Digital System Design

Architecture Level Improvement

Lecturer: 王景平

Advisor: Prof. An-Yeu Wu

Date: 2025/04/17

What .synopsys_dc.setup defined

- search_path: the path to search for unsolved reference library or design
- link_library: the library used for interpreting input description
  - Any cells instantiated in your HDL code (Macro…)
- target_library: the ASIC technology that the design is mapped to (standard cell)
- symbol_library: used during schematic generation
- synthetic_library: designware library to be used

- Other variables

.db versus .lib file

slow.db

```
SOH SOH SOH   BS ETX 6 x8F SOH BS SOH ETX STX EN
taint_license SOH ETX BS contents EOT BEL slow
2000.11-SP1-1 NUL STX DC4 SOH ENQ ACK CAN libr
operations
processors
FF data_classes VT
library_types FF SI implementations
EOT slow ENQ ETX DC2 ENQ SOH SO EOT type ENQ SI DC
references NAK
netlist_cells SYN FF netlist_nets ETB
graphics_view CAN BEL aliases EM VT constrain
ACCSHCINX2 STX NUL NUL NUL NUL EOT NAK SOH SOH C
pin_number STX EOT STX"    pin_class STX ENQ
rise_power
0-STX ENQ . BS template/DC3 energy_template_
FS=!C|=#GSq=$ENQ-=$NUL xFC=#   =""xD5=RS
fall_power VT 06 STX /ETX RS ETX BEL=,BS1=x87
0 NUL SOH ENQ ; STX !B   SO0-STX/ETX RS ETX BEL=
x95=EOT騢=ENQfACK=ENQ   =ENQbxE1=EOT搧=
xB8 =VT|5=VT愔=VT   =
  =BSψ=EOT摞=
}g=VTJxED=VT扈=VTcVT=
wGS=BS   =EOT愁=VT3xDB=FF DC2'=FF   =FFZx
Z\=
Ec=FFx91BS=
椪=BEL0@=SIxC96=DLE舀=DC1;\=DC1/xD3=DL
E<US@xA3<DLExEAx9E<-xA3<<-6/<,dw<*xE2x
6xB5<
```

slow.lib

```
cell (AND2X1) {
  cell_footprint : and2;
  area : 6.789600;
  pin(A) {
    direction : input;
    capacitance : 0.001327;
  }
  pin(B) {
    direction : input;
    capacitance : 0.001542;
  }
  pin(Y) {
    direction : output;
    capacitance : 0.0;
    function : "(A B)";
    internal_power() {
      related_pin : "A";
      rise_power(energy_template_7x7) {
        index_1 ("0.042, 0.066, 0.112, 0.206, 0.
```

Area

Power

```
  timing() {
    related_pin : "A";
    timing_sense : positive_unate;
    cell_rise(delay_template_7x7) {
      index_1 ("0.042, 0.066, 0.112, 0.206, 0.
      index_2 ("0.00079, 0.002054, 0.00474, 0.
      values ( \
```

Timing (delay)

fast.lib versus slow.lib file

fast.lib ➔ for HOLD time

```
cell (AND2X1) {
  cell_footprint : and2;
  area : 6.789600;
```

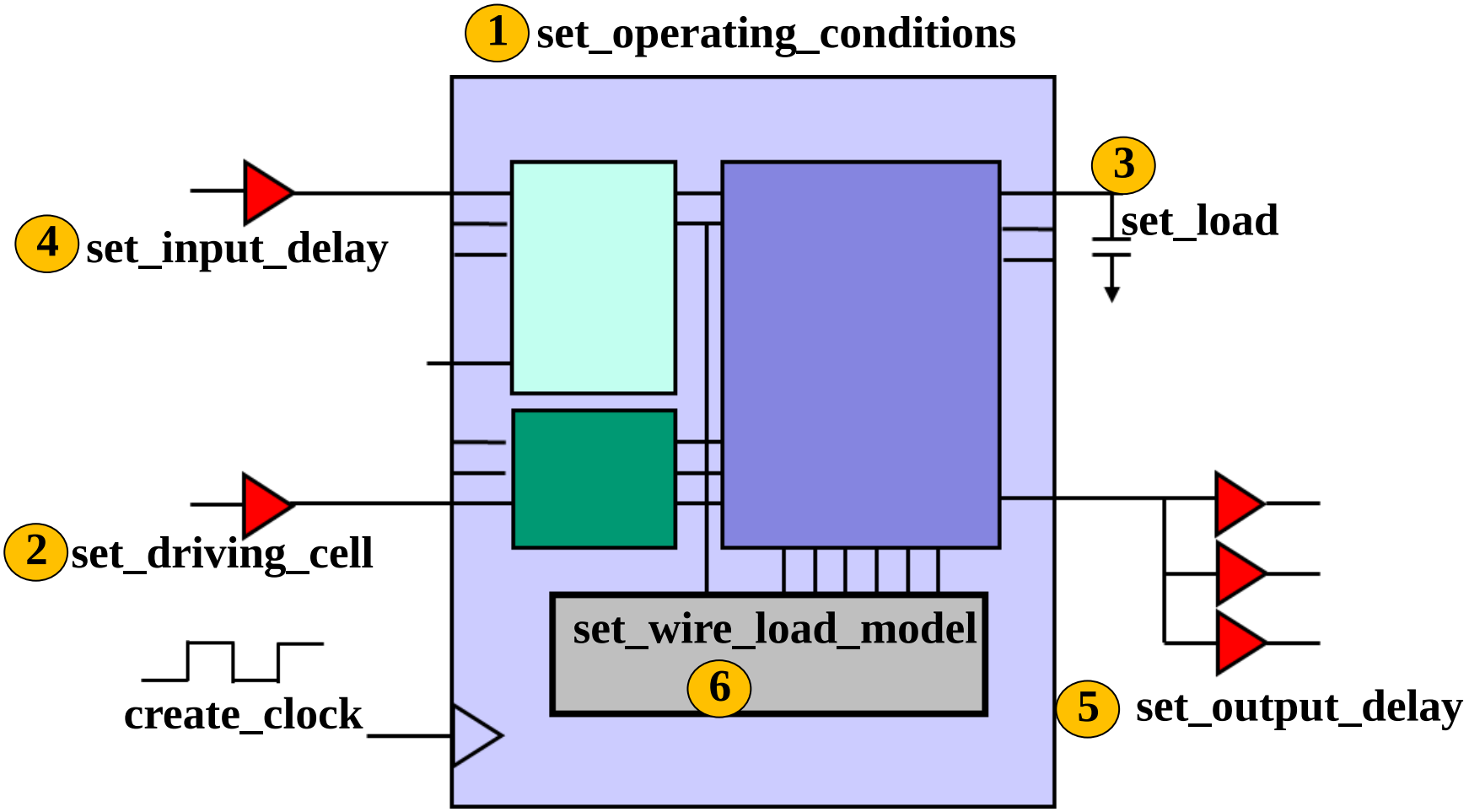
```
timing() {
  related_pin : "A";
  timing_sense : positive_unate;
  cell_rise(delay_template_7x7) {
    index_1 ("0.02, 0.032, 0.056, 0.102
    index_2 ("0.00079, 0.002054, 0.0047
    values ( \
      "0.045522, 0.050983, 0.061495, 0.
      "0.047351, 0.052798, 0.063319, 0.
      "0.051040, 0.056467, 0.066916, 0.
      "0.055823, 0.061330, 0.071888, 0.
      "0.059203, 0.064842, 0.075497, 0.
      "0.057208, 0.063229, 0.074353, 0.
      "0.039006, 0.045647, 0.057870, 0.
  }
```

slow.lib ➔ for SETUP time

```
cell (AND2X1) {
  cell_footprint : and2;
  area : 6.789600;
```

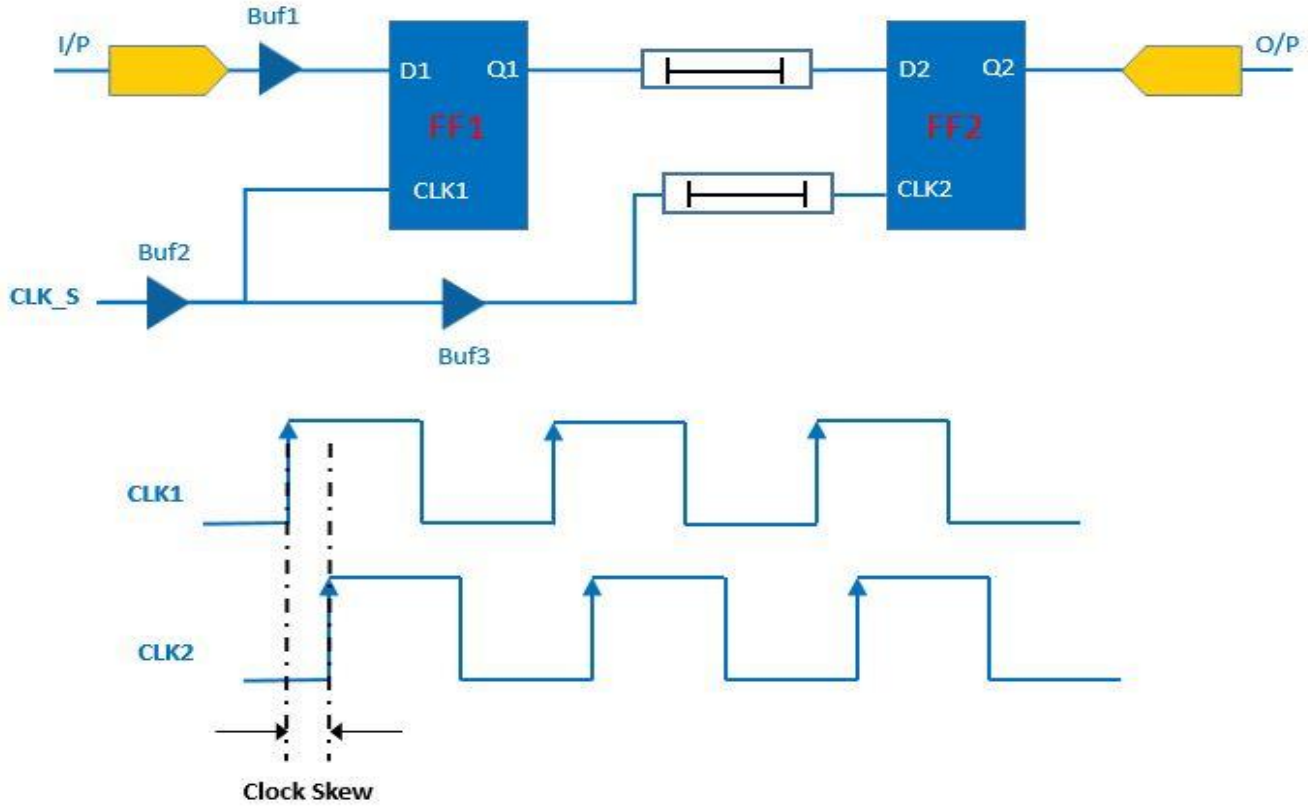
```
timing() {
  related_pin : "A";
  timing_sense : positive_unate;
  cell_rise(delay_template_7x7) {
    index_1 ("0.042, 0.066, 0.112, 0.2
    index_2 ("0.00079, 0.002054, 0.004
    values ( \
      "0.120617, 0.134063, 0.159187, 0
      "0.125564, 0.138998, 0.164099, 0
      "0.135198, 0.148596, 0.173633, 0
      "0.153769, 0.167154, 0.192143, 0
      "0.177698, 0.191732, 0.217528, 0
      "0.199998, 0.215214, 0.242429, 0
      "0.207585, 0.224996, 0.255239, 0
  }
```

Describing Design Environment (2/2)



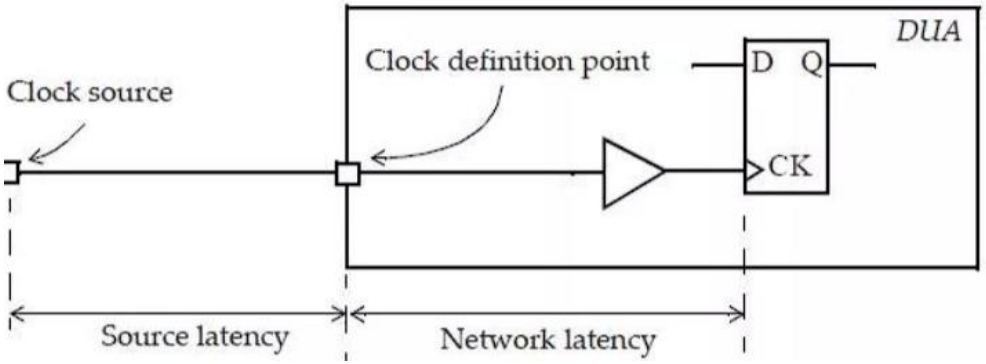
Clock Uncertainty (Skew)

- The maximum difference between the arrival of clock signals at sequential cells
  - `set_clock_uncertainty <UNCERTAINTY> [get_clocks <CLOCK NAME>]`



Clock Latency

- The delay from the clock source to the devices
  - `set_clock_latency 0.5 [-source] [get_clocks <CLOCK NAME>]`
- I/O delay is set with respect to the clock edge
- Considered if there is more than 1 clock in the system
  - Latency of source and sink canceled out for 1 clock system



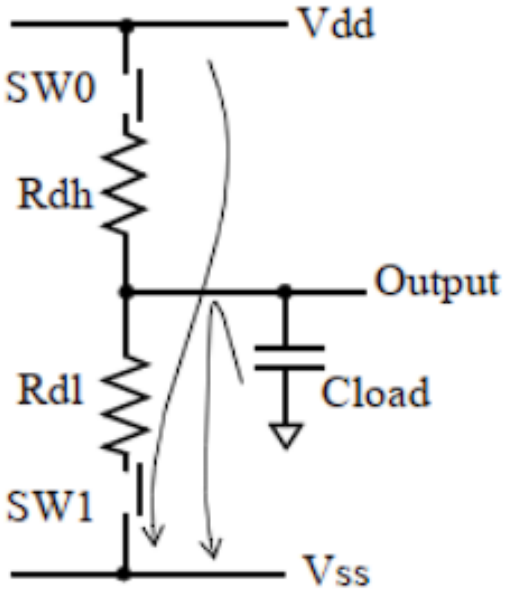
Setting the Fanout Load

- Vendors impose design rules that restrict how many cells are connected to one another based on capacitance, transition, and fanout

- Fanout
  - Number of load gates connected to the output of the driving gate

```
set_max_capacitance 0.2 [all_inputs]
set_max_transition  0.2 [all_inputs]
set_max_fanout      6   [all_inputs]
```

The functions among these three command are same, you just need to use one



Common Collection Command

all_clocks	get_cells
all_ideal_nets	get_clocks
all_inputs	get_designs
all_outputs	get_nets
all_registers	get_pins
	get_ports

- **get_* [-hier|-hierarchical] [NAME PATTERN]**
  - **get_cells "inst*"** → all cells start with "inst"
  - **get_nets "*valid*"** → all nets contain "valid"
  - **get_pins "top/U1/Q"** → pin Q under top/U1
  - **get_pins "*Q"** → pins end with "Q" of top design(no match)
  - **get_pins "*/Q"** → all pins Q under instances of top design
  - **get_pins "*/Q" -hier** → all pins Q under all instances

Outline

- Architecture Level Improvement
  - Performance metrics
  - Timing Issue
    - Pipelining
    - Retiming
    - Parallelizing
  - Area Issue
  - Power Issue

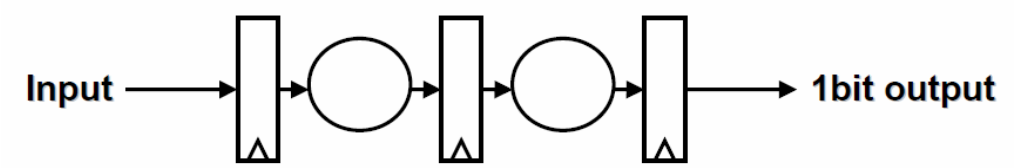
Introduction

- When design in RTL, the designer need to be aware of timing, area and power issues.
- Meeting timing requirement is the most critical goal in design. Only optimize for power or area after timing is met.
- Synthesis tools operate in gate level, and cannot resolve all timing, area and power issues.
  - Performance optimization is done better at algorithm level and architecture level.

Performance Metrics

- Latency – How many cycles (or how long) does it take from input to output?

- Throughput – How many operations can be completed per second?
  - Throughput = clock frequency x operation's bit width



clock frequency = 100MHz

Latency = 3 clock cycles, 30ns
Throughput = 100Mbits/second

Timing Issue

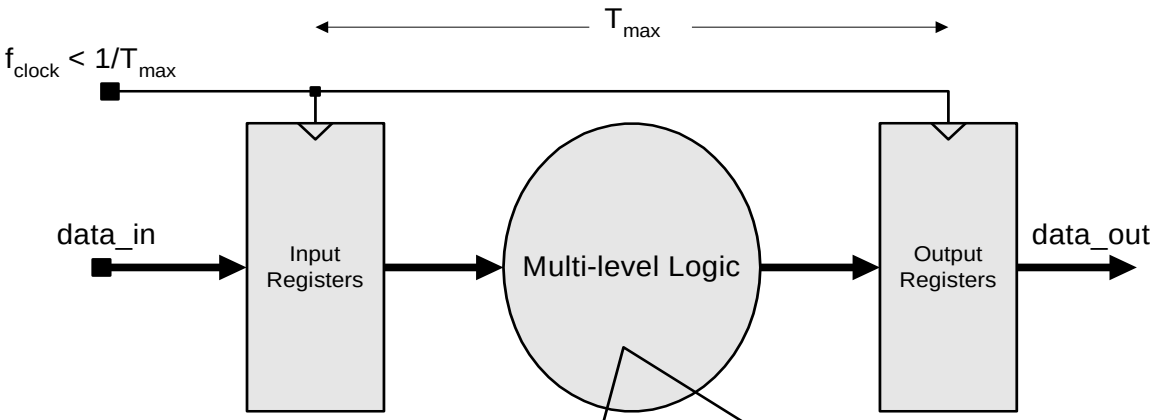
- To fit system throughput, the timing (clock period) must be smaller than some value.
- In IC design industry, the design must meet timing with margin, and using worst-case library model.
- If the post-synthesis simulation (under a clock period larger than target one) cannot pass the verification to deliver certain throughput, we need to improve the timing.

How to Improve Timing in Design

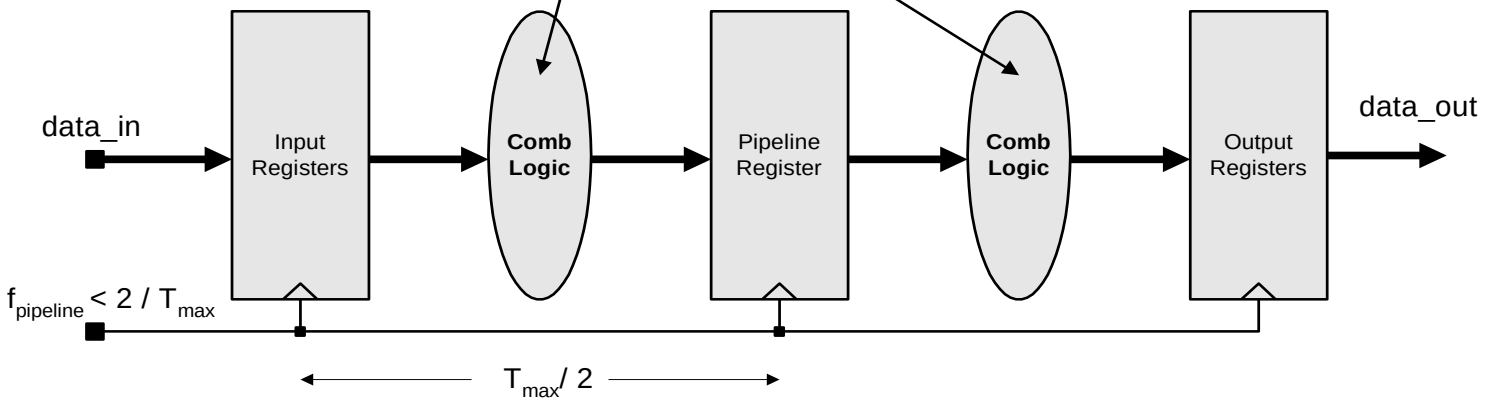
- **Pipelining** : Exploits temporal parallelism
  - Insert pipeline registers without changing the coherence of the data
  - Shorten critical path
  - Reduce achievable clock period
  - Trade off latency (in cycles) to improve throughput
- **Parallelizing** :
  - Duplicate function units working in parallel
  - Improve throughput without shortening critical path

Concept of Pipelining

Original



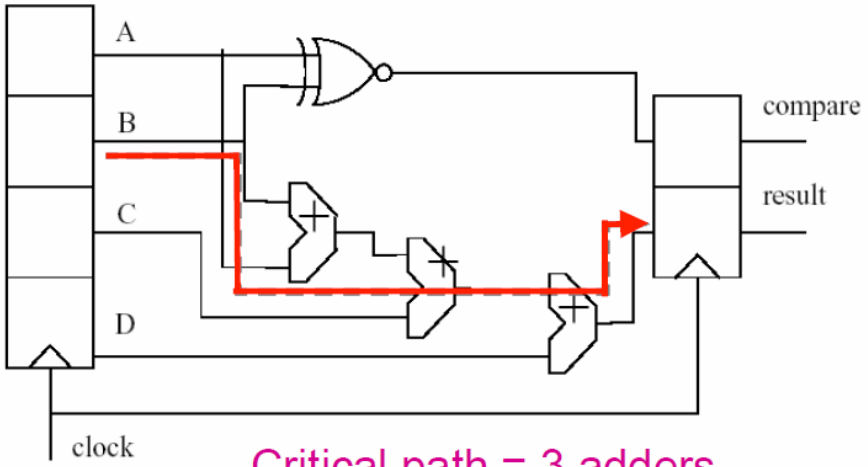
2-stage pipelining



Example 1: Simple Circuit

original

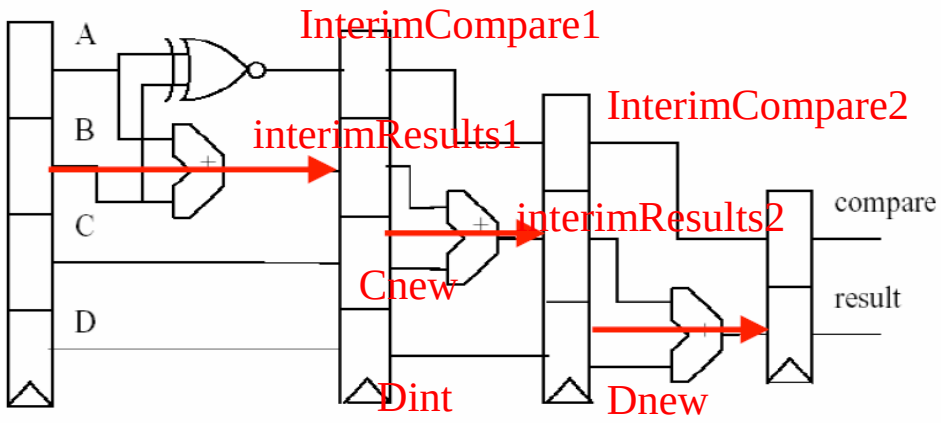
```
always@(posedge clock)
  begin
    result <= A + B + C + D;
    compare <=A ^ B;
  end
```



Critical path = 3 adders

3-stage pipelining

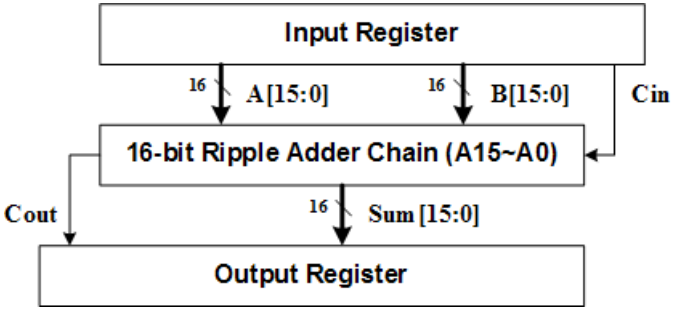
```
always@(posedge clock)
  begin
    interimResult1 <= A + B;
    Cnew <= C;
    Dint <= D;
    Dnew <= Dint;
    interimResult2 <= InterimResult1 + Cnew;
    Result <= InterimResult2 + Dnew;
    InterimCompare1 <= A^B;
    InterimCompare2 <= InterimCompare1;
    Compare <= InterimCompare2;
  end
```



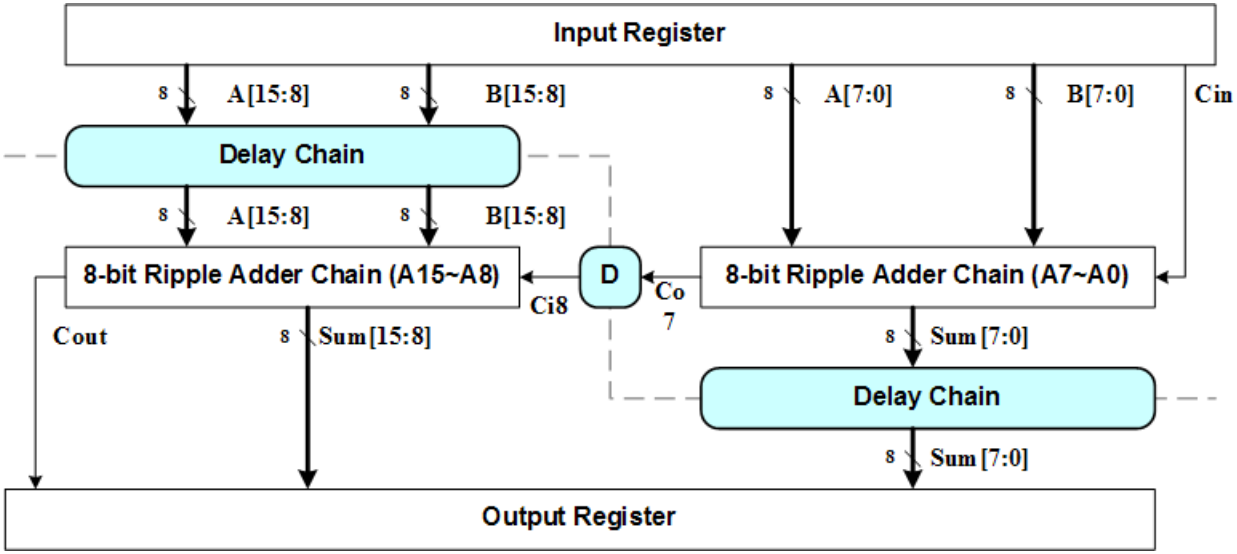
Critical path = 1 adders

Example 2: Pipelined 16-bit Adder

Original



2-stage pipelining
(fine-grain pipeline)



Drawbacks of Pipelining

- Increase area
  - The number of pipeline registers increases
- Increase latency (in cycle)
  - Insert N stage pipeline registers, add N cycle latency

How to do Pipelining?

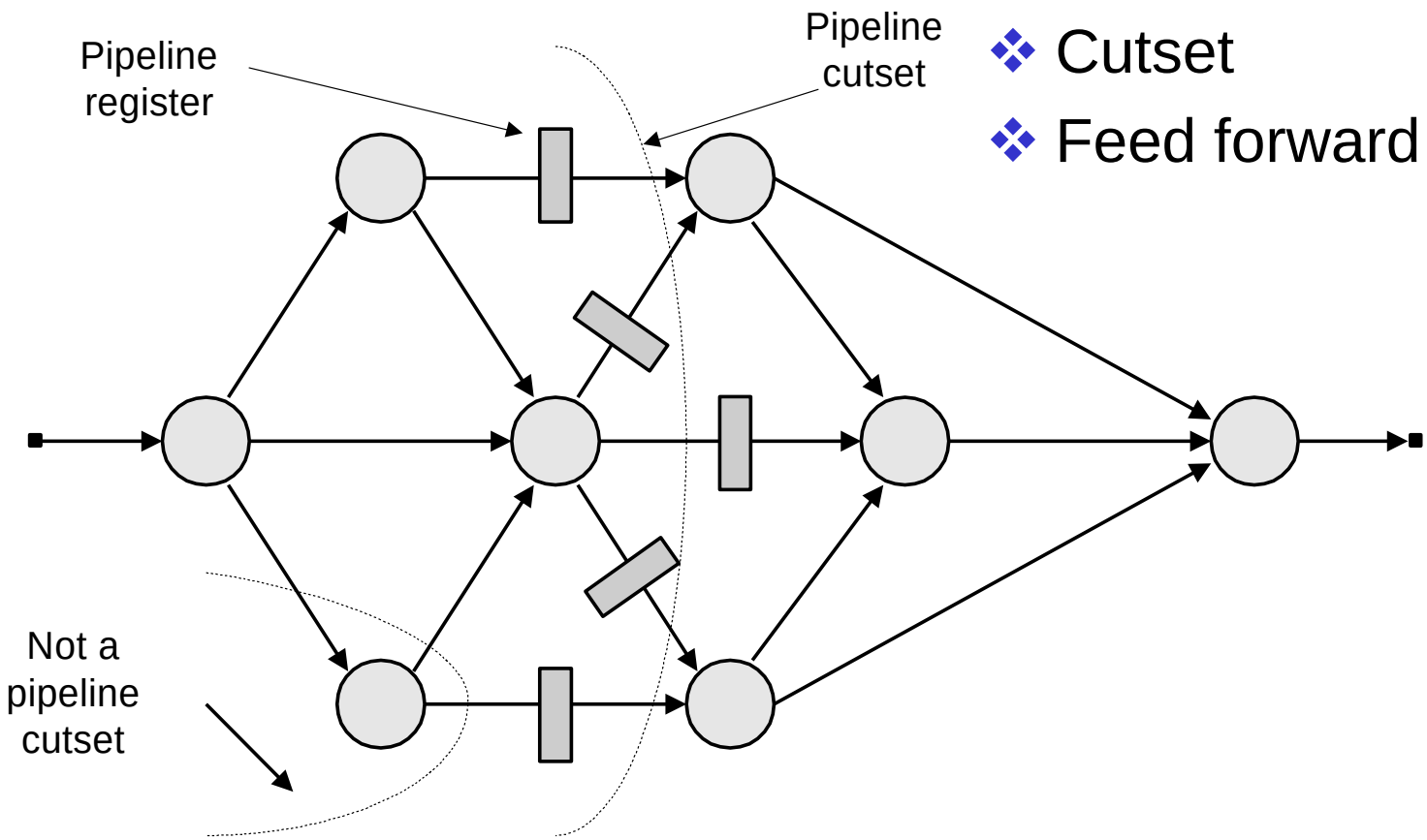
- Draw the circuit diagram in to a directed graph
- Put pipeline registers on feed-forward cut-set of the graph

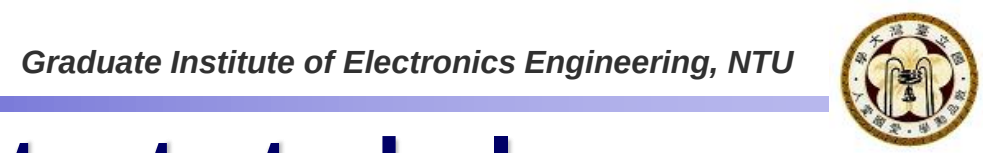
- A cutset is a set of edges of a graph such that if they are removed from the graph, the graph becomes disjoint.

- Feed-forward cut-set: all the removed edges have the same direction from one disjoint set to another. We call it pipeline cut-set.

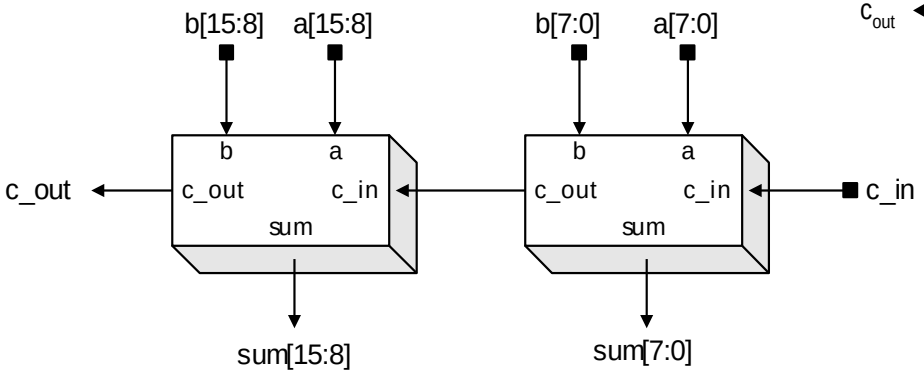
Example

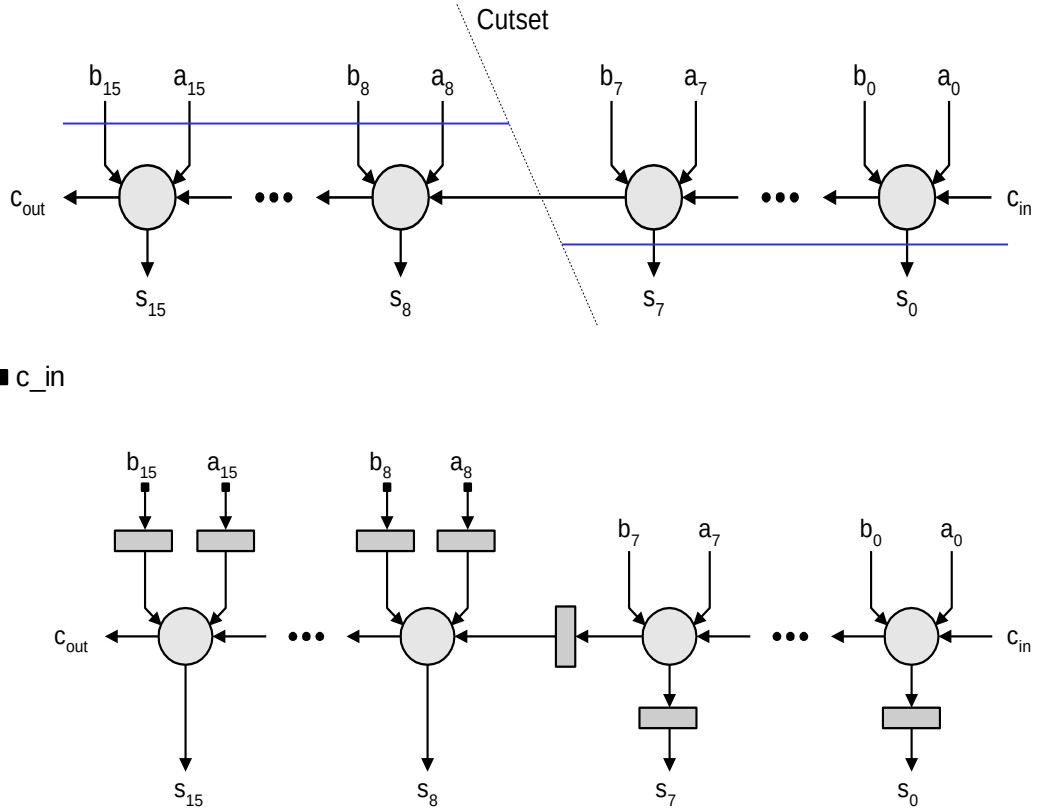
- Direct graph
- Cutset
- Feed forward cutset



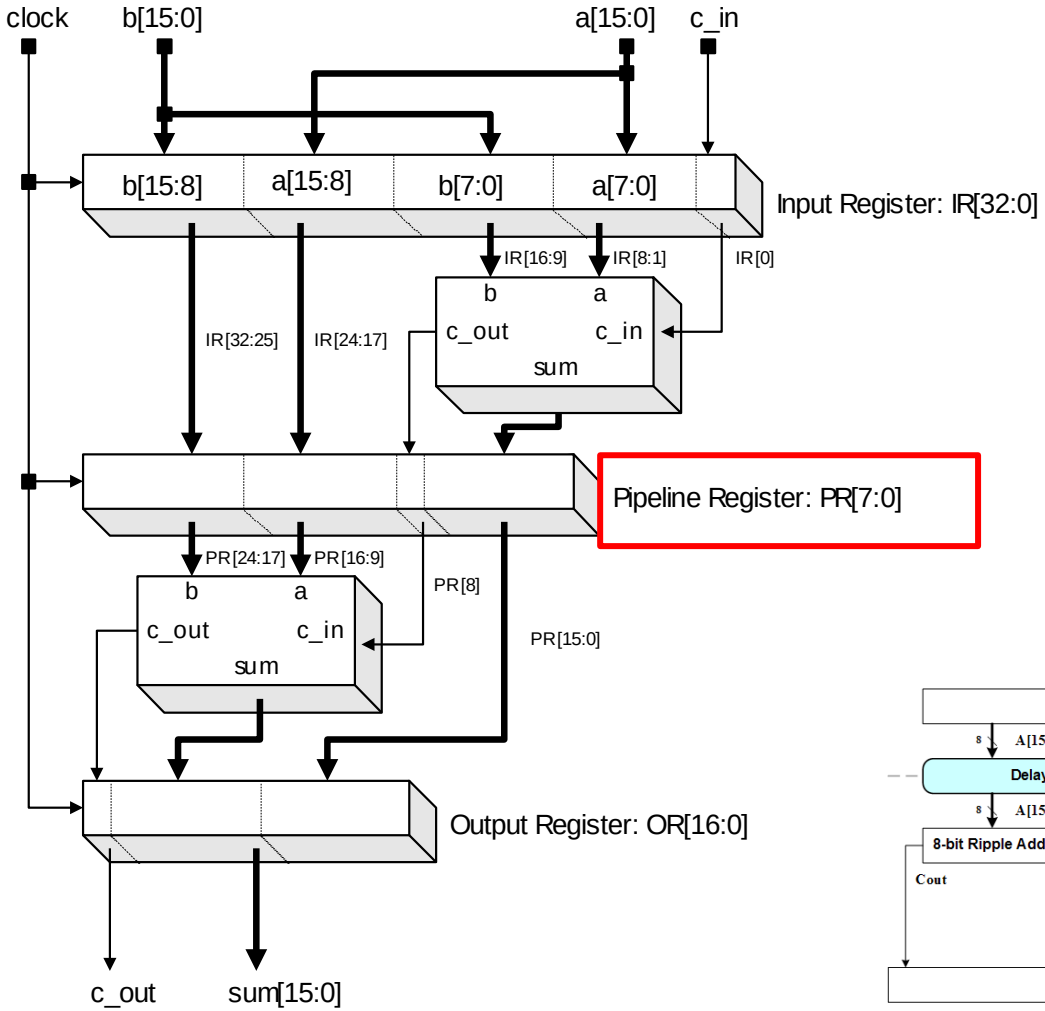


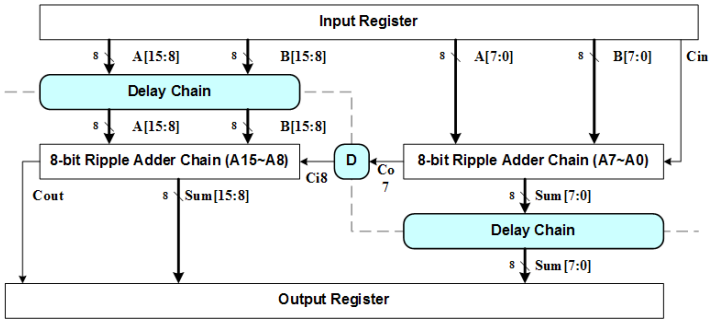
Choose cutsets to balance performance between stages





Pipelined 16-bit Adder Structure





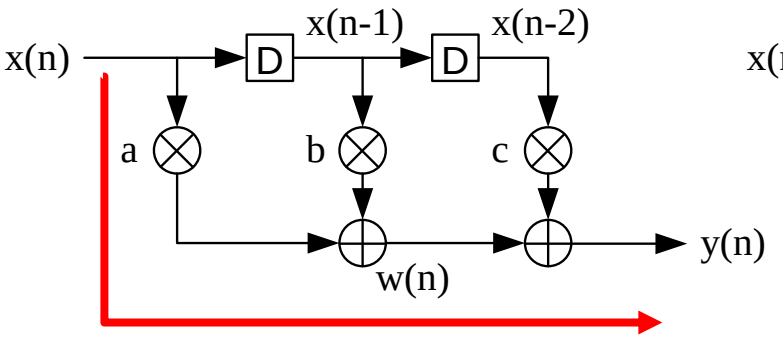
Example 3: FIR Filter

Original

$w(n) = ax(n) + bx(n-1)$

$\mathbf{y(n)} = cx(n-2) + w(n)$

$\mathbf{= ax(n) + bx(n-1) + cx(n-2)}$



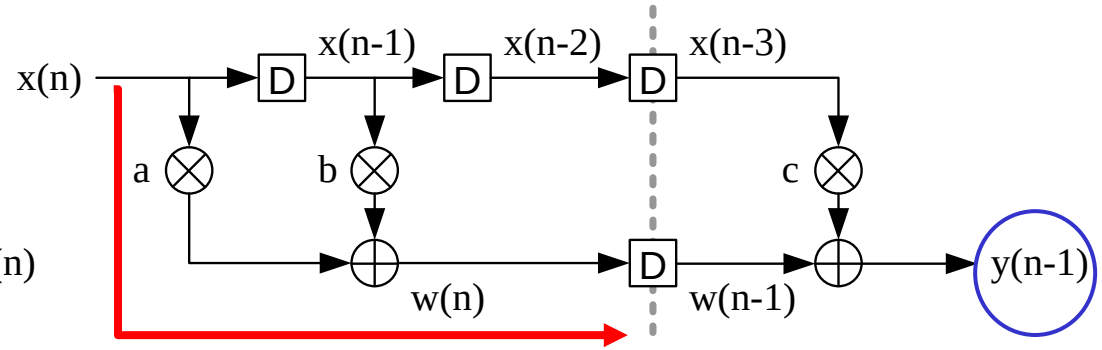
critical path = $T_{MUL} + 2T_{ADD}$

2-stage pipelining

$w(n-1) = ax(n-1) + bx(n-2)$

$\mathbf{y(n-1)} = cx(n-3) + w(n-1)$

$\mathbf{= ax(n-1) + bx(n-2) + cx(n-3)}$



Delay
1 cycle

critical path = $T_{MUL} + T_{ADD}$

Notes for Pipelining

- $T_{clk} = T_{sample}$
- Still has some limitations
  - Pipeline bubbles, data hazard…
- Effective pipelining
  - Put pipelining registers on the critical path
  - To approach the ideal separation, we needs to know architecture (gate-level) timing well for balancing the partition of the critical path.

How to deal with non feed forward cutset?

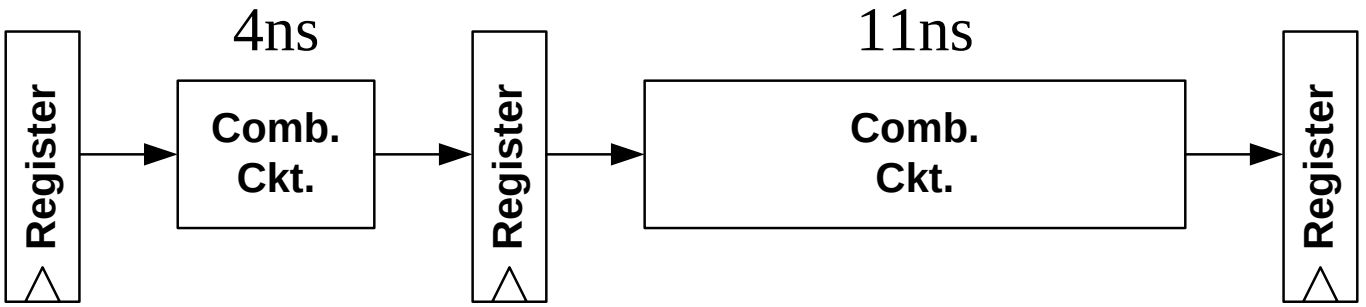
Introduction of Retiming

- Retiming is a technique to change the location of registers (delay elements) without affecting the input/output characteristics.

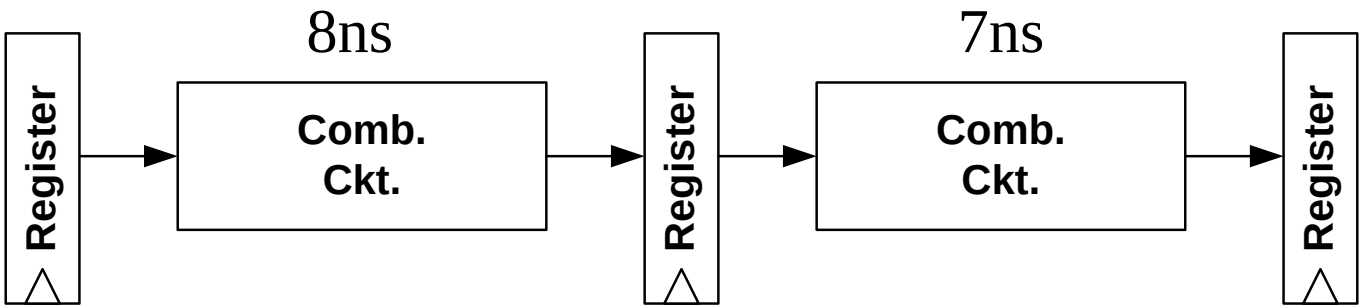
- Used to
  - Balance latency on different combinational path
  - Reduce clock period
- Note
  - May change the number of register in design
  - Usually applied by synthesis tool

Concept of Retiming

Original Clock period > 11ns

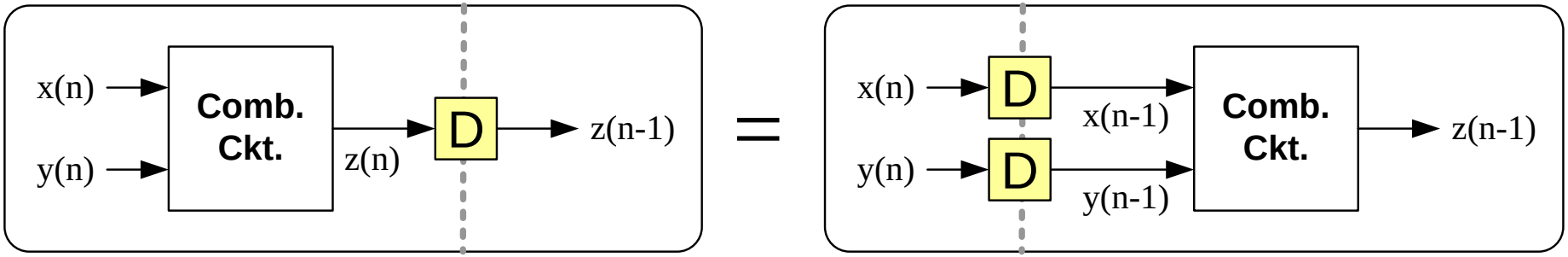


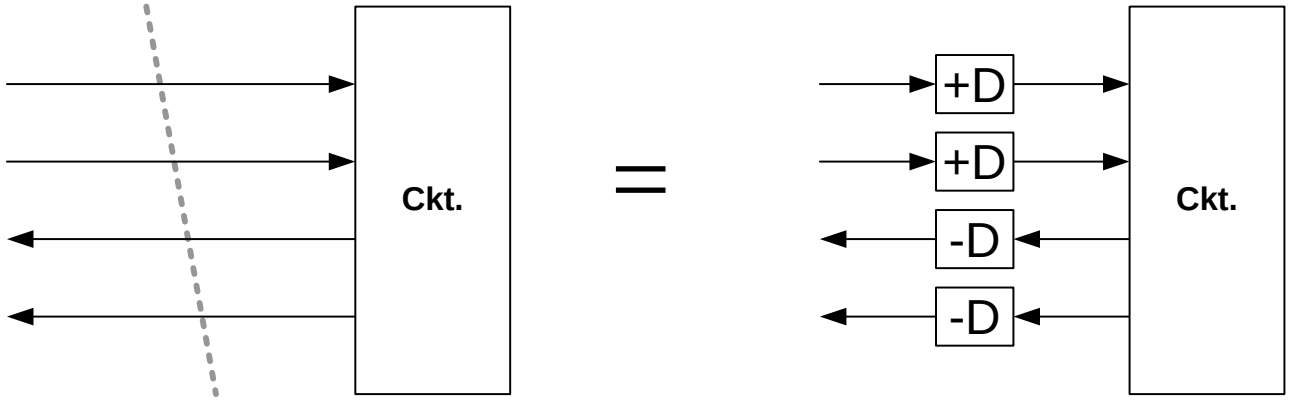
After Retiming Clock period > 8ns



How to do retiming?

- We simply introduce an approach called: “cut-set retiming”



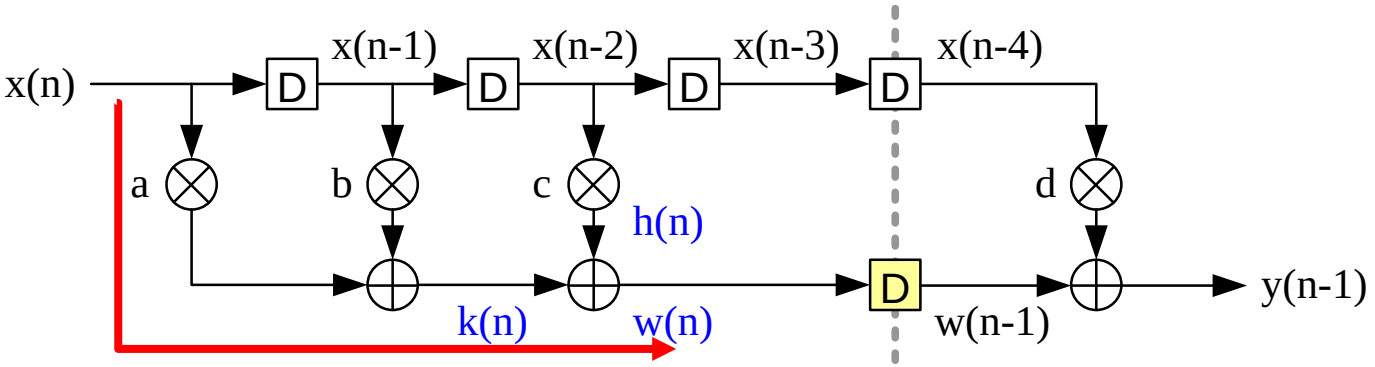


A cutset

Example of Retiming

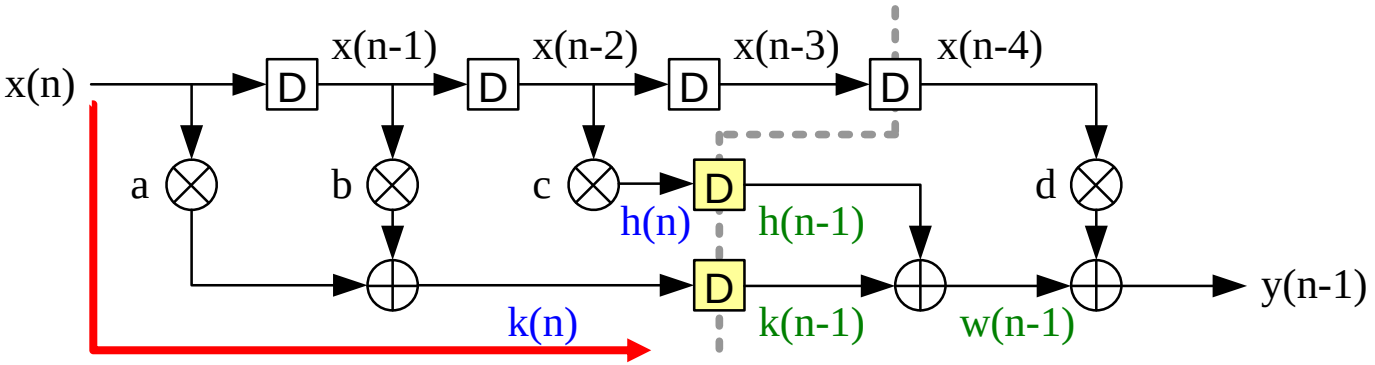
Original

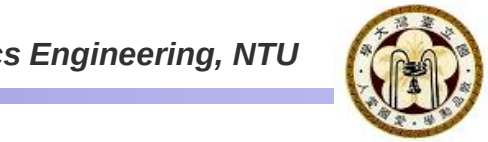
Clock period = $T_{MUL} + 2T_{ADD}$



After Retiming

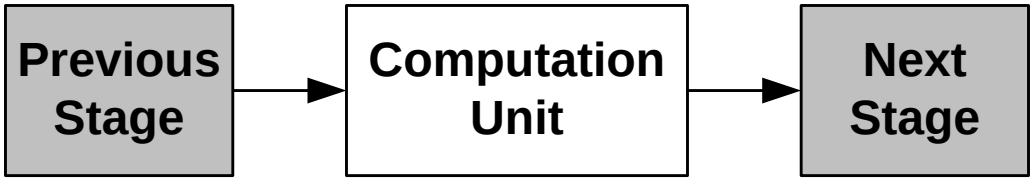
Clock period = $T_{MUL} + T_{ADD}$



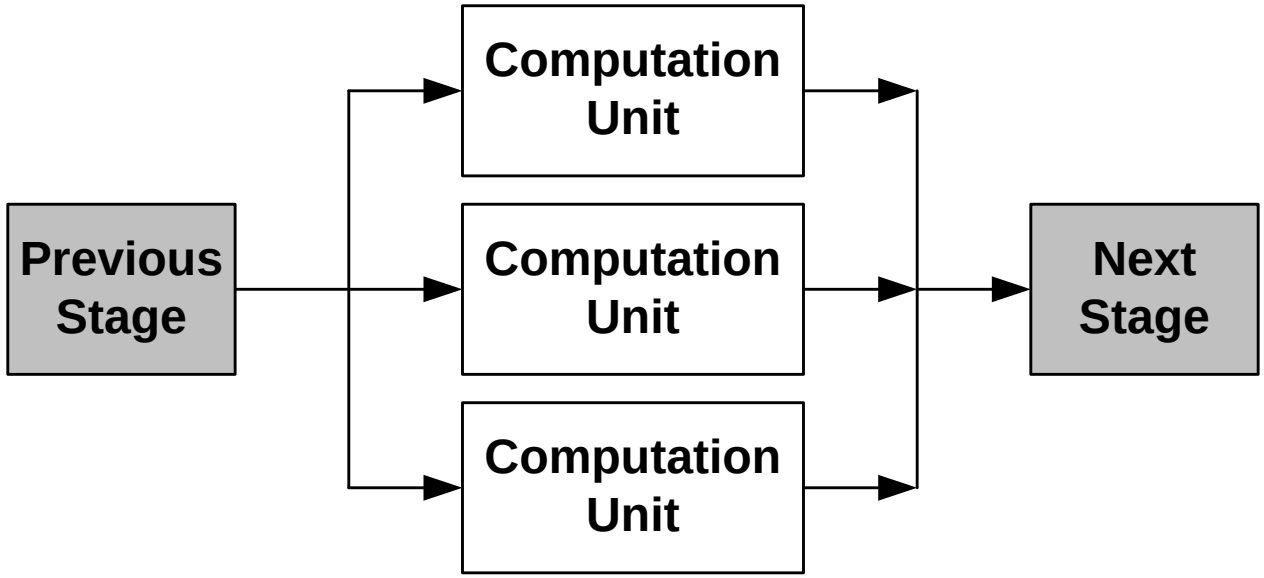


Concept of Parallelizing

Original



Parallelize by 3



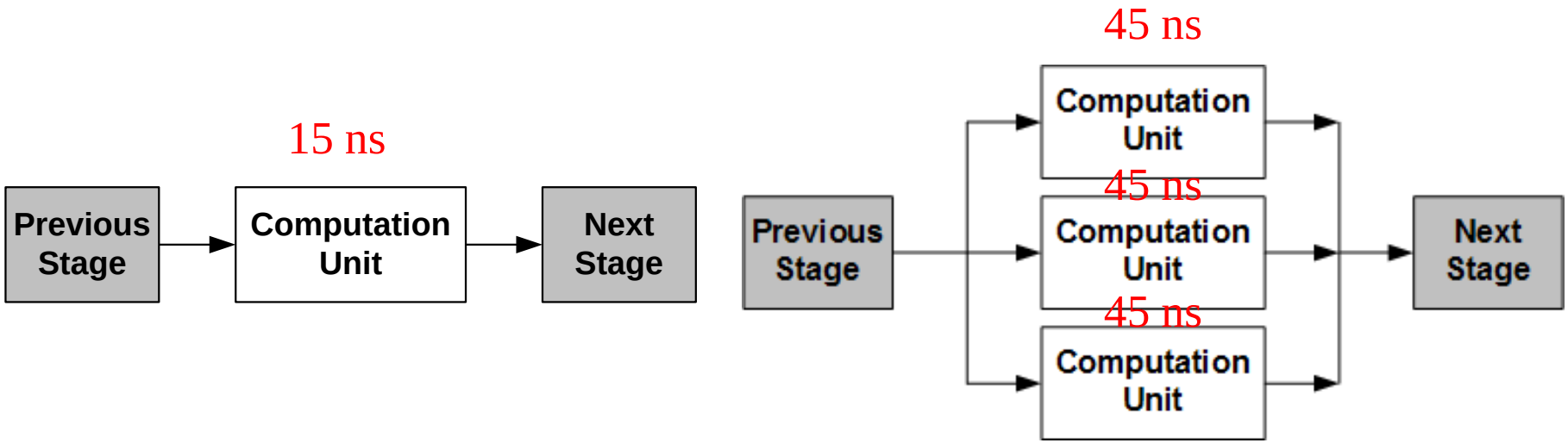
Drawback of Parallelizing

- Duplicated area!
- Duplicated **I/O bandwidth**! (limitation)
- Duplicated static power and dynamic power!!

- May not speed up as parallelizing factor
  - Dependency between data
  - Dependency between operations
  - Limitation of I/O bandwidth

Notes for Parallelizing

- $T_{clk} > T_{sample}$
- Large hardware cost
- The input/output data access scheme should be carefully designed, it will cost a lot sometimes

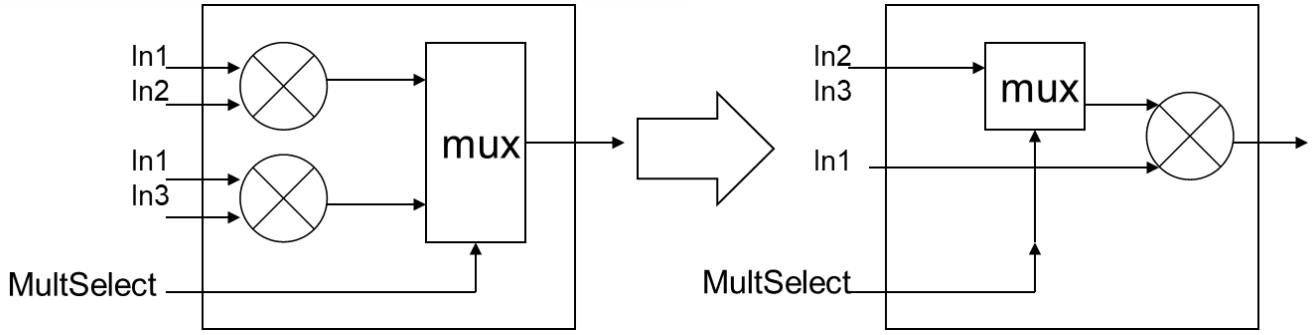


Area Issues

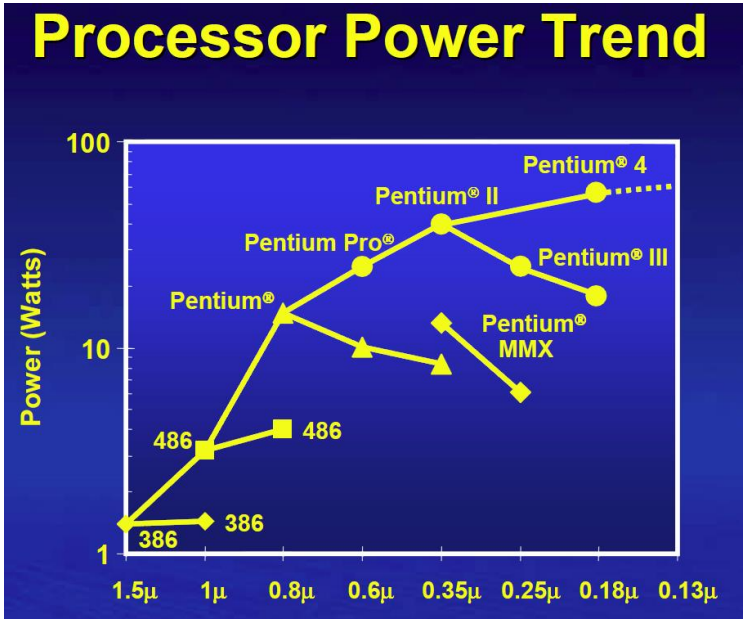
- Area = Cost.
- During the design process, the designer should be conscious of area.
- Resource sharing is the basic approach

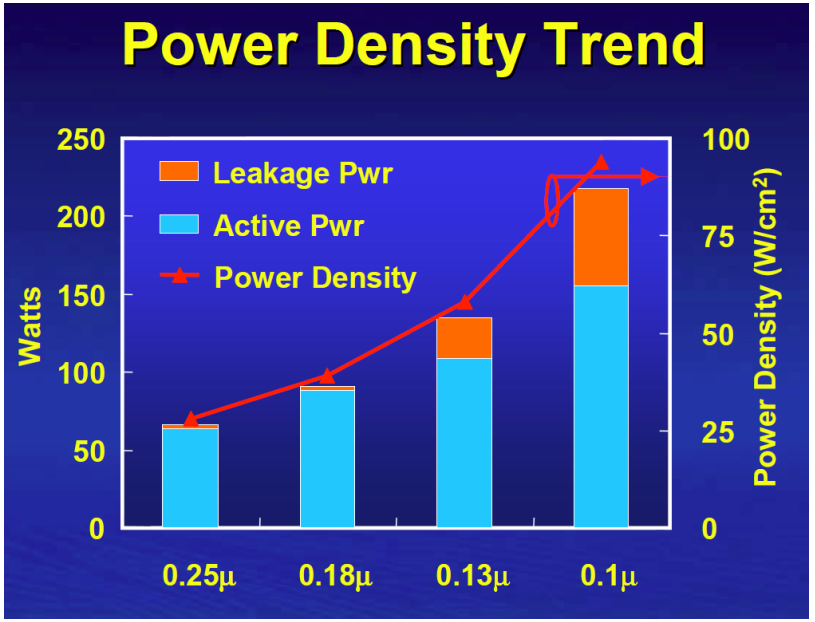
```
wire [15:0] A,B,C;
wire [7:0]  In1, In2, In3;
wire        MultSelect;

assign A = In1 * In2;
assign B = In1 * In3;
assign C = MultSelect ? A : B;
```

```
wire [15:0] C;
wire [7:0]  In1, In2, In3, InSelect;
wire        MultSelect;

assign InSelect = MultSelect ? In2 : In3;
assign C = In1 * InSelect;
```



Power Limitation





- Low power design is more and more important in today's chip design due to heat dissipation, packaging, and portability needs.
- Here we consider low power approaches to reduce dynamic power

Dynamic Power Consumption

- N_{node} is the switching activity
- f_{clock} is the clock frequency
- C_L is the node capacitance
- V_{dd} is the power supply voltage

$$P = N_{node} * C_L * V_{dd}^2 * f_{clock}$$

Strategy for Low Power Design (1/2)

- Lower V_{dd}
  - Determines transistor driving capability/speed
  - Dynamic voltage scaling
  - Usually technology-dependent

- C_L can only be minimized by back-end.

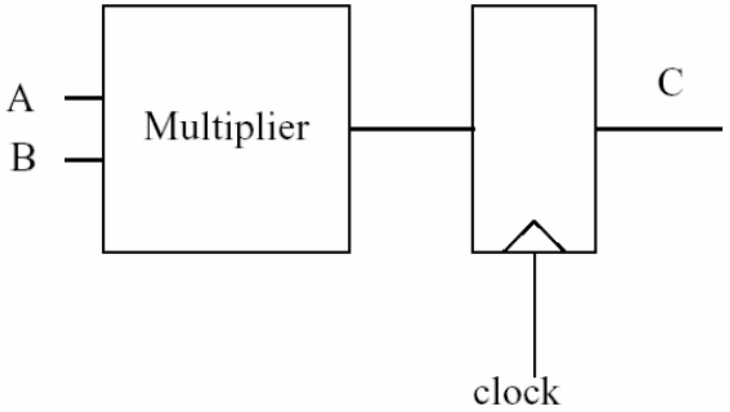
Strategy for Low Power Design (2/2)

- Reducing Clock Frequency
  - Design with clock rate that is 'just right'
  - **Clock Gating**
  - Slow down clock in power saving mode
- Reducing Switching Activity
  - Avoid unnecessary circuit switching
  - **Data Gating**
  - Reducing switching activity at I/O pins
- Assume the sampling frequency is the same, Pipeline and Parallel can reduce power
  - Cooperate with **voltage scaling technique**

Example: Reducing Activity for Multiplier

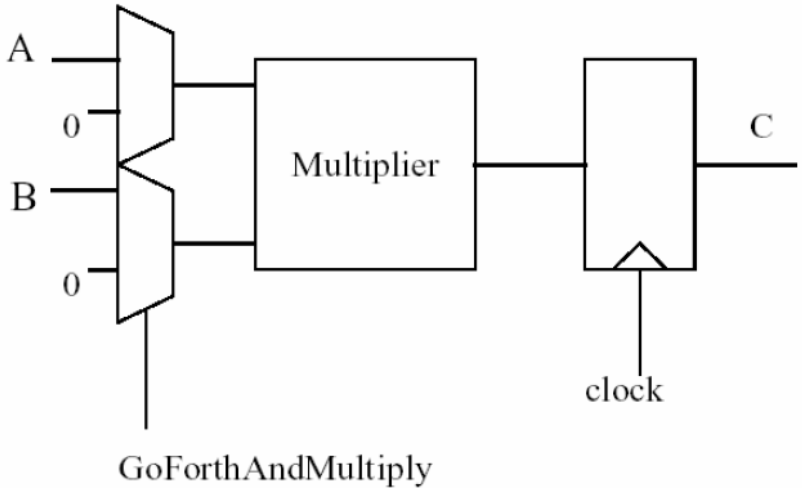
original

```
reg [31:0] C;
reg [15:0] A,B;

always@(posedge clock)
  C = A * B;
```



Low-power design

```
reg [31:0] C;
reg [15:0] A,B;
wire [15:0] Ain, Bin;
assign Ain = GoForthAndMultiply ? A : 16'h0;
assign Bin = GoForthAndMultiply ?  B : 16'h0;
always@(posedge clock)
    C = Ain * Bin;
```



Pipeline and Parallelizing for Low Power

- Pipelining and parallel processing can reduce power consumption by voltage scaling
- Propagation delay

On the critical path

$$T_{pd} = \frac{C_{charge} V_0}{k(V_0 - V_t)^2}$$

- Power consumption

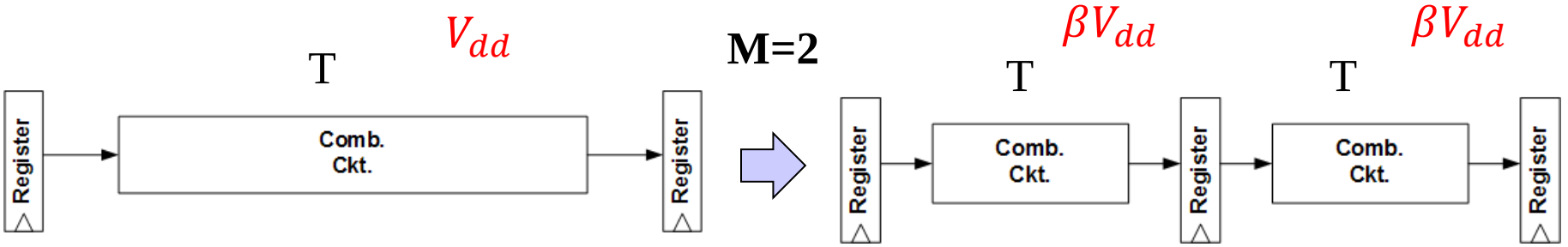
All capacitances

$$P = C_{total} V_0^2 f$$

- Assume the sampling frequency (input data rate) remains unchanged

Pipelining to Scale Down Vdd (1/2)

- For M-level pipelining
  - Critical path is reduced to 1/M
  - The capacitance in the critical path is also reduced by 1/M
  - The supply voltage can be reduced to βV_{dd} when the sampling period remains unchanged



Pipelining to Scale Down Vdd (2/2)

- Power consumption

$$P_{pip} = C_{total}\beta^2 V_{DD}^2 f = \beta^2 P$$

- How to determine β

$$T = \frac{C_{charge} V_{dd}}{k(V_{dd} - V_t)^2}$$

||

$$T_{pip} = \frac{\frac{C_{charge}}{M}\beta V_{dd}}{k(\beta V_{dd} - V_t)^2}$$

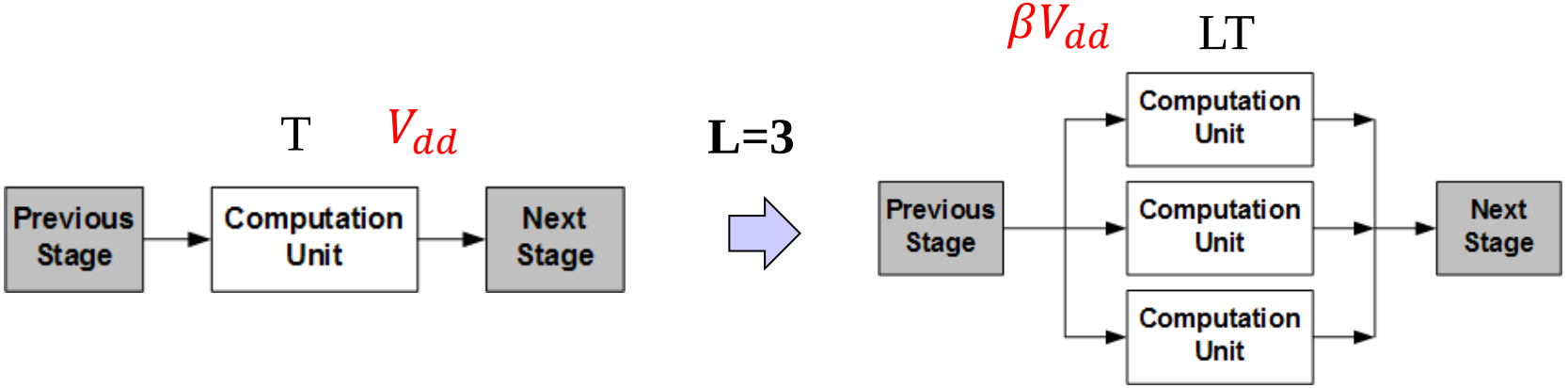
Given Vdd, Vt, M, we can solve β with this equation

$$M(\beta V_{dd} - V_t)^2 = \beta(V_{dd} - V_t)^2$$

If effective pipelining

Parallelizing to Scale Down Vdd (1/2)

- For L-parallel design
  - $T_{par} = LT$
  - The capacitance in the critical path remains unchanged
  - Total capacitances increase by L
  - Have more time to charge the capacitance, the supply voltage can be lower βV_{dd}



Parallelizing to Scale Down Vdd (1/2)

- Power consumption

$$P_{par} = (LC_{total})\beta^2 V_{DD}^2 \frac{f}{L} = \beta^2 P$$

- How to determine β

$$T = \frac{C_{charge} V_{dd}}{k(V_{dd} - V_t)^2}$$

Given Vdd, Vt, L, we can solve β with this equation

$$\mathrm{L}(\beta V_{dd} - V_t)^2 = \beta (V_{dd} - V_t)^2$$

substitute

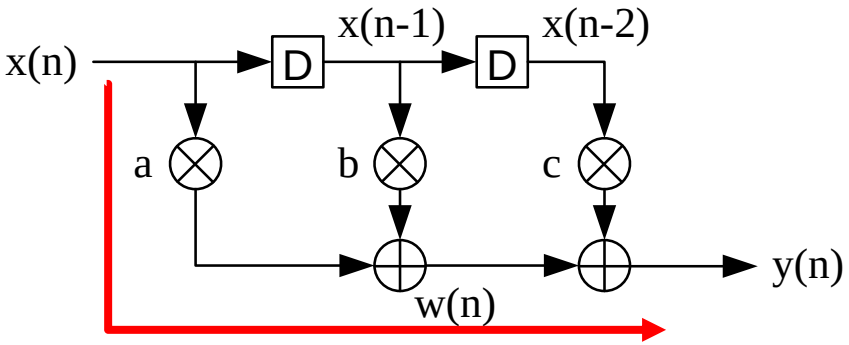
$$T_{par} = LT = \frac{C_{charge} \beta V_{dd}}{k(\beta V_{dd} - V_t)^2}$$

Example: Voltage Scaling on Pipelining (1/2)

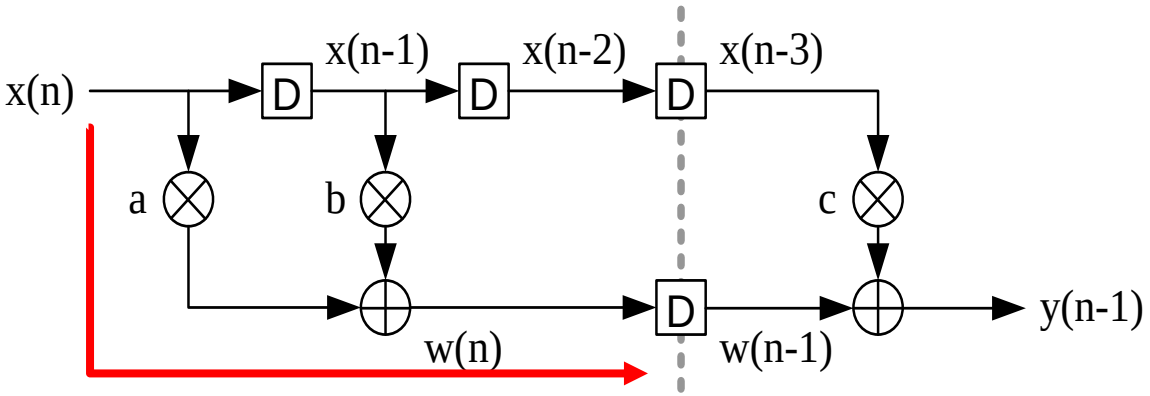
- Parameter
  - T_M=10u.t.
  - T_A=2 u.t.
  - T_{sample}=14 u.t.
  - C_M= $5C_A$
  - V_t=0.6V
  - Normal V_{dd}=5V

(a) New Vdd?

(b) Power saving percentage?



Assume same sampling rate



Example: Voltage Scaling on Pipelining (2/2)

- Parameter
  - T_M = 10 u.t.
  - T_A = 2 u.t.
  - T_{sample}=14 u.t.
  - $C_M = 5C_A$
  - V_t = 0.6V
  - Normal V_{dd} = 5V

$$T = \frac{7CV_{dd}}{k(V_{dd}-V_t)^2}$$

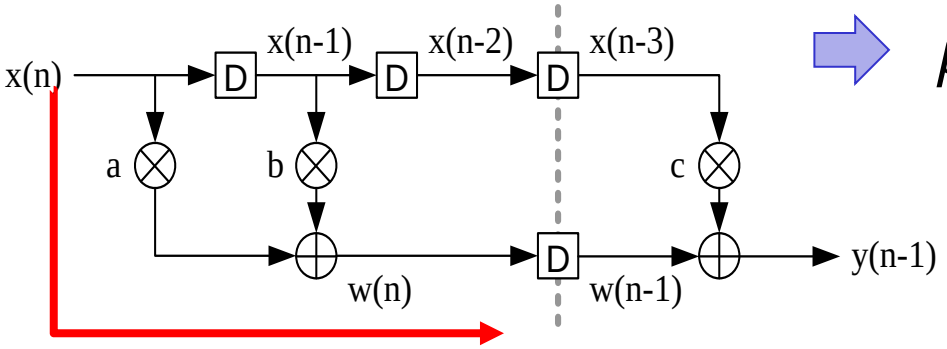
$$\| $$

$$T_{pip} = \frac{6C\beta V_{dd}}{k(\beta V_{dd}-V_t)^2}$$

$$\Rightarrow \frac{7}{6}(5\beta - 0.6)^2 = \beta(5-0.6)^2$$

$$\Rightarrow \beta = 0.8875 \text{ or } 0.016$$

Invalid, <Vt



$\Rightarrow$ New V_{dd}=4.4375 V

Power saving=1-β^2=21.2%