begin
    if (reset) begin
      …… end
    else if (ctrl1) begin
      …… end
    else if (~ctrl2) begin
      …… end
  end
end
```

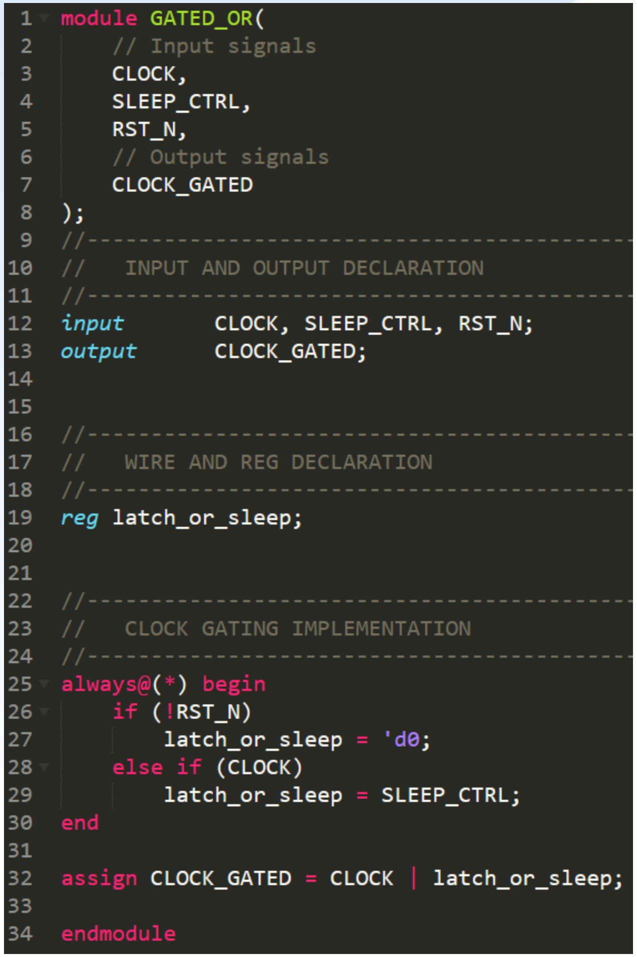
G_sleep = 0
G_chg_en = 0
Data bypass !!



Dynamic Clock Gating Example

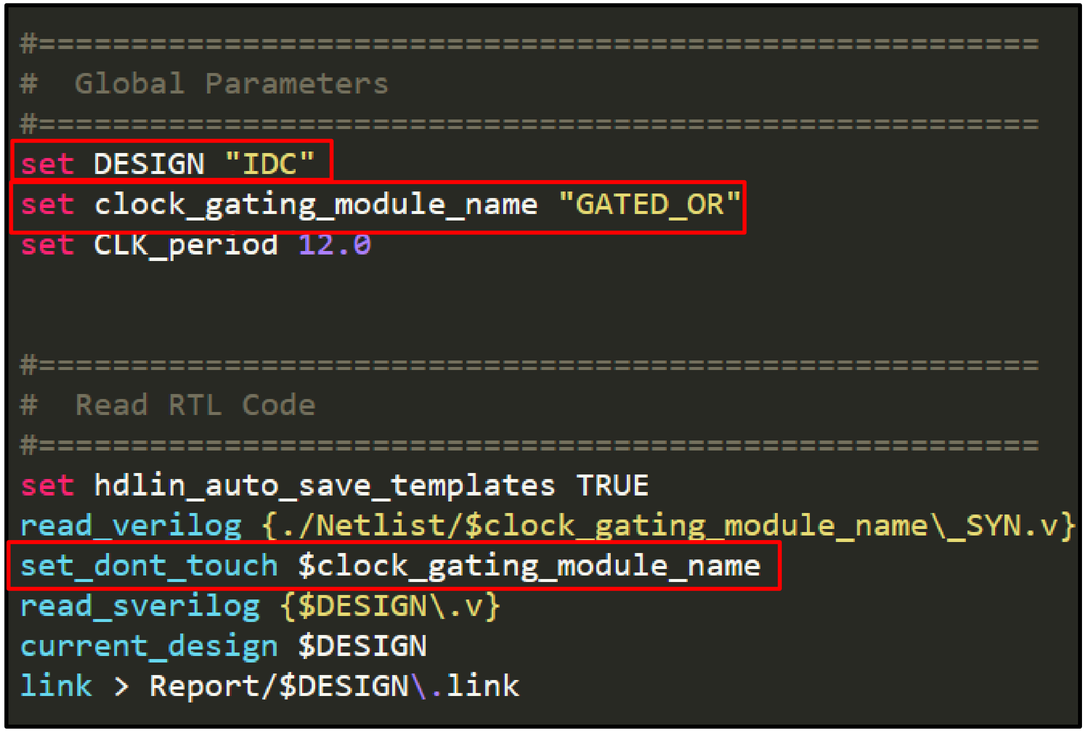
set_dont_touch $clock_gating_module_name

GATED_OR.v



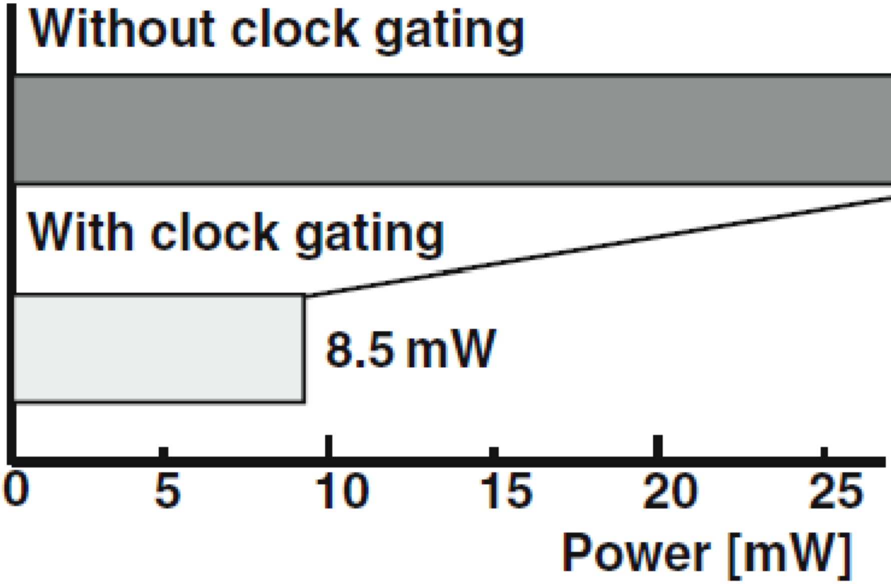
```verilog
module GATED_OR(
    // Input signals
    CLOCK,
    SLEEP_CTRL,
    RST_N,
    // Output signals
    CLOCK_GATED
);
//---------------------------------------------
//   INPUT AND OUTPUT DECLARATION
//---------------------------------------------
input        CLOCK, SLEEP_CTRL, RST_N;
output       CLOCK_GATED;


//---------------------------------------------
//   WIRE AND REG DECLARATION
//---------------------------------------------
reg latch_or_sleep;


//---------------------------------------------
//   CLOCK GATING IMPLEMENTATION
//---------------------------------------------
always@(*) begin
    if (!RST_N)
        latch_or_sleep = 'd0;
    else if (CLOCK)
        latch_or_sleep = SLEEP_CTRL;
end

assign CLOCK_GATED = CLOCK | latch_or_sleep;

endmodule
```

clk_syn.tcl

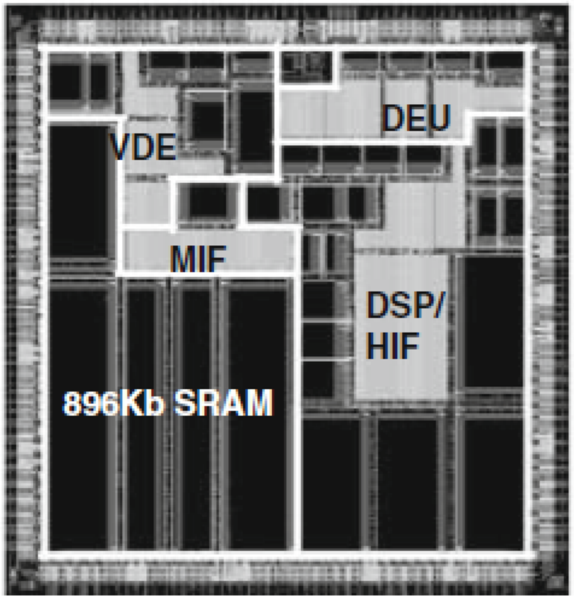


```tcl
#===============================================================
#  Global Parameters
#===============================================================
set DESIGN "IDC"
set clock_gating_module_name "GATED_OR"
set CLK_period 12.0


#===============================================================
#  Read RTL Code
#===============================================================
set hdlin_auto_save_templates TRUE
read_verilog {./Netlist/$clock_gating_module_name\_SYN.v}
set_dont_touch $clock_gating_module_name
read_sverilog {$DESIGN\.v}
current_design $DESIGN
link > Report/$DESIGN\.link
```

Clock Gating Efficiently Reduces Power



MPEG-4 decoder



90% of FFs clock-gated.

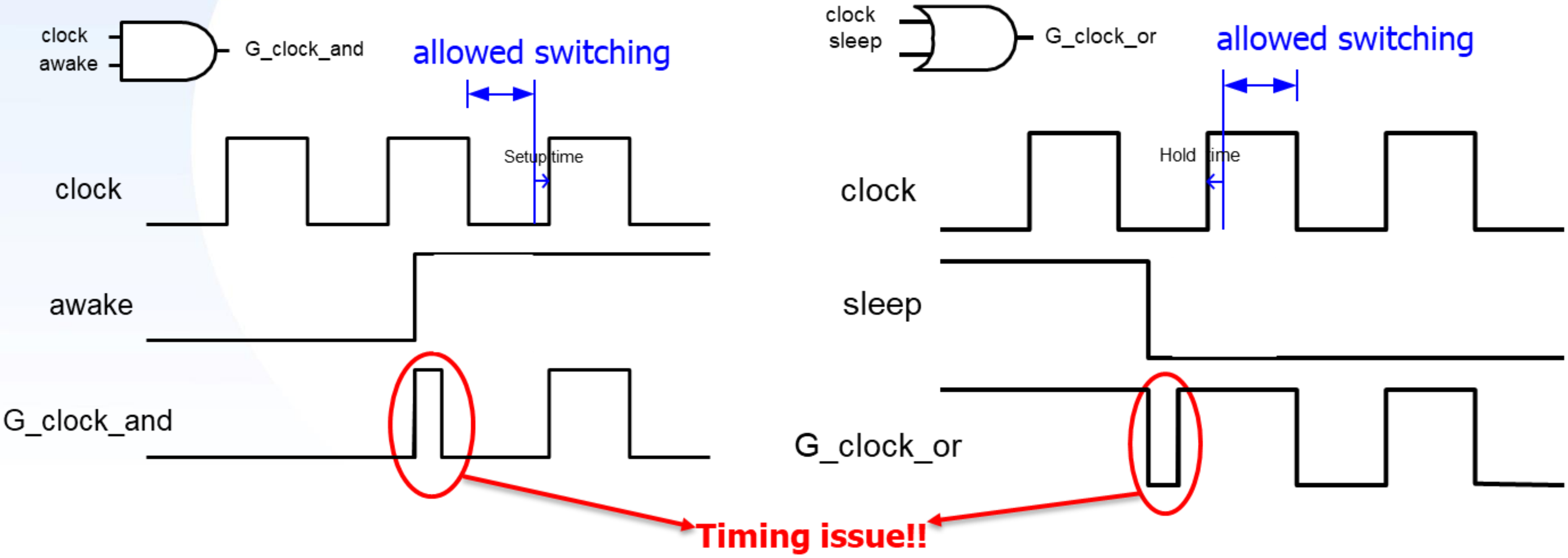
70% power reduction by clock gating alone.

[Ref: M. Ohashi, ISSCC'02]

© IEEE 2002

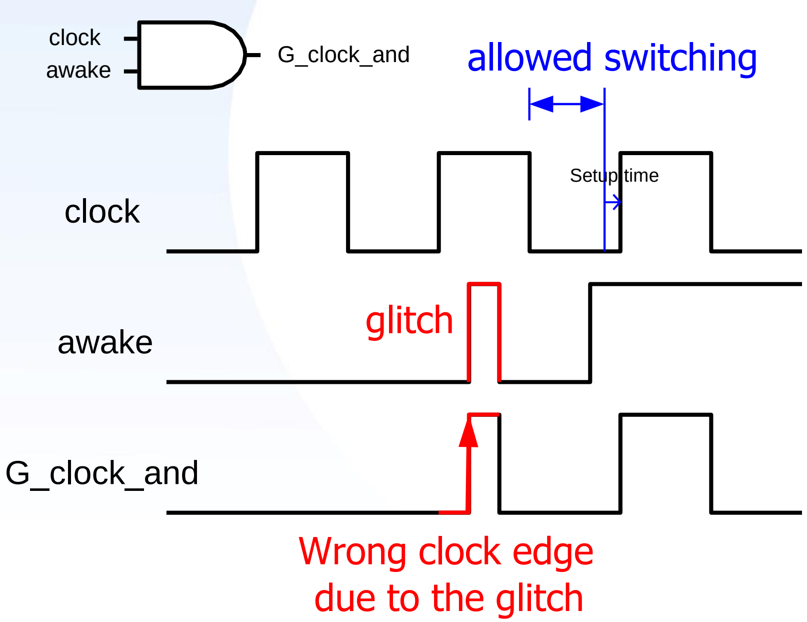
Dynamic Clock Gating

- ✓ **Clock gating can be implemented by either AND-gating or OR-gating**
- ✓ **The control signal can only change in specific half cycle**
  - For clock rising edge trigger registers design
    - For AND gate, the allow switching cycle is clock at low period
    - For OR gate, the allow switching cycle is clock at high period

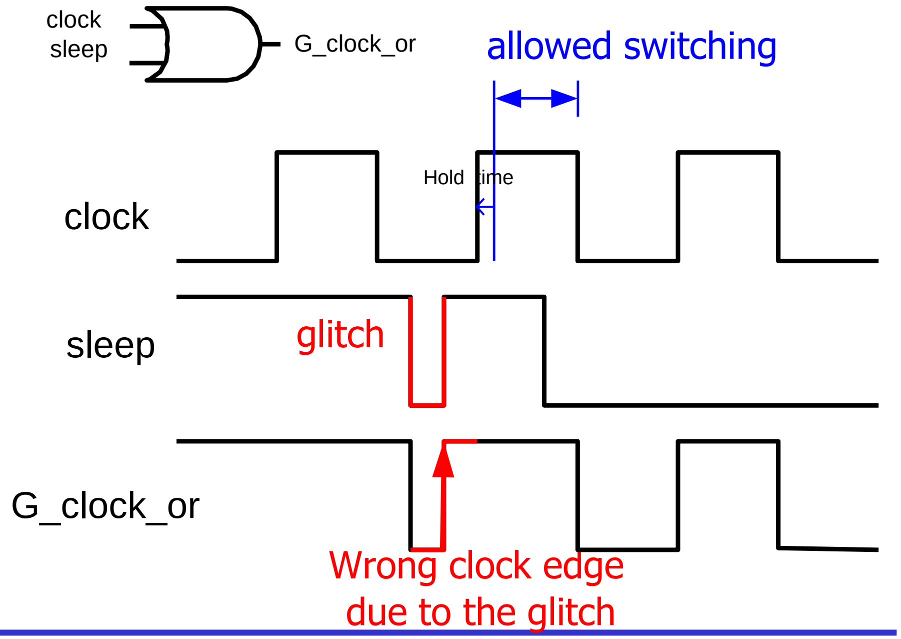


Dynamic Clock Gating

- ✓ **Clock gating can be implemented by either AND-gating or OR-gating**
- ✓ **The control signal can only change in specific half cycle**
  - For clock rising edge trigger registers design
    - For AND gate, the allow switching cycle is clock at low period
    - For OR gate, the allow switching cycle is clock at high period



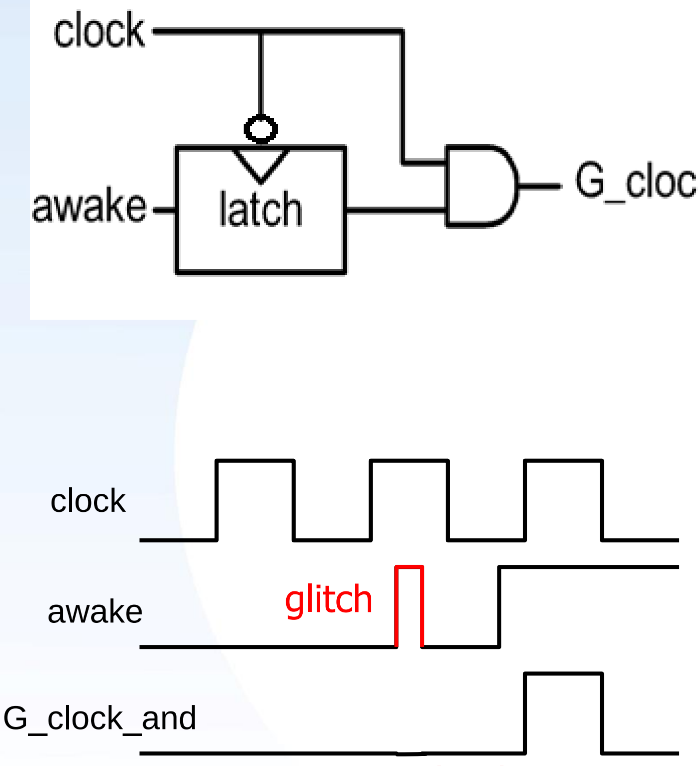
Wrong clock edge due to the glitch



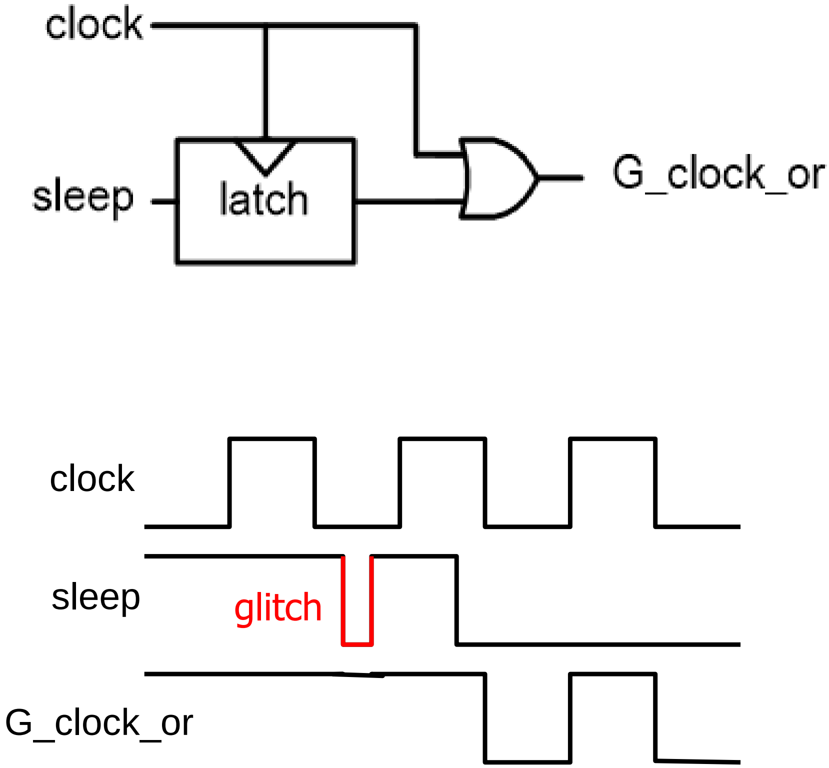
Wrong clock edge due to the glitch

Dynamic Clock Gating

✓ The method of avoid glitch

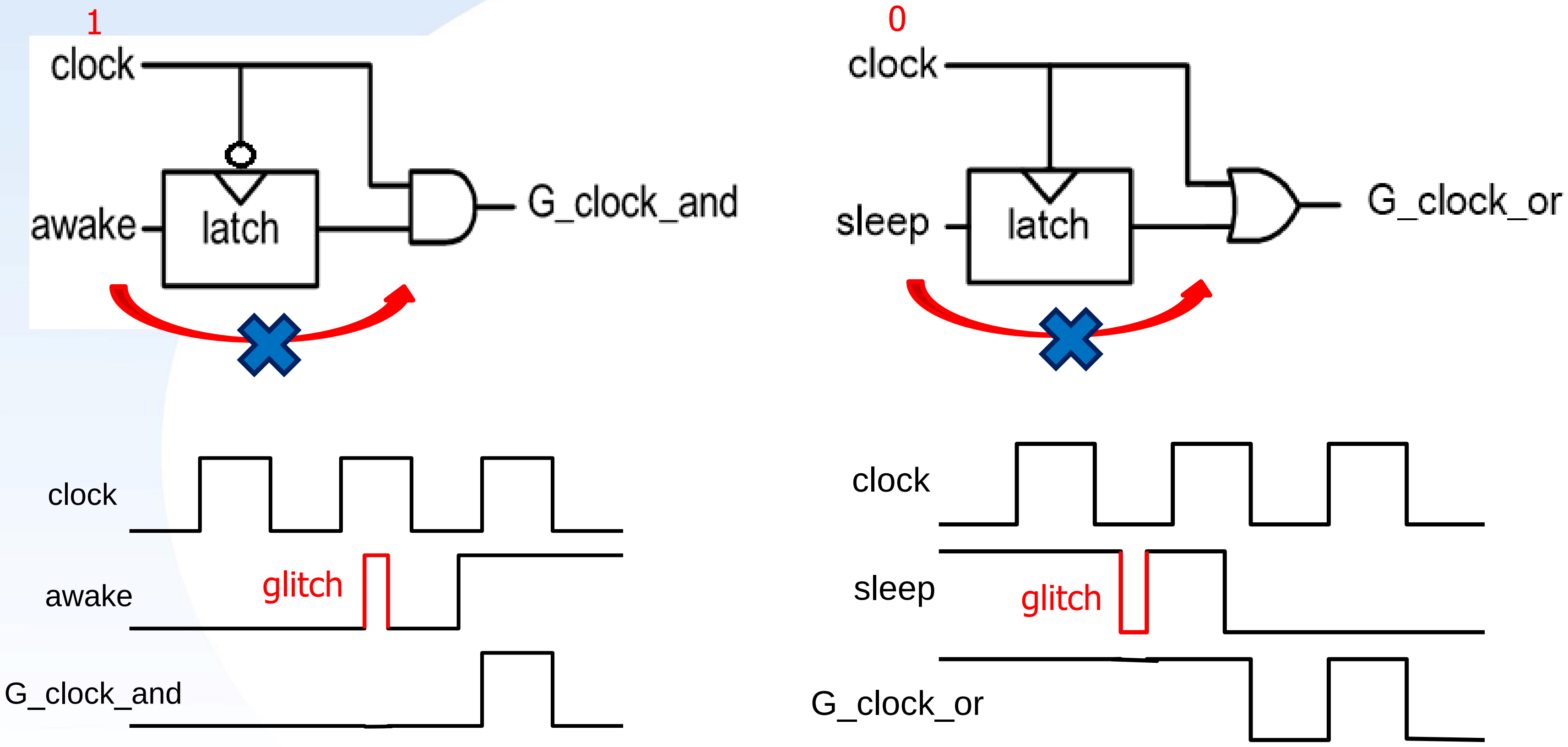


Wrong clock edge
due to the glitch



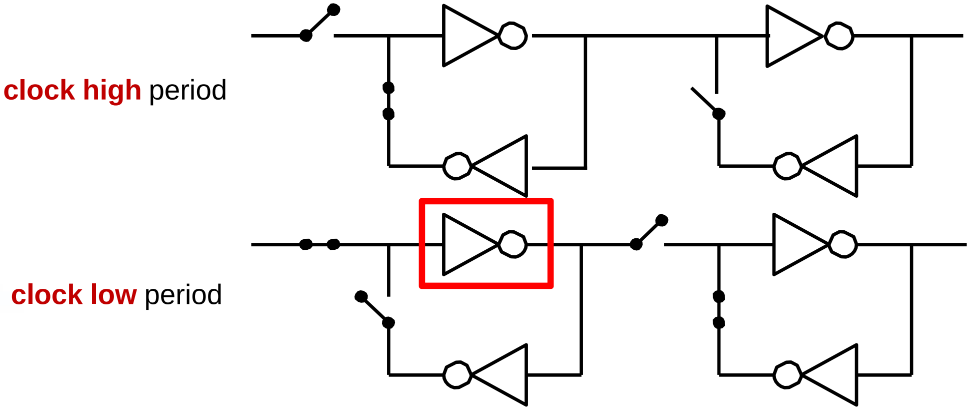
Dynamic Clock Gating

✓ **The method of avoid glitch**



Dynamic Clock Gating

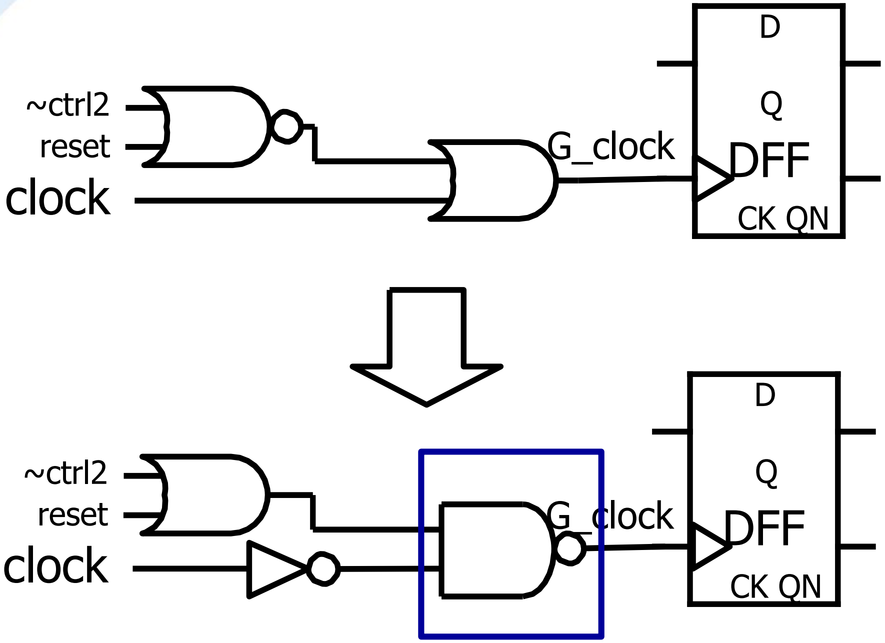
- ✓ **OR-gating is better**
  - **OR-gating consumes less power** than AND-gating
    - For OR-gating, the gated clock is tied at high when the register is turned off
    - No matter data is toggling, the first latch circuits will not be toggled
    - For AND-gating, the gated clock is tied at low when the register is turned off
    - The first latch circuits will consume power as data input is switching
  - The gating control signals should be generated from clock rising edge
  - Flip-flops → stable gating the clock as clock high period
    - Consistent with original clock rising edge trigger registers design
    - It is easier for timing control and analysis



Dynamic Clock Gating

✓ Notes

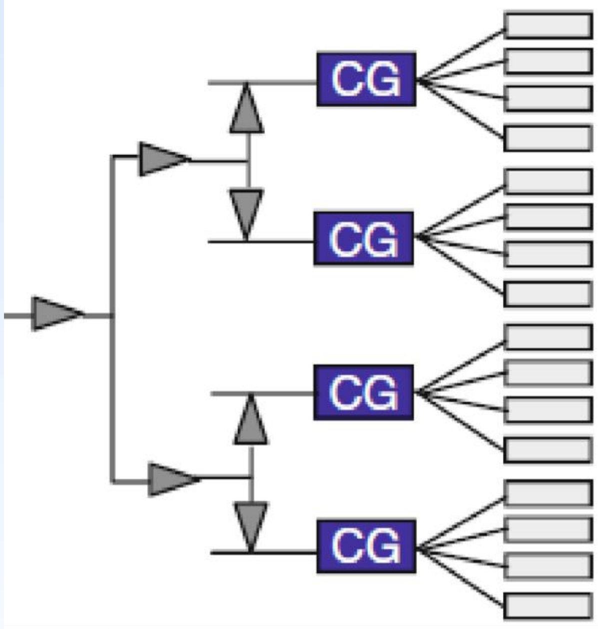
- Writing OR gate in a single module can prevent the OR gate form being optimized with other logics during the synthesis



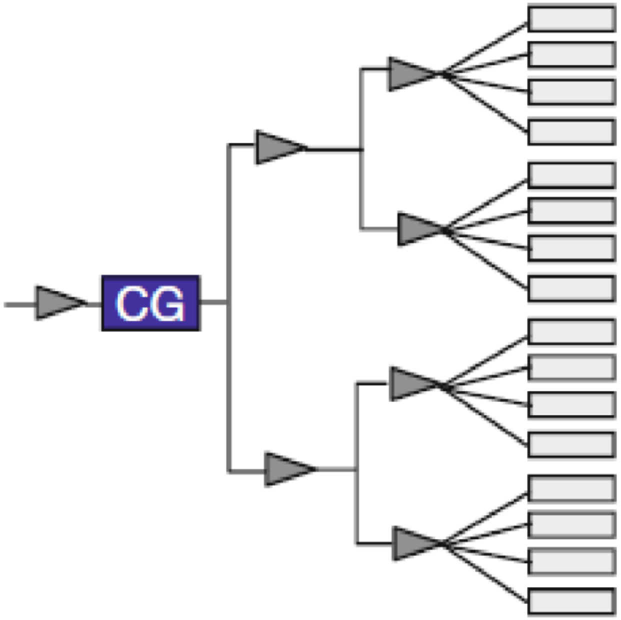
Not OR gate

Clock Gating

✓ Power Saving



✓ Simpler skew management, less area



- Keep the gates close to the registers. This allows for a fine-grain control on what to turn off and when. It comes at the expense of a more complex skew control and extra area.
- Move the gating devices higher up in the tree, which has the added advantage that the clock distribution network of the sub-tree is turn off. Modules cannot be turn off as often

Reference

- ✓ Prof. S.J. Jou, “Low-Power Digital-IC”, The lecture note of DIC, 2014.
- ✓ S. S. Wang, “Logic Synthesis with Design Compiler”, The lecture of CIC course training, August 2008.
- ✓ M. Keating, D. Flynn, R. Aitken, A. Gibbons, K. shi, “Low Power Methodology Manual”.
- ✓ Arman Vassighi, Manoj Sachdev, “Thermal Runaway in Integrated Circuits” , June 2006.

Introduction to Power Simulation by Using PrimeTime

NYCU-EE IC LAB Fall-2024

NCTU OASIS Lab
founded in 2003

Lecturer: Yen-Ning Tung

Outline

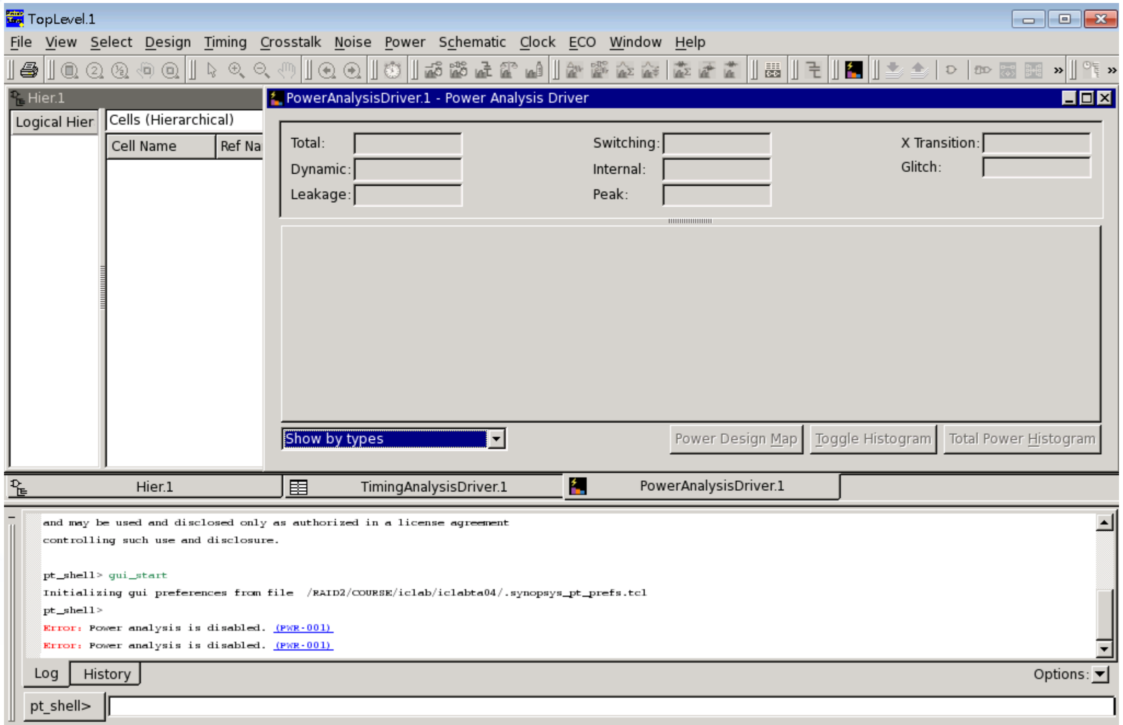
- ✓ Start PrimeTime
- ✓ Simulation Flow of PrimeTime
  - Phase 1 – relative file preparartion
  - Phase 2 – power analysis
- ✓ Reports Analysis
- ✓ A Power Analysis Script Example

Start Prime Time

✓ Interfaces of PrimeTime

- **Command-line interface**
  - pt_shell
- **Graphical user interface**
  - pt_shell -gui

```
pt_shell>
```



```
and may be used and disclosed only as authorized in a license agreement
controlling such use and disclosure.

pt_shell> gui_start
Initializing gui preferences from file  /RAID2/COURSE/iclab/iclabta04/.synopsys_pt_prefs.tcl
pt_shell>
Error: Power analysis is disabled. (PWR-001)
Error: Power analysis is disabled. (PWR-001)
```

Start PrimeTime

- ✓ **PrimeTime supports command in tcl mode**
  - Tcl: tool command language (tcl) has a straightforward syntax
- ✓ **Start using PrimeTime**
  - Start pt_shell (the PrimeTime command-line interface)
    - Command : pt_shell
  - Start PrimeTime graphical user interface (GUI)
    - Command : pt_shell -gui
  - Run scripts in pt_shell gui
    - Command : source filename.tcl
  - Get command help
    - Command : man command_name
  - Exit PrimeTime
    - Command : exit

Simulation Flow

Value Change Dump

Switching Activity Interchange Format

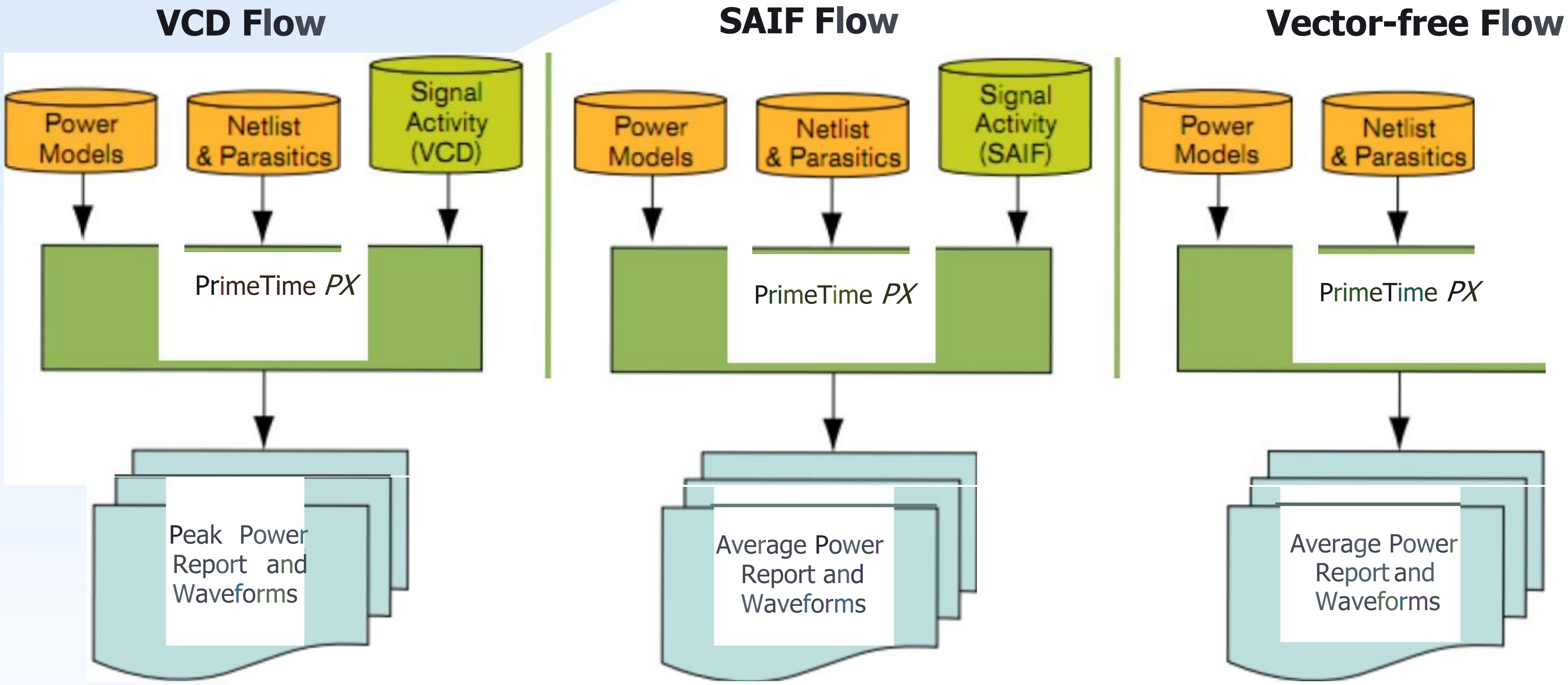


Figure 4:Power Analysis Flows.

VCD file : Contains value changes of a signal. i.e. at what times signals changes their values.

SAIF file : contains toggle counts and time information like how much time a signal was in 1 state, 0 state , x state.
It contains cumulative information of vcd.

Ref:http://www.researchgate.net/post/Whats_the_difference_between_a_SAIF_and_VCD_file_used_to_evaluate_power_consumption

Simulation Flow

✓ **PrimeTime simulation flow contains two phases**

- **Phase 1 – relative file preparation**
  - Technology library
  - Gate-level netlist
  - Switching activity file (Value change Dumpfile or FSDB file)
  - Synopsys design constraint (SDC file)
  - Standard delay format (SDF file)
  - Parasitic file (after APR) (RSPF, SPEF, or DSPF file)

- **Phase 2 – power analysis**
  - Specify search path and link libraries
  - Read design and link design
  - Read switching activity file
  - Set transition time and annotate parasitic
  - Generate power report

Simulation Flow: Phase 1

✓ **Phase 1 – relative file preparation**

- **Technology library**
  - The file contains library cells, which has timing, power and characterization information
  - Internal and leakage power are in the library
  - The library used for power analysis should be the same with the library used for synthesis
  - The technology libraries are provided by foundry, and are specified in the startup file.

Simulation Flow: Phase 2

✓ Phase 2 – power analysis

- **Specify search path**
  - Search path is a list of directories for PrimeTime to read design and library files
  - Command: set search_path {absolute_path or relative_path}
    - ◆ Example: set search_path {. ./ library ./source_code}

- **Specify link library**
  - The link_library variable specifies where and in what order PrimeTime looks for design files and library files for linking the design
  - Command: set link_library {* library_name}
    - ◆ Using an asterisk(*) to make PrimeTime search for designs in memory
    - ◆ Example: set link_library {* slow.db}

Simulation Flow: Phase 2

- **Read design**
  - The design for power analysis must be a structural, fully mapped gate-level netlist
  - The netlist cannot contain any high-level constructs
  - Command: read_verilog file_names
    - ◆ Example: read_Verilog CHIP.v

- **Set current design**
  - Before link design, specify the current design first
  - Command: current_design design_name
    - ◆ Example: current_design YOUR_DESIGN

- **Link design**
  - Resolve references in a design
  - Command: link_design [design_name]
    - ◆ design_name: specify design name to be linked (Defualt is current design)
    - ◆ Example: link_design

Simulation Flow: Phase 2

- **Read switching activity file**
  - The switching activity information used for power calculation is in the VCD or FSDB format generated and dumped out from the simulation
  - Command: read_vcd [-strip_path strip] file_name
    - [-strip_path strip]: specifies a path prefix that is to be stripped from all the object
    - names read from the VCD file or FSDB file
    - PrimeTime will transform FSDB file into VCD file automatically
    - Example : if the VCD or FSDB file is generated from module TESTBED and the instance name of the design is I_YOUR_DESIGN
    - ➔ read_vcd –strip_path TESTBED/I_YOUR_DESIGN CHIP.vcd

- **Read design constraint file**
  - It is unnecessary to re-specify design constraints. Designer can simply include the SDC file generated during synthesis
  - Command: read_sdc [-syntax_only] file_name
    - [-syntax_only]: only process the syntax check of SDC file
    - Example : read_sdc CHIP.sdc

Simulation Flow: Phase 2

- **Annotate delay information**
  - Designer can include the delay information generated during synthesis
  - Command: read_sdf [-load_delay net | cell] [-syntax_only] file_name
    - [-load_delay net | cell]: indicate whether load delays are included in net delays
    - or in cell delays
    - [-syntax_only]: only process the syntax check of SDC file
    - Example : read_sdf –load_delay net CHIP.sdf

- **Read design constraint file**
  - PrimeTime uses the transition time defined at each input port to calculate the cell power consumption and transition times for the following logic stages
  - Command : set_input_transition transition_time port_list
    - Example : set_input_transition 0.1 [all_inputs]

Simulation Flow: Phase 2

- **Generate power waveforms**
  - To save the power values over time, designer can make PrimeTime stores the power for every event within a given time interval in a waveform file.
  - Command : set_power_analysis_options –waveform_interval [-format fsdb | out] [-file file_prefix]
    - ◆ waveform_interval: specify the sampling interval for power calculation (unit : ns)
    - ◆ [-file file_prefix]: set the prefix for the output files (default : PrimeTime)
    - ◆ [-format fsdb | out]: specify the waveform format
      - » fsdb: PrimeTime default value, can be viewed by nWave
      - » out: PrimeTime will output a text .out file
      - » rpt: PrimeTime does not output waveform, but display peak power in .rpt file
    - ◆ Example: set_power_analysis_options -waveform_interval 1 -waveform_format out -waveform_output vcd

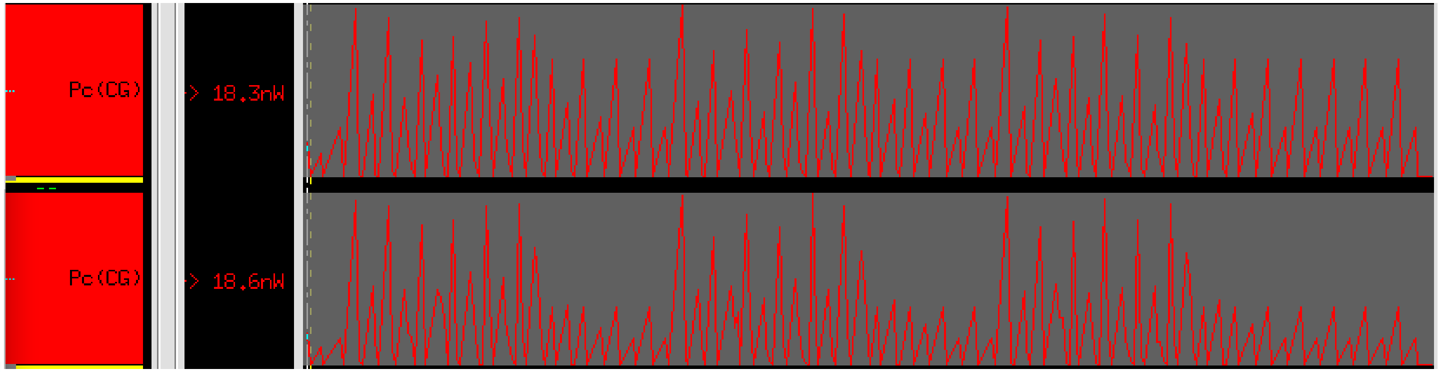
Will create file ''vcd.out''

Simulation Flow: Phase 2

- **Generate power waveforms**
  - To save the power values over time, designer can make PrimeTime stores the power for every event within a given time interval in a waveform file.
  - Command : set_power_analysis_options –waveform_interval [-file file_prefix] [-format fsdb | out]
    - -waveform_output fsdb example

w/o clock gating

w/ clock gating

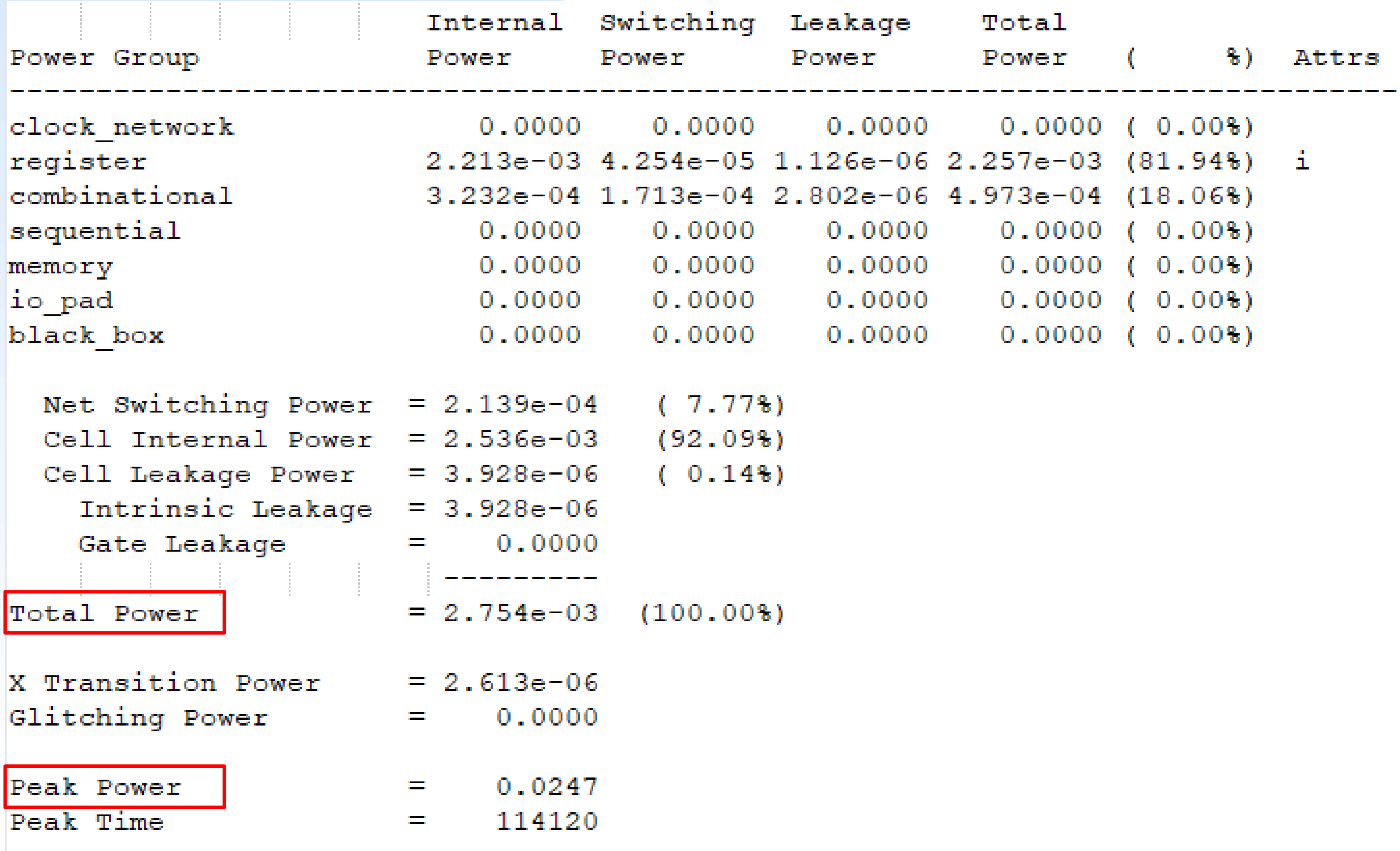


Simulation Flow: Phase 2

- **Generate default power reports**
  - PrimeTime can generate a wide range of reports that provide information about the power consumption
  - Command: report_power [-file file_prefix] [-leaf] [-nosplit]
    [-sortby [power | toggle | name]]
    - [-leaf]: reports power for leaf instances
    - [-sortby [power | toggle | name]]: sort the instances in the .rpt file
      - » Power: sort by power values
      - » Toggle: sort by toggle counts
      - » Name sort by name
    - [-nosplit]: to prevent line-splitting or section-breaking
    - Example: report_power –file CHIP –sortby power -leaf

Reports Analysis

- ✓ **Default power report analusis**



```
                          Internal  Switching  Leakage    Total
Power Group               Power     Power      Power      Power    (      %)  Attrs
--------------------------------------------------------------------------------
clock_network                0.0000    0.0000    0.0000    0.0000 ( 0.00%)
register                  2.213e-03 4.254e-05 1.126e-06 2.257e-03 (81.94%)  i
combinational             3.232e-04 1.713e-04 2.802e-06 4.973e-04 (18.06%)
sequential                   0.0000    0.0000    0.0000    0.0000 ( 0.00%)
memory                       0.0000    0.0000    0.0000    0.0000 ( 0.00%)
io_pad                       0.0000    0.0000    0.0000    0.0000 ( 0.00%)
black_box                    0.0000    0.0000    0.0000    0.0000 ( 0.00%)

  Net Switching Power  = 2.139e-04   ( 7.77%)
  Cell Internal Power  = 2.536e-03   (92.09%)
  Cell Leakage Power   = 3.928e-06   ( 0.14%)
    Intrinsic Leakage  = 3.928e-06
    Gate Leakage       =    0.0000
                         ---------
Total Power            = 2.754e-03  (100.00%)

X Transition Power     = 2.613e-06
Glitching Power        =    0.0000

Peak Power             =    0.0247
Peak Time              =    114120
```

Appendix A
Dynamic Clock Gating For Different Syntheses

Dynamic Clock Gating

- ✓ **A general design Verilog coding**
  - **For ASIC synthesis**
    - **'define FPGA_SYN 0**
    - **Bypass data, gated clock**
  - **For FPGA synthesis**
    - **'define FPGA_SYN 1**
    - **Gated data, bypass clock**

```
`ifdef SUPPORT_POWER_DOWN wire
   G_clock;
   wire G_sleep = ~( reset | ctrl1 | (~ctrl2) );
   wire G_gated = G_sleep & ~( `FPGA_SYN );
   wire G_chk_en = ( `FPGA_SYN == 0 )? 1'b0 : G_sleep;
   GATED_OR GATED_G (
       .CLOCK( clock ),
       .SLEEP_CTRL( G_gated ),
       .CLOCK_GATED( G_clock )
   );
`else
   wire G_clock = clock; wire
   G_chk_en = 0;
`endif

always@(posedge G_clock)  begin
   if ( !G_chk_en) begin
      if (reset) begin
         ...... end
      else if (ctrl1) begin
         ...... end
      else if (~ctrl2) begin
         ...... end
   end end
```

Dynamic Clock Gating

- ✓ **A general design Verilog coding**
  - **For ASIC synthesis**
    - **'define FPGA_SYN 0**
    - **Bypass data, gated clock**
  - **For FPGA synthesis**
    - **'define FPGA_SYN 1**
    - **Gated data, bypass clock**

```
`ifdef SUPPORT_POWER_DOWN wire
   G_clock;
   wire G_sleep = ~( reset | ctrl1 | (~ctrl2) );
   wire G_gated = G_sleep & ~( `FPGA_SYN );
   wire G_chk_en = ( `FPGA_SYN == 0 )? 1'b0 : G_sleep;
   GATED_OR GATED_G (
       .CLOCK( clock ),
       .SLEEP_CTRL( G_gated ),
       .CLOCK_GATED( G_clock )
   );
`else
   wire G_clock = clock; wire
   G_chk_en = 0;
`endif

always@(posedge G_clock)  begin
  if ( !G_chk_en) begin
     if (reset) begin
        ...... end
     else if (ctrl1) begin
        ...... end
     else if (~ctrl2) begin
        ...... end
  end end
```

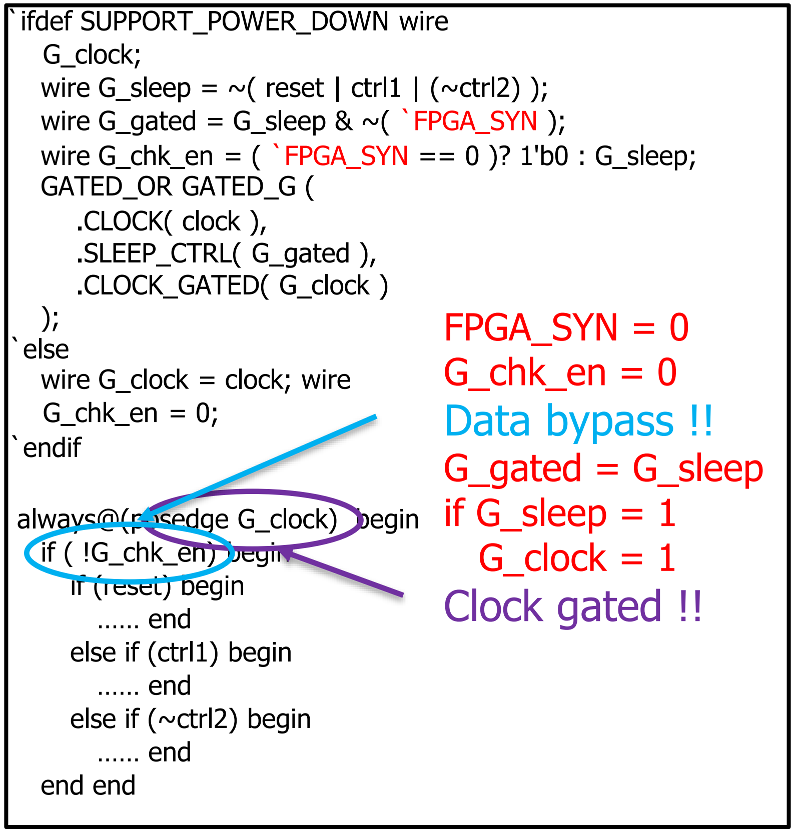
FPGA_SYN = 0
G_chk_en = 0
Data bypass !!

Dynamic Clock Gating

✓ **A general design Verilog coding**

- **For ASIC synthesis**
  - **'define FPGA_SYN 0**
  - **Bypass data, gated clock**

- **For FPGA synthesis**
  - **'define FPGA_SYN 1**
  - **Gated data, bypass clock**



```
`ifdef SUPPORT_POWER_DOWN wire
   G_clock;
   wire G_sleep = ~( reset | ctrl1 | (~ctrl2) );
   wire G_gated = G_sleep & ~( `FPGA_SYN );
   wire G_chk_en = ( `FPGA_SYN == 0 )? 1'b0 : G_sleep;
   GATED_OR GATED_G (
       .CLOCK( clock ),
       .SLEEP_CTRL( G_gated ),
       .CLOCK_GATED( G_clock )
   );
`else
   wire G_clock = clock; wire
   G_chk_en = 0;
`endif

always@(posedge G_clock) begin
   if ( !G_chk_en) begin
      if (reset) begin
         ...... end
      else if (ctrl1) begin
         ...... end
      else if (~ctrl2) begin
         ...... end
   end end
```

FPGA_SYN = 0
G_chk_en = 0
Data bypass !!
G_gated = G_sleep
if G_sleep = 1
   G_clock = 1
Clock gated !!

Revision History

2007	Kuan-Ling Kuo (amos@si2lab.org)
2008	Yao-Lin Chen (arryz@si2lab.org)
2013	Yu-Tao Yang (futurestar@si2lab.org)
	Rong-Jie Liu (johnny510.ee98@g2.nctu.edu.tw)
2014	Hsin_Yi Yu (cindy19212002@gmail.com)
2015	Sheng Wan (vjod@si2lab.org)
	Renxuan Yu (yurx1123@gmail.com)
2017	Tsu-Jui Hsu (johnson711309@gmail.com)
2018	Po-Yu Huang (hpy35269@gmail.com)
	Chien-Hao Chen (coreldraw8083@gmail.com)
2019	Chi-Yuan Sung (sam850325@gmail.com)
2020	Cheng-Han Huang (huang50216@gmail.com)
2021	Shao-Wen Cheng (shaowen0213@gmail.com)
2022	Yu-Lun Hsu (huang50216@gmail.com)
	Fang-Jui Chang (eric9686.ee07@nycu.edu.tw)
2023	Kuan-Wei Chen (bakerchen1018.ee10@nycu.edu.tw)
2024	Yen-Ning Tung (tongcookie777.ee12@nycu.edu.tw)