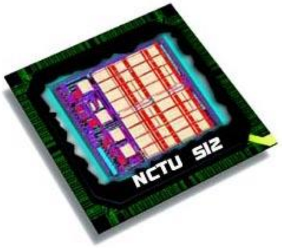
Advanced Sequential Circuit Design

NYCU-EE IC Lab Fall 2023

Lecturer : Jia-Yu, Lee



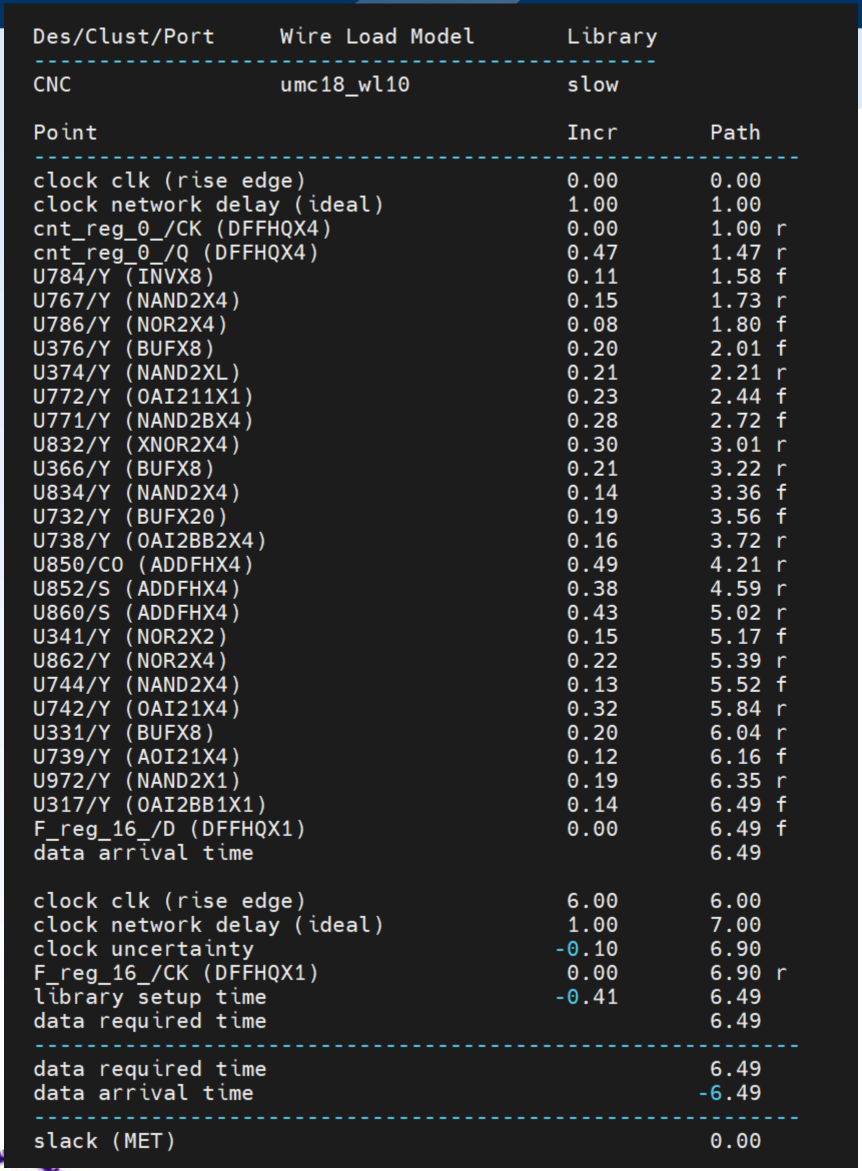
Outline

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

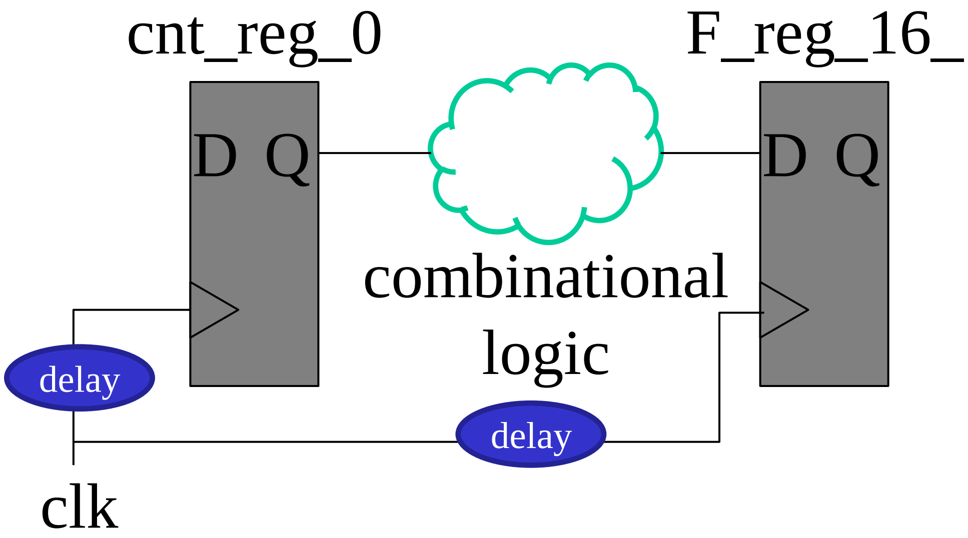
Outline

- ✓ **Section 1- Timing**
  - Setup/hold time
  - Pipeline
- ✓ **Section 2- Designware**

Recall

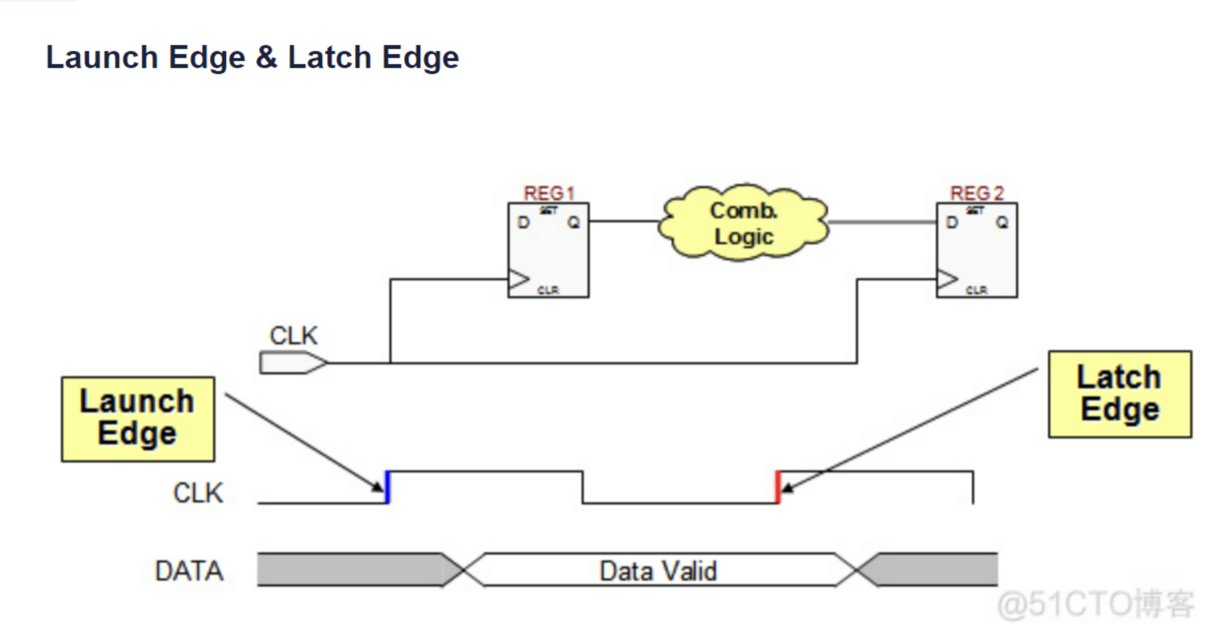


```
Des/Clust/Port      Wire Load Model      Library
------------------------------------------------
CNC                 umc18_wl10           slow

Point                                     Incr        Path
-----------------------------------------------------------
clock clk (rise edge)                     0.00        0.00
clock network delay (ideal)               1.00        1.00
cnt_reg_0_/CK (DFFHQX4)                   0.00        1.00 r
cnt_reg_0_/Q (DFFHQX4)                    0.47        1.47 r
U784/Y (INVX8)                            0.11        1.58 f
U767/Y (NAND2X4)                          0.15        1.73 r
U786/Y (NOR2X4)                           0.08        1.80 f
U376/Y (BUFX8)                            0.20        2.01 f
U374/Y (NAND2XL)                          0.21        2.21 r
U772/Y (OAI211X1)                         0.23        2.44 f
U771/Y (NAND2BX4)                         0.28        2.72 f
U832/Y (XNOR2X4)                          0.30        3.01 r
U366/Y (BUFX8)                            0.21        3.22 r
U834/Y (NAND2X4)                          0.14        3.36 f
U732/Y (BUFX20)                           0.19        3.56 f
U738/Y (OAI2BB2X4)                        0.16        3.72 r
U850/CO (ADDFHX4)                         0.49        4.21 r
U852/S (ADDFHX4)                          0.38        4.59 r
U860/S (ADDFHX4)                          0.43        5.02 r
U341/Y (NOR2X2)                           0.15        5.17 f
U862/Y (NOR2X4)                           0.22        5.39 r
U744/Y (NAND2X4)                          0.13        5.52 f
U742/Y (OAI21X4)                          0.32        5.84 r
U331/Y (BUFX8)                            0.20        6.04 r
U739/Y (AOI21X4)                          0.12        6.16 f
U972/Y (NAND2X1)                          0.19        6.35 r
U317/Y (OAI2BB1X1)                        0.14        6.49 f
F_reg_16_/D (DFFHQX1)                     0.00        6.49 f
data arrival time                                     6.49

clock clk (rise edge)                     6.00        6.00
clock network delay (ideal)               1.00        7.00
clock uncertainty                        -0.10        6.90
F_reg_16_/CK (DFFHQX1)                    0.00        6.90 r
library setup time                       -0.41        6.49
data required time                                    6.49
-----------------------------------------------------------
data required time                                    6.49
data arrival time                                    -6.49
-----------------------------------------------------------
slack (MET)                                           0.00
```



Recall

- ✓ **Launch Edge:**
  - The clock rising edge used by Register 1 for generating data
- ✓ **Latch Edge:**
  - The clock rising edge used by Register 2 for receiving data will introduce a delay of one clock cycle from Launch Edge



Timing Issue

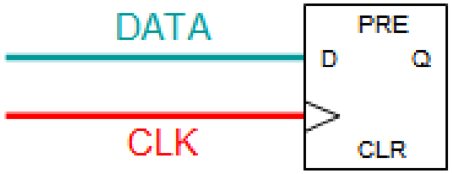
✓ Terminology

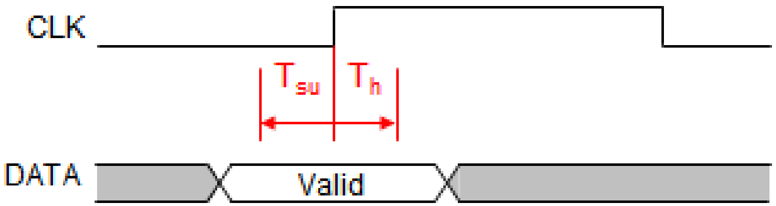
- Setup time (t_{setup})

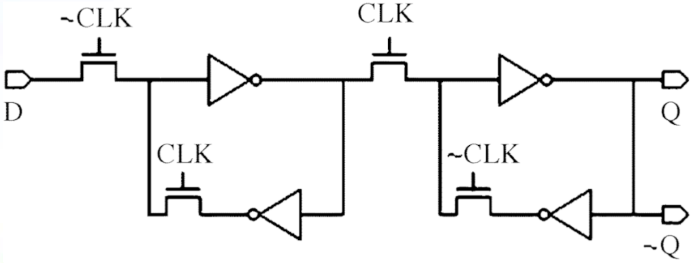
The time that the input signal must be stabilized before the clock edge.

- Hold time (t_{hold})

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

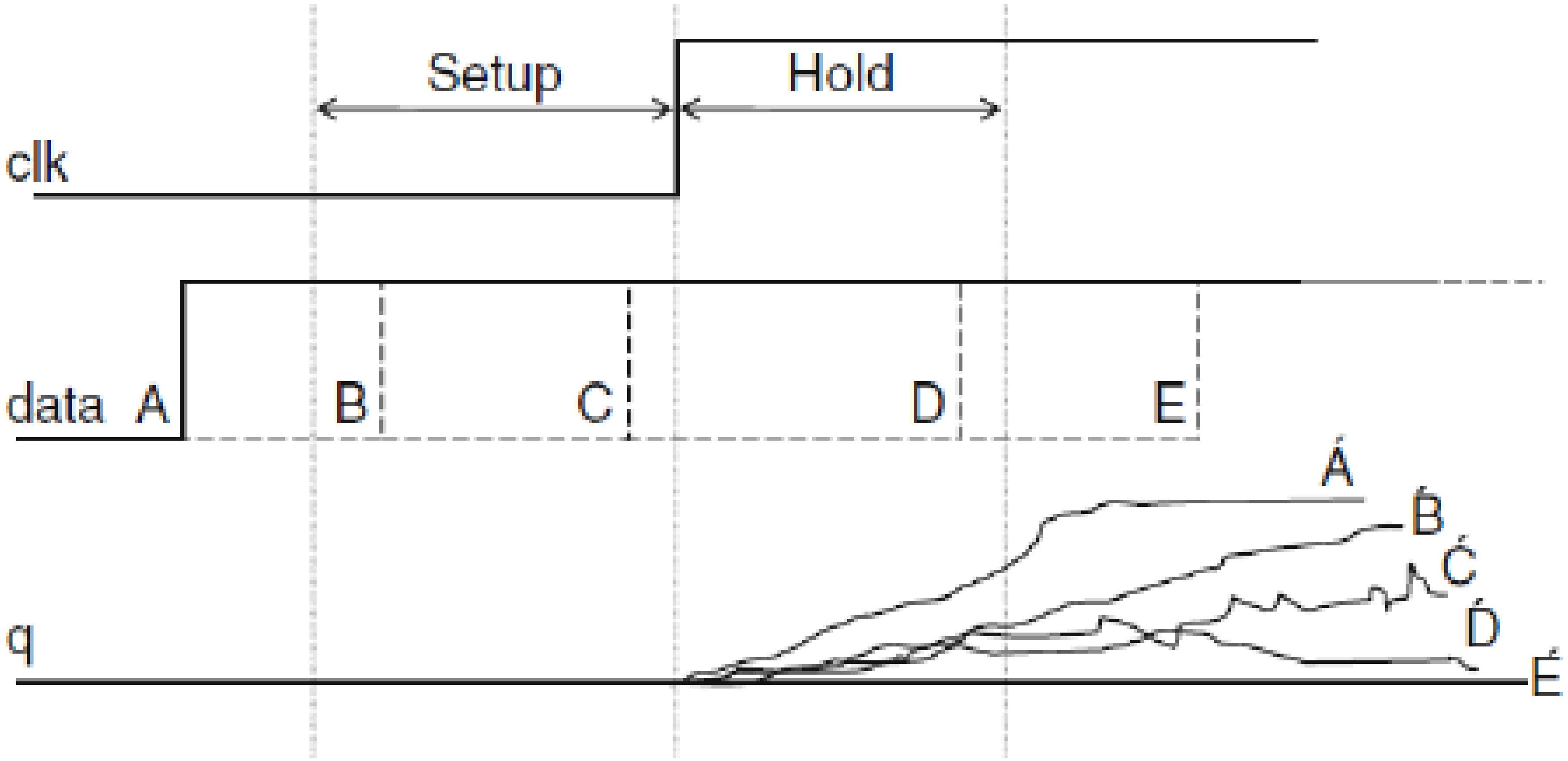






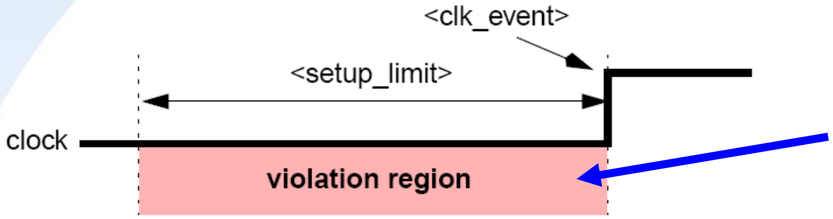
Timing Issue

- ✓ **Metastability**



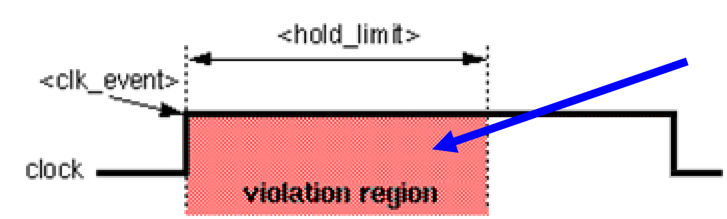
Timing Check (1/2)

- ✓ **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.



data can’t be changed
in this red region

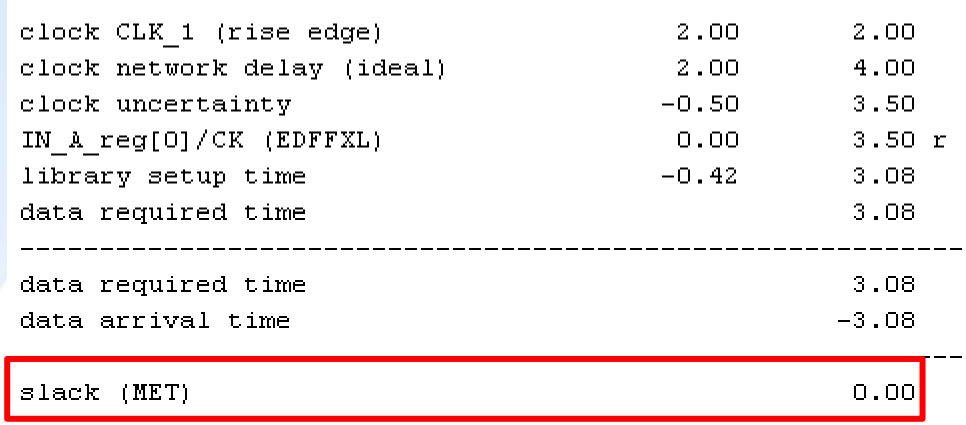
- ✓ **Hold time check**
  - The tool determines whether a data signal remains stable for a minimum specified time (i.e., violation region) after a transition in an enabling signal, such as a clock event.



data can’t be changed
in this red region

Timing Check (2/2)

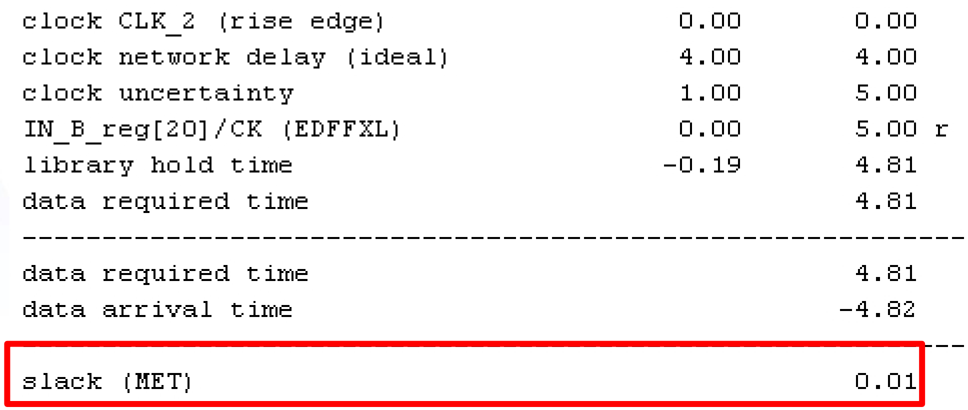
✓ Timing report: setup time



```
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

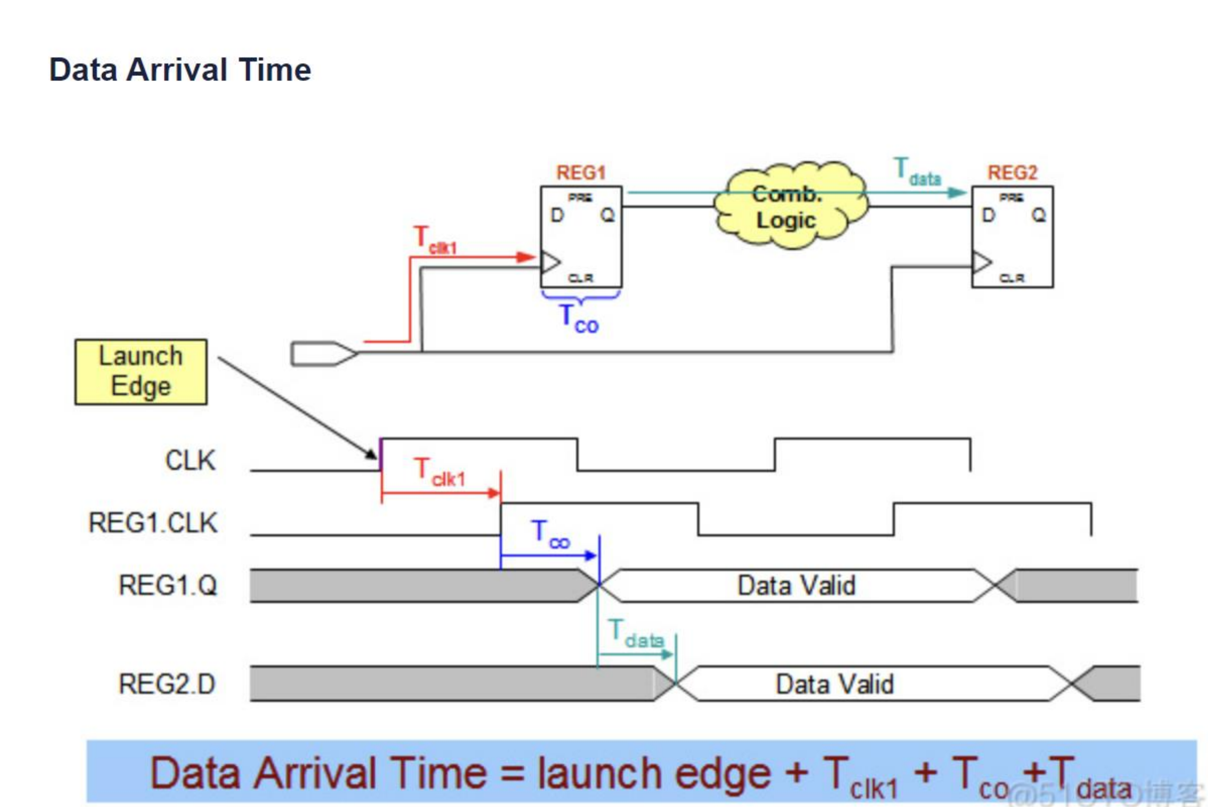
Slacks should be MET!
(non-negative)



```
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
```

Terminology

- **Data Arrival Time**
  - The time at which data actually arrives at the input D of Register 2

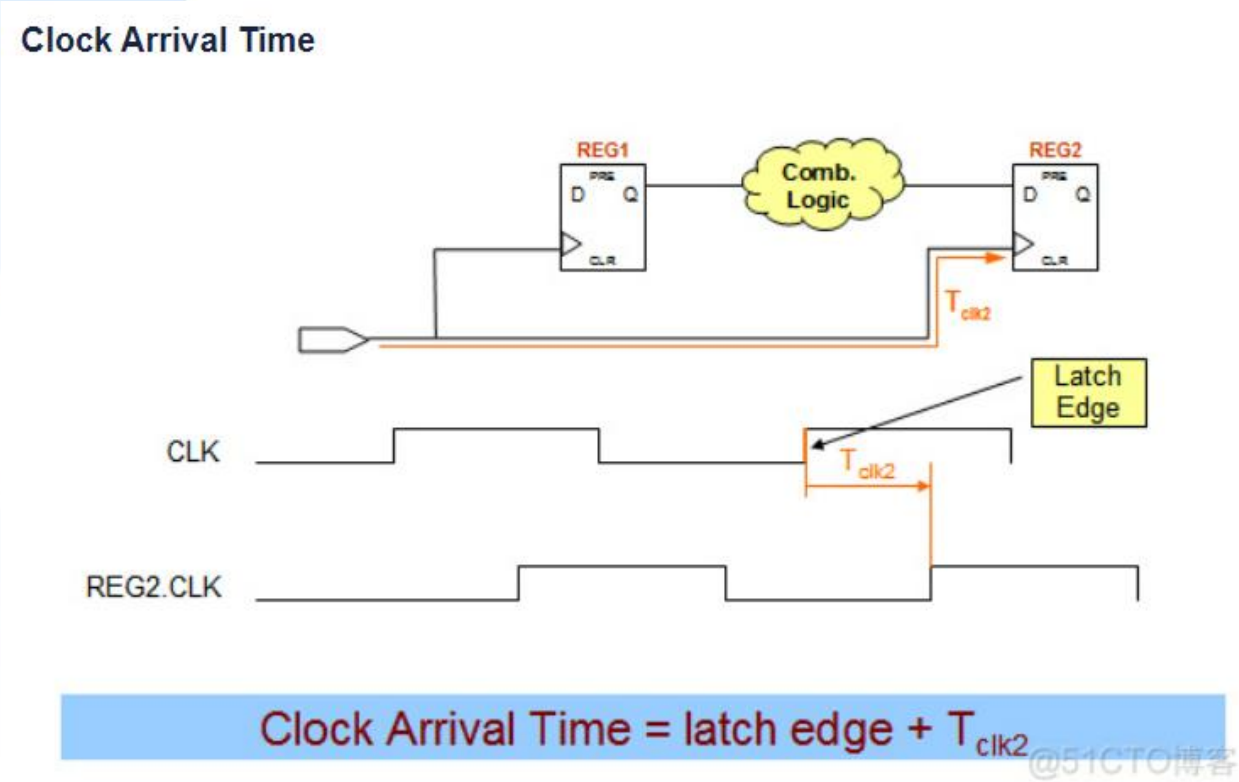


Data Arrival Time

Data Arrival Time = launch edge + T_{clk1} + T_{co} +T_{data}

Terminology

- ✓ **Clock Arrival Time**
  - The actual time at which the clock signal arrives at the input of Register 2

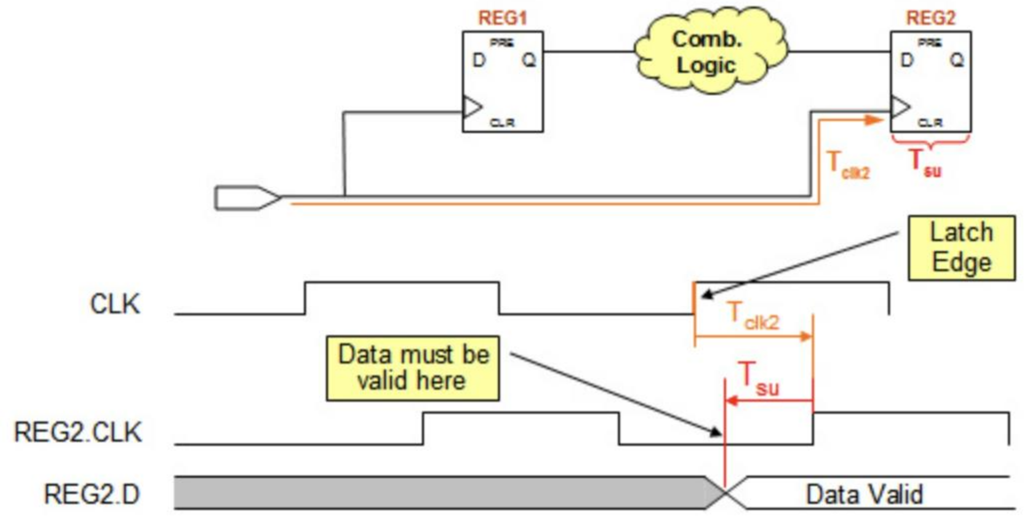


Clock Arrival Time

Clock Arrival Time = latch edge + T_{clk2}

Terminology

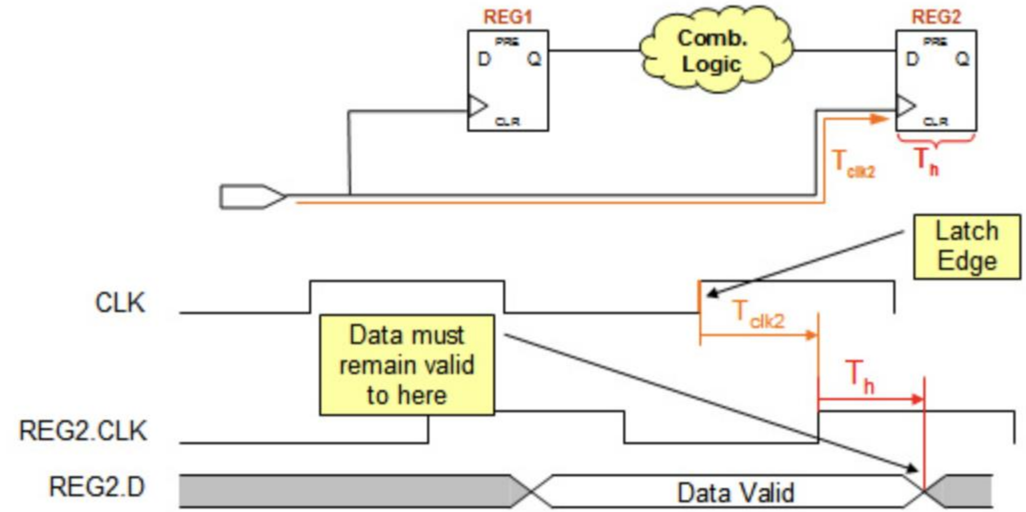
- ✓ **Data required Time (setup time)**
  - – The latest time by which the data must be ready



Data Required Time = Clock Arrival Time - T_{su} - Setup Uncertainty

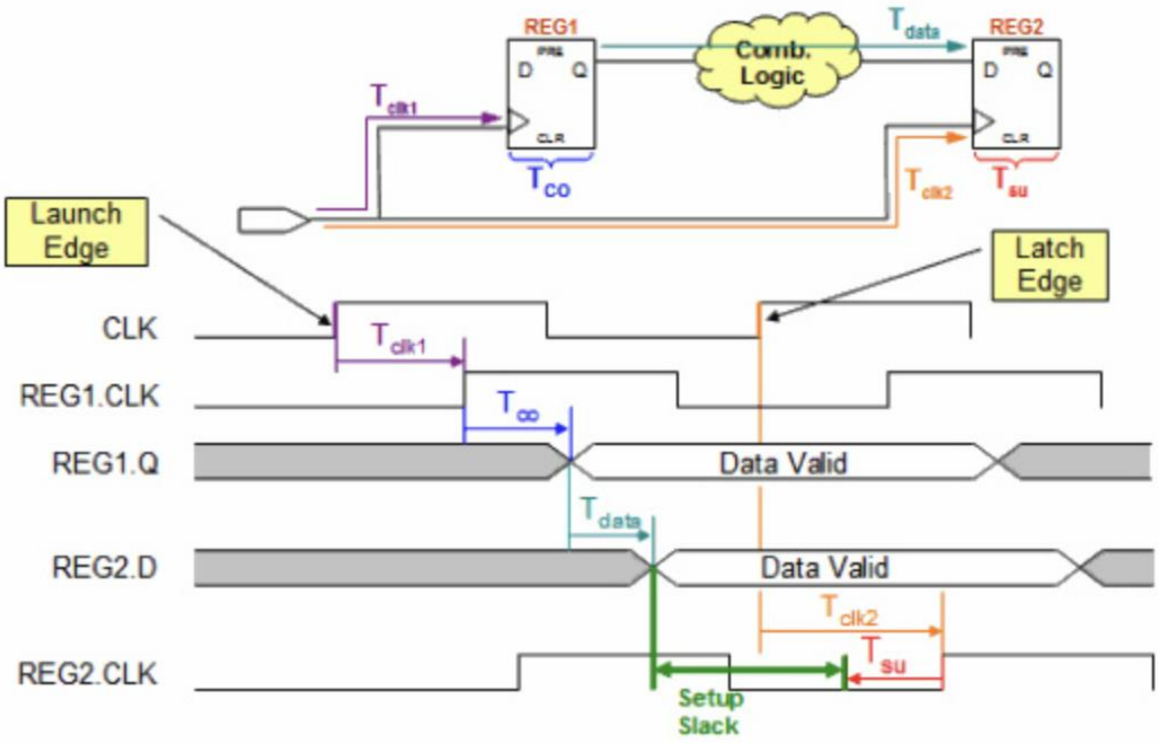
Terminology

- ✓ **Data required Time (hold time)**
  - Until what time at least does the data need to be maintained



Data Required Time = Clock Arrival Time + T_h + Hold Uncertainty

Setup Slack

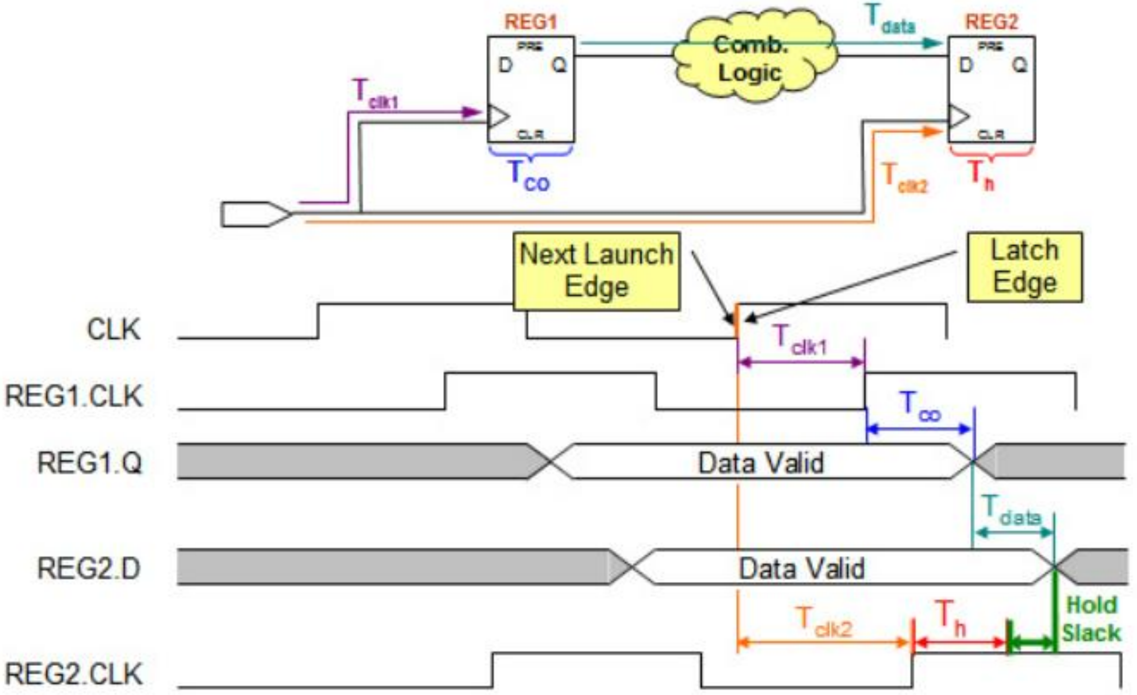


Setup Slack = Data Required Time – Data Arrival Time

Positive slack
Timing requirement me

Negative slack
Timing requirement not met

Hold Slack



Hold Slack = Data Arrival Time – Data Required Time

Positive slack
Timing requirement me

Negative slack
Timing requirement not met

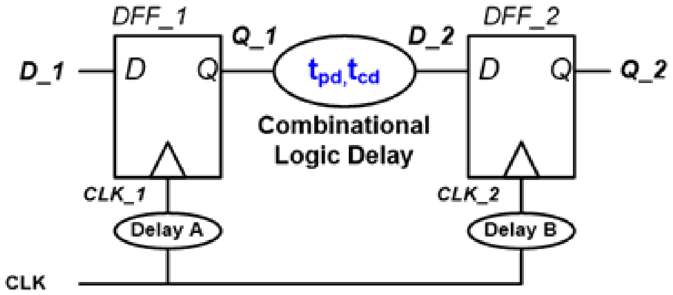
Timing Issue

✓ 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})

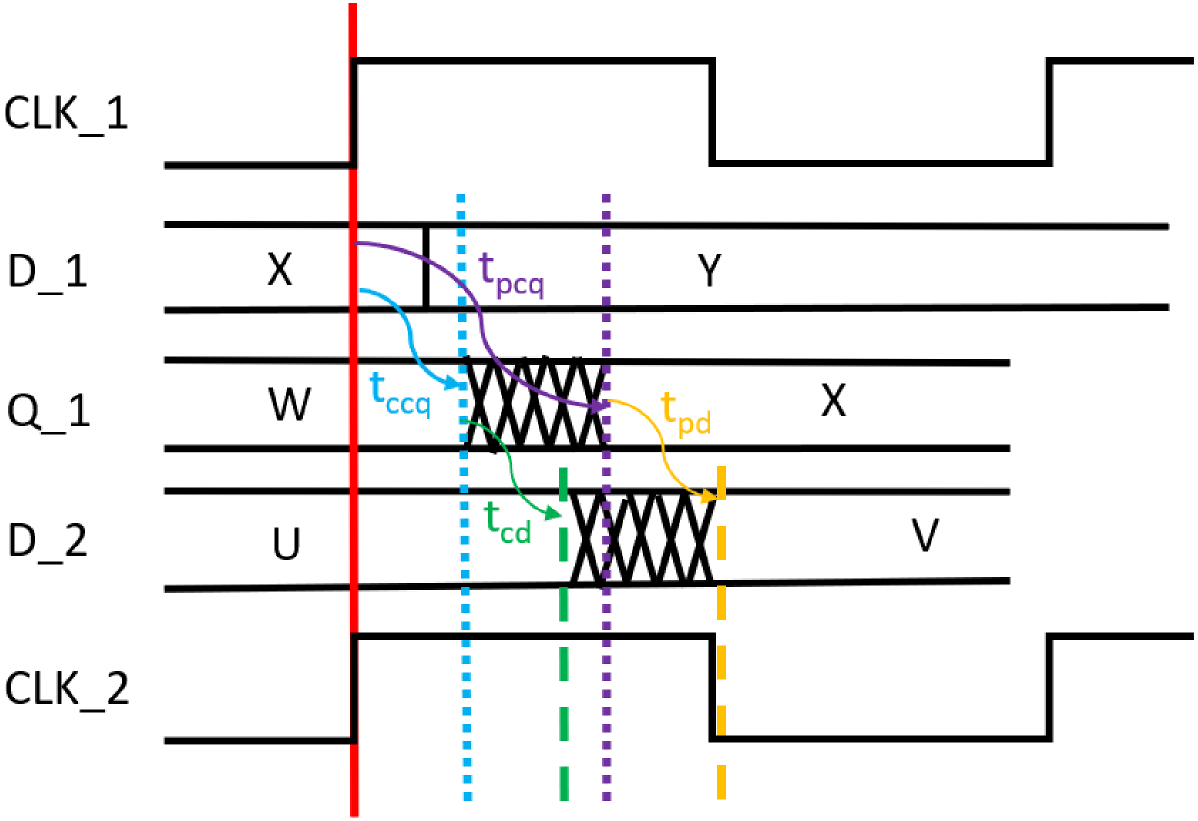


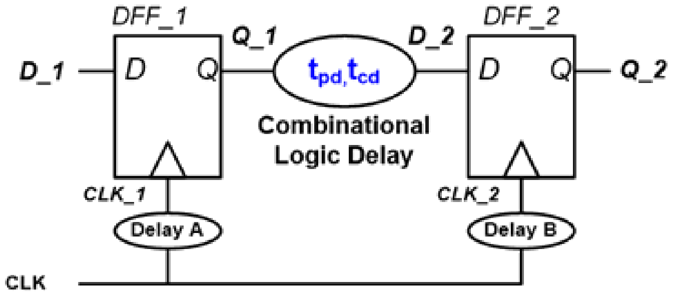
– Propagation delay

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})

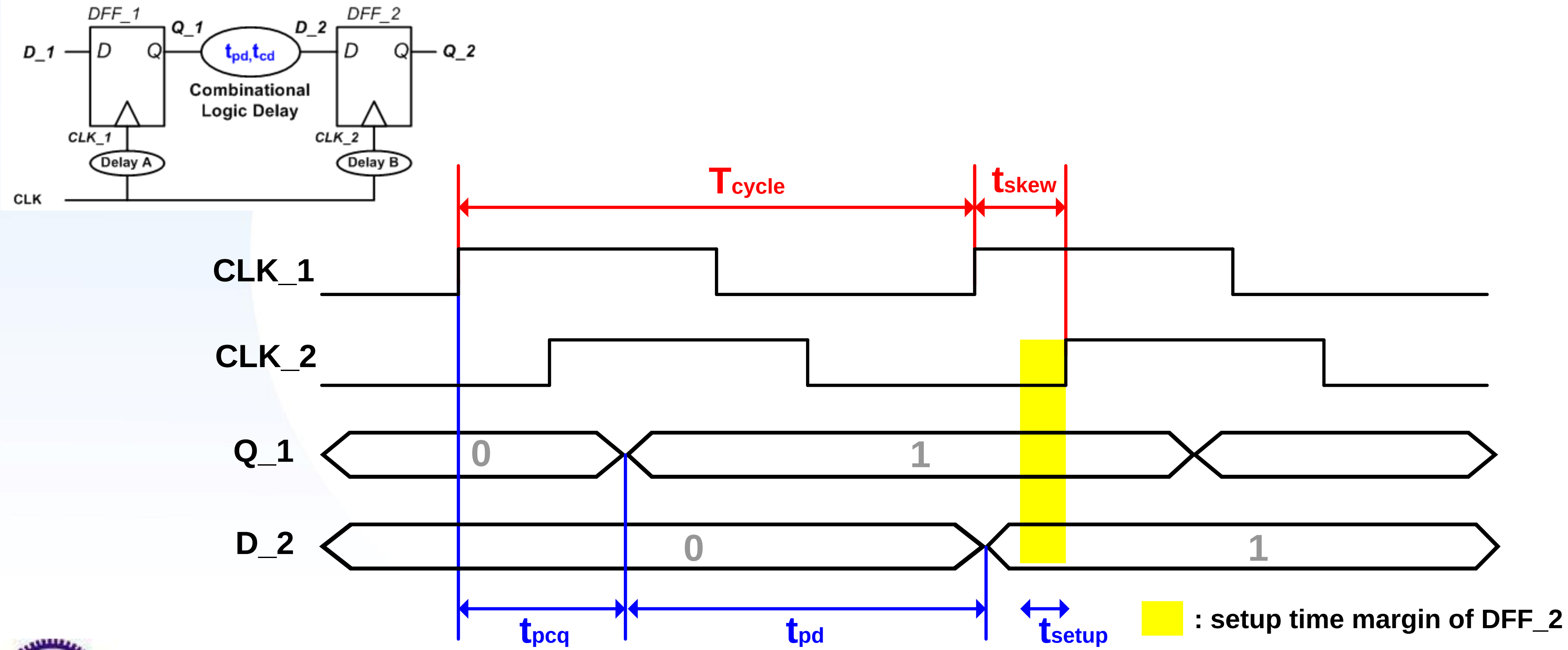
Timing Issue





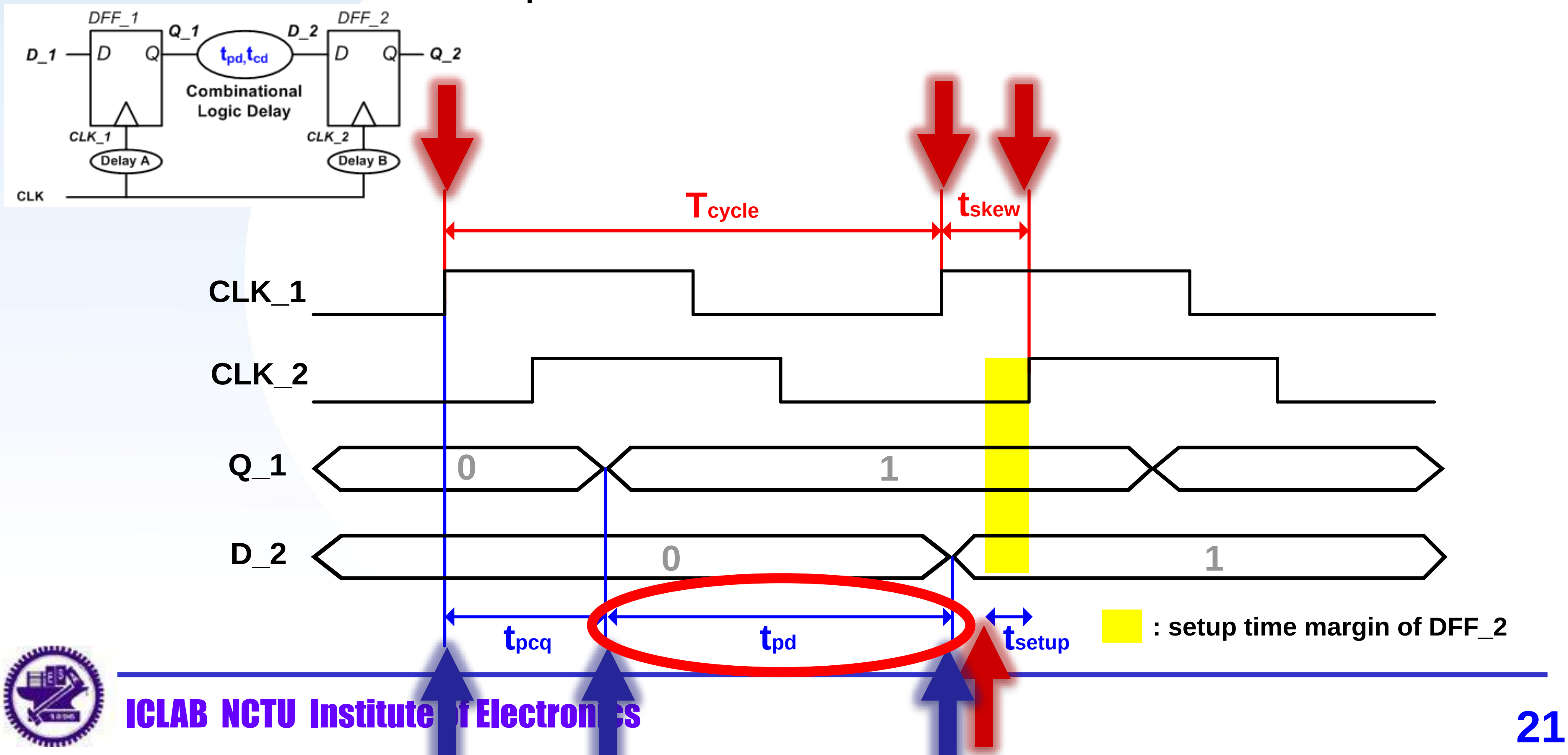
Setup Time Criterion

- ✓ **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



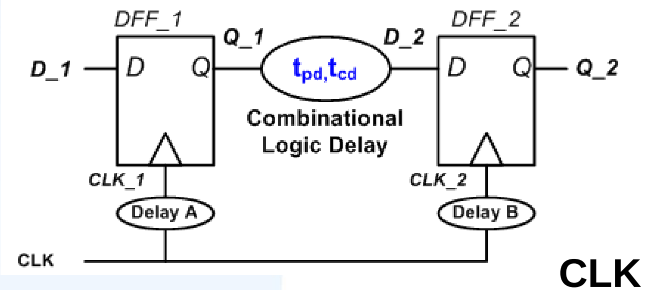
Setup Time Criterion

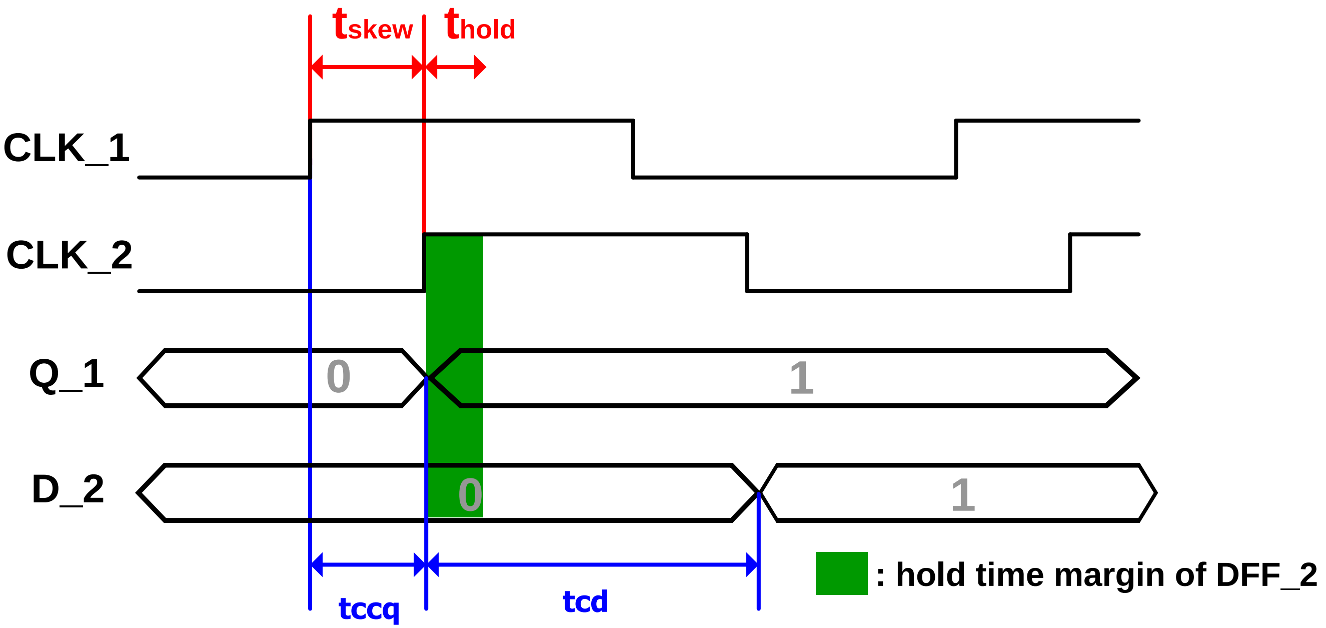
- ✓ **Setup time criterion: $(T_{cycle} + t_{skew})$ > $(t_{pcq} + t_{pd} + t_{setup})$**
  - data required time = $T_{cycle} + t_{skew} - t_{setup}$
  - data arrival time = $t_{pcq} + t_{pd}$
  - Slack = data required time - data arrival time



Hold Time Criterion

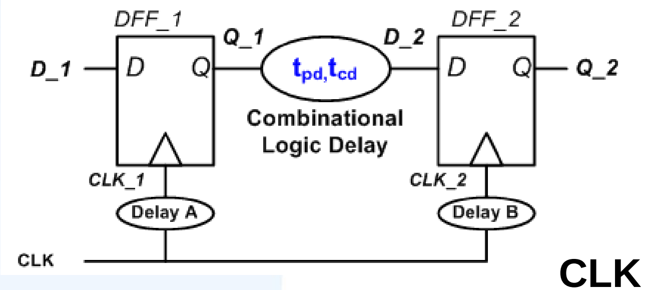
- ✓ **Hold time criterion: $(t_{ccq} + t_{cd}) > (t_{hold} + t_{skew})$**
  - data required time = $t_{skew} + t_{hold}$
  - data arrival time = $t_{ccq} + t_{cd}$
  - Slack = data arrival time – data required time

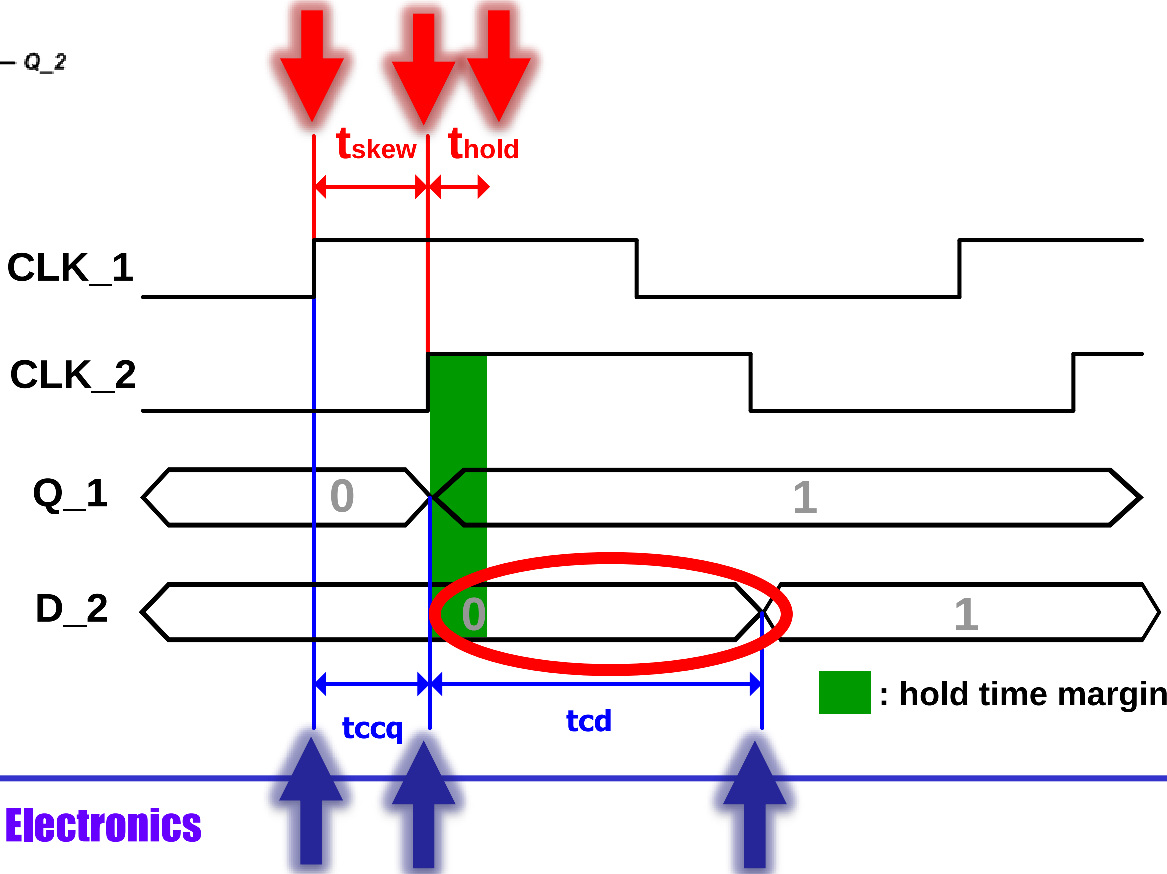




Hold Time Criterion

- ✓ **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





🟩 : hold time margin of DFF_2

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)**

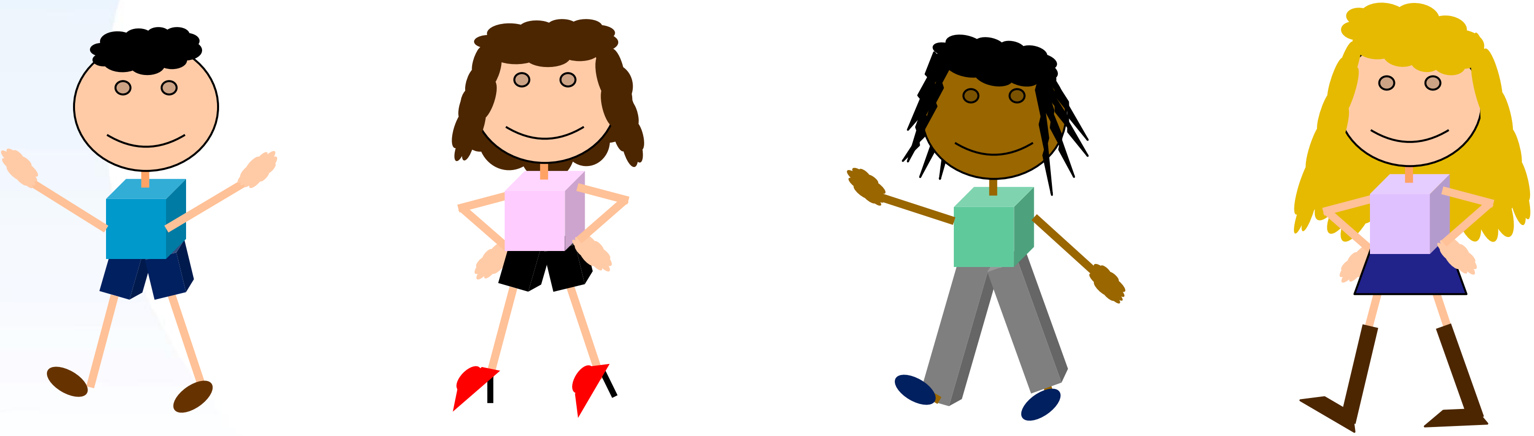
Outline

- ✓ **Section 1- Timing**
  - – Setup/hold time
  - – Pipeline
- ✓ **Section 2- Designware**

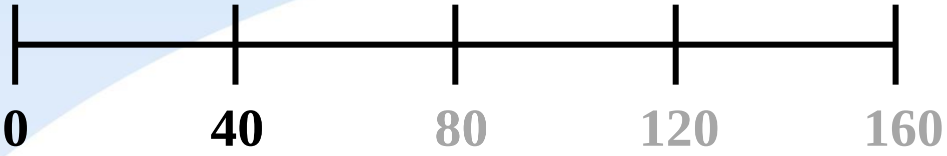
Trade-off between Area and Timing

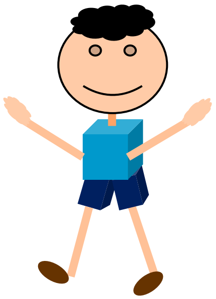
Area: 1 unit

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



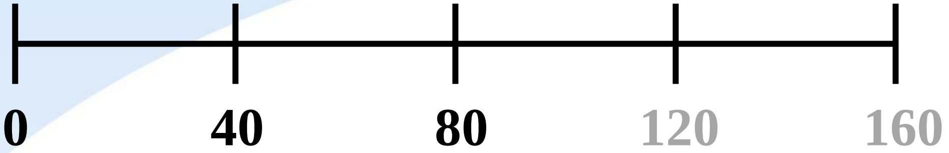
Trade-off between Area and Timing

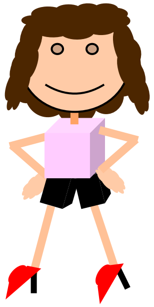




Wash and Dry = 40 mins

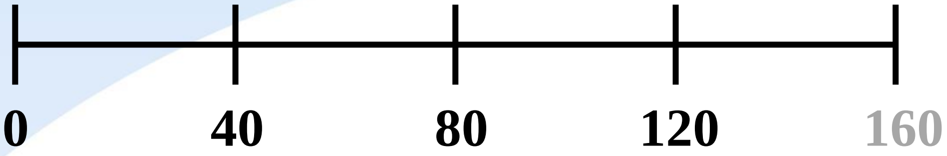
Trade-off between Area and Timing



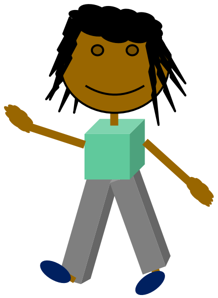


Wash and Dry = 40 mins

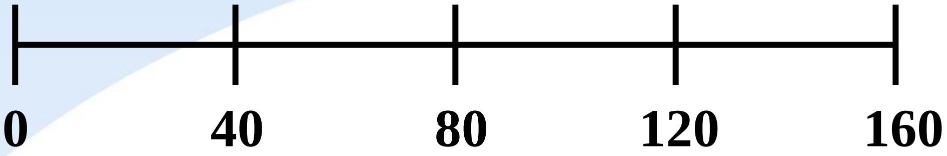
Trade-off between Area and Timing



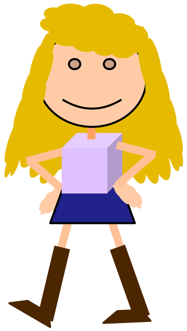
Wash and Dry = 40 mins



Trade-off between Area and Timing



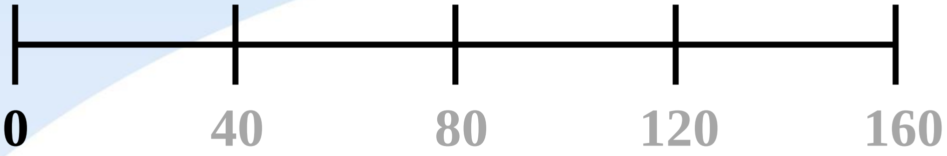
Wash and Dry = 40 mins



Area: 1 unit

Time: 160 mins

Trade-off between Area and Timing



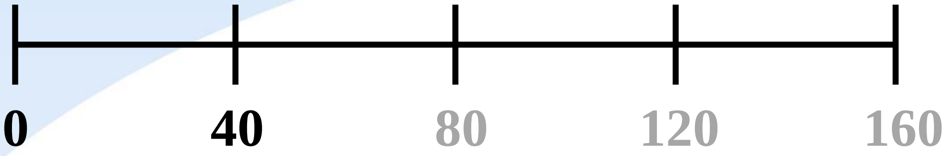
Wash and Dry = 40 mins

Area: 1 unit

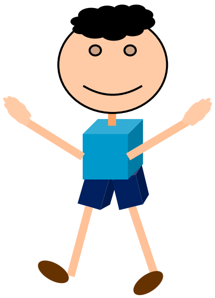
Time: 160 mins

Wash and Dry = 40 mins

Trade-off between Area and Timing



Wash and Dry = 40 mins

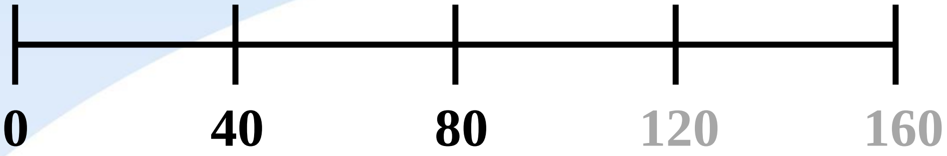


Area: 1 unit

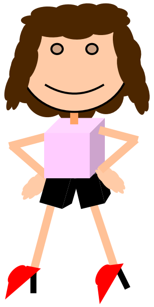
Time: 160 mins

Wash and Dry = 40 mins

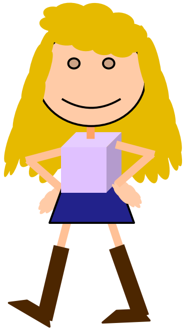
Trade-off between Area and Timing



Wash and Dry = 40 mins



Wash and Dry = 40 mins



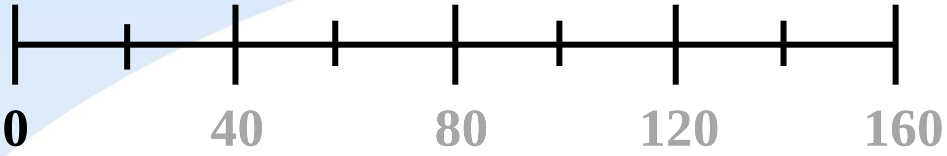
~~Area: 1 unit~~

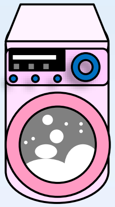
~~Time: 160 mins~~

Area: 2 units

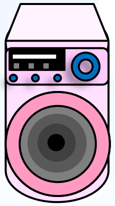
Time: 80 mins

Trade-off between Area and Timing





Wash 20 mins
Area 0.7 units

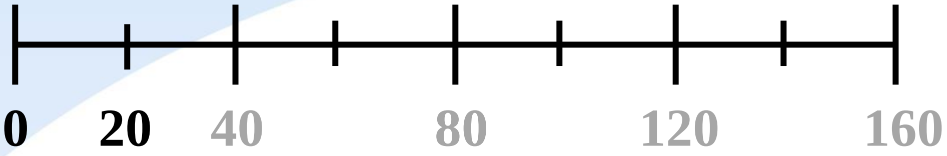


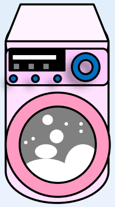
Dry 20 mins
Area 0.7 units

Area: 1 unit

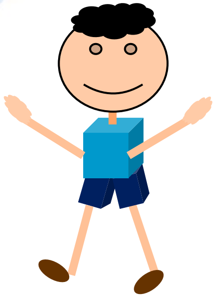
Time: 160 mins

Trade-off between Area and Timing



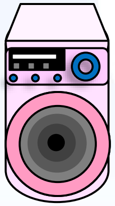


Wash 20 mins



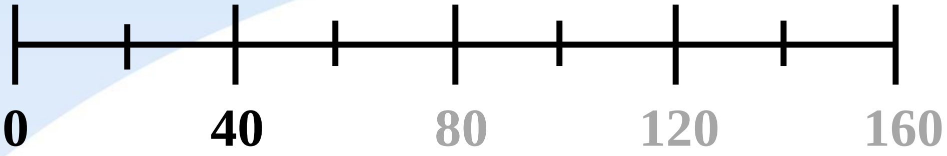
Area: 1 unit

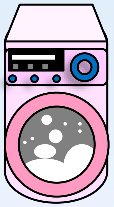
Time: 160 mins



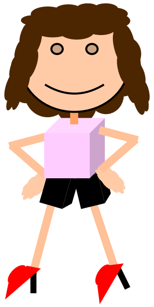
Dry 20 mins

Trade-off between Area and Timing



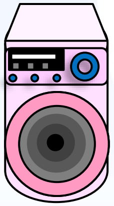


Wash 20 mins

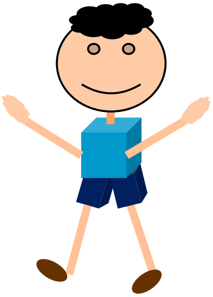


Area: 1 unit

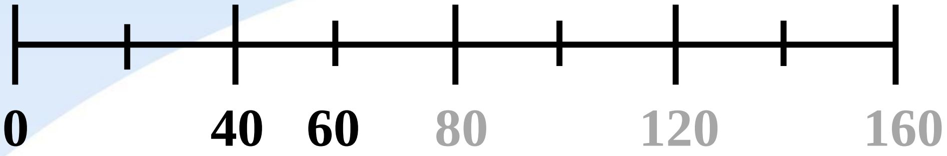
Time: 160 mins

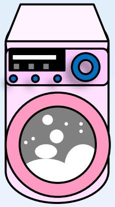


Dry 20 mins

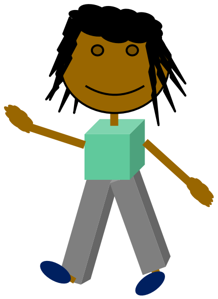


Trade-off between Area and Timing



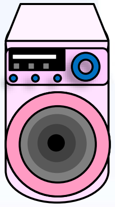


Wash 20 mins

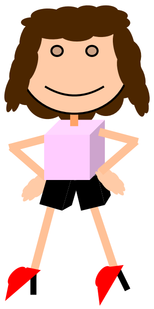


Area: 1 unit

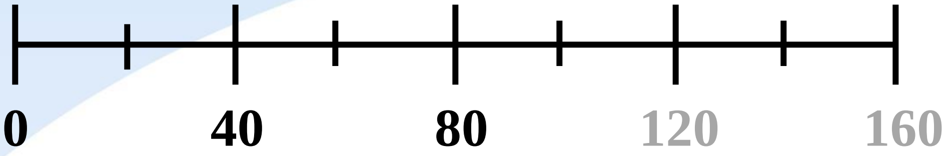
Time: 160 mins

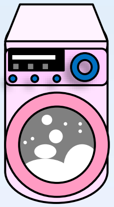


Dry 20 mins

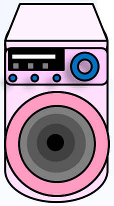


Trade-off between Area and Timing





Wash 20 mins

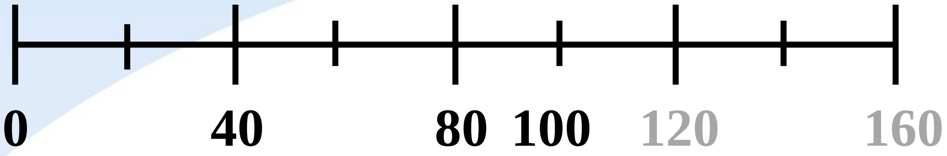


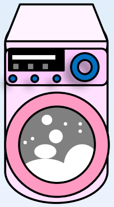
Dry 20 mins

Area: 1 unit

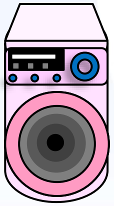
Time: 160 mins

Trade-off between Area and Timing

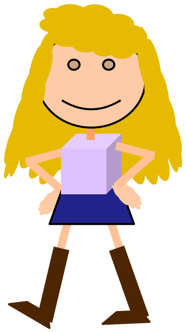




Wash 20 mins



Dry 20 mins



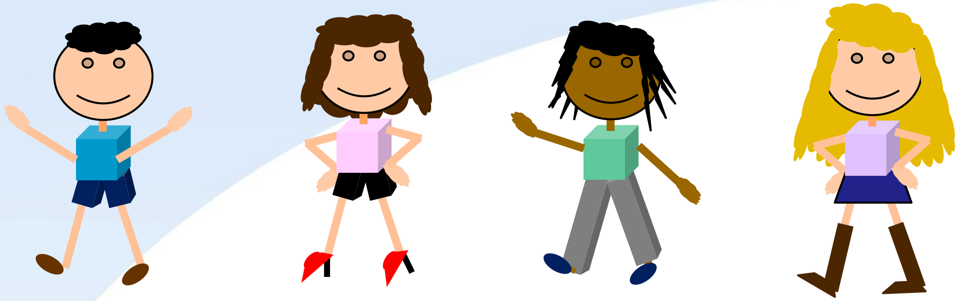
~~Area: 1 unit~~

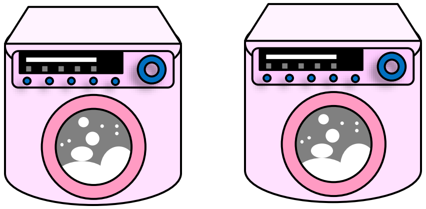
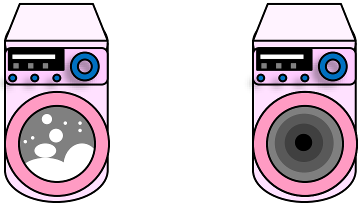
~~Time: 160 mins~~

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

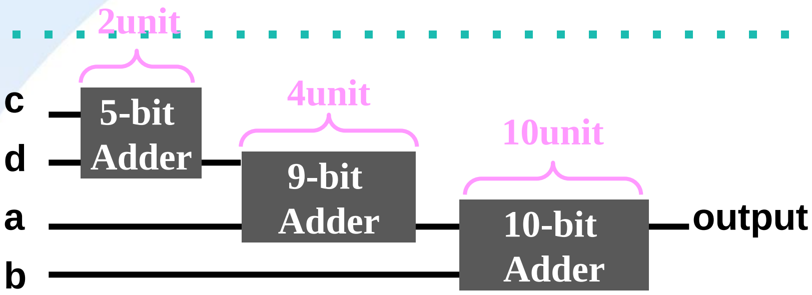
✓ a [7:0] , b [7:0] , c [3:0] , d [3:0]

✓ Q: (a + b + c + d) x 1000 iterations ?

Basic		
Parallel		
Pipeline		

Trade-off between Area and Timing

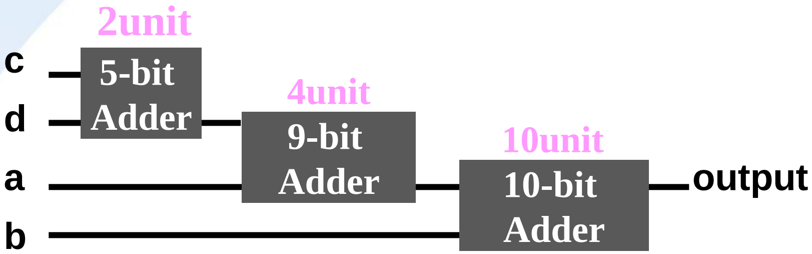
- ✓ a [7:0] , b [7:0] , c [3:0] , d [3:0]
- ✓ Q: (a + b + c + d) x 1000 iterations ?

Basic		Area: 24 units Time: 16 x 1000 = 16000 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) x 1000 iterations ?

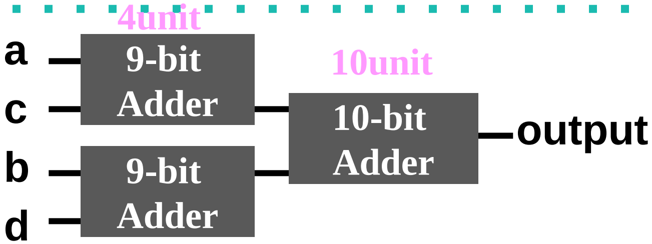
Basic



Area: 24 units

Time: 16 x 1000 = 16000 units

Parallel



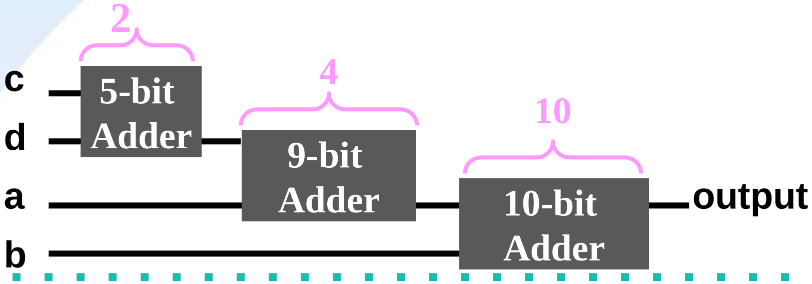
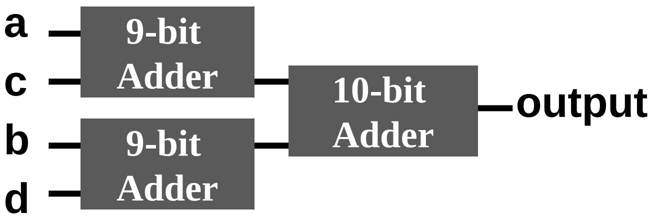
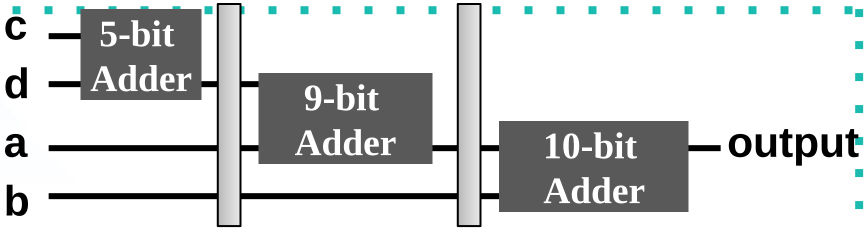
Area: 28 units

Time: 14 x 1000 = 14000 units

Pipeline

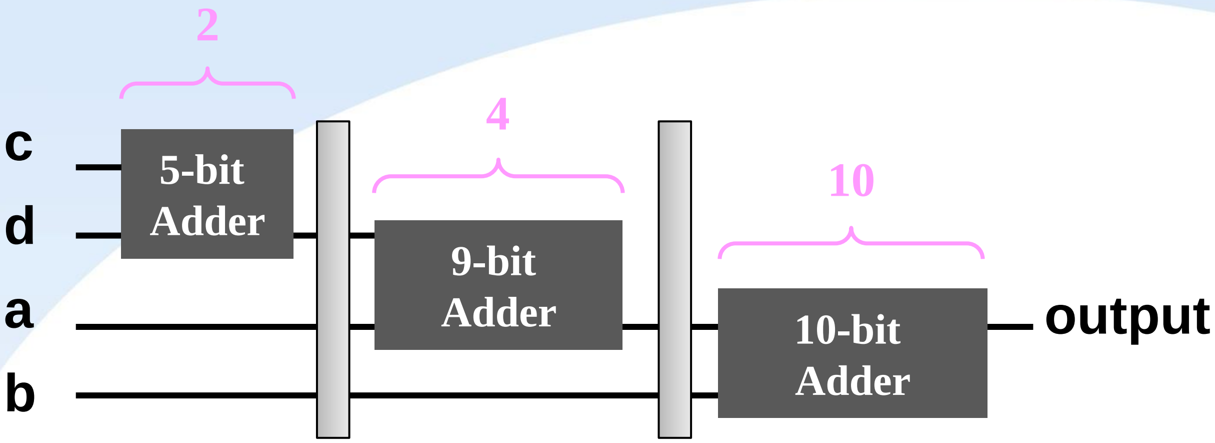
Trade-off between Area and Timing

- ✓ a [7:0] , b [7:0] , c [3:0] , d [3:0]
- ✓ Q: (a + b + c + d) x 1000 iterations ?

Basic	 2 4 10 c d a b 5-bit Adder 9-bit Adder 10-bit Adder output	Area: 24 units Time: 16 x 1000 = 16000 units
Parallel	 a c b d 9-bit Adder 9-bit Adder 10-bit Adder output	Area: 28 units Time: 14 x 1000 = 14000 units
Pipeline	 c d a b 5-bit Adder 9-bit Adder 10-bit Adder output	

Trade-off between Area and Timing

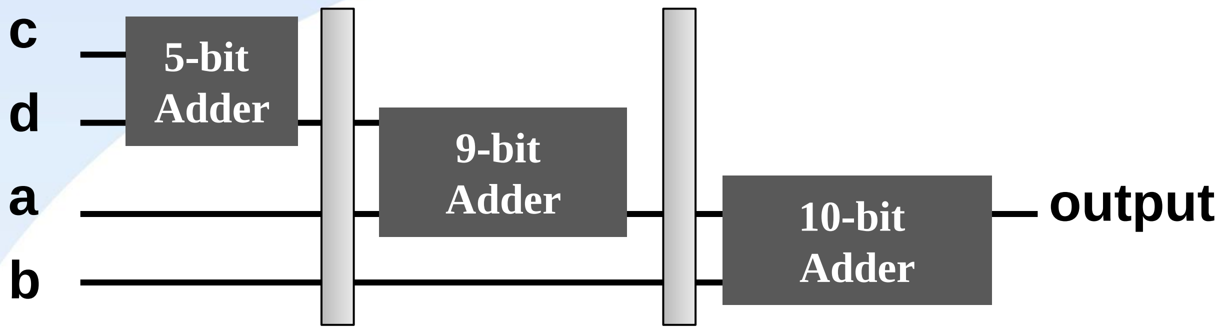
Pipeline



T=0	c1 + d1		

Trade-off between Area and Timing

Pipeline



c

d

a

b

5-bit Adder

9-bit Adder

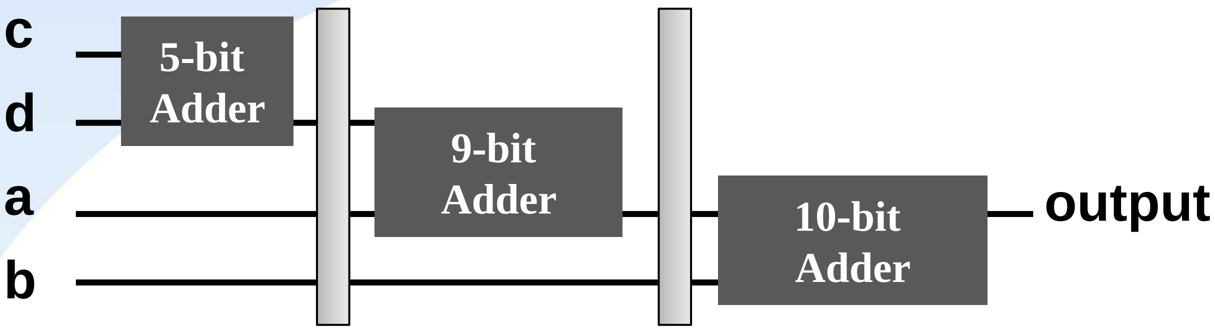
10-bit Adder

output

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

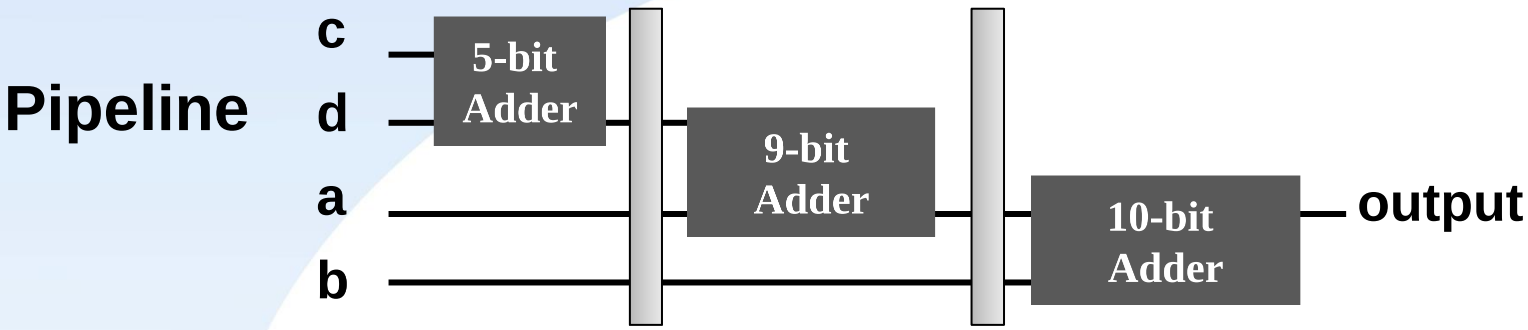
Trade-off between Area and Timing

Pipeline



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

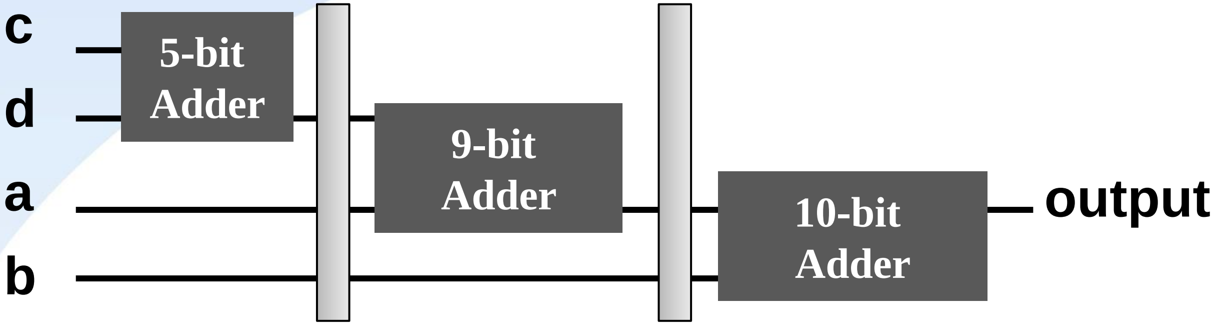
Trade-off between Area and Timing



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

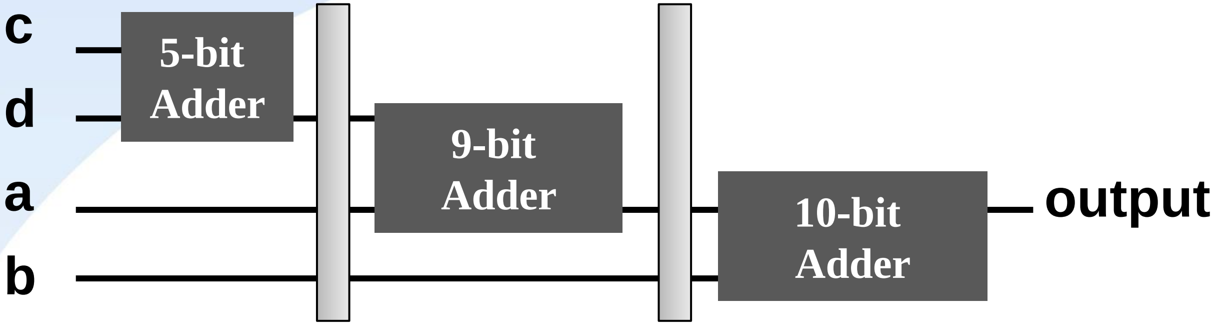
Pipeline



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)

Trade-off between Area and Timing

Pipeline

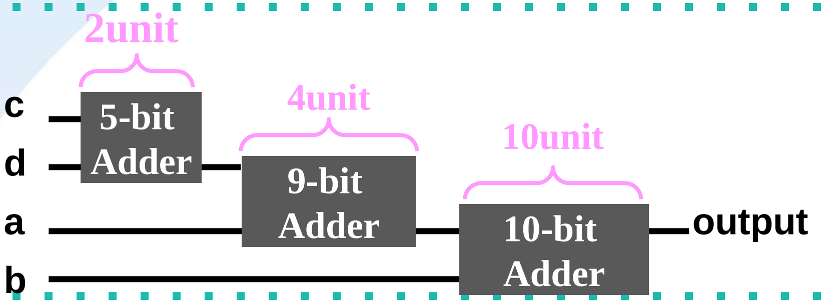


T=0	c1 + d1		
T=1	c2 + d2	a1 + (c1 + d1)	
T=2	c3 + d3	a2 + (c2 + d2)	b1 + (a1 + c1 + d1)
T=3	c4 + d4	a3 + (c3 + d3)	b2 + (a2 + c2 + d2)
T=4		a4 + (c4 + d4)	b3 + (a3 + c3 + d3)
T=5			b4 + (a4 + c4 + d4)

Trade-off between Area and Timing

- ✓ a [7:0] , b [7:0] , c [3:0] , d [3:0]
- ✓ Q: (a + b + c + d) x 1000 iterations ?

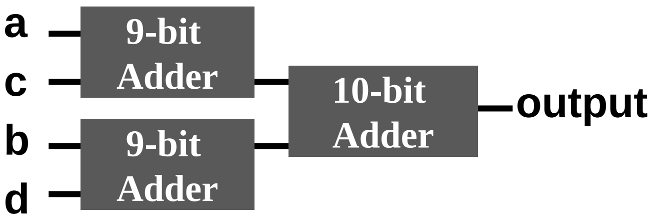
Basic



Area: 24 units

Time: 16 x 1000 = 16000 units

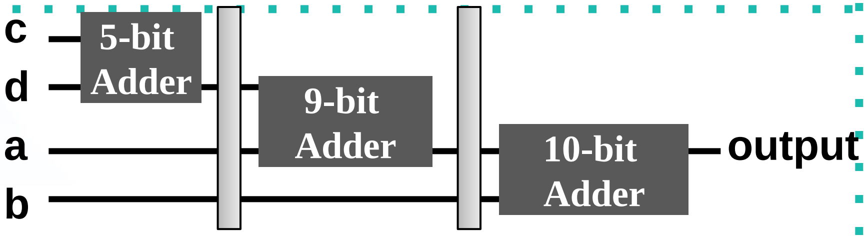
Parallel



Area: 28 units

Time: 14 x 1000 = 14000 units

Pipeline

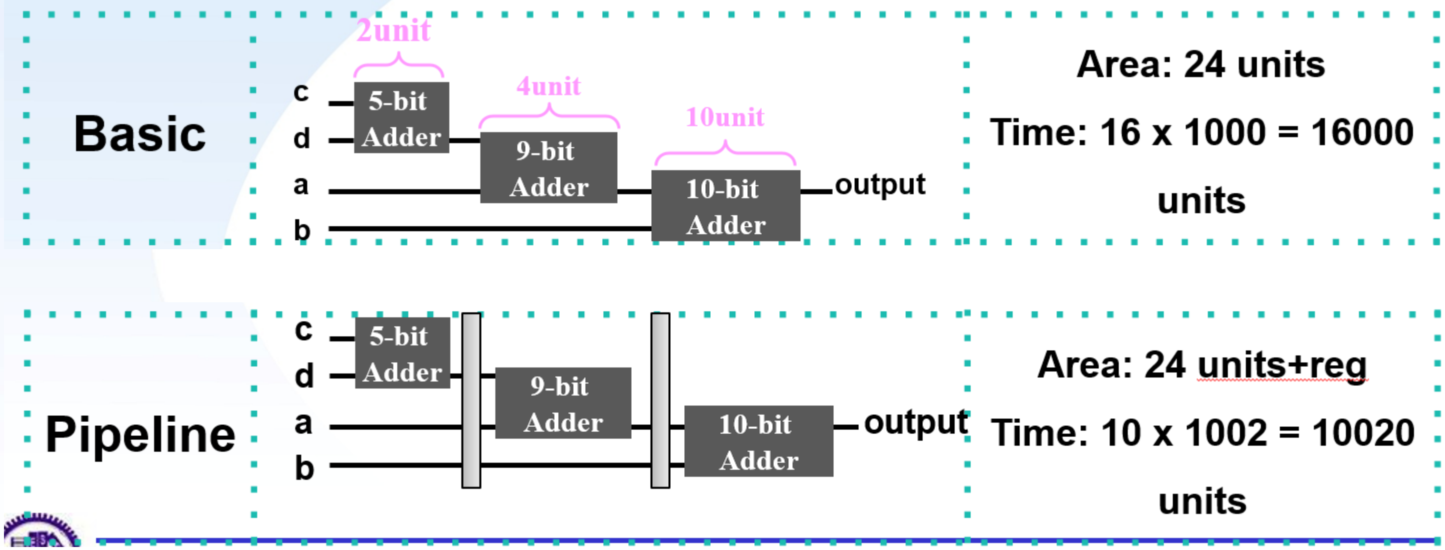


Area: 24 units+reg

Time: 10 x 1002 = 10020 units

Pipeline Speedup

- ✓ **Latency of single task** ☹ ✗
- ✓ **Throughput → Speed Up** ☺ ✓
  - Potential speedup = Number pipe stages, if all stages are balanced.
  - Pipeline rate limited by slowest stage
  - Note the overhead of pipeline



Area: 24 units

Time: 16 x 1000 = 16000 units

Area: 24 units+reg

Time: 10 x 1002 = 10020 units

Outline

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

Overview of DesignWare

- ✓ **IP (Intellectual Property)**
  - – Soft IP : RTL design, requires verification.
  - – Firm IP : Netlist resource, less used.
  - – Hard IP : GDSII format, high performance but technology dependent.
- ✓ **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 SYNOPSYS 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
- 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
    - Supply default function only, can not use special function.
  - Instantiate IP
    - Use SYNOPSYS 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:

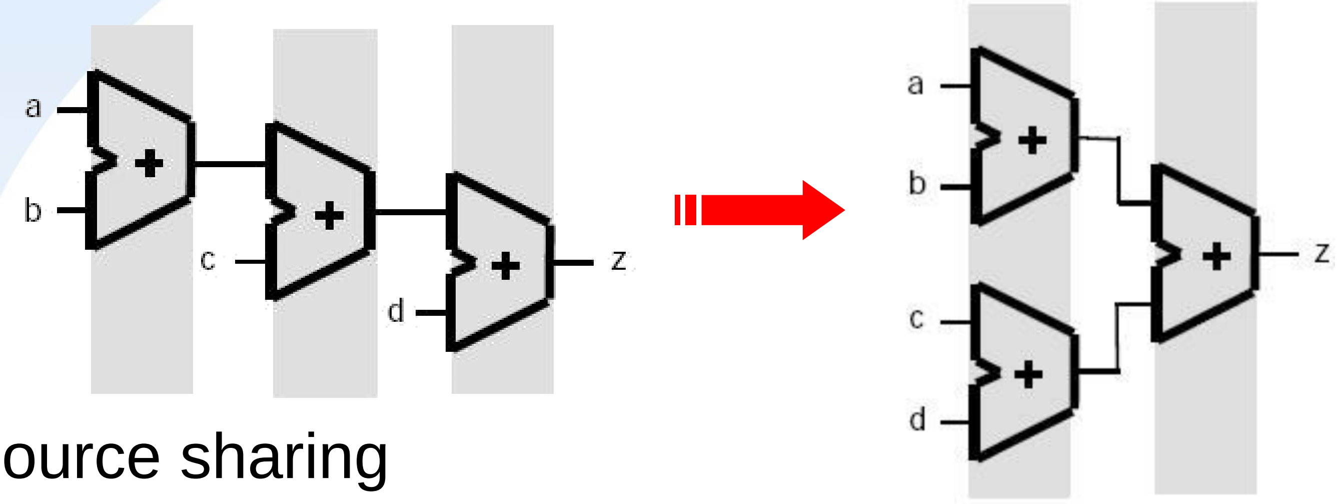
Synthetic Operators	HDL Operator
adder	**+, +1**
subtractor	**-, -1**
comparator	**==, <, <=, >, >=**
multiplier	*****
selector	**If, case**

Operator Inference (2/3)

✓ **High-level synthesis**

– Arithmetic optimization

- Arithmetic level optimization, ex: a+b+c+d -> (a+b)+(c+d)

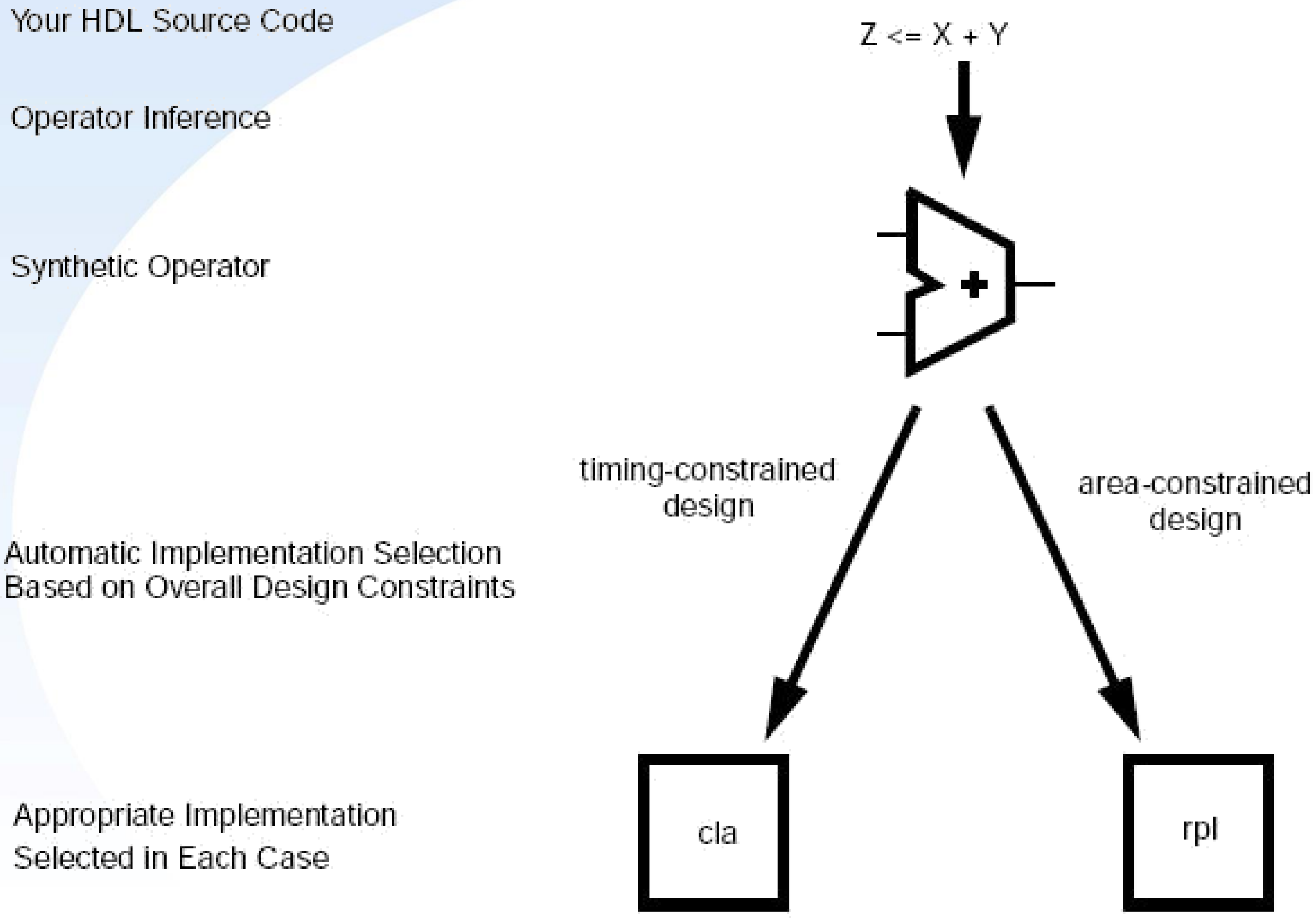


– Resource sharing

- 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

Z <= X + Y

timing-constrained design

area-constrained design

cla

rpl

Instantiate IP (1/9)

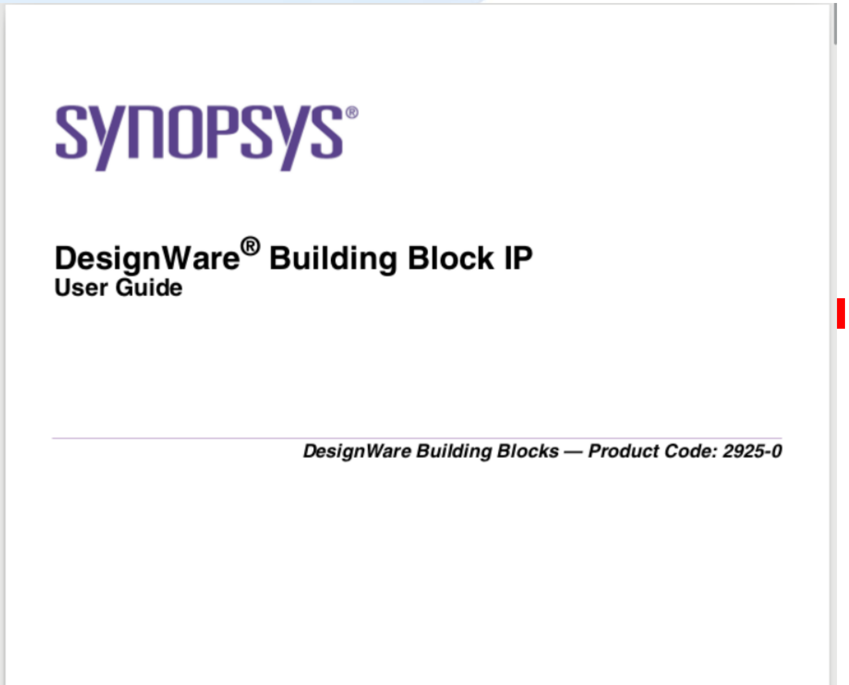
- ✓ **Instantiation IP**
  - To instantiate a synthetic module manually and explicitly.
  - Need to include a reference to the synthetic module in HDL code.

- ✓ **SYNOPSYS online document**
  - Command:

```
evince /usr/cad/synopsys/synthesis/cur/dw/doc/manuals/dwbb_userguide.pdf &
```

Instantiate IP (2/9)

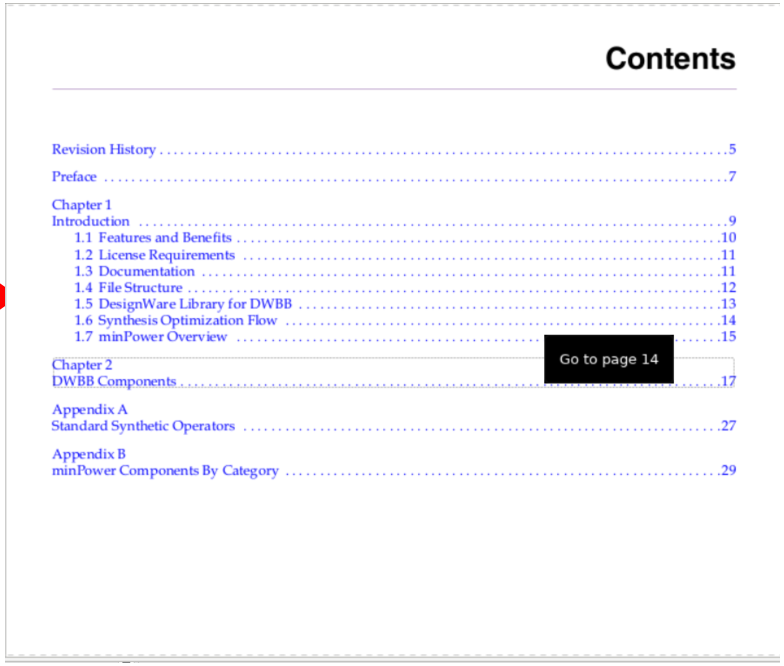
- ✓ **SYNOPSYS online document**
  - Select section2.0



SYNOPSYS®

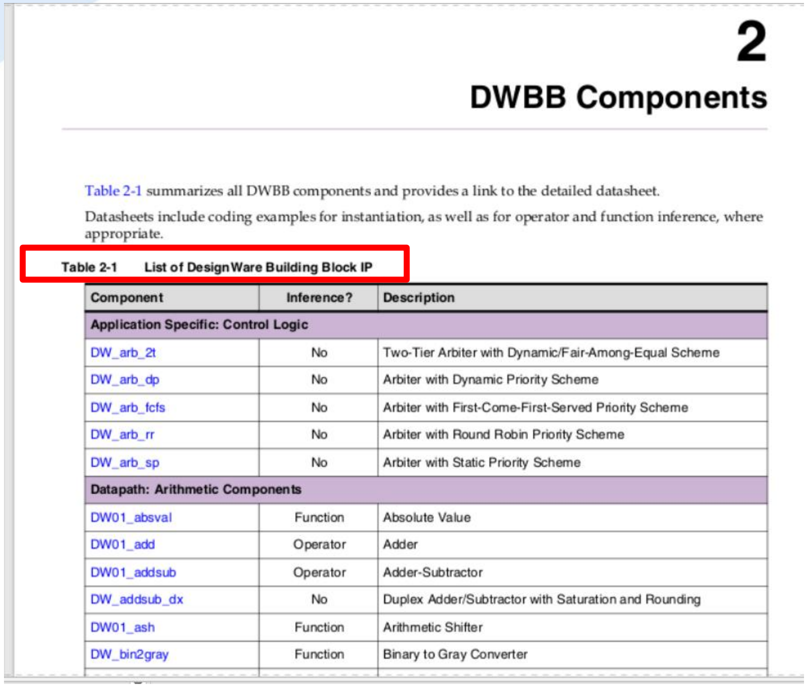
DesignWare® Building Block IP
User Guide

DesignWare Building Blocks — Product Code: 2925-0



Contents

Instantiate IP (3/9)



2

DWBB Components

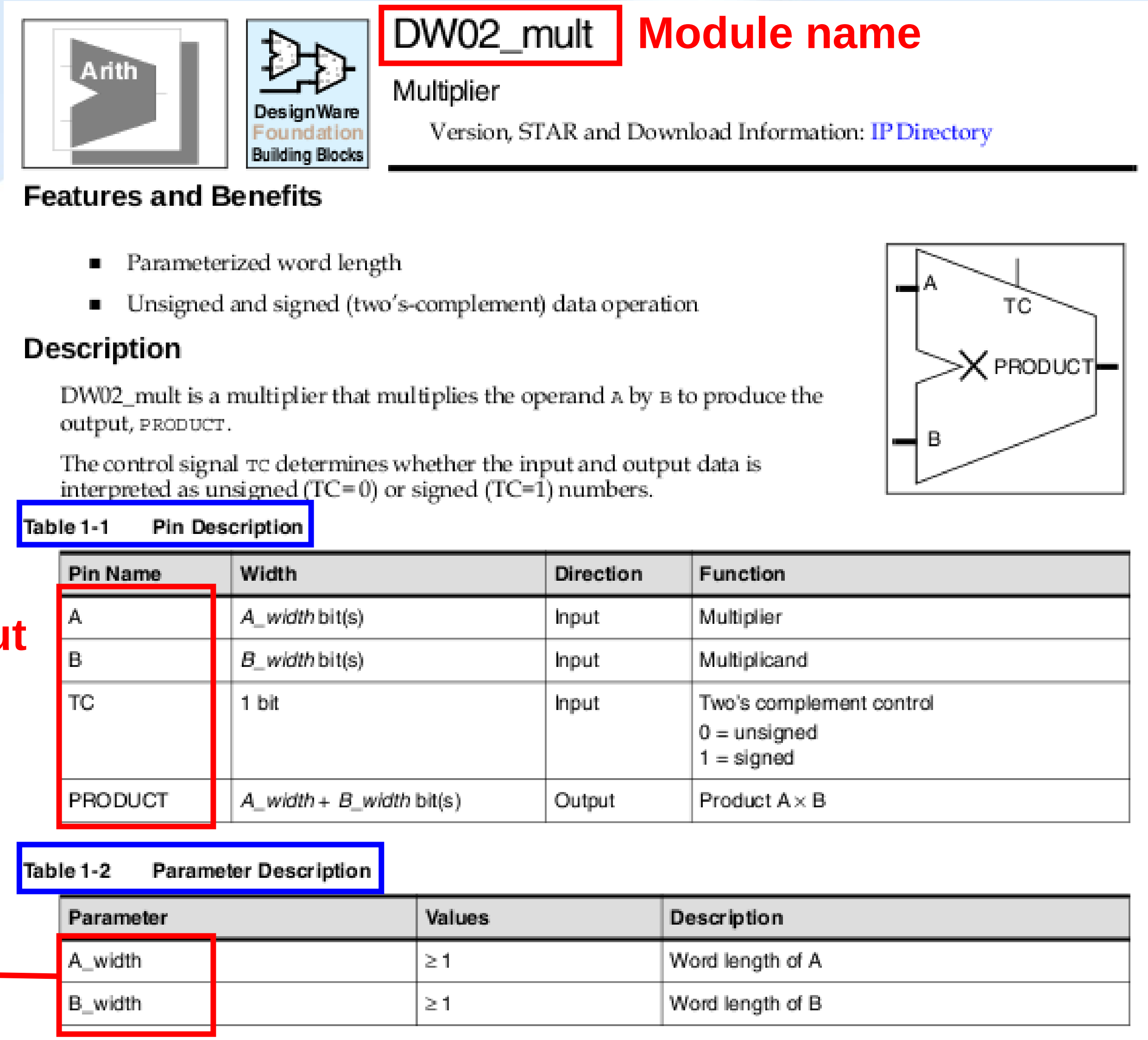
Table 2-1 summarizes all DWBB components and provides a link to the detailed datasheet.

Datasheets include coding examples for instantiation, as well as for operator and function inference, where appropriate.

Table 2-1 List of DesignWare Building Block IP

Component	Inference?	Description
Application Specific: Control Logic		
DW_arb_2t	No	Two-Tier Arbiter with Dynamic/Fair-Among-Equal Scheme
DW_arb_dp	No	Arbiter with Dynamic Priority Scheme
DW_arb_fcfs	No	Arbiter with First-Come-First-Served Priority Scheme
DW_arb_rr	No	Arbiter with Round Robin Priority Scheme
DW_arb_sp	No	Arbiter with Static Priority Scheme
Datapath: Arithmetic Components		
DW01_absval	Function	Absolute Value
DW01_add	Operator	Adder
DW01_addsub	Operator	Adder-Subtractor
DW_addsub_dx	No	Duplex Adder/Subtractor with Saturation and Rounding
DW01_ash	Function	Arithmetic Shifter
DW_bin2gray	Function	Binary to Gray Converter

Instantiate IP (4/9)



Arith

DesignWare Foundation Building Blocks

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	*A_width* bit(s)	Input	Multiplier
B	*B_width* bit(s)	Input	Multiplicand
TC	1 bit	Input	Two's complement control 0 = unsigned 1 = signed
PRODUCT	*A_width* + *B_width* bit(s)	Output	Product A × B

input & output

Table 1-2 Parameter Description

Parameter	Values	Description
A_width	≥ 1	Word length of A
B_width	≥ 1	Word length of B

Argument assignment:

DW02_ mult #(N,N)

mult01(…, …, …, …);

Instantiate IP (5/9)

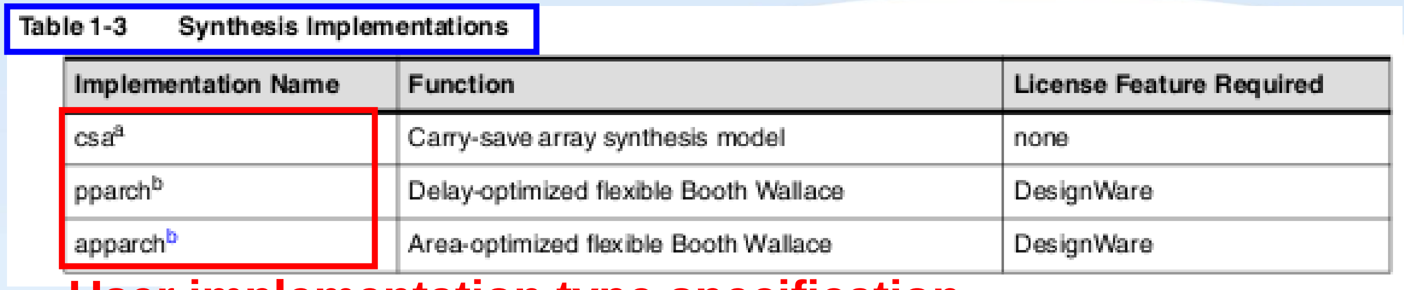


Table 1-3 Synthesis Implementations

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

User implementation type specification

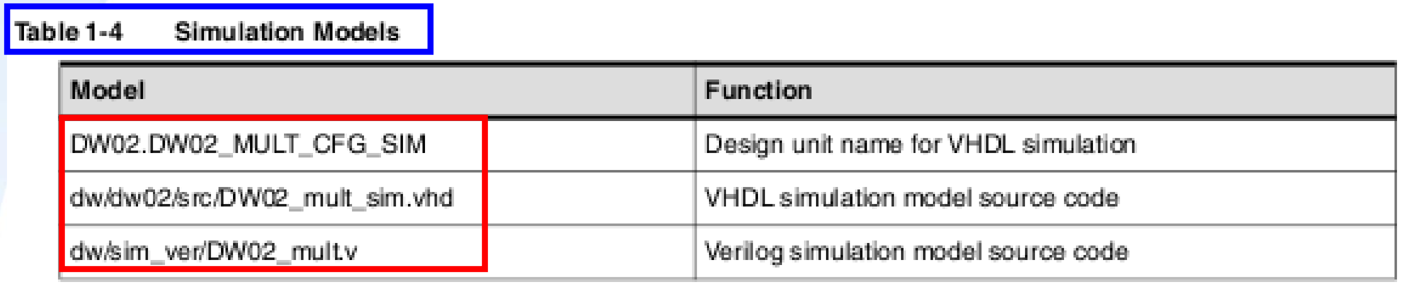


Table 1-4 Simulation Models

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

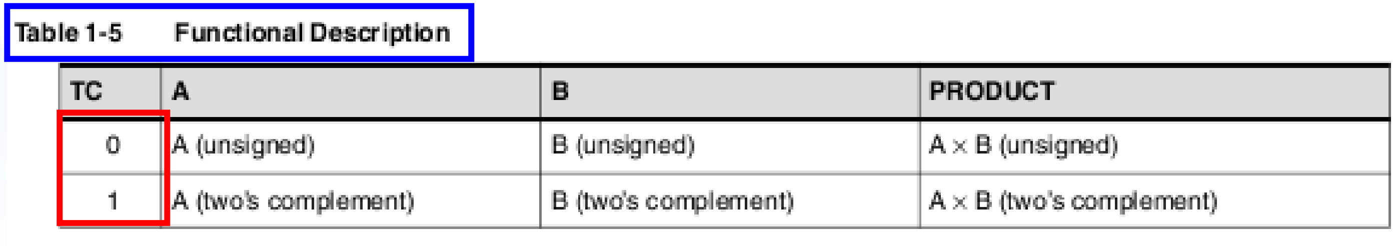


Table 1-5 Functional Description

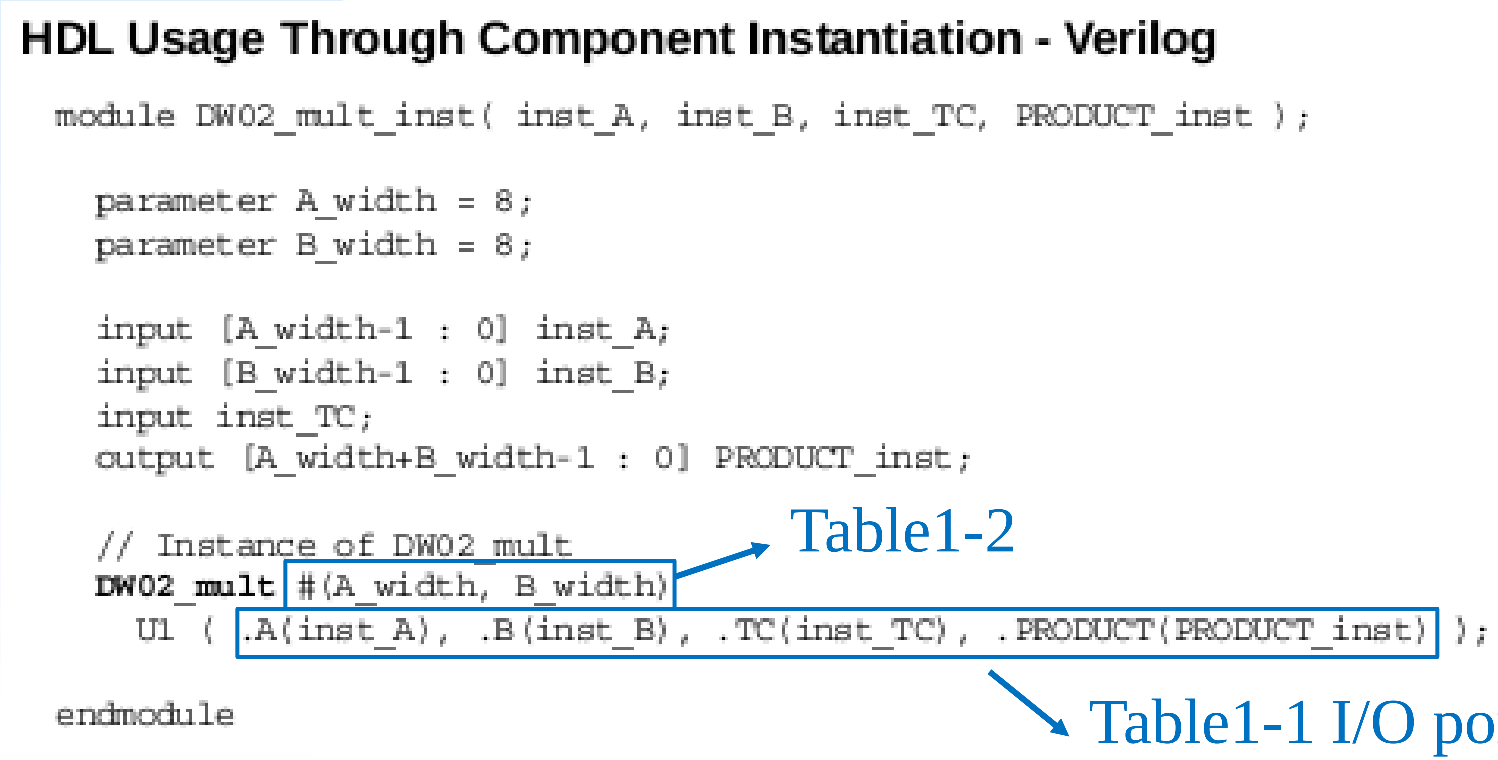
TC	A	B	PRODUCT
0	A (unsigned)	B (unsigned)	A × B (unsigned)
1	A (two's complement)	B (two's complement)	A × B (two's complement)

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 ”

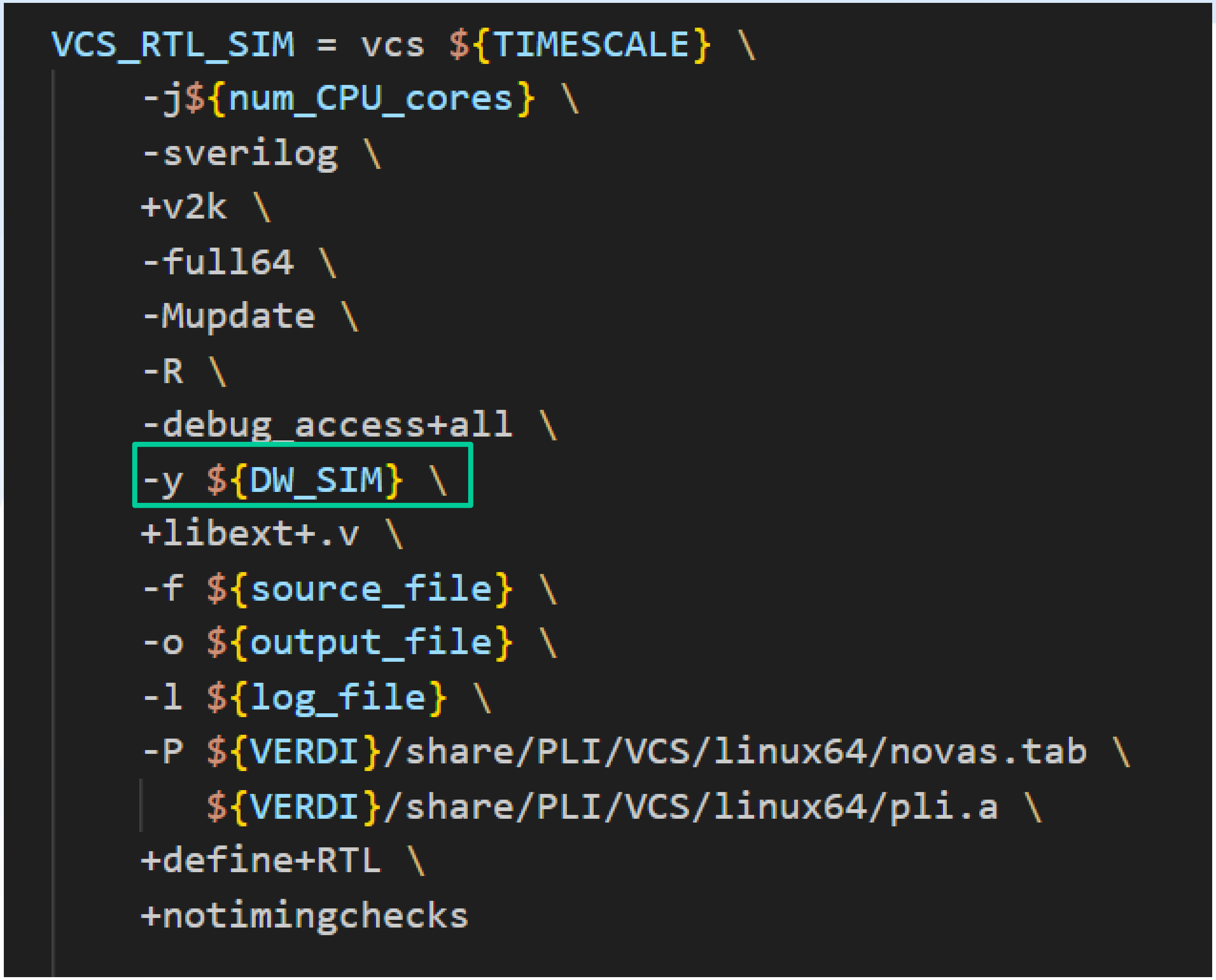
```
`include /usr/synthesis/dw/sim_ver/DW02_mult.v“
```

- ✓ **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 (7/9)



```
VCS_RTL_SIM = vcs ${TIMESCALE} \
    -j${num_CPU_cores} \
    -sverilog \
    +v2k \
    -full64 \
    -Mupdate \
    -R \
    -debug_access+all \
    -y ${DW_SIM} \
    +libext+.v \
    -f ${source_file} \
    -o ${output_file} \
    -l ${log_file} \
    -P ${VERDI}/share/PLI/VCS/linux64/novas.tab \
       ${VERDI}/share/PLI/VCS/linux64/pli.a \
    +define+RTL \
    +notimingchecks
```

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

```
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
```

Reference

- https://blog.51cto.com/u_15076209/4702482?fbclid=IwAR3V4tEEPMQ_NJKI-2AFvaEksIUzPBvww5E7yqvpRDmujNUTkDat7bBCKQ0

- https://zhuanlan.zhihu.com/p/278523793?fbclid=IwAR2W0cuazZ8Ci5G_C1QCMDMiBqBC42YasmEW67hLJZuaRdU6VeCPjOuoYgE