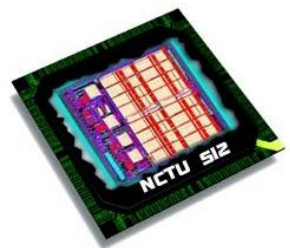


# Advanced Sequential Circuit Design

NCTU-EE IC Lab Fall 2020

Lecturer : Jing-Yang, Huang



# Outline

- ✓ **Section 1- Timing**
- ✓ **Section 2- Designware**



# Outline

## ✓ **Section 1- Timing**

- Setup/hold time
- Pipeline

## ✓ **Section 2- Designware**



# Timing Issue

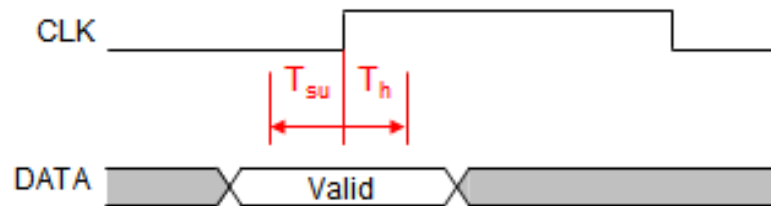
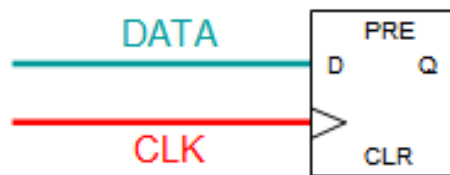
## ✓ Terminology

- Setup time ( $t_{setup}$ )

The time that the input signal must be stabilized **before** the clock edge.

- Hold time ( $t_{hold}$ )

The time that the input signal must be stabilized **after** the clock edge.



# Timing Issue

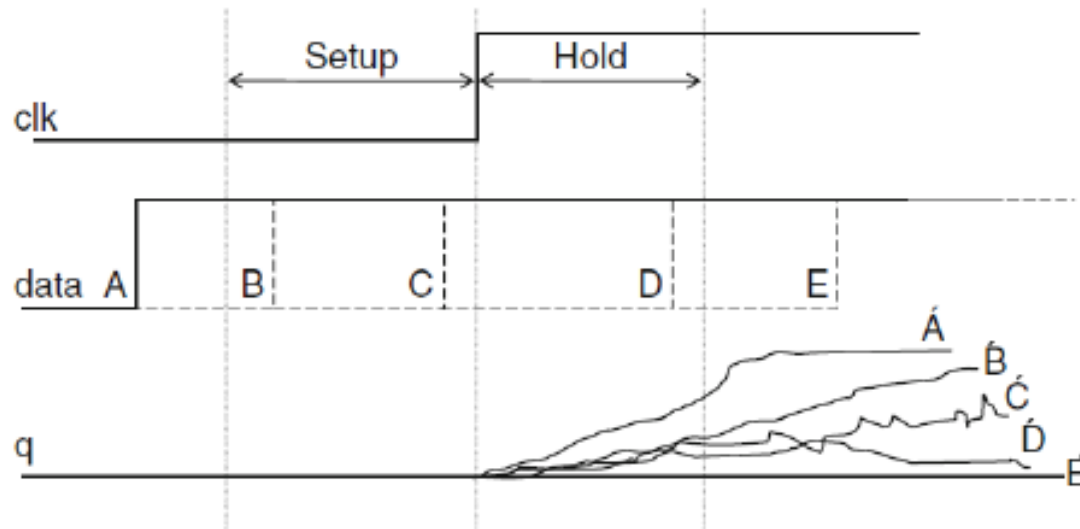
## ✓ Terminology

- Setup time ( $t_{setup}$ )

The time that the input signal must be stabilized **before** the clock edge.

- Hold time ( $t_{hold}$ )

The time that the input signal must be stabilized **after** the clock edge.



# Timing Issue

## ✓ Terminology

### – Contamination delay

The minimum amount of time from an **input changes** until **any output starts to change** its value.

- Clk-to-Q contamination delay ( $t_{ccq}$ )
- Logic contamination delay ( $t_{cd}$ )

### – Propagation delay

The maximum amount of time from **input changes** until **all output reaches steady state**.

- Clk-to-Q propagation delay ( $t_{pcq}$ )
- Logic propagation delay ( $t_{pd}$ )



# Timing Issue

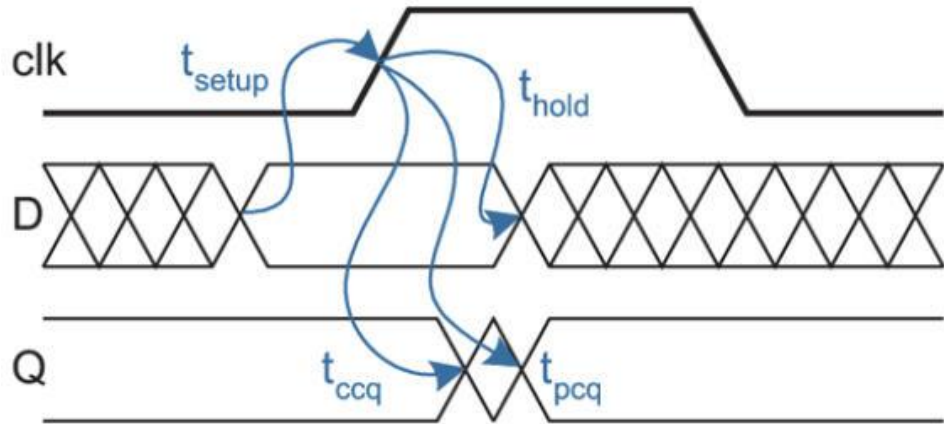
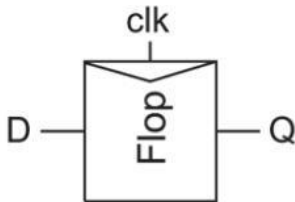
## ✓ Term definition

- Clk-to-Q contamination delay ( $t_{ccq}$ )

The minimum amount of time from an **clock edge** until **Q starts to change** its value.

- Clk-to-Q propagation delay ( $t_{pcq}$ )

The maximum amount of time from an **clock edge** until **Q reaches steady state**.



# Timing Issue

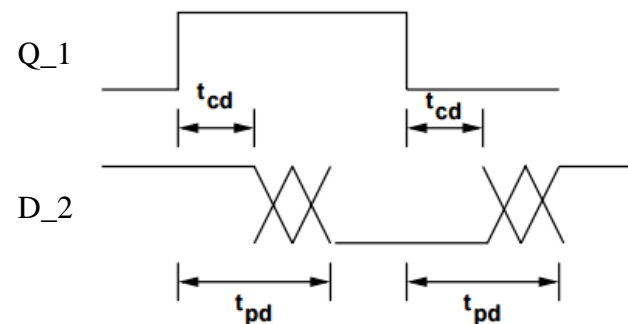
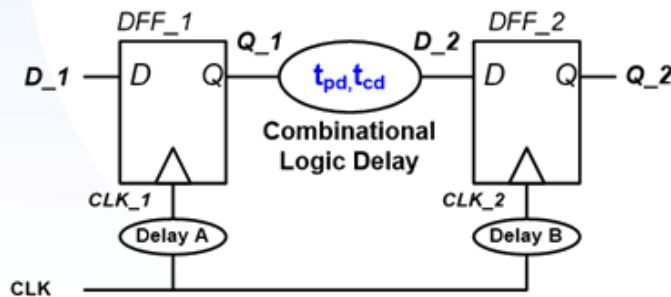
## ✓ Term definition

- Logic contamination delay ( $t_{cd}$ )

The minimum amount of time from a combinational logic input (Q\_1) changes until combinational logic output (D\_2) starts to change its value.

- Logic propagation delay ( $t_{pd}$ )

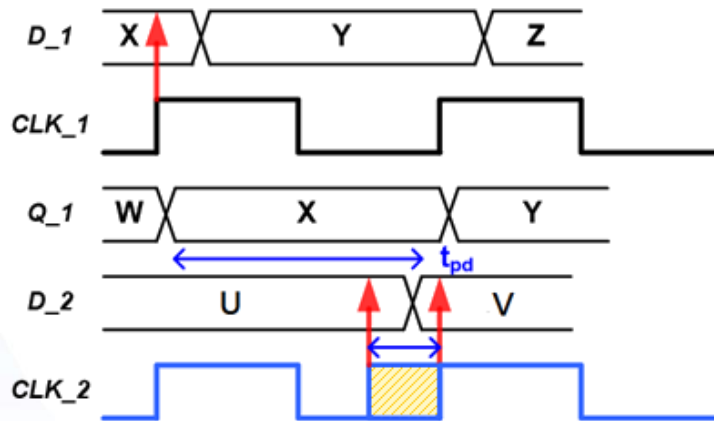
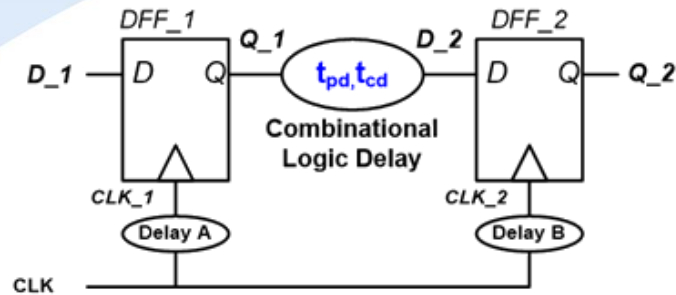
The maximum amount of time from a combinational logic input (Q\_1) changes until combinational logic output (D\_2) reaches steady state.



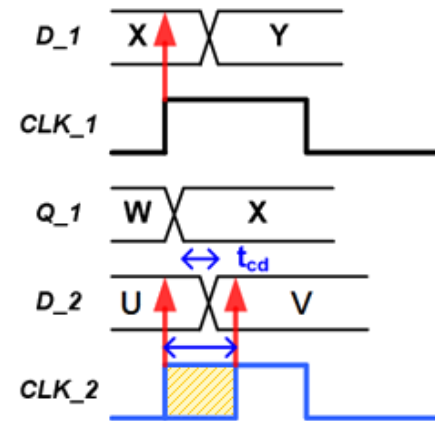


# Timing Violation

## ✓ Timing violation



Setup Time Violation

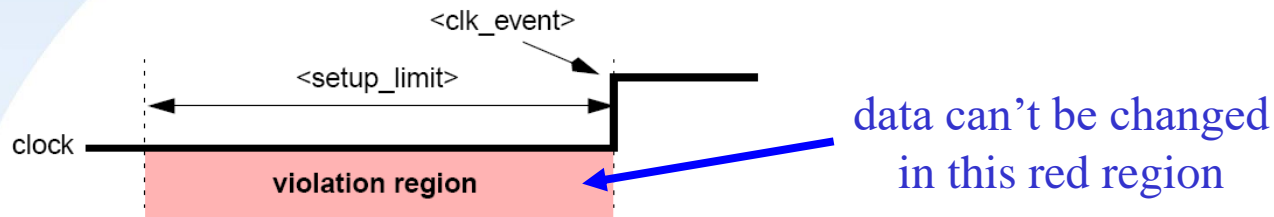


Hold Time Violation

# Timing Check (1/2)

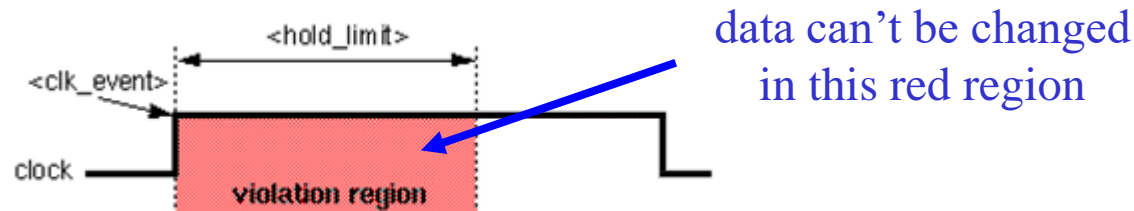
## ✓ Setup time check

- The tool determines whether a data signal **remains stable** for a minimum specified time (i.e., violation region) **before** a transition in an enabling signal, such as a clock event.



## ✓ Hold time check

- The tool determines whether a data signal **remains stable** for a minimum specified time (i.e., violation region) **after** a transition in an enabling signal, such as a clock event.



# Timing Check (2/2)

## ✓ Timing report: setup time

clock CLK_1 (rise edge)	2.00	2.00
clock network delay (ideal)	2.00	4.00
clock uncertainty	-0.50	3.50
IN_A_reg[0]/CK (EDFFXL)	0.00	3.50 r
library setup time	-0.42	3.08
data required time		3.08
-----		
data required time		3.08
data arrival time		-3.08
-----		
slack (MET)		0.00

## ✓ Timing report: hold time

Slacks should be **MET!**  
(non-negative)

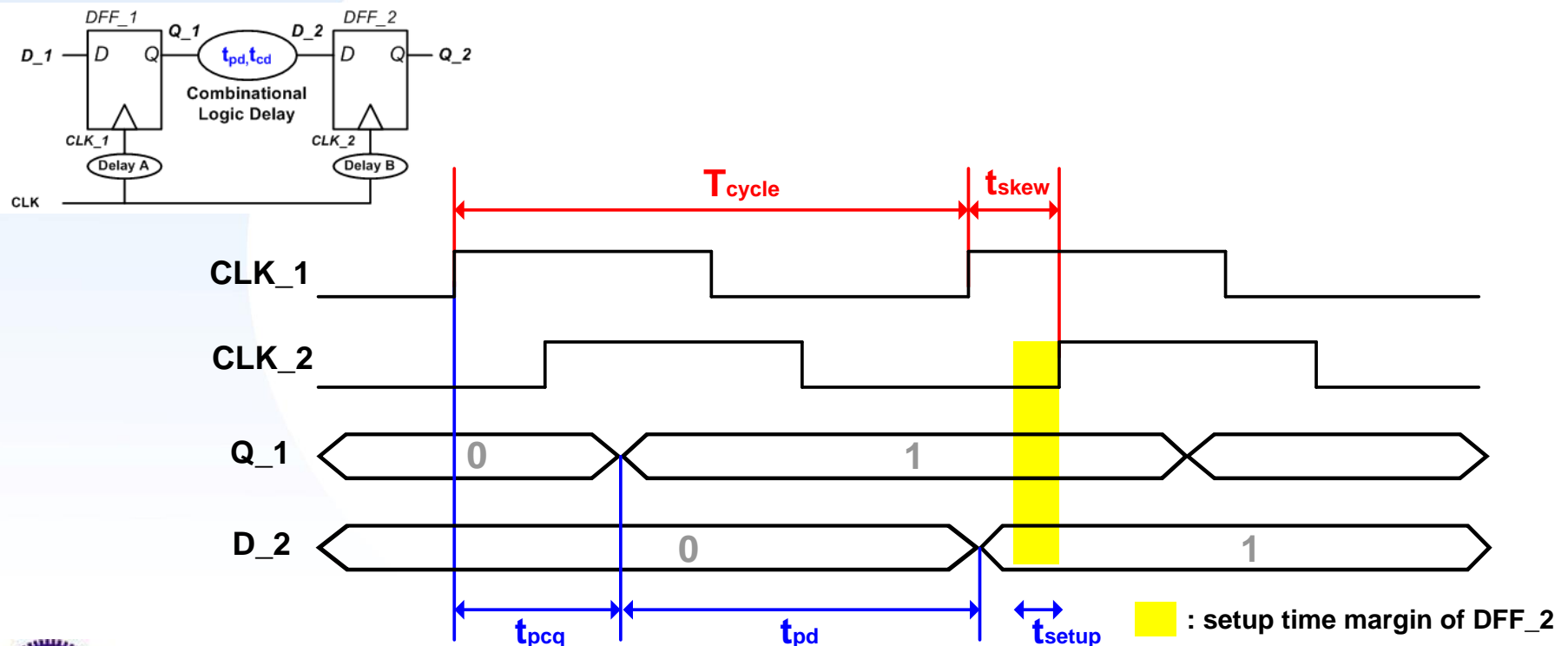
clock CLK_2 (rise edge)	0.00	0.00
clock network delay (ideal)	4.00	4.00
clock uncertainty	1.00	5.00
IN_B_reg[20]/CK (EDFFXL)	0.00	5.00 r
library hold time	-0.19	4.81
data required time		4.81
-----		
data required time		4.81
data arrival time		-4.82
-----		
slack (MET)		0.01



# Setup Time Criterion

✓ **Setup time criterion:**  $(T_{cycle} + t_{skew}) > (t_{pcq} + t_{pd} + t_{setup})$

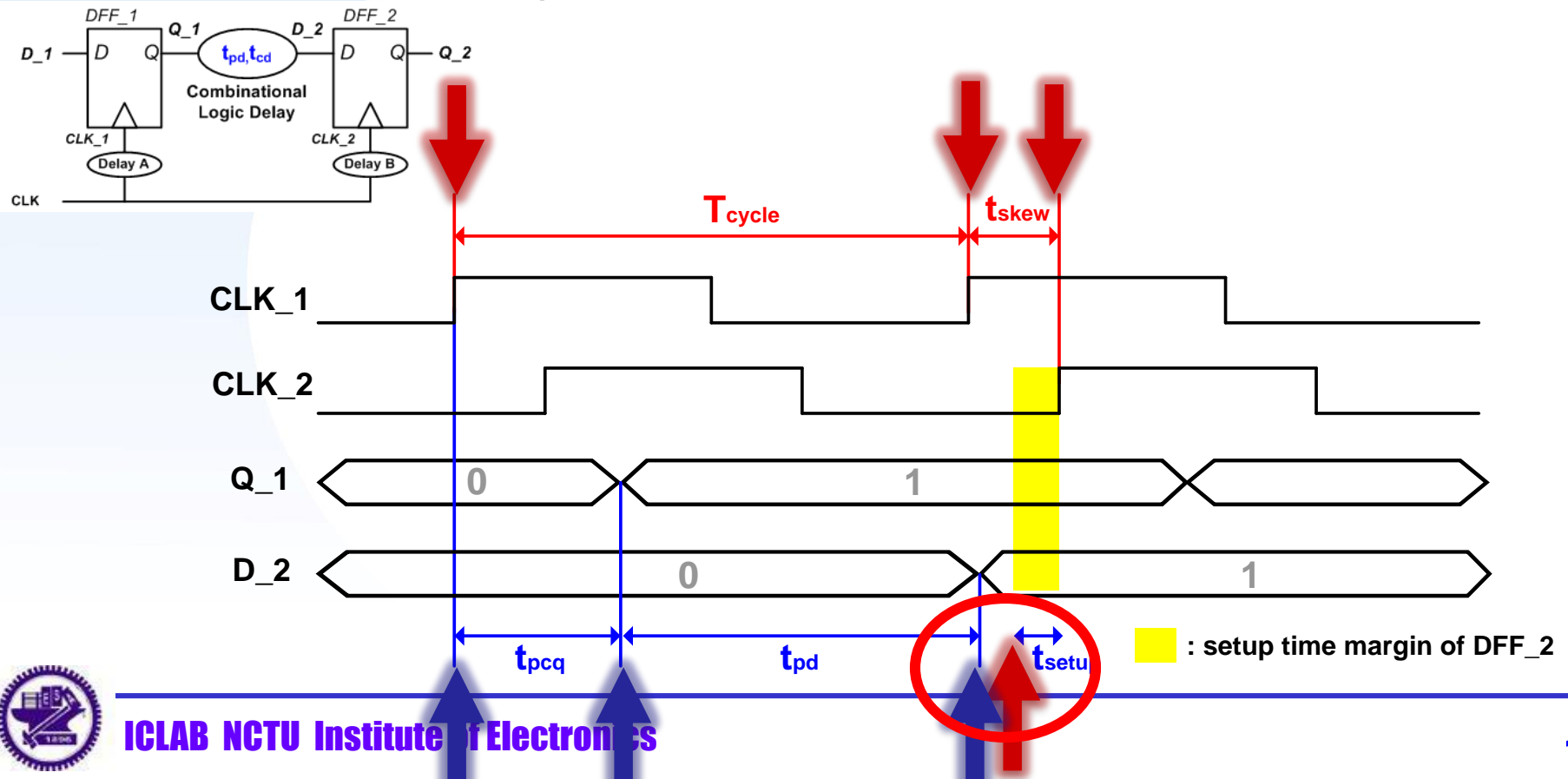
- data required time =  $T_{cycle} + t_{skew} - t_{setup}$
- data arrival time =  $t_{pcq} + t_{pd}$
- Slack = data required time - data arrival time



# Setup Time Criterion

✓ **Setup time criterion:**  $(T_{cycle} + t_{skew}) > (t_{pcq} + t_{pd} + t_{setup})$

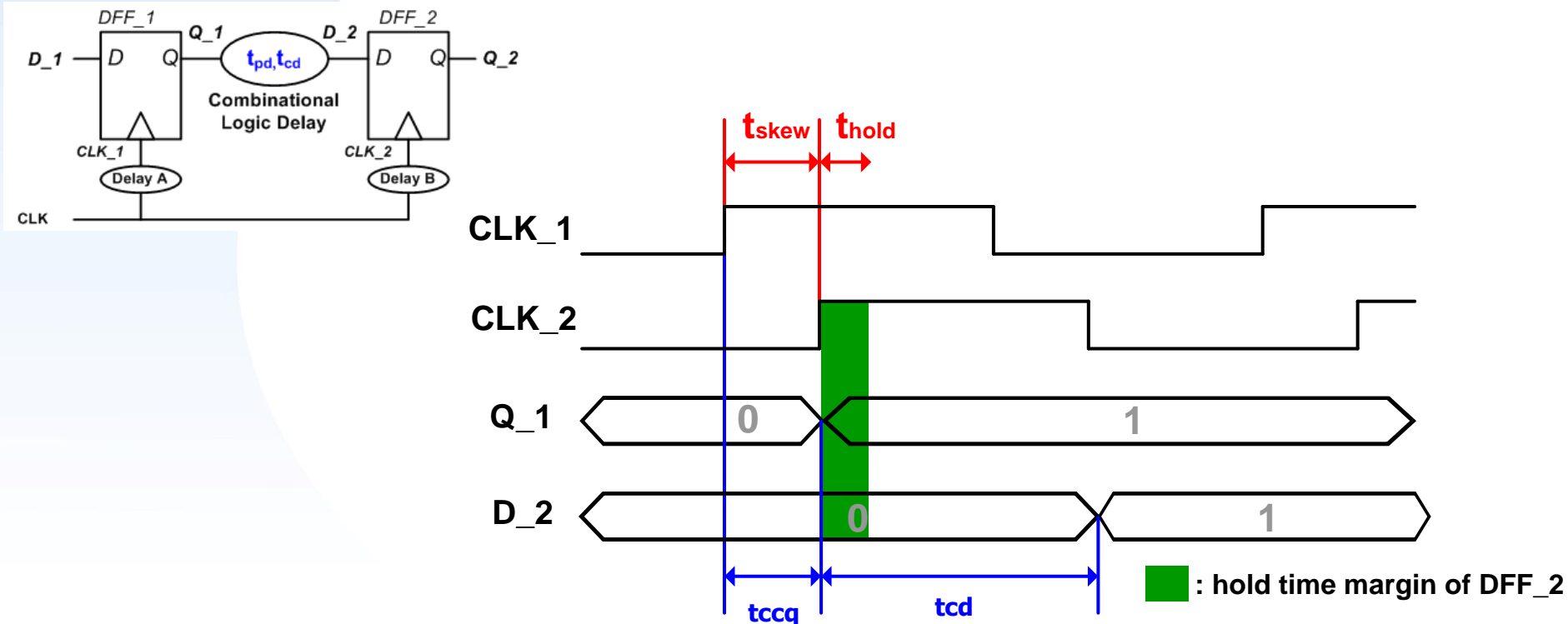
- data required time =  $T_{cycle} + t_{skew} - t_{setup}$
- data arrival time =  $t_{pcq} + t_{pd}$
- Slack = data required time - data arrival time



# Hold Time Criterion

✓ **Hold time criterion:**  $(t_{ccq} + t_{cd}) > (t_{hold} + t_{skew})$

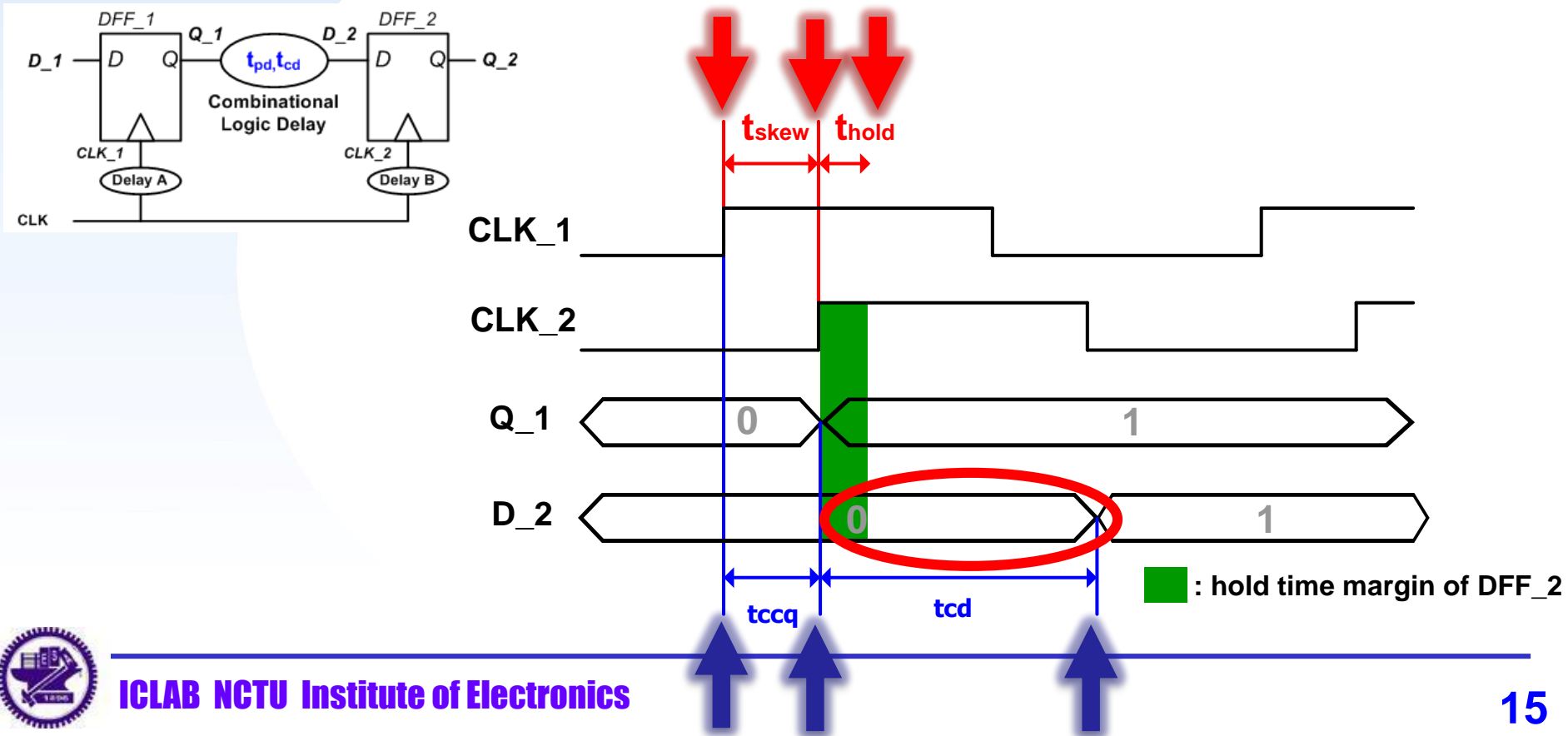
- data required time =  $t_{skew} + t_{hold}$
- data arrival time =  $t_{ccq} + t_{cd}$
- Slack = data arrival time – data required time



# Hold Time Criterion

✓ Hold time criterion:  $(t_{ccq} + t_{cd}) > (t_{hold} + t_{skew})$

- data required time =  $t_{skew} + t_{hold}$
- data arrival time =  $t_{ccq} + t_{cd}$
- Slack = data arrival time – data required time



# When Timing Violation Occurs...

- ✓ **Adjust data path to meet the constraints**
  - Setup violation → too many works in one cycle
    - Apply pipelining
  - Hold violation → insufficient delay
    - add delays to the violated path, such as buffers/inverters/Muxes
- ✓ **Increase clock period for setup violation**
- ✓ **In most practical cases, hold violations are fixed during the backend work (after clock tree synthesis)**





# Outline

## ✓ **Section 1- Timing**

- Setup/hold time
- Pipeline

## ✓ **Section 2- Designware**

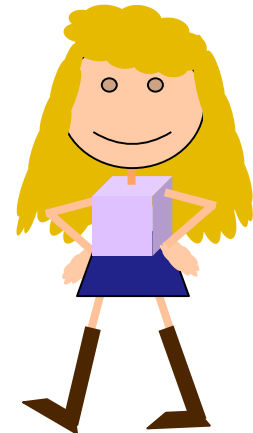
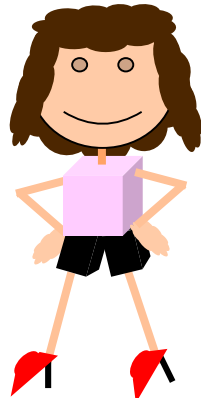
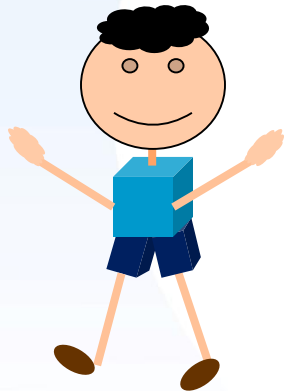


# Trade-off between Area and Timing

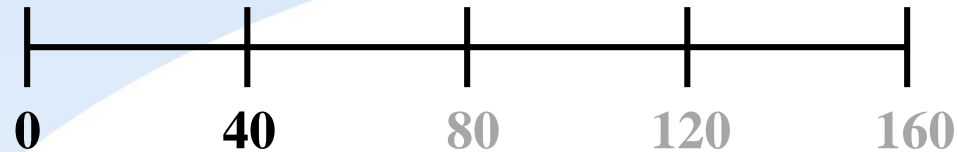


Area: 1 unit

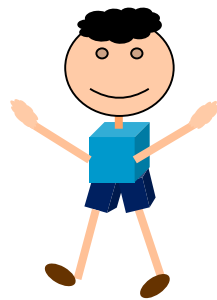
Time: 40 mins (Wash: 20 mins + Dry: 20 mins)



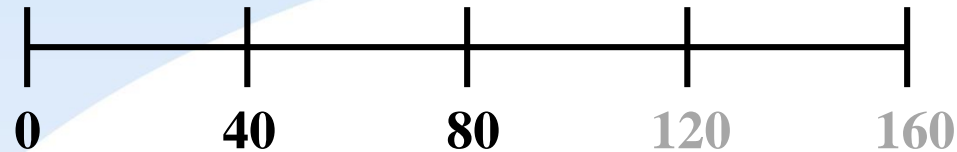
# Trade-off between Area and Timing



**Wash and Dry = 40 mins**



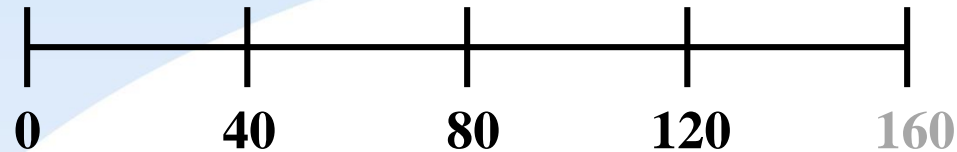
# Trade-off between Area and Timing



**Wash and Dry = 40 mins**



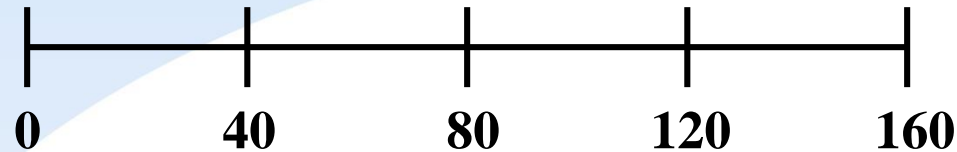
# Trade-off between Area and Timing



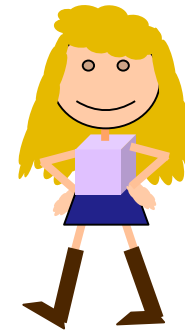
Wash and Dry = 40 mins



# Trade-off between Area and Timing



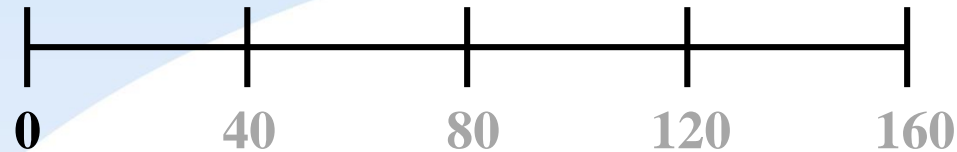
**Wash and Dry = 40 mins**



**Area: 1 unit**

**Time: 160 mins**

# Trade-off between Area and Timing



**Wash and Dry = 40 mins**

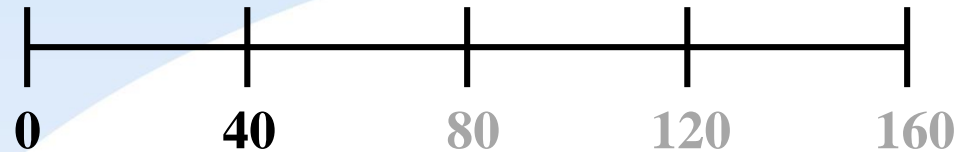
**Area: 1 unit**

**Time: 160 mins**

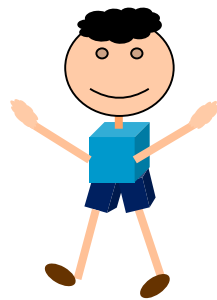


**Wash and Dry = 40 mins**

# Trade-off between Area and Timing



**Wash and Dry = 40 mins**

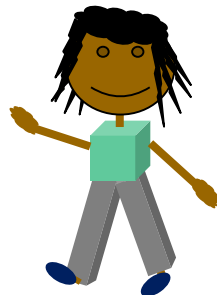


**Area: 1 unit**

**Time: 160 mins**

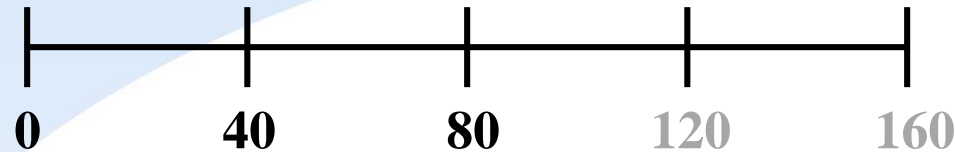


**Wash and Dry = 40 mins**





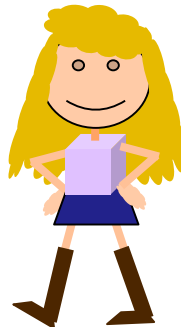
# Trade-off between Area and Timing



**Wash and Dry = 40 mins**



**Wash and Dry = 40 mins**

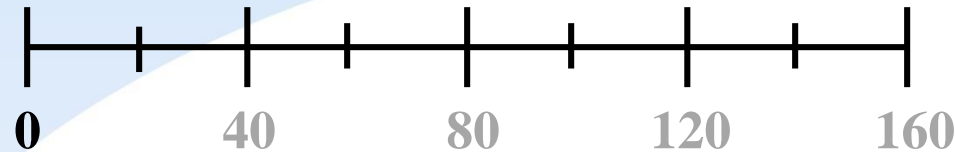


~~Area: 1 unit  
Time: 160 mins~~



**Area: 2 units  
Time: 80 mins**

# Trade-off between Area and Timing



**Wash 20 mins**  
**Area 0.7 units**

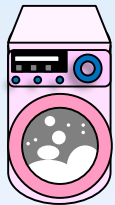
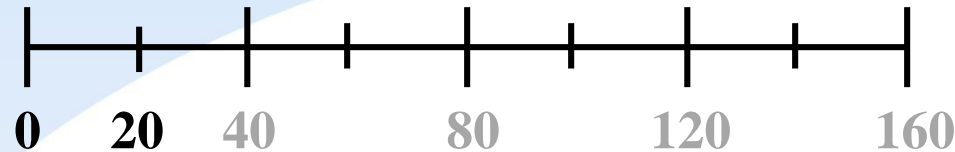


**Dry 20 mins**  
**Area 0.7 units**

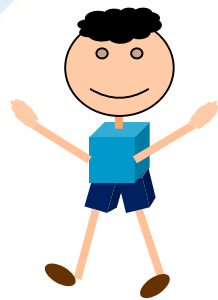
**Area: 1 unit**

**Time: 160 mins**

# Trade-off between Area and Timing



**Wash 20 mins**



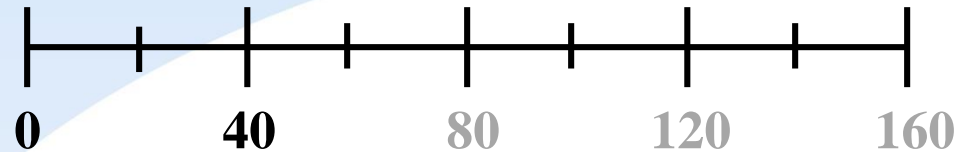
**Area: 1 unit**

**Time: 160 mins**



**Dry 20 mins**

# Trade-off between Area and Timing



**Wash 20 mins**

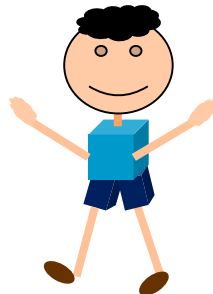


**Area: 1 unit**

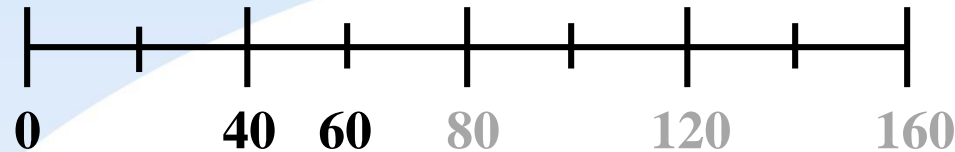
**Time: 160 mins**



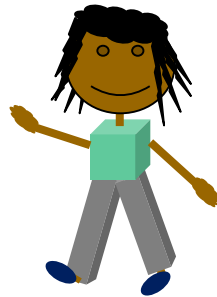
**Dry 20 mins**



# Trade-off between Area and Timing



**Wash 20 mins**



**Area: 1 unit**

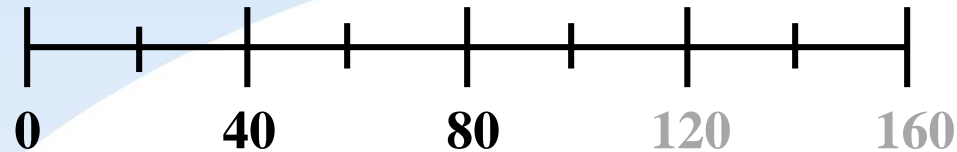
**Time: 160 mins**



**Dry 20 mins**



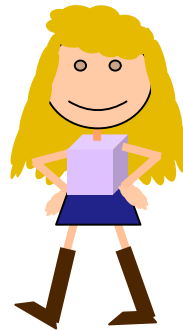
# Trade-off between Area and Timing



**Wash 20 mins**



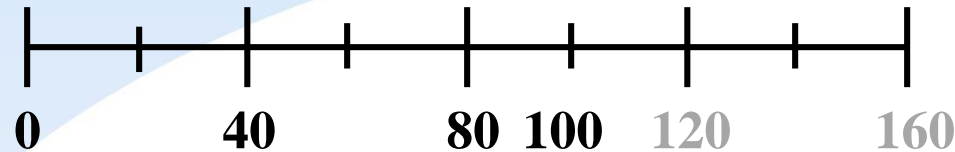
**Dry 20 mins**



**Area: 1 unit**

**Time: 160 mins**

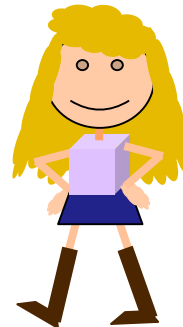
# Trade-off between Area and Timing



**Wash 20 mins**



**Dry 20 mins**



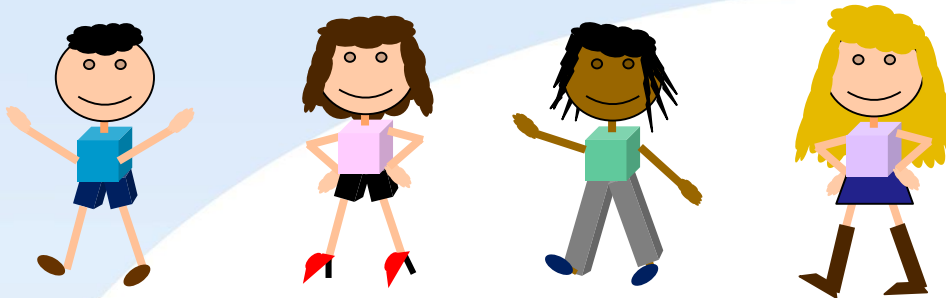
~~Area: 1 unit  
Time: 160 mins~~



**Area:  $0.7+0.7 = 1.4$  units**

**Time: 100 mins**

# Trade-off between Area and Timing

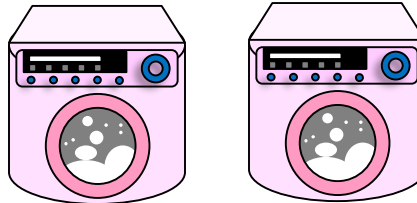


**Basic**



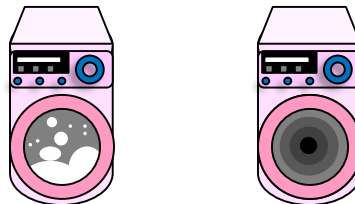
**Area: 1 unit**  
**Time: 160 mins**

**Parallel**



**Area: 2 units**  
**Time: 80 mins**

**Pipeline**



**Area:  $0.7+0.7 = 1.4$  units**  
**Time: 100 mins**



# Trade-off between Area and Timing

- ✓  $a [7:0]$  ,  $b [7:0]$  ,  $c [3:0]$  ,  $d [3:0]$
- ✓ Q:  $(a + b + c + d) \times 100$  iterations ?

Basic

Parallel

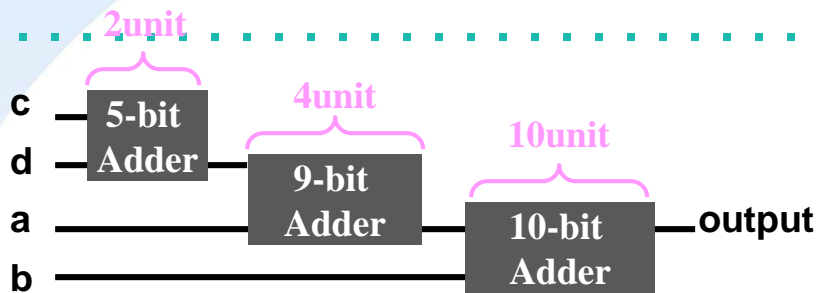
Pipeline



# Trade-off between Area and Timing

- ✓  $a [7:0]$  ,  $b [7:0]$  ,  $c [3:0]$  ,  $d [3:0]$
- ✓ Q:  $(a + b + c + d) \times 100$  iterations ?

**Basic**



Area: 24 units

Time:  $16 \times 100 = 1600$   
units

**Parallel**

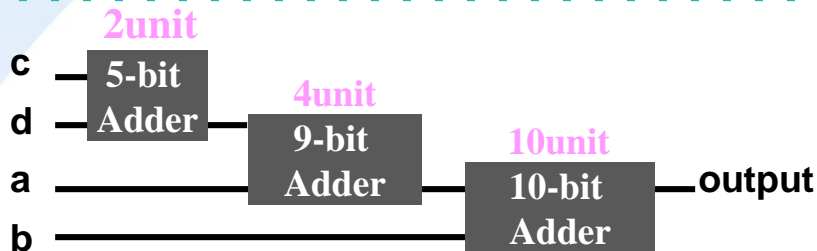
**Pipeline**



# Trade-off between Area and Timing

- ✓ a [7:0] , b [7:0] , c [3:0] , d [3:0]
- ✓ Q:  $(a + b + c + d) \times 100$  iterations ?

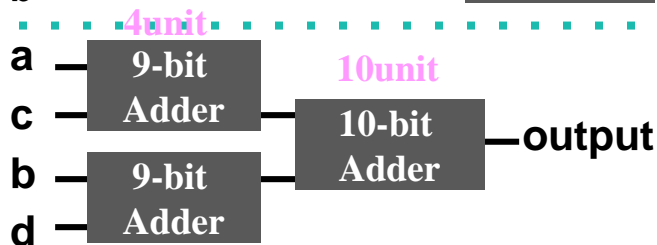
## Basic



Area: 24 units

Time:  $16 \times 100 = 1600$   
units

## Parallel



Area: 28 units

Time:  $14 \times 100 = 1400$   
units

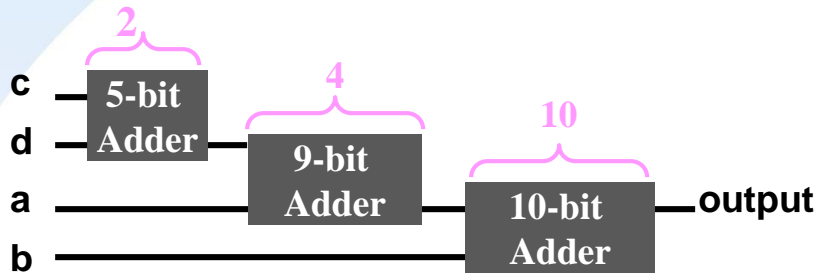
## Pipeline



# Trade-off between Area and Timing

- ✓ a [7:0] , b [7:0] , c [3:0] , d [3:0]
- ✓ Q:  $(a + b + c + d) \times 100$  iterations ?

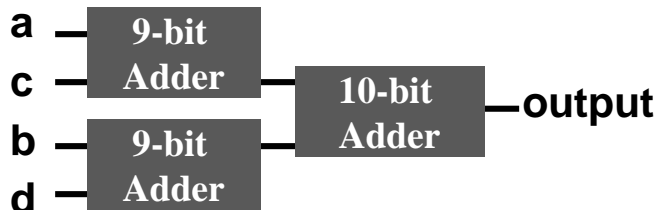
## Basic



Area: 24 units

Time:  $16 \times 100 = 1600$   
units

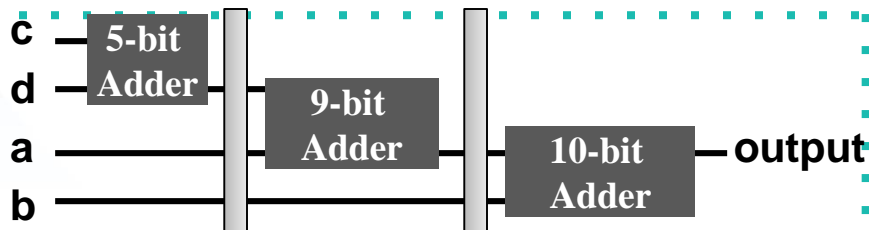
## Parallel



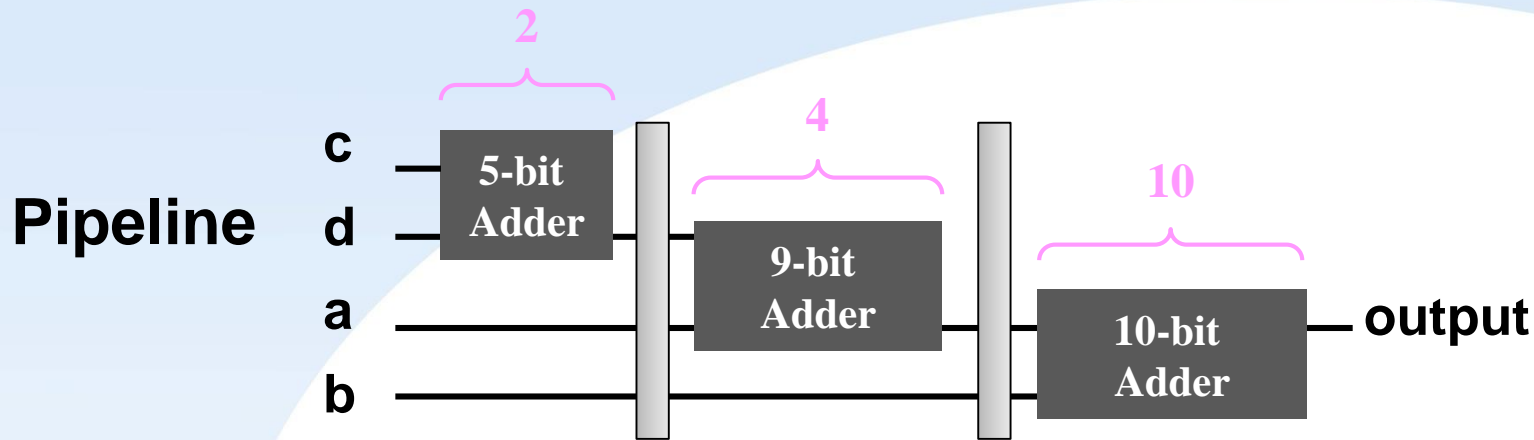
Area: 28 units

Time:  $14 \times 100 = 1400$   
units

## Pipeline



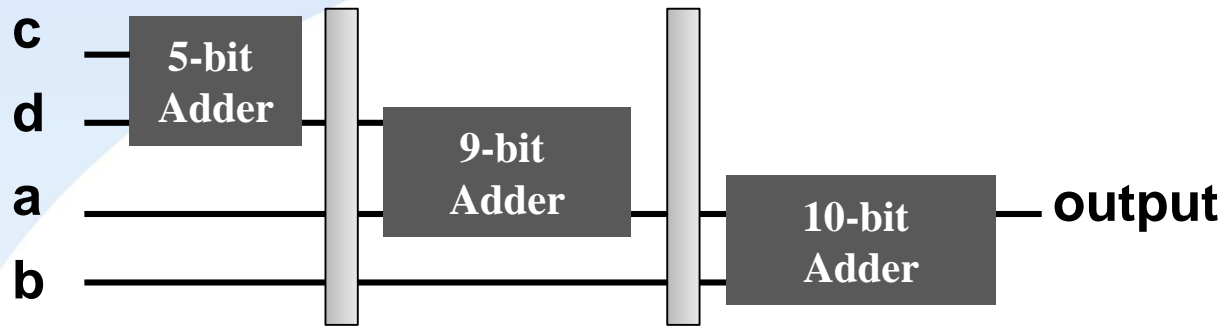
# Trade-off between Area and Timing



<b>T=0</b>	$c_1 + d_1$		

# Trade-off between Area and Timing

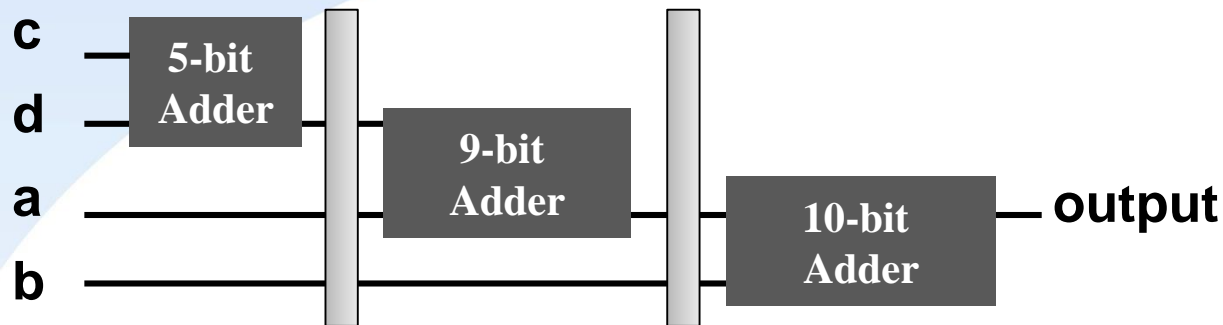
Pipeline



T=0	$c1 + d1$		
T=1	$c2 + d2$	$a1 + (c1 + d1)$	

# Trade-off between Area and Timing

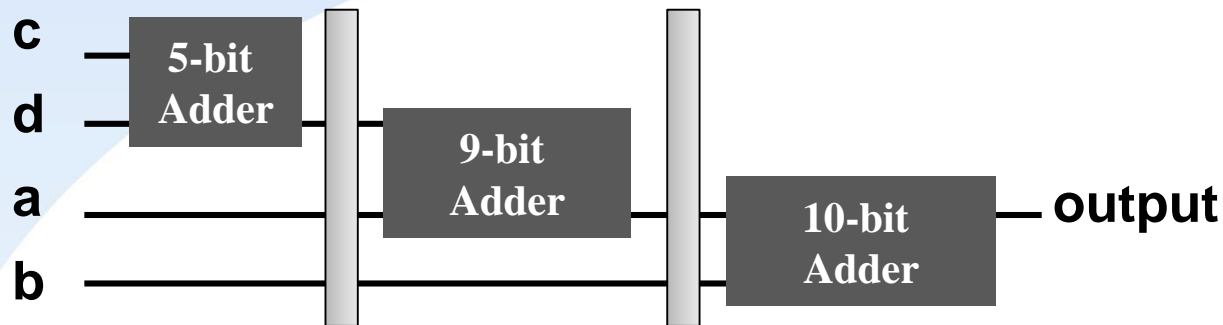
Pipeline



T=0	$c1 + d1$		
T=1	$c2 + d2$	$a1 + (c1 + d1)$	
T=2	$c3 + d3$	$a2 + (c2 + d2)$	$b1 + (a1 + c1 + d1)$

# Trade-off between Area and Timing

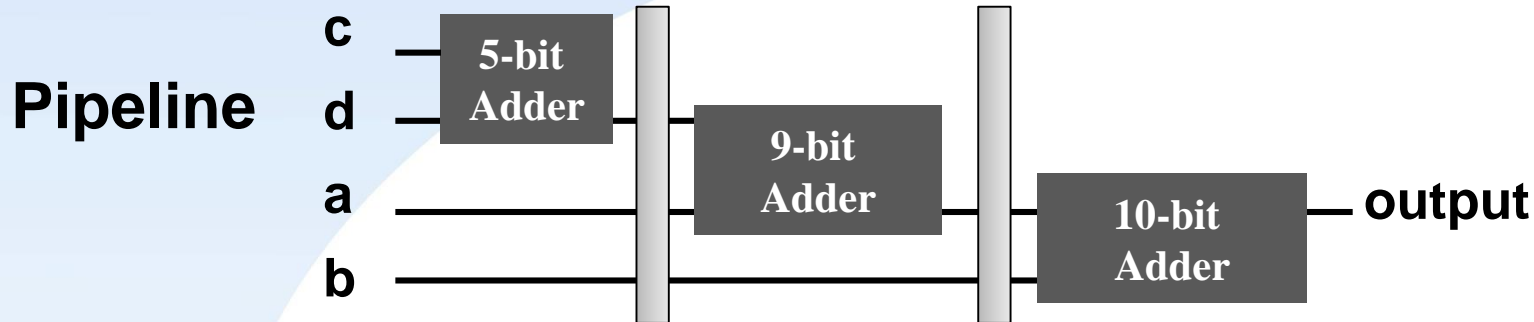
Pipeline



T=0	$c1 + d1$		
T=1	$c2 + d2$	$a1 + (c1 + d1)$	
T=2	$c3 + d3$	$a2 + (c2 + d2)$	$b1 + (a1 + c1 + d1)$
T=3	$c4 + d4$	$a3 + (c3 + d3)$	$b2 + (a2 + c2 + d2)$



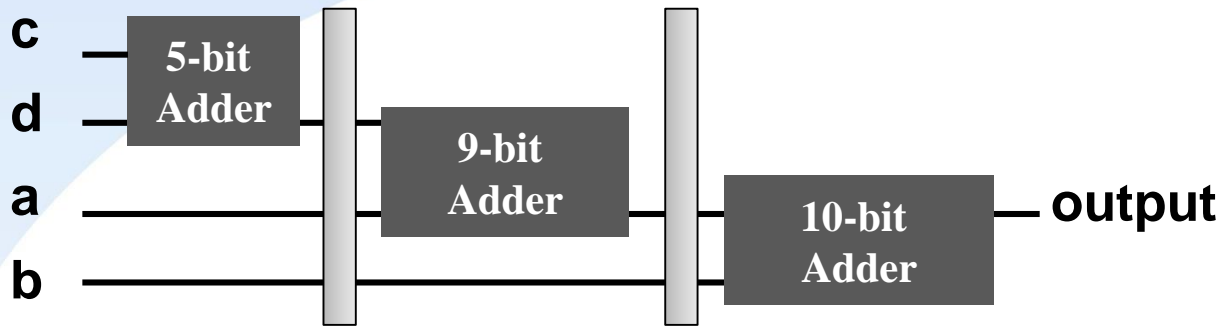
# Trade-off between Area and Timing



T=0	$c1 + d1$		
T=1	$c2 + d2$	$a1 + (c1 + d1)$	
T=2	$c3 + d3$	$a2 + (c2 + d2)$	$b1 + (a1 + c1 + d1)$
T=3	$c4 + d4$	$a3 + (c3 + d3)$	$b2 + (a2 + c2 + d2)$
T=4		$a4 + (c4 + d4)$	$b3 + (a3 + c3 + d3)$

# Trade-off between Area and Timing

Pipeline



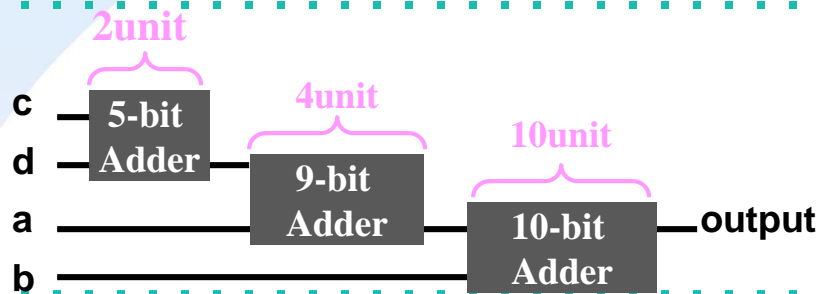
T=0	$c1 + d1$		
T=1	$c2 + d2$	$a1 + (c1 + d1)$	
T=2	$c3 + d3$	$a2 + (c2 + d2)$	$b1 + (a1 + c1 + d1)$
T=3	$c4 + d4$	$a3 + (c3 + d3)$	$b2 + (a2 + c2 + d2)$
T=4		$a4 + (c4 + d4)$	$b3 + (a3 + c3 + d3)$
T=5			$b4 + (a4 + c4 + d4)$

# Trade-off between Area and Timing

✓ a [7:0] , b [7:0] , c [3:0] , d [3:0]

✓ Q:  $(a + b + c + d) \times 100$  iterations ?

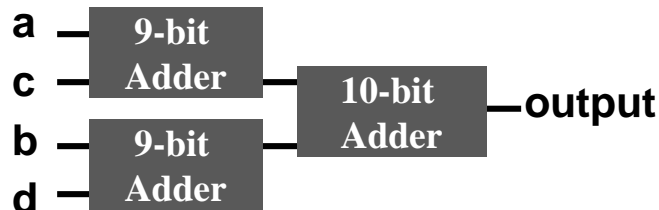
## Basic



Area: 24 units

Time:  $16 \times 100 = 1600$   
units

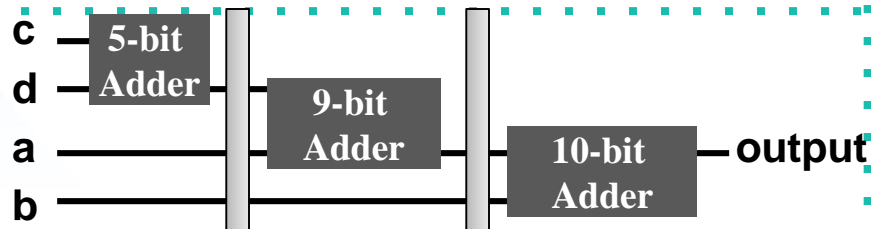
## Parallel



Area: 28 units

Time:  $14 \times 100 = 1400$   
units

## Pipeline



Area: 24 units+reg

Time:  $10 \times 102 = 1020$   
units



# Outline

- ✓ Section 1- Timing
- ✓ **Section 2- Designware**



# Overview of DesignWare

## ✓ IP (Intellectual Property )

- Hard IP : GDSII format, high performance but technology dependent.
- Firm IP : Netlist resource, less used.
- Soft IP : RTL design, requires verification.

## ✓ DesignWare library

- Provides synthesizable and verification IPs.
- Supports the method to optimize the area or the speed and reduce the timing.

## ✓ DesignWare IP library categories

- Building Block IPs (formally called Foundation Library)
- CoreTools
- Implementation IPs
- Smart Model Library
- Memory Models
- AMBA OCB Family
- Verification IPs



# DesignWare Building Block IPs (1/2)

## ✓ DesignWare building block IPs

- A collection of reusable IP blocks integrated into the SYNOPSIS synthesis environment.

## ✓ Characteristics

- Pre-verified for quality and better quality of results (QOR) in synthesis, decreasing design and technology risk.
- Allows high-level optimization of performance during synthesis.
- Increased design reusability, productivity
- Parameterized in size and also in functionality for some IP
- Technology-independent
- Provide synthesizable models, simulation models, datasheets, and examples.

# DesignWare Building Block IPs (2/2)

## ✓ Library categories

- Basic Library : A set of components bundled with HDL Compiler that implements several common arithmetic and logic functions.
- Logic : Combinational and sequential components
- Math : Arithmetic and trigonometric components
- Memory : Registers, FIFOs, and FIFO controllers, sync. And async. RAMs and stack components.
- DSP Library : Digital filters for digital signal processing (DSP) applications, ex: FIR, IIR filter
- Application Specific: Data integrity, interface, and JTAG components.
- GTECH Library : Genetic technology library, a technology-Independent, gate-level library.



# Usage of DesignWare Building Block IP

## ✓ Usage of DesignWare Building Block IP

- Operator inference
  - Convenient, but sometimes it is inefficient when synthesizing.
  - Supply default function only, can not use special function.
- Instantiate IP
  - Use SYNOPSIS design compiler shell script.
  - Supply different architecture for implementation.
  - Applying pre-compiling sub-blocks speeds up the synthesis for large design.





# Operator Inference (1/3)

## ✓ Operator inference

- Use the HDL operator in description, and the operator must include in *synthetic operator* definition.
- HDL compiler will infer synthetic operator in HDL code.
- HDL compiler supply high-level synthesis.
- The " / " operator is required for the DesignWare license.
- The HDL operator defined in standard synthetic operator:

Synthetic Operators	HDL Operator
<b>adder</b>	<b>+, +1</b>
<b>subtractor</b>	<b>-, -1</b>
<b>comparator</b>	<b>==, &lt;, &lt;=, &gt;, &gt;=</b>
<b>multiplier</b>	<b>*</b>
<b>selector</b>	<b>If, case</b>

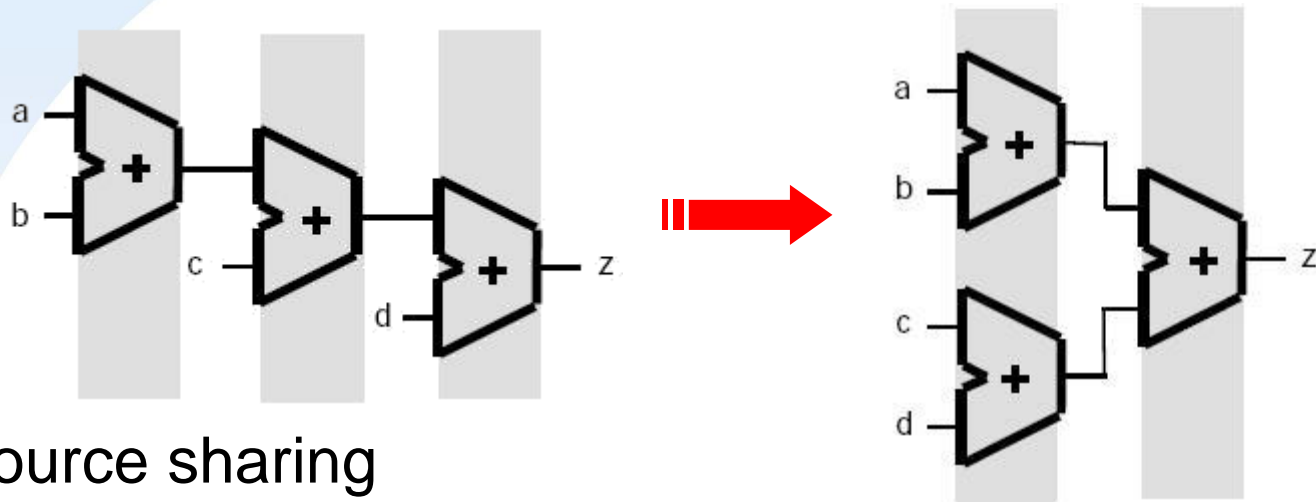


# Operator Inference (2/3)

## ✓ High-level synthesis

### — Arithmetic optimization

- Arithmetic level optimization, ex:  $a+b+c+d \rightarrow (a+b)+(c+d)$



### — Resource sharing

- Allows similar operations that do not overlap in time to be carried out by the same physical hardware.

# Operator inference (3/3)

## ✓ High-level synthesis flow

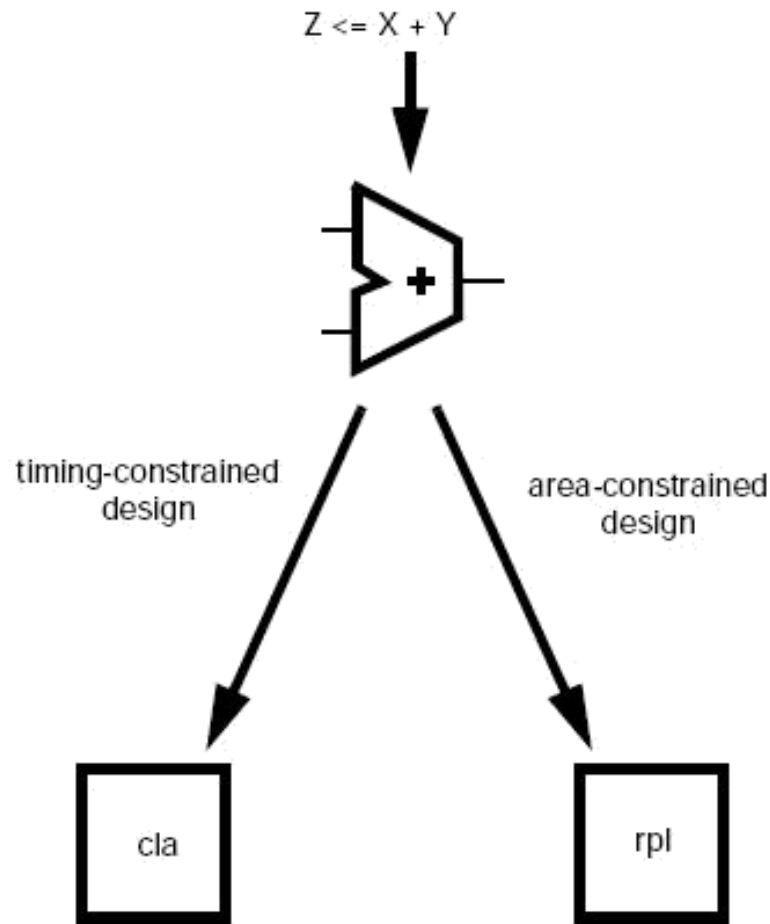
Your HDL Source Code

Operator Inference

Synthetic Operator

Automatic Implementation Selection  
Based on Overall Design Constraints

Appropriate Implementation  
Selected in Each Case



# Instantiate IP (1/9)

## ✓ Instantiation IP

- To instantiate a synthetic module manually and explicitly.
- Need to include a reference to the synthetic module in HDL code.

## ✓ SYNOPSYS online document

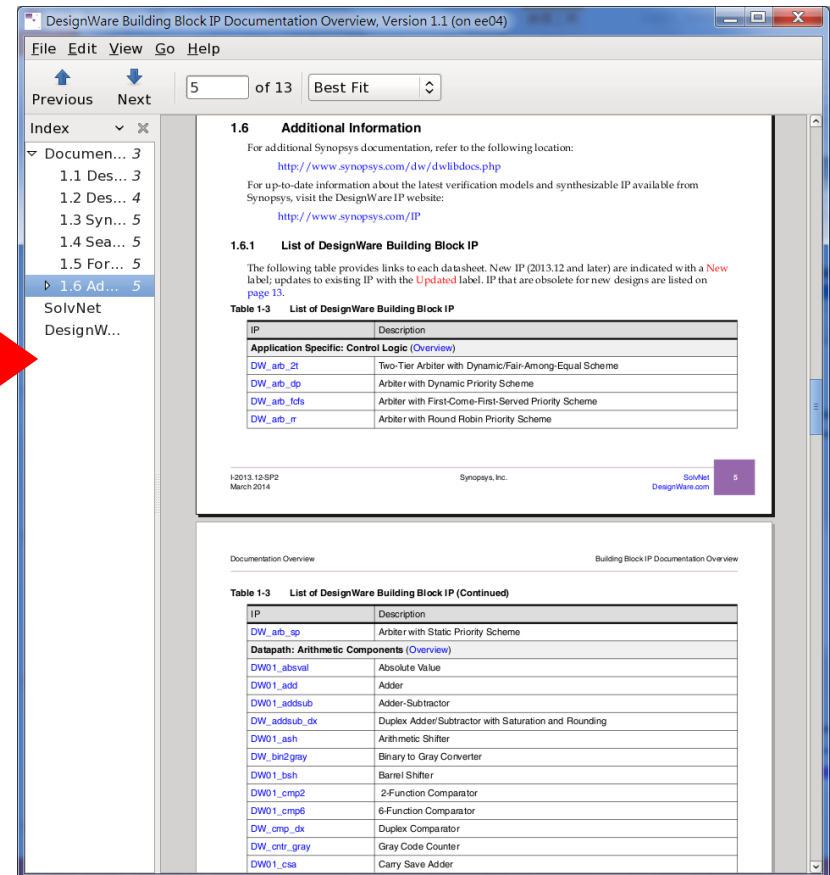
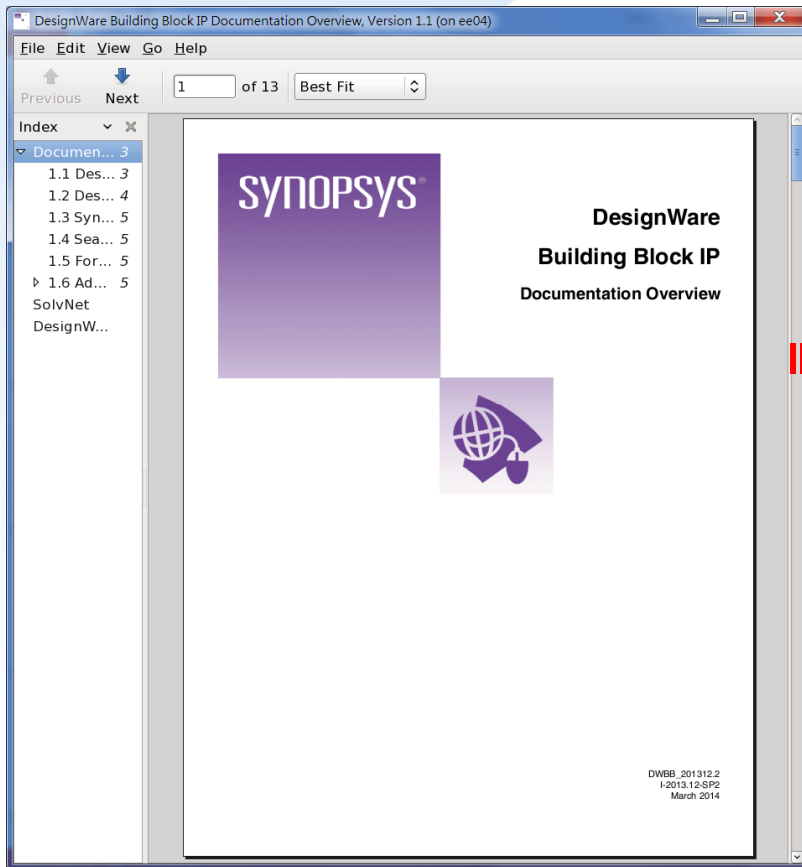
- Command:

```
evince /RAID2/EDA/synopsys/synthesis/cur/dw/doc/dwbb_overview.pdf &  
remember execute Xwin and setenv DISPLAY your_IP:0
```



# Instantiate IP (2/9)

- ✓ **SYNOPSYS online document**
  - Select section 1.6



# Instantiate IP (3/9)

## 1.6.1 List of DesignWare Building Block IP

The following table provides links to each datasheet. New IP (2013.12 and later) are indicated with a **New** label; updates to existing IP with the **Updated** label. IP that are obsolete for new designs are listed on [page 13](#).

Table 1-3 List of DesignWare Building Block IP

IP	Description
<b>Application Specific: Control Logic</b> ( <a href="#">Overview</a> )	
<a href="#">DW_arb_2t</a>	Two-Tier Arbiter with Dynamic/Fair-Among-Equal Scheme
<a href="#">DW_arb_dp</a>	Arbiter with Dynamic Priority Scheme
<a href="#">DW_arb_fcds</a>	Arbiter with First-Come-First-Served Priority Scheme
<a href="#">DW_arb_rr</a>	Arbiter with Round Robin Priority Scheme
<b>Datapath: Arithmetic Components</b> ( <a href="#">Overview</a> )	
<a href="#">DW01_absval</a>	Absolute Value
<a href="#">DW01_add</a>	Adder
<a href="#">DW01_addsub</a>	Adder-Subtractor
<a href="#">DW_addsub_dx</a>	Duplex Adder/Subtractor with Saturation and Rounding
<a href="#">DW01_ash</a>	Arithmetic Shifter
<a href="#">DW_bin2gray</a>	Binary to Gray Converter
<a href="#">DW01_bsh</a>	Barrel Shifter
<a href="#">DW01_cmp2</a>	2-Function Comparator
<a href="#">DW01_cmp6</a>	6-Function Comparator
<a href="#">DW_cmp_dx</a>	Duplex Comparator
<a href="#">DW_cntr_gray</a>	Gray Code Counter
<a href="#">DW01_csa</a>	Carry Save Adder
<a href="#">DW01_dec</a>	Decrementer
<a href="#">DW_div</a>	Combinational Divider
<a href="#">DW_div_sat</a>	Combinational Divider with Saturation ( <b>New</b> )
<a href="#">DW_div_pipe</a>	Stallable Pipelined Divider
<a href="#">DW_exp2</a>	Base 2 Exponential (2a)
<a href="#">DW_gray2bin</a>	Gray to Binary Converter
<a href="#">DW01_inc</a>	Incrementer
<a href="#">DW01_indec</a>	Incrementer-Decrementer
<a href="#">DW_inc_gray</a>	Gray Incrementer
<a href="#">DW_inv_sqrt</a>	Reciprocal of Square-Root
<a href="#">DW_lbsh</a>	Barrel Shifter with Preferred Left Direction
<a href="#">DW_ln</a>	Natural Logarithm (ln(a))
<a href="#">DW_log2</a>	Base 2 Logarithm (log <sub>2</sub> (a)) ( <b>Updated datasheet</b> )
<a href="#">DW02_mac</a>	Multiplier-Accumulator
<a href="#">DW_minmax</a>	Minimum/Maximum Value
<a href="#">DW02_mult</a>	Multiplier
<a href="#">DW02_multp</a>	Partial Product Multiplier

Table 1-3 List of DesignWare Building Block IP (Continued)

IP	Description
<a href="#">DW02_mult_2_stage</a>	Two-Stage Pipelined Multiplier
<a href="#">DW02_mult_3_stage</a>	Three-Stage Pipelined Multiplier
<a href="#">DW02_mult_4_stage</a>	Four-Stage Pipelined Multiplier
<a href="#">DW02_mult_5_stage</a>	Five-Stage Pipelined Multiplier
<a href="#">DW02_mult_6_stage</a>	Six-Stage Pipelined Multiplier
<a href="#">DW_mult_dx</a>	Duplex Multiplier
<a href="#">DW_mult_pipe</a>	Stallable Pipelined Multiplier
<a href="#">DW_norm</a>	Normalization for Fractional Input
<a href="#">DW_norm_rnd</a>	Normalization and Rounding
<a href="#">DW_piped_mac</a>	Pipelined Multiplier-Accumulator
<a href="#">DW02_prod_sum</a>	Generalized Sum of Products
<a href="#">DW02_prod_sum1</a>	Multiplier-Adder
<a href="#">DW_prod_sum_pipe</a>	Stallable Pipelined Generalized Sum of Products
<a href="#">DW_rash</a>	Arithmetic Shifter with Preferred Right Direction
<a href="#">DW_rbsh</a>	Barrel Shifter with Preferred Right Direction
<a href="#">DW01_satrnd</a>	Arithmetic Saturation and Rounding Logic
<a href="#">DW_shifter</a>	Combined Arithmetic and Barrel Shifter
<a href="#">DW_sla</a>	Arithmetic Shifter with Preferred Left Direction (VHDL style)
<a href="#">DW_sra</a>	Arithmetic Shifter with Preferred Right Direction (VHDL style)
<a href="#">DW_square</a>	Integer Squarer
<a href="#">DW_squarep</a>	Partial Product Integer Squarer
<a href="#">DW_sqrt</a>	Combinational Square Root
<a href="#">DW_sqrt_pipe</a>	Stallable Pipelined Square Root
<a href="#">DW01_sub</a>	Subtractor
<a href="#">DW02_sum</a>	Vector Adder
<a href="#">DW02_tree</a>	Wallace Tree Compressor
<b>Datapath: Floating Point</b> ( <a href="#">Overview</a> )	
<a href="#">DW_fp_add</a>	Floating Point Adder
<a href="#">DW_fp_addsub</a>	Floating Point Adder/Subtractor
<a href="#">DW_fp_cmp</a>	Floating Point Comparator
<a href="#">DW_fp_div</a>	Floating Point Divider



# Instantiate IP (4/9)



**DW02\_mult**

**Module name**

Multiplier

Version, STAR and Download Information: [IP Directory](#)

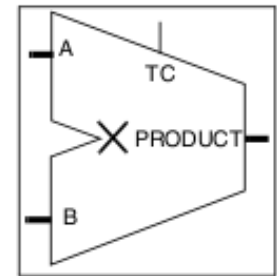
## Features and Benefits

- Parameterized word length
- Unsigned and signed (two's-complement) data operation

## Description

DW02\_mult is a multiplier that multiplies the operand *A* by *B* to produce the output, *PRODUCT*.

The control signal *TC* determines whether the input and output data is interpreted as unsigned (*TC*=0) or signed (*TC*=1) numbers.



**Table 1-1 Pin Description**

Pin Name	Width	Direction	Function
A	<i>A_width</i> bit(s)	Input	Multiplier
B	<i>B_width</i> bit(s)	Input	Multiplicand
TC	1 bit	Input	Two's complement control 0 = unsigned 1 = signed
PRODUCT	<i>A_width</i> + <i>B_width</i> bit(s)	Output	Product $A \times B$

input & output

**Argument assignment:**  
**DW02\_mult #(N,N)**  
**mult01(..., ..., ..., ...);**

**Table 1-2 Parameter Description**

Parameter	Values	Description
<i>A_width</i>	$\geq 1$	Word length of A
<i>B_width</i>	$\geq 1$	Word length of B



# Instantiate IP (5/9)

**Table 1-3 Synthesis Implementations**

Implementation Name	Function	License Feature Required
csa <sup>a</sup>	Carry-save array synthesis model	none
pparch <sup>b</sup>	Delay-optimized flexible Booth Wallace	DesignWare
apparch <sup>b</sup>	Area-optimized flexible Booth Wallace	DesignWare

**User implementation type specification**

**Table 1-4 Simulation Models**

Model	Function
DW02.DW02_MULT_CFG_SIM	Design unit name for VHDL simulation
dw/dw02/src/DW02_mult_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW02_mult.v	Verilog simulation model source code

**Simulation model path specification**

**Table 1-5 Functional Description**

TC	A	B	PRODUCT
0	A (unsigned)	B (unsigned)	$A \times B$ (unsigned)
1	A (two's complement)	B (two's complement)	$A \times B$ (two's complement)

**Functional parameter specification**



# Instantiate IP (6/9)

## ✓ Instantiate module

- Instantiate the synthetic module and specify parameters defined in document.

### HDL Usage Through Component Instantiation - Verilog

```
module DW02_mult_inst( inst_A, inst_B, inst_TC, PRODUCT_inst );  
  
    parameter A_width = 8;  
    parameter B_width = 8;  
  
    input [A_width-1 : 0] inst_A;  
    input [B_width-1 : 0] inst_B;  
    input inst_TC;  
    output [A_width+B_width-1 : 0] PRODUCT_inst;  
  
    // Instance of DW02_mult  
    DW02_mult #(A_width, B_width)  
        U1 ( .A(inst_A), .B(inst_B), .TC(inst_TC), .PRODUCT(PRODUCT_inst) );  
  
endmodule
```

Table1-2

Table1-1 I/O port



# Instantiate IP (7/9)

## ✓ RTL behavior simulation

- Specify the behavioral simulation models (Table1-4).
  - Absolute path
  - Relative path

## ✓ Absolute path

- ``include "/usr/synthesis/dw/sim_ver/<model_name>.v "`  
``include /usr/synthesis/dw/sim_ver/DW02_mult.v“`

## ✓ Relative path

- ``include "<model_name>.v "`  
``include "DW02_mult.v“`
- Command: `irun <file_name>.v –incdir <directory>`
  - Ex : `irun DW02_multi_inst.v –incdir /usr/synthesis/dw/sim_ver/`



# Instantiate IP (8/9)

## ✓ Synthesis

- Apply `//synopsys translate_off`  
`//synopsys translate_on`

```
//synopsys translate_off (DA synthesis off)
..... (the code won't be synthesis)
//synopsys translate_on (DA synthesis on)
```

## ✓ Set the implementation type of IP

- User specify the implementation type of IP manually.

```
.....
//synopsys dc_script_begin
//set_implementation wall U1 (instance name of IP)
implementation type from (Table1-3)
//synopsys dc_script_end
.....
```



# Instantiate IP (9/9)

## ✓ Example

- RTL/Gate simulation description

```
//synopsys translate_off
`include "/usr/synthesis/dw/sim_ver/DW02_mult.v" (Table1-4)
//synopsys translate_on

module SignedMultiplier(a, b, product);
  input [7 : 0] a;
  input [7 : 0] b;
  output [15: 0] product;

  DW02_mult #(8, 8)  U1 (.A(a), .B(b), .TC(1'b1), .PRODUCT(product));
  (cell name)      (Table1-2)                      (Table1-1)

  //synopsys dc_script_begin
  //set_implementation csa U1
                                     (Table1-3)
  //synopsys dc_script_end

endmodule
```

Note : If you use Designware, you should use clean command after each simulation. (./09\_clean\_up)!

