Chapter 2.8-2.12
Introduction to VHDL

Version: 2023/11/22

Chapter 2 Introductin to VHDL

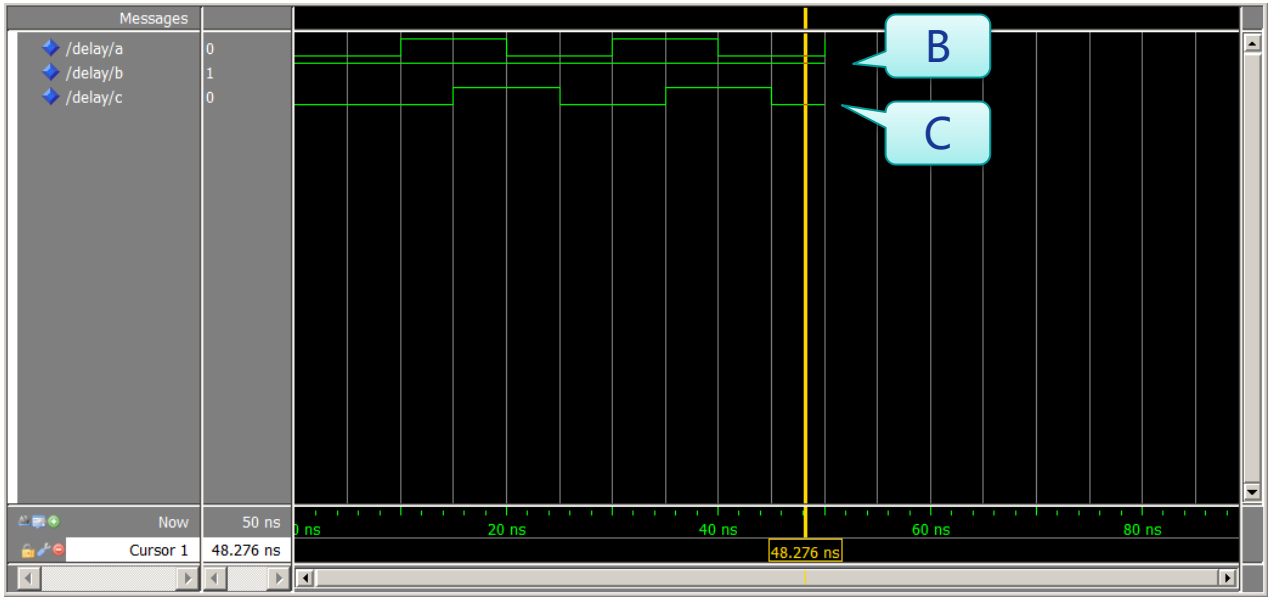
1	Computer-aided design	11	Simple synthesis examples
2	Hardware description language	12	VHDL models for multiplexers
3	VHDL description of combinational circuits	13	VHDL libraries
4	VHDL modules	14	Modeling registers and counters using VHDL processes
5	Sequential statements and VHDL processes	15	Behavioral and structural VHDL
6	Modeling flip-flop using VHDL processes	16	Variables, signals, and constants
7	Processes using wait statemetns	17	Arrays
8	**Two types of VHDL delays: Transport and inertial delays**	18	Loops in VHDL
9	Complication,simulation, and synthesis of VHDL code	19	Assert and report statements
10	VHDL data types and operatos		

2.8 Two types of VHDL delays

```
entity delay is
  port(
    A: buffer bit;
    B: in bit;
    C: out bit);
end delay;

architecture equ of delay is

begin
  A <= not A after 10 ns;
  C <= A and B after 5 ns;
end equ;
```

Square wave with period 20 ns

AND gate with propagation delay of 5 ns



2.8 Two types of VHDL delays

```
entity delay is
  port(
    A: buffer bit;
    B: in bit;
    C: out bit);
end delay;

architecture equ of delay is

begin
  A <= not A after 1 ns;
  C <= A and B after 5 ns;
end equ;
```

10 ns → 1 ns

If AND gate is simulated with inputs that change very often in comparison to the gate delay, the simulation output will **NOT** show the changes

How VHDL **delays** work?

2.8 Two types of VHDL delays

Delay types	
Transport delays (传输延时)	
Inertial delays (惯性延时)	default

Inertial delay models gates and other devices that do not propagate short pulses from input to output

If a gate has an ideal inertial delay T, in addition to delaying the input signals by T, **any pulse with a width less than T is rejected**

- Real devices do not behave in this way
- Perhaps they would reject very narrow spurious pulses, but it might be unreasonable to assume that all pulses narrower than the delay duration will be rejected

2.8 Two types of VHDL delays

```
signal_name <= expression after  delay-time;
```

```
signal_name <= reject pulse-width inertial expression after
          delay-time;
```

It evaluates the expression, rejects any pulses whose width is less than pulse-width, and then sets the signal equal to the result after a delay-time

rejection pulse width ≤ delay time

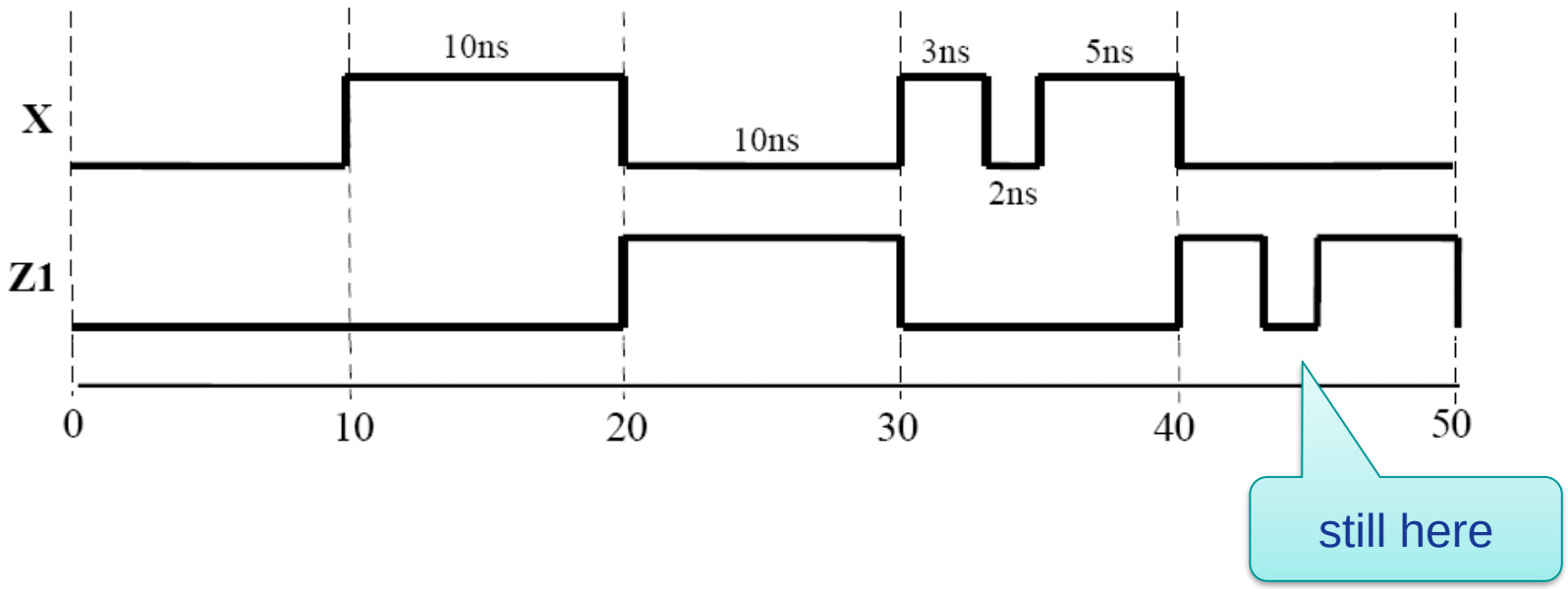
Transport delay

```
signal_name <= transport expression after delay-time;
```

- **Transport delay** is intended to model the delay introduced by wiring, simply delays an input signal by specified delay time
- In order to model this delay, the key word **transport** must be specified in the code

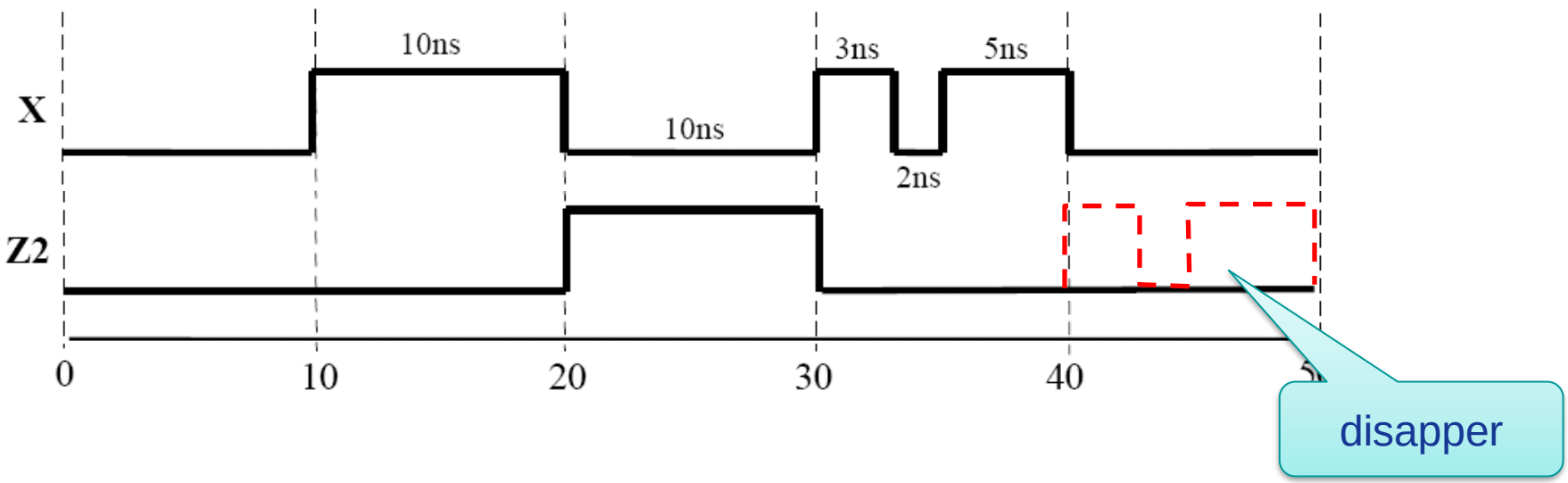
2.8 Two types of VHDL delays

```
Z1 <= transport X after 10 ns;
```



2.8 Two types of VHDL delays

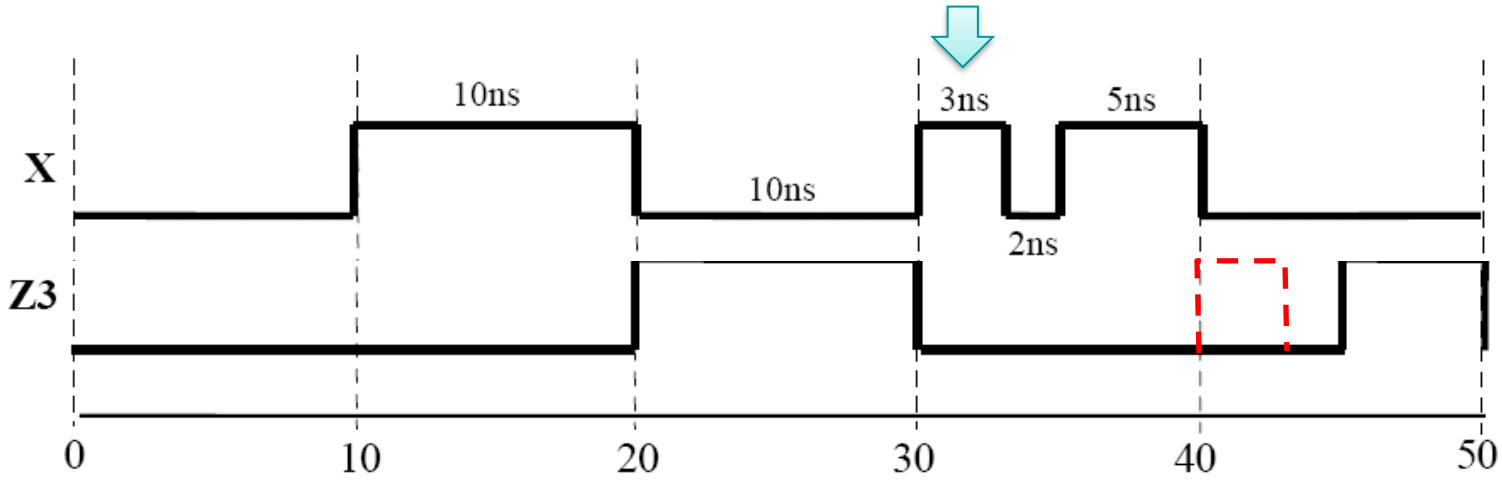
Z2 <= X **after** 10 ns; --Inertial delay



- The pulse rejection associated with inertial delay can inhibit many output changes
- In simulations with basic gates and simple circuits, one should make sure that test sequence that you apply are wider than the inertial delays of the modeled devices

2.8 Two types of VHDL delays

Z3 <= **reject** 4 ns **inertial** X **after** 10 ns;

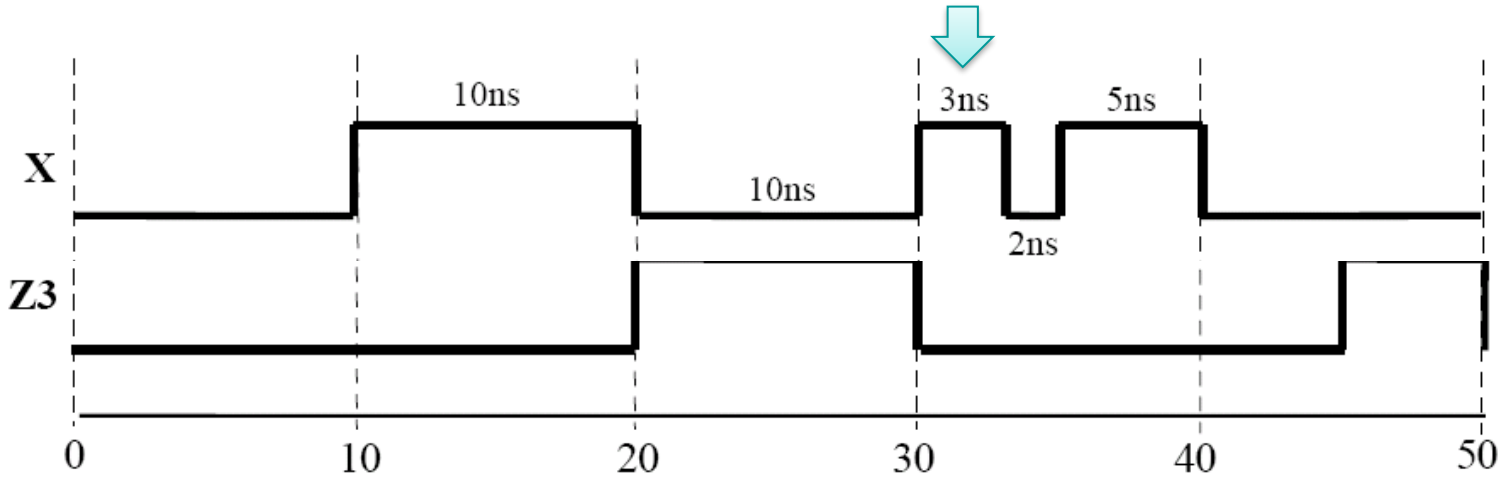


In general, using **reject** is equivalent to using a combination of an inertial and a transport delay

```
Zm <= X after _?_ ns; -- inertial delay rejects short pulses
Z3 <= transport Zm after _?_ ns; -- total delay is 10 ns
```

2.8 Two types of VHDL delays

Z3 <= **reject** 4 ns **inertial** X **after** 10 ns;

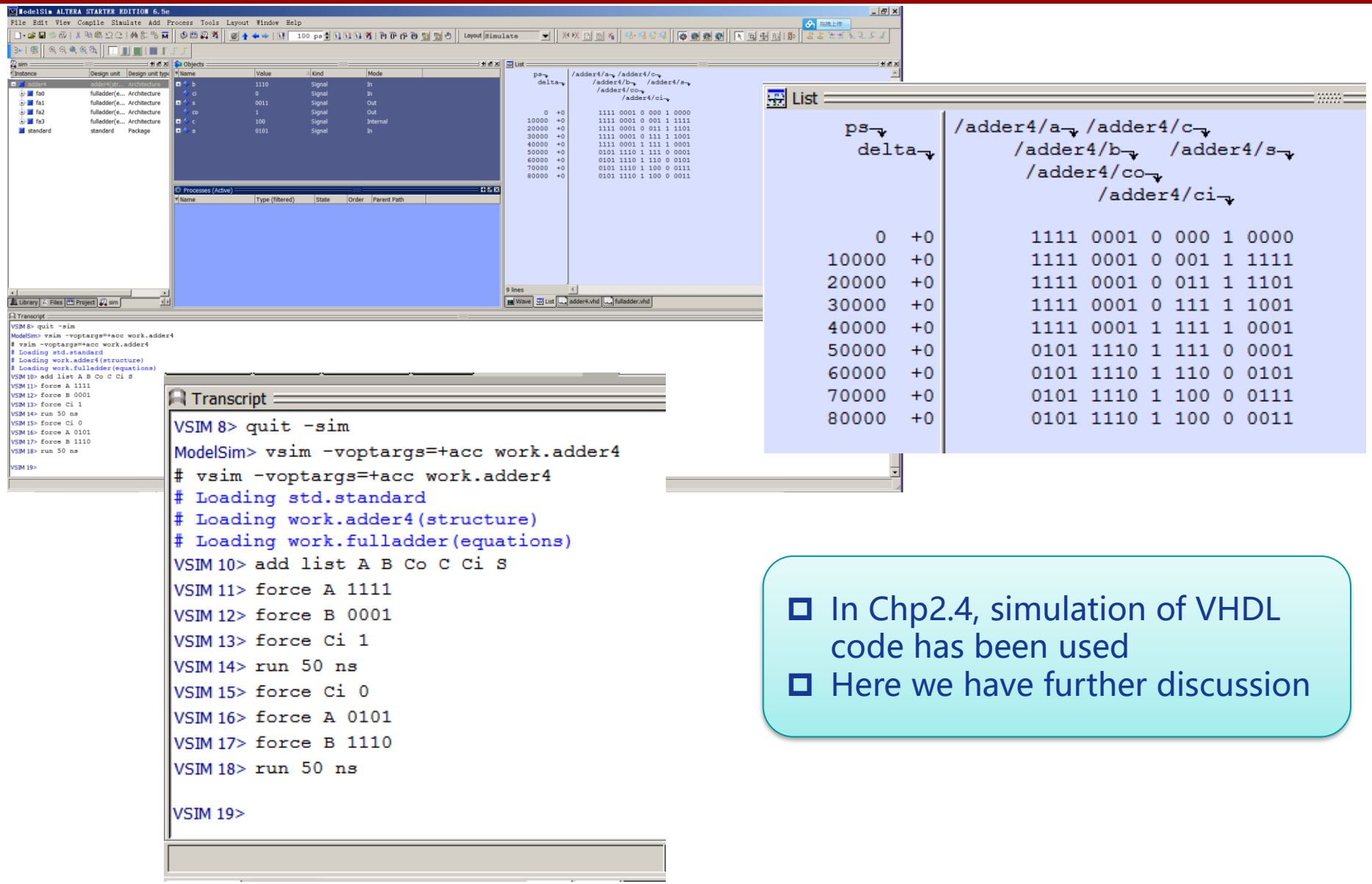



```
Zm <= X after 4 ns; -- inertial delay rejects short pulses
Z3 <= transport Zm after 6 ns; -- total delay is 10 ns
```

Chapter 2 Introductin to VHDL

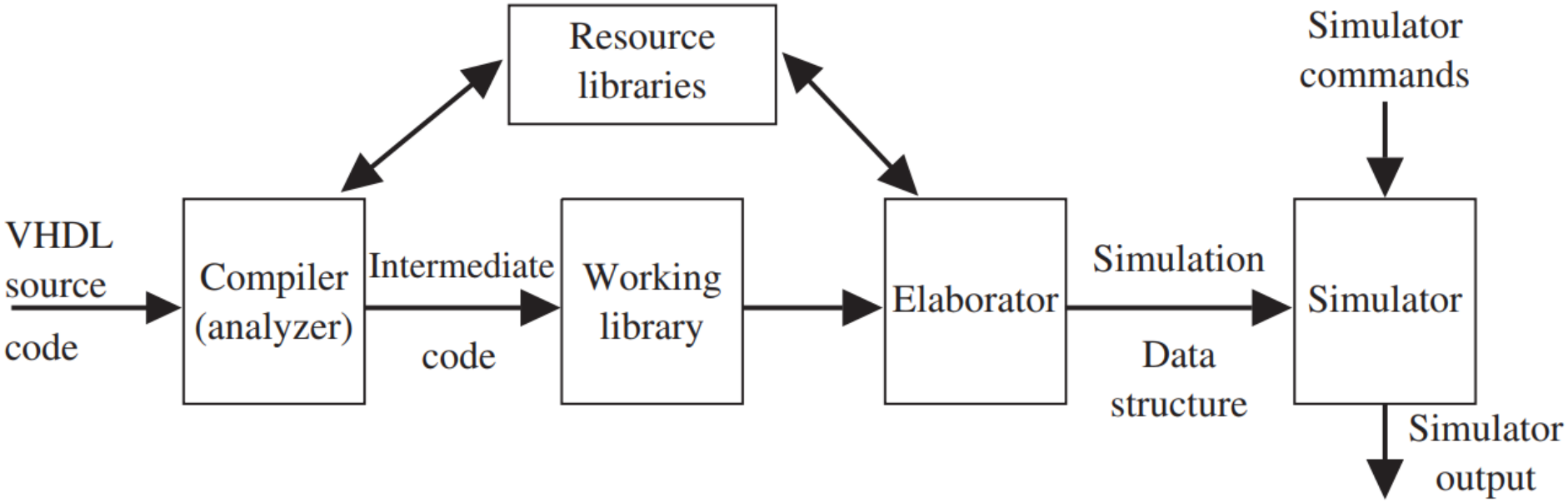
1	Computer-aided design	11	Simple synthesis examples
2	Hardware description language	12	VHDL models for multiplexers
3	VHDL description of combinational circuits	13	VHDL libraries
4	VHDL modules	14	Modeling registers and counters using VHDL processes
5	Sequential statements and VHDL processes	15	Behavioral and structural VHDL
6	Modeling flip-flop using VHDL processes	16	Variables, signals, and constants
7	Processes using wait statemetns	17	Arrays
8	Two types of VHDL delays: Transport and inertial delays	18	Loops in VHDL
9	**Complication, simulation, and synthesis of VHDL code**	19	Assert and report statements
10	VHDL data types and operatos		

2.9 Compilation, simulation, and synthesis of VHDL code



- In Chp2.4, simulation of VHDL code has been used
- Here we have further discussion

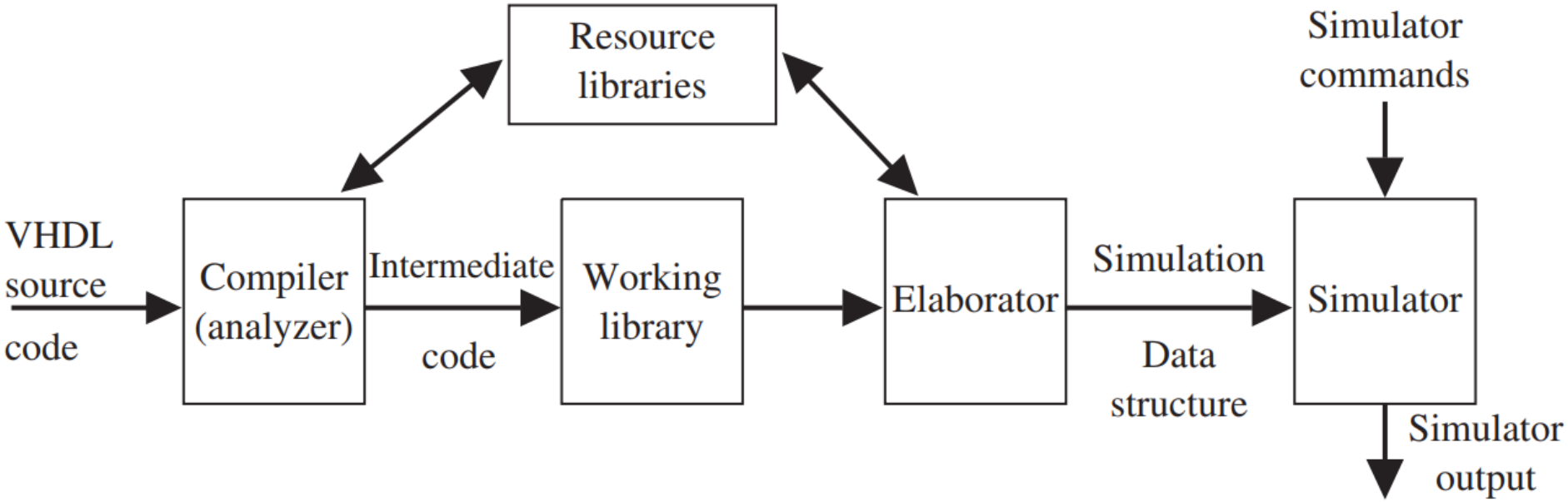
2.9 Compilation, simulation, and synthesis of VHDL code



Simulation of VHDL code is important

- to verify the VHDL code correctly implements intended design
- to verify that the design meets its specifications

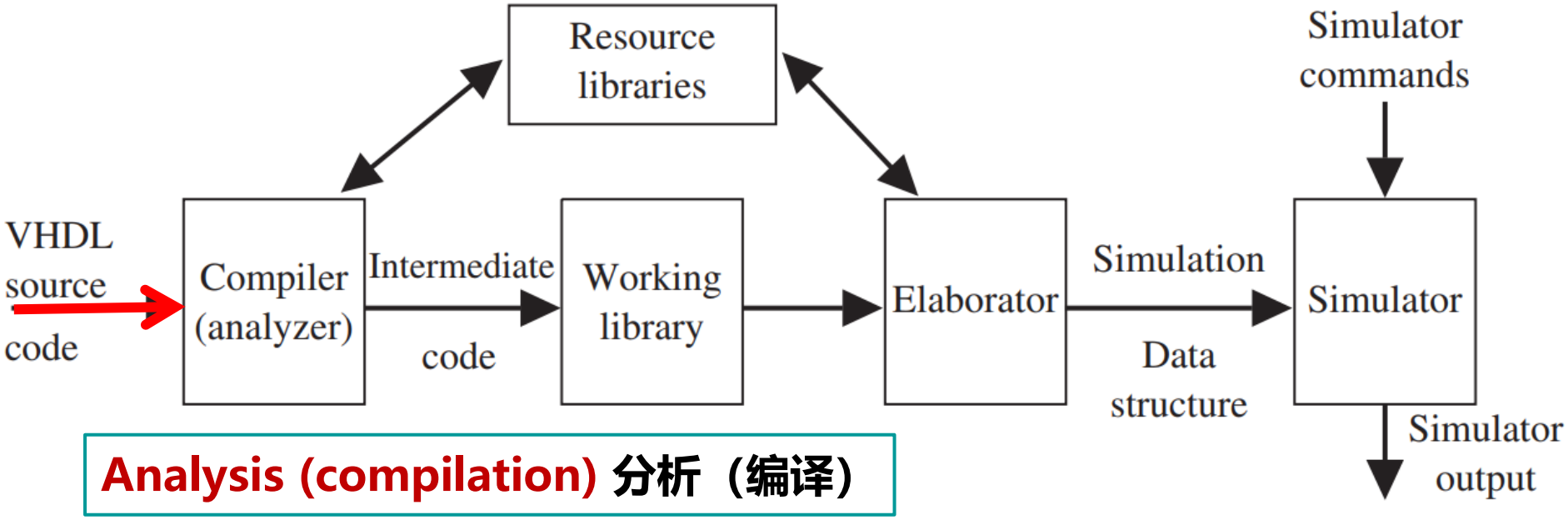
2.9 Compilation, simulation, and synthesis of VHDL code



Simulation of VHDL code:

1. **Analysis (compilation) 分析（编译）**
2. **Elaboration 细化**
3. **Simulation 仿真**

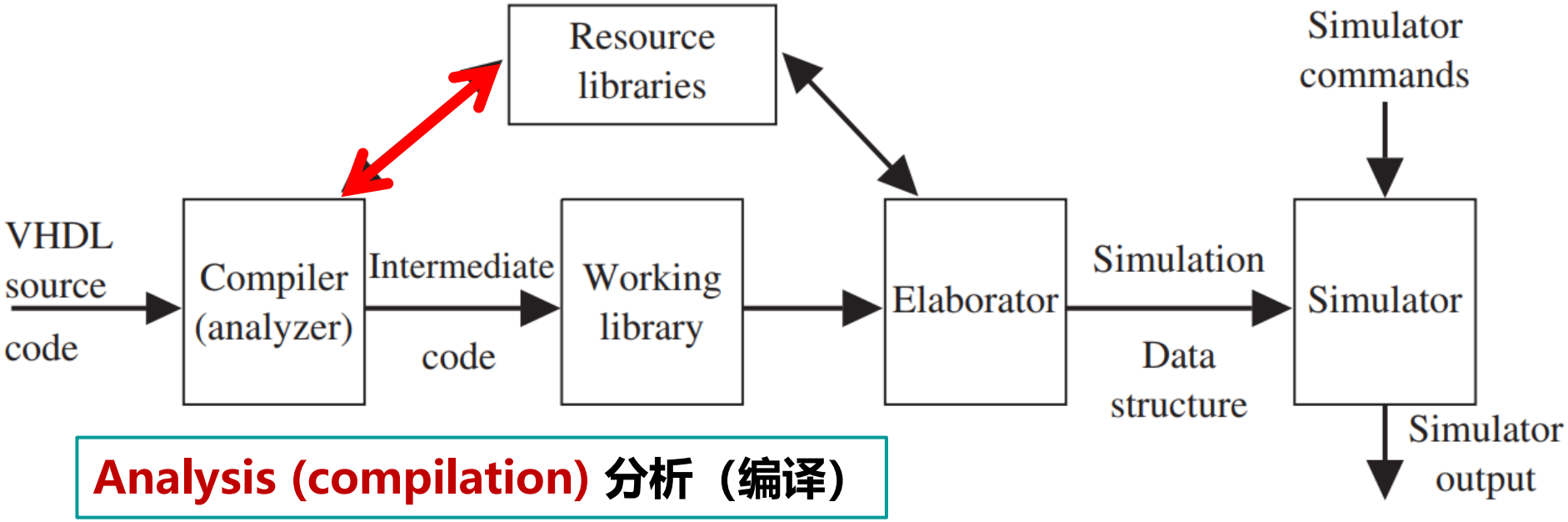
2.9 Compilation, simulation, and synthesis of VHDL code



Analysis (compilation) 分析（编译）

Compiler（**analyzer**）checks source code to see that it conforms to the syntax and semantic rules

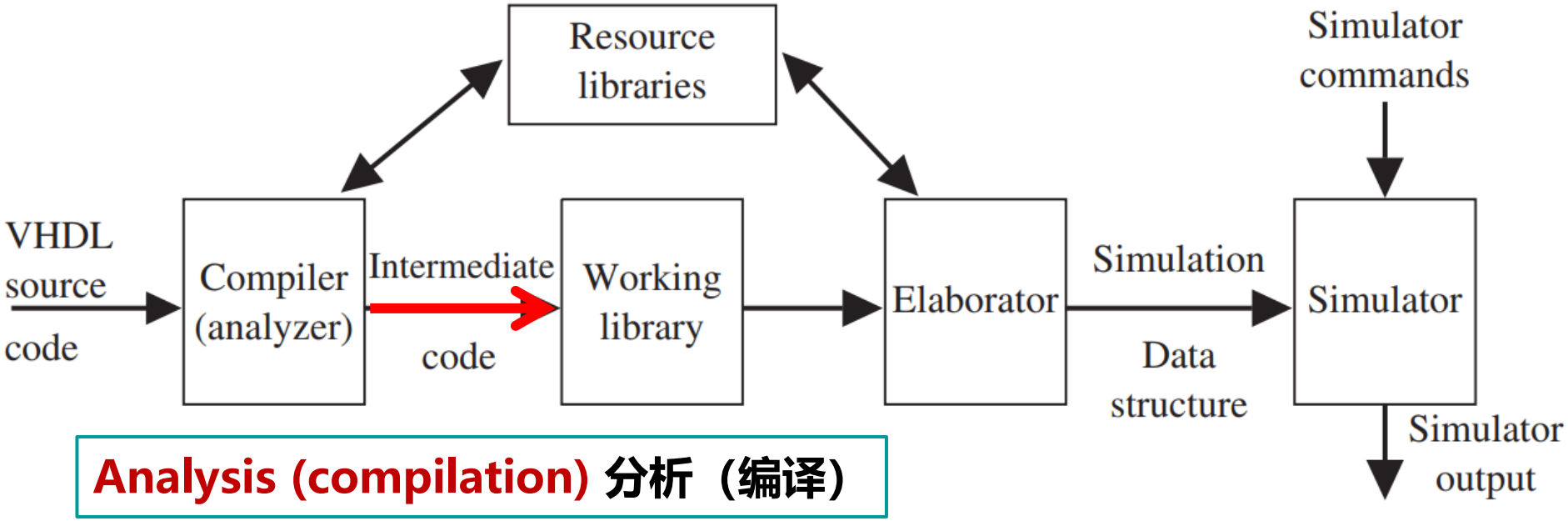
2.9 Compilation, simulation, and synthesis of VHDL code



Analysis (compilation) 分析（编译）

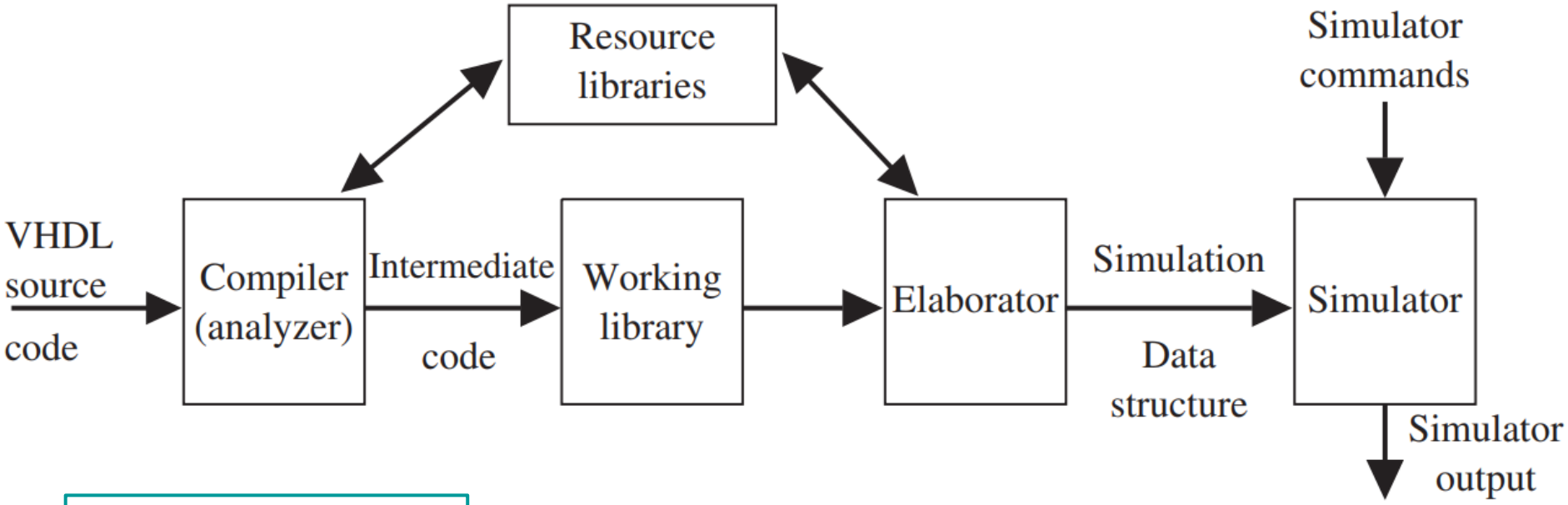
Compiler also checks to see that references to libraries are correct

2.9 Compilation, simulation, and synthesis of VHDL code



If VHDL code conforms to all of the rules, compiler generates **intermediate code**

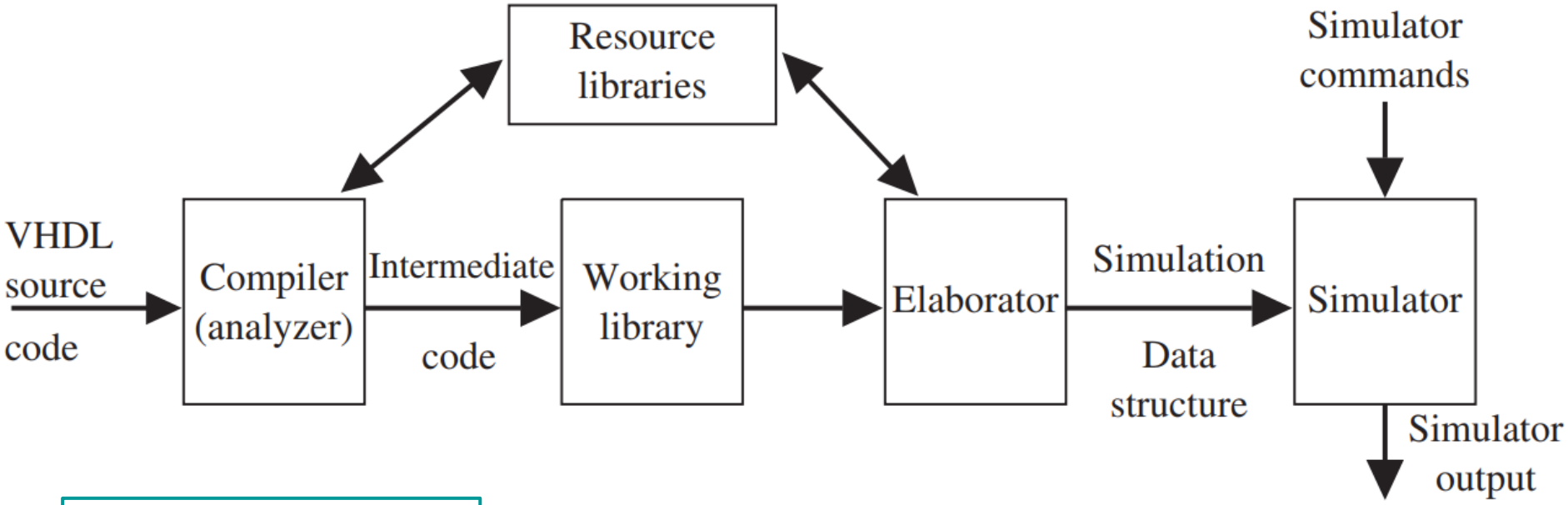
2.9 Compilation, simulation, and synthesis of VHDL code



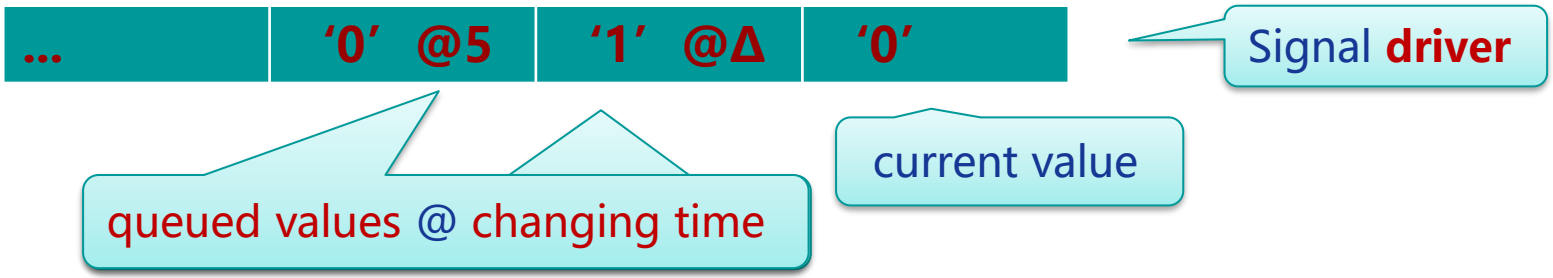
Elaboration 细化

Intermediate code are converted to a form which can be used by simulator

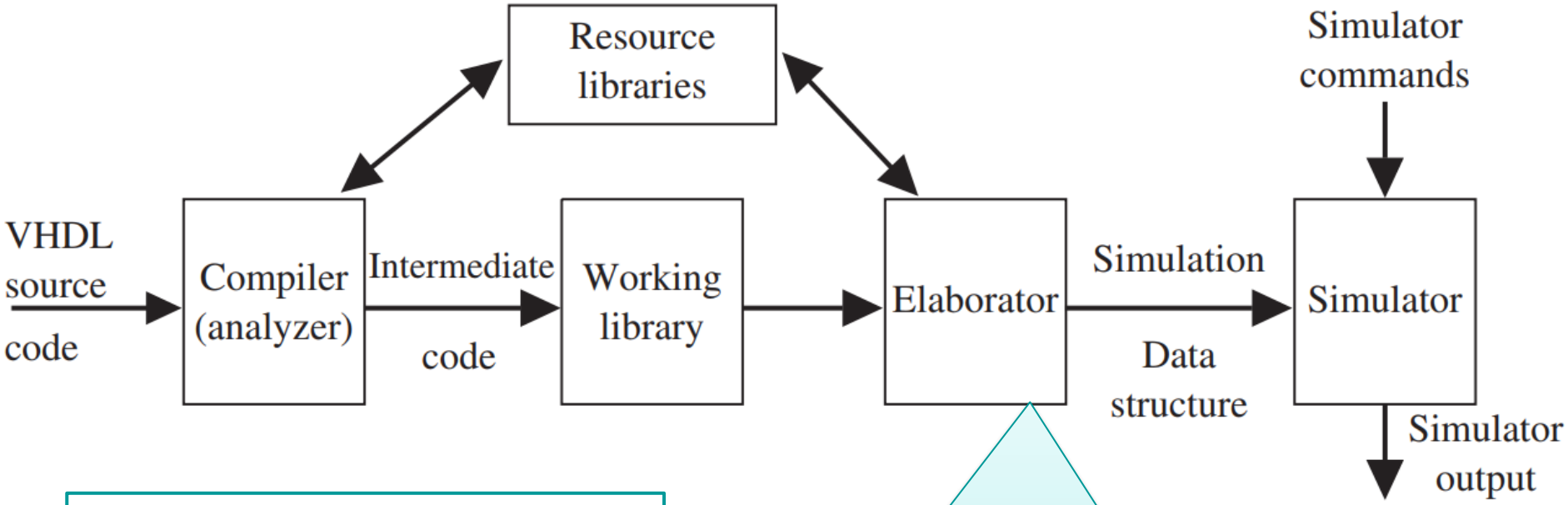
2.9 Compilation, simulation, and synthesis of VHDL code



Elaboration 细化



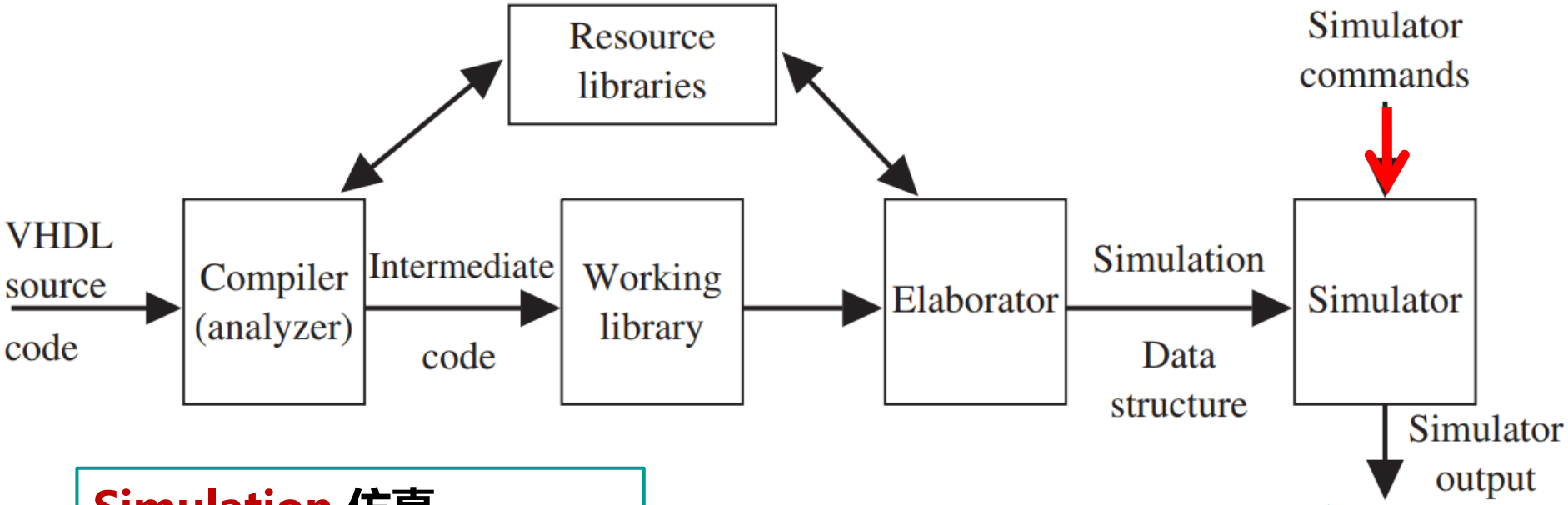
2.9 Compilation, simulation, and synthesis of VHDL code



Elaboration 细化

- **Ports** are created for each instance of a component
- **Memory storage** is allocated for the required signals
- The **interconnections** among the port signals are specified
- A mechanism is established for executing VHDL statements in the **proper sequence**
- The resulting **data structure** represents the digital system being simulated

2.9 Compilation, simulation, and synthesis of VHDL code



Simulation 仿真

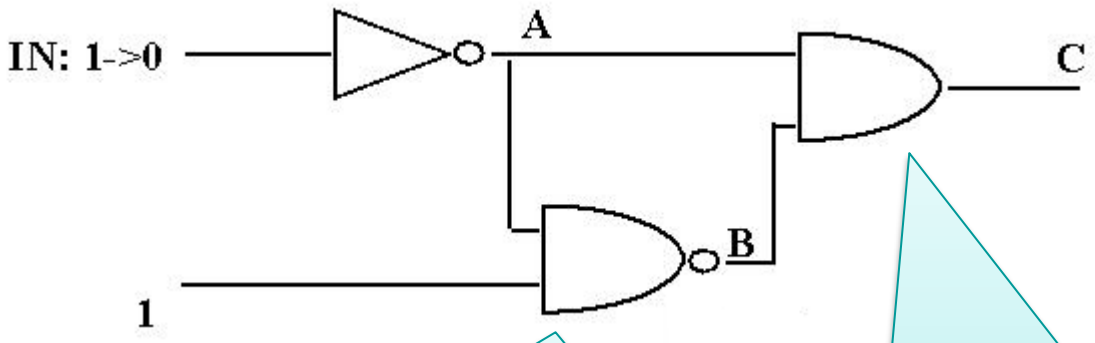
- **Initialization phase**
- **Actual simulation**

The simulator accepts simulation commands, which control the simulation of the digital system and which specify the desired simulator output

2.9 Compilation, simulation, and synthesis of VHDL code

Understanding the role of **Δ delays** is important when interpreting output from a VHDL simulator

An example without Delta delay



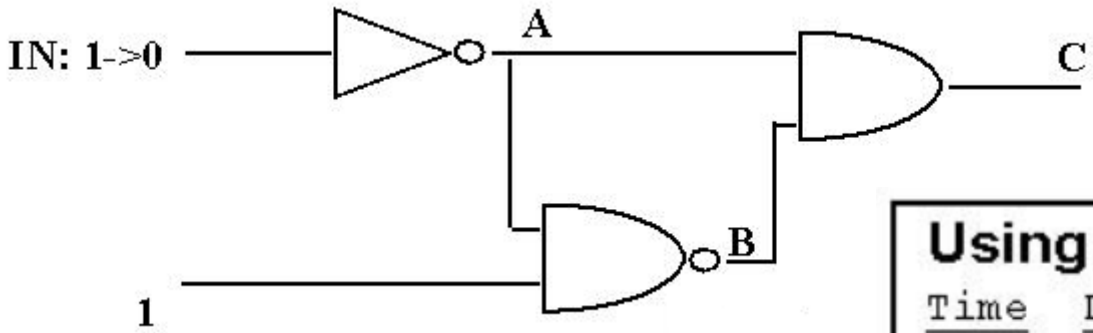
What is the behavior of C?

If NAND gate evaluated first:
IN: 1->0
A : 0->1
B : 1->0
C : 0->0

If AND gate evaluated first:
IN: 1->0
A : 0->1
C : 0->1
B : 1->0
C : 1->0

2.9 Compilation, simulation, and synthesis of VHDL code

An example with Delta delay

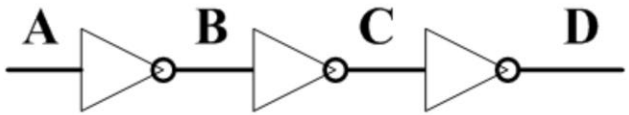


Using delta delay scheduling

Time	Delta	Event
0 ns	1	IN: 1->0 eval INVERTER
	2	A: 0->1 eval NAND, AND
	3	B: 1->0 C: 0->1 eval AND
	4	C: 1->0
1 ns		

Δ delays are used to make sure that signals are proceed in the proper sequence

2.9 Compilation, simulation, and synthesis of VHDL code

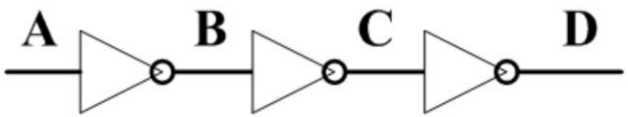



```
1   B <= not A;
2   C <= not B;
3   D <= not C after 5ns;
```

```
1  B <= not A (after Δ);
2  C <= not B (after Δ);
3  D <= not C after 5 ns;
```

Although Δ delay do not show up on waveform outputs from the simulator, they show up on listing outputs

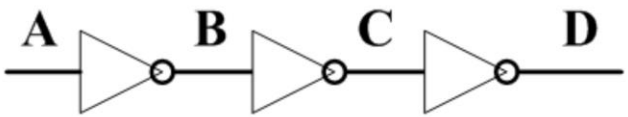
2.9 Compilation, simulation, and synthesis of VHDL code




```
1   B <= not A;
2   C <= not B;
3   D <= not C after 5ns;
```

ns	delta	A	B	C	D
0	+0	0	1	0	1
3	+0	1	1	0	1

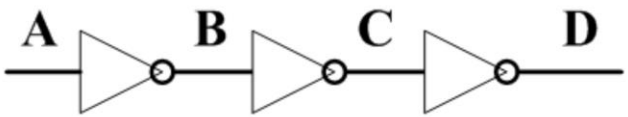
2.9 Compilation, simulation, and synthesis of VHDL code




```
1   B <= not A;
2   C <= not B;
3   D <= not C after 5ns;
```

ns	delta	A	B	C	D
0	+0	0	1	0	1
3	+0	1	1	0	1
3	+1	1	0	0	1

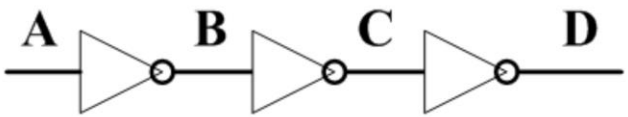
2.9 Compilation, simulation, and synthesis of VHDL code




```
1   B <= not A;
2   C <= not B;
3   D <= not C after 5ns;
```

ns	delta	A	B	C	D
0	+0	0	1	0	1
3	+0	1	1	0	1
3	+1	1	0	0	1
3	+2	1	0	1	1

2.9 Compilation, simulation, and synthesis of VHDL code




```
1   B <= not A;
2   C <= not B;
3   D <= not C after 5ns;
```

ns	delta	A	B	C	D
0	+0	0	1	0	1
3	+0	1	1	0	1
3	+1	1	0	0	1
3	+2	1	0	1	1
8	+0	1	0	1	0

When time advances a finite amount (as opposed to delta, which is infinitesimal), the delta counter is reset, i.e., 3 + 2 delta + 5 = 8

2.9.1 Simulation with Multiple Processes

```
entity simulation_example is
end simulation_example;

architecture test1 of simulation_example is
signal A, B: bit;
begin
  P1: process(B)
  begin
    A <= '1';
    A <= transport '0' after 5 ns;
  end process P1;

  P2: process(A)
  begin
    if A = '1' then B <= not B after 10 ns;
    end if;
  end process P2;
end test1;
```

- If a model contains more than one process, all processes execute concurrently with other processes
- If there are concurrent statements outside processes, they also execute concurrently

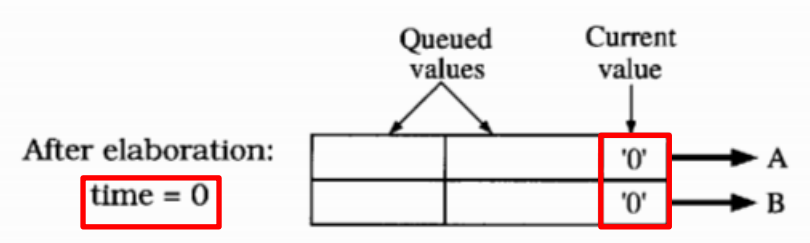
2.9.1 Simulation with Multiple Processes

```
entity simulation_example is
end simulation_example;

architecture test1 of simulation_example is
signal A, B: bit;
begin
  P1: process(B)
  begin
    A <= '1';
    A <= transport '0' after 5 ns;
  end process P1;

  P2: process(A)
  begin
    if A = '1' then B <= not B after 10 ns;
    end if;
  end process P2;
end test1;
```

- Statements inside of each process execute sequentially
- A process **takes no time to execute** unless it has wait statements in it
- Signals take **Δ time to update** when no delay is specified

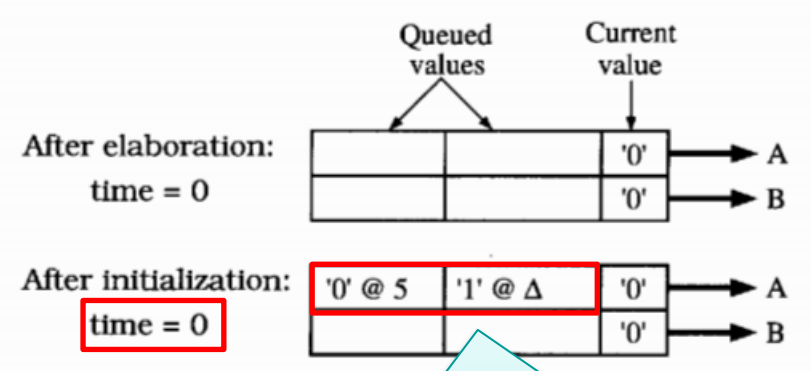
2.9.1 Simulation with Multiple Processes

```
entity simulation_example is
end simulation_example;

architecture test1 of simulation_example is
signal A, B: bit;
begin
  P1: process(B)
  begin
    A <= '1';
    A <= transport '0' after 5 ns;
  end process P1;

  P2: process(A)
  begin
    if A = '1' then B <= not B after 10 ns;
    end if;
  end process P2;
end test1;
```



2.9.1 Simulation with Multiple Processes

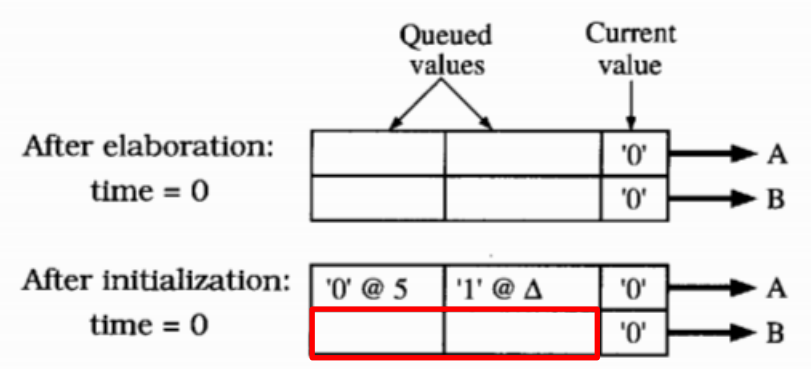
```
entity simulation_example is
end simulation_example;

architecture test1 of simulation_example is
signal A, B: bit;
begin
  P1: process(B)
  begin
    A <= '1';
    A <= transport '0' after 5 ns;
  end process P1;

  P2: process(A)
  begin
    if A = '1' then B <= not B after 10 ns;
    end if;
  end process P2;
end test1;
```



After a VHDL simulator is initialized, it executes each process with a sensitivity list one time through

2.9.1 Simulation with Multiple Processes

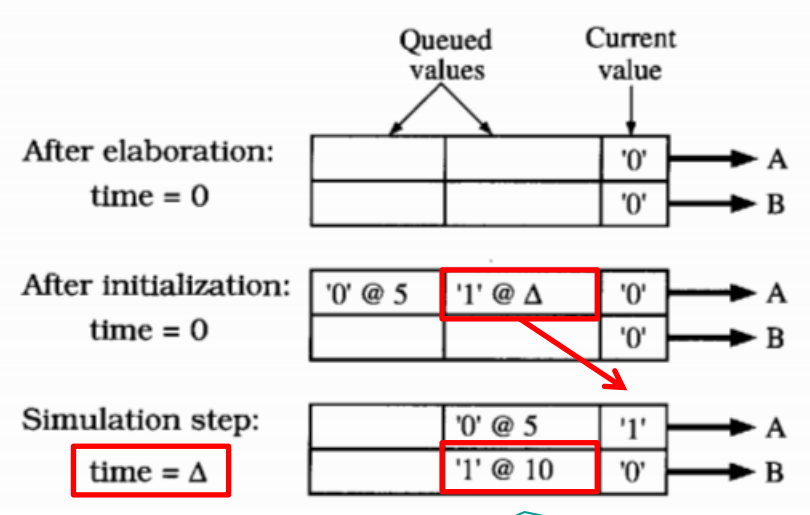
```
entity simulation_example is
end simulation_example;

architecture test1 of simulation_example is
signal A, B: bit;
begin
  P1: process(B)
  begin
    A <= '1';
    A <= transport '0' after 5 ns;
  end process P1;

  P2: process(A)
  begin
    if A = '1' then B <= not B after 10 ns;
    end if;
  end process P2;
end test1;
```



At time = 0, no change in B, since A still '0' during execution at time 0 ns

2.9.1 Simulation with Multiple Processes

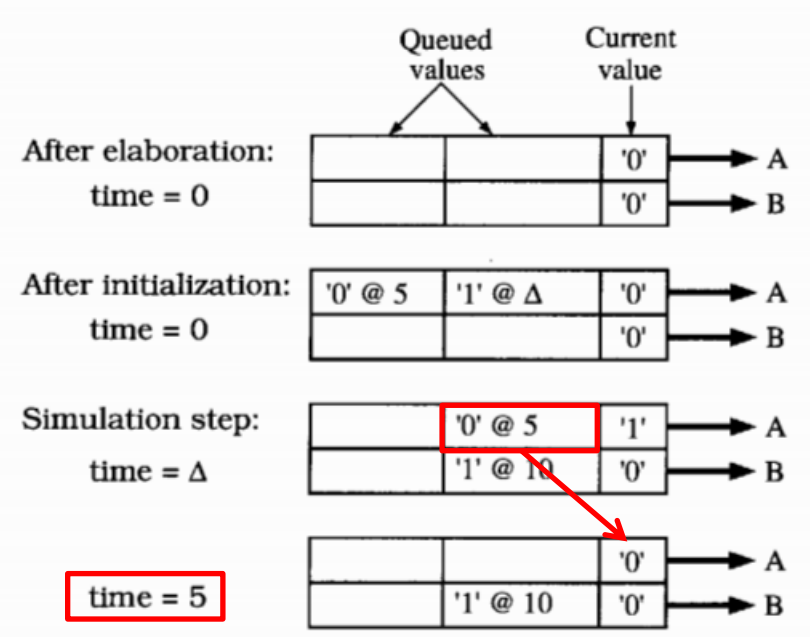
```
entity simulation_example is
end simulation_example;

architecture test1 of simulation_example is
signal A, B: bit;
begin
  P1: process(B)
  begin
    A <= '1';
    A <= transport '0' after 5 ns;
  end process P1;

  P2: process(A)
  begin
    if A = '1' then B <= not B after 10 ns;
    end if;
  end process P2;
end test1;
```



A = '1', B is scheduled to change to '1' at 10 ns

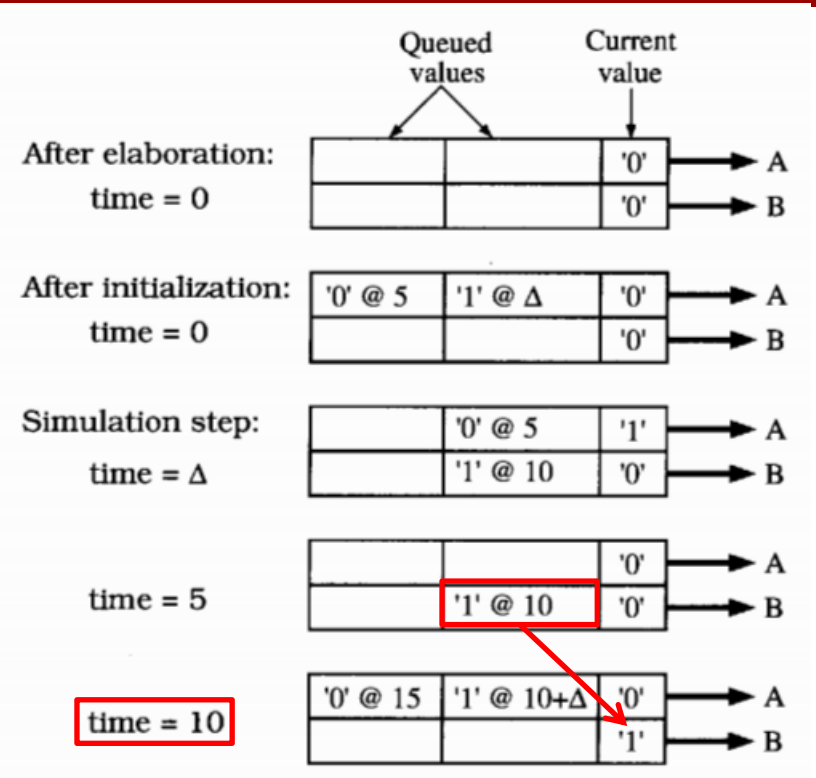
2.9.1 Simulation with Multiple Processes

```
entity simulation_example is
end simulation_example;

architecture test1 of simulation_example is
signal A, B: bit;
begin
  P1: process(B)
  begin
    A <= '1';
    A <= transport '0' after 5 ns;
  end process P1;

  P2: process(A)
  begin
    if A = '1' then B <= not B after 10 ns;
    end if;
  end process P2;
end test1;
```



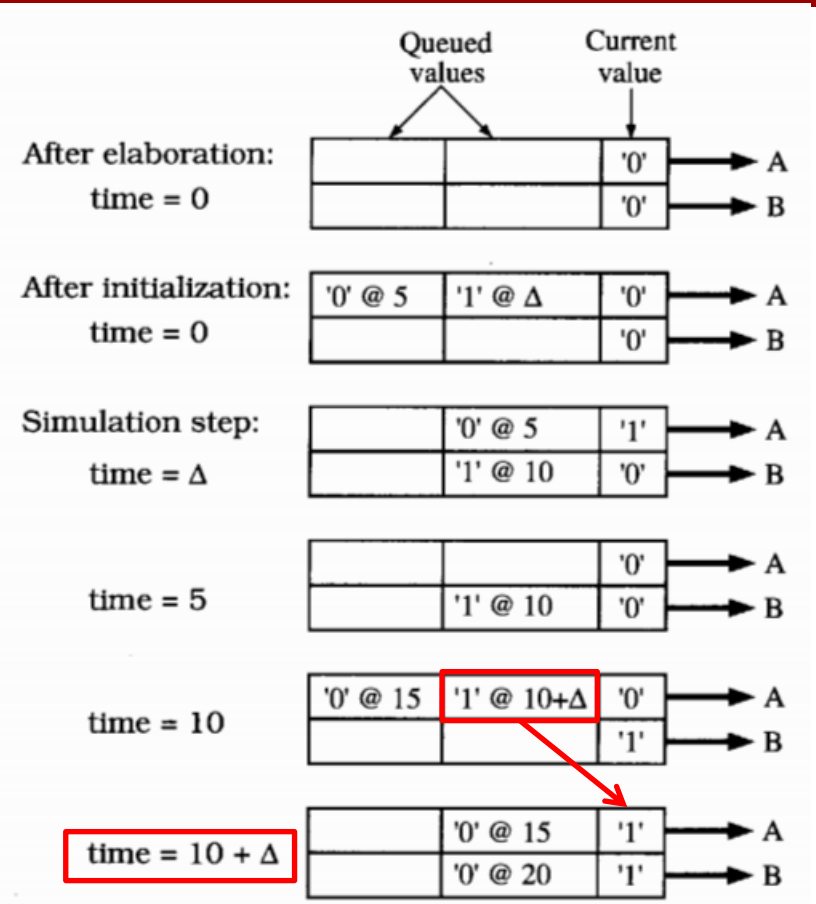
2.9.1 Simulation with Multiple Processes

```
entity simulation_example is
end simulation_example;

architecture test1 of simulation_example is
signal A, B: bit;
begin
  P1: process(B)
  begin
    A <= '1';
    A <= transport '0' after 5 ns;
  end process P1;

  P2: process(A)
  begin
    if A = '1' then B <= not B after 10 ns;
    end if;
  end process P2;
end test1;
```



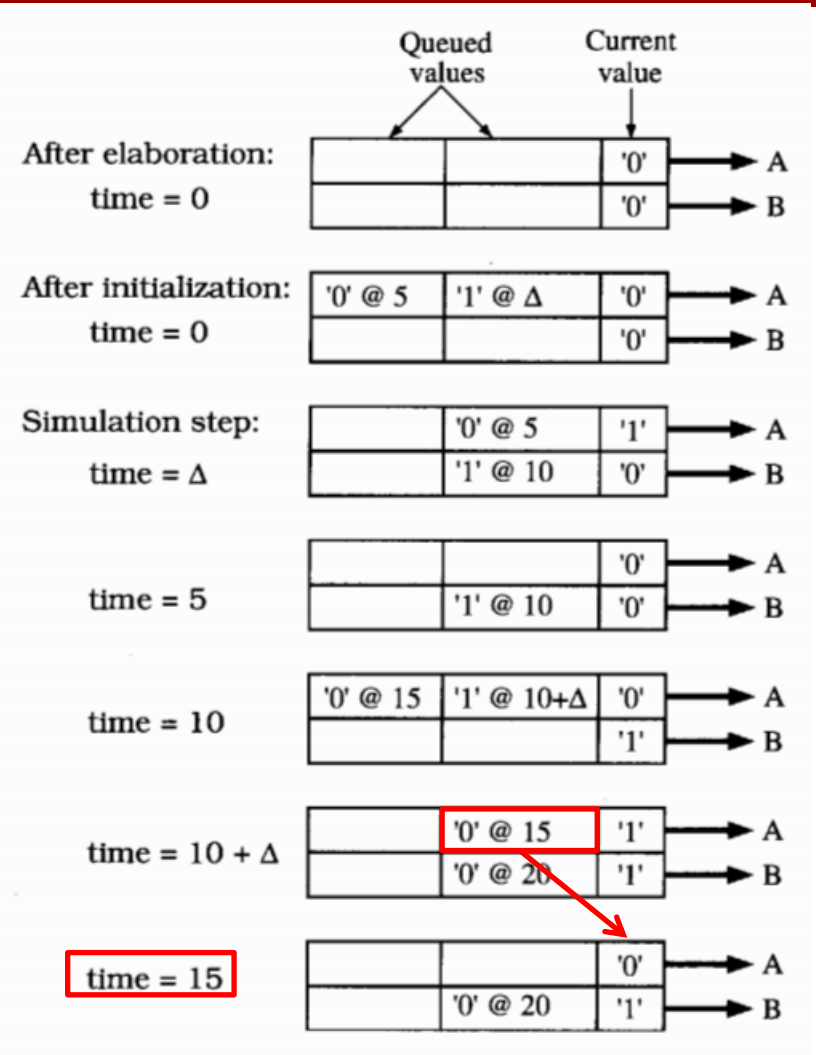
2.9.1 Simulation with Multiple Processes

```
entity simulation_example is
end simulation_example;

architecture test1 of simulation_example is
signal A, B: bit;
begin
  P1: process(B)
  begin
    A <= '1';
    A <= transport '0' after 5 ns;
  end process P1;

  P2: process(A)
  begin
    if A = '1' then B <= not B after 10 ns;
    end if;
  end process P2;
end test1;
```



2.9.1 Simulation with Multiple Processes

```
entity simulation_example is
end simulation_example;

architecture test1 of simulation_example is
signal A, B: bit;
begin
  P1: process(B)
  begin
    A <= '1';
    A <= transport '0' after 5 ns;
  end process P1;

  P2: process(A)
  begin
    if A = '1' then B <= not B after 10 ns;
    end if;
  end process P2;
end test1;
```

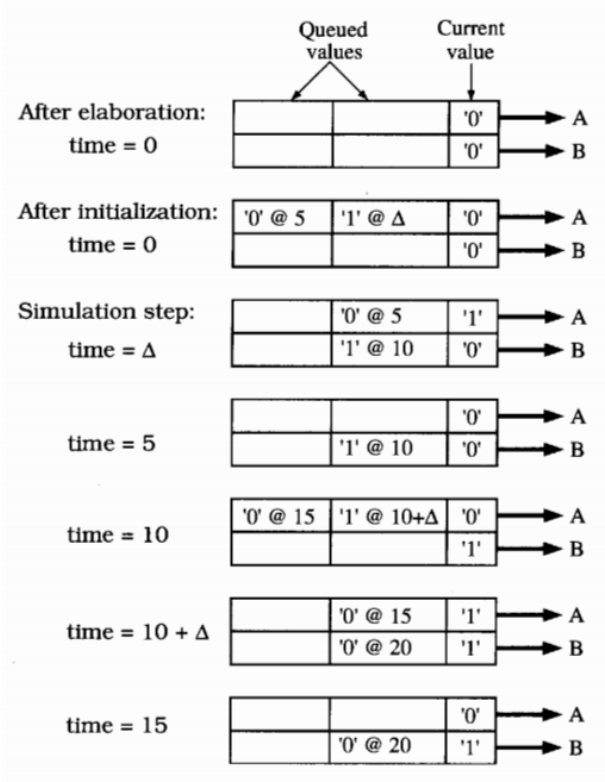


2.9.1 Simulation with Multiple Processes

```
entity simulation_example is
end simulation_example;

architecture test1 of simulation_example is
signal A, B: bit;
begin
  P1: process(B)
  begin
    A <= '1';
    A <= transport '0' after 5 ns;
  end process P1;

  P2: process(A)
  begin
    if A = '1' then B <= not B after 10 ns;
    end if;
  end process P2;
end test1;
```

ps	delta	/simulation_example/a	/simulation_example/b
0	+0	0	0
0	+1	1	0
5000	+0	0	0
10000	+0	0	1
10000	+1	1	1
15000	+0	0	1
20000	+0	0	0
20000	+1	1	0
25000	+0	0	0
30000	+0	0	1
30000	+1	1	1
35000	+0	0	1
40000	+0	0	0
40000	+1	1	0
45000	+0	0	0
50000	+0	0	1
50000	+1	1	1

2.9.1 Simulation with Multiple Processes

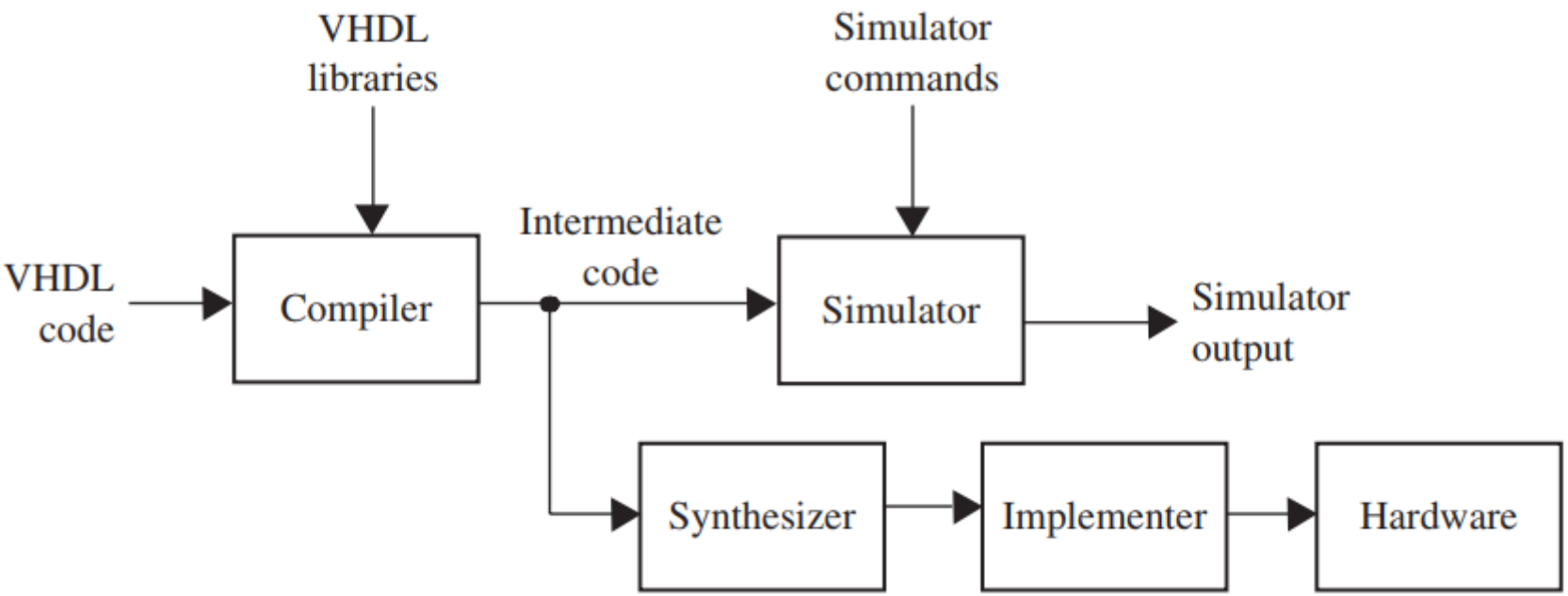


VHDL simulators use event-driven simulation

- A change in a signal is referred to as an event
- Each time an event occurs, any processes that have been waiting on the event are executed in zero time, and any resulting signal changes are queued up to occur at some future time

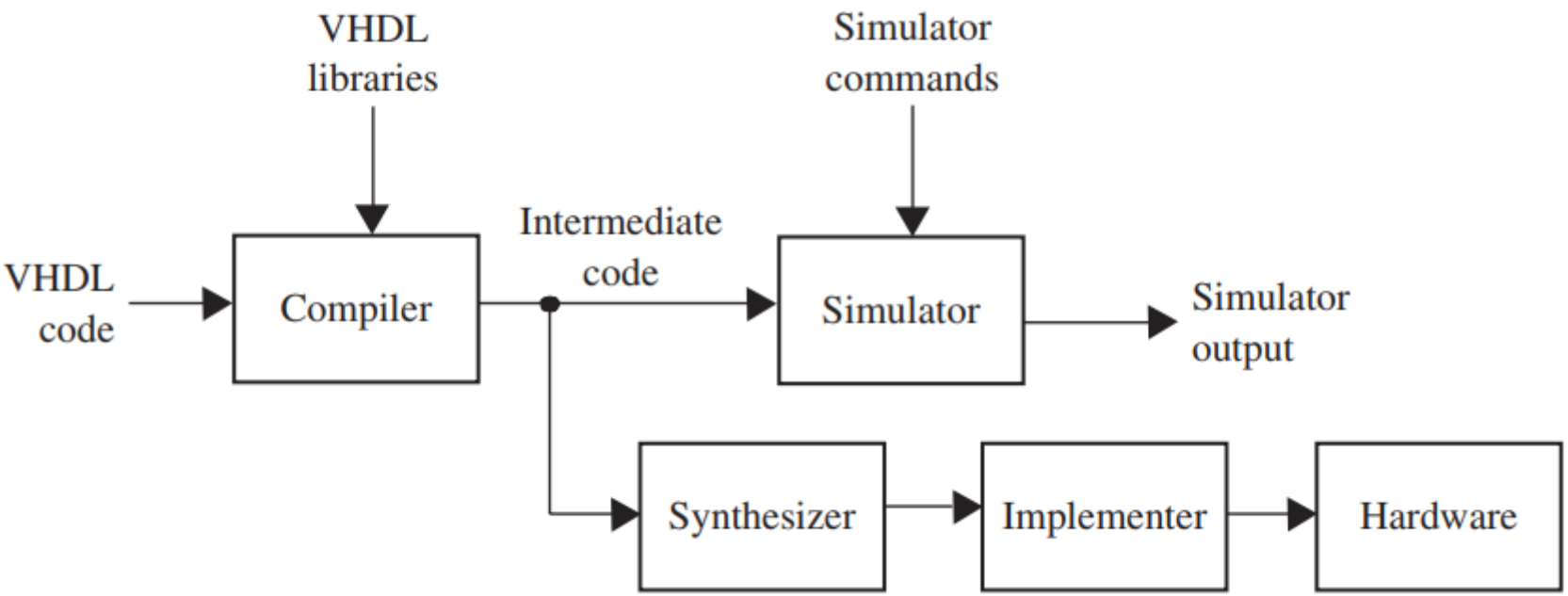
- When all the active processes are finished executing, simulation time is advanced to the time for which the next event is scheduled, and the simulator processes that event
- This continuous until either no more events have been scheduled or the simulation time limit is reached

2.9 Compilation, Simulation, and Synthesis of VHDL Code



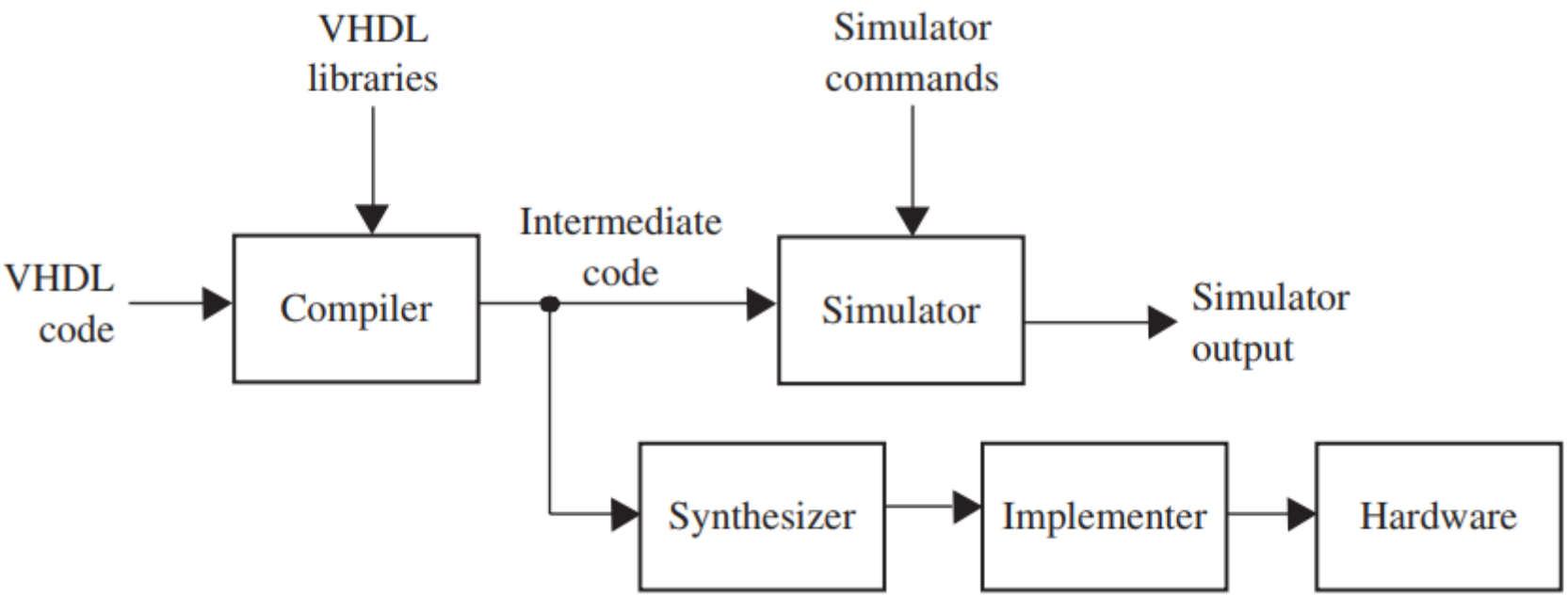
Nowadays, one of the most important uses of VHDL is to **synthesize** or automatically create hardware from a VHDL description

2.9 Compilation, Simulation, and Synthesis of VHDL Code



The **synthesis** software for VHDL translates the VHDL code to a circuit description that specifies the needed components and the connections between the components

2.9 Compilation, Simulation, and Synthesis of VHDL Code



The synthesizer output can then be used to implement the digital system using specific hardware

CPLD, **FPGA**, ASIC, etc.

Chapter 2 Introductin to VHDL

1	Computer-aided design	11	Simple synthesis examples
2	Hardware description language	12	VHDL models for multiplexers
3	VHDL description of combinational circuits	13	VHDL libraries
4	VHDL modules	14	Modeling registers and counters using VHDL processes
5	Sequential statements and VHDL processes	15	Behavioral and structural VHDL
6	Modeling flip-flop using VHDL processes	16	Variables, signals, and constants
7	Processes using wait statemetns	17	Arrays
8	Two types of VHDL delays: Transport and inertial delays	18	Loops in VHDL
9	Complication,simulation, and synthesis of VHDL code	19	Assert and report statements
10	**VHDL data types and operators**		

2.10.1 Data Types

In order to write VHDL code efficiently, it is essential to know what date types are allowed, and how to specify and use them

Data type	Value	Example
bit	'0' or '1'	
boolean	FALSE or TRUE	
integer	$-(2^{31}-1)$ to $+(2^{31}-1)$	128
real	-1.0E38 to +1.0E38	1E3
character	upper- and lowercase letters, digits, and special characters	'd', '7', '+'
time	an integer with units fs, ps, ns, ms, sec, min, or hr	10 ns

Real and time types are not synthesizable

2.10.1 Data Types

Enumeration(枚举)

user-defined date type

Example

default value

```
type state_type is (S0, S1, S2, S3, S4, S5);
signal state : state_type := S1;
```

VHDL: Strongly typed language

Example

```
A <= B or C
```

A, B, and C have the same type

If types do not match, explicit type conversion should be performed, or "overloaded operators" should be created

2.10.2 VHDL Operators (运算符)

	Operator class	Operators	Precedence
1	Binary logical	and or nand nor xor xnor	Lowest
2	Relational	= /= < <= > >=	
3	Shift	sll srl sla sra rol ror	
4	Adding	+ - &	
5	Unary sign	+ -	
6	Multiplying	* / mod rem	
7	Miscellaneous	not abs **	Highest

Example

If A = "110", B = "111", C = "011000", D = "111011"

```
(A & not B or C ror 2 and D) = "110010"
```

= "000"

	Operator class	Operators	Precedence
1	Binary logical	and or nand nor xor xnor	Lowest
2	Relational	= /= < <= > >=	
3	Shift	sll srl sla sra rol ror	
4	Adding	+ - &	
5	Unary sign	+ -	
6	Multiplying	* / mod rem	
7	Miscellaneous	not abs **	Highest

Example

If A = "110", B = "111", C = "011000", D = "111011"

```
(A & "000" or C ror 2 and D) = "110010"
```

& : concatenate two vectors (or an element and a vector, or two elements) to form a longer vector

	Operator class	Operators	Precedence
1	Binary logical	and or nand nor xor xnor	Lowest
2	Relational	= /= < <= > >=	
3	Shift	sll srl sla sra rol ror	
4	Adding	+ - &	
5	Unary sign	+ -	
6	Multiplying	* / mod rem	
7	Miscellaneous	not abs **	Highest

Example

If A = "110", B = "111", C = "011000", D = "111011"

```
("110000" or C ror 2 and D) = "110010"
```

= "000110"

	Operator class	Operators	Precedence
1	Binary logical	and or nand nor xor xnor	Lowest
2	Relational	= /= < <= > >=	
3	Shift	sll srl sla sra rol ror	
4	Adding	+ - &	
5	Unary sign	+ -	
6	Multiplying	* / mod rem	
7	Miscellaneous	not abs **	Highest

Example

If A = "110", B = "111", C = "011000", D = "111011"

```
("110000" or "000110" and D) = "110010"
```

= "110110"

	Operator class	Operators	Precedence
1	Binary logical	and or nand nor xor xnor	Lowest
2	Relational	= /= < <= > >=	
3	Shift	sll srl sla sra rol ror	
4	Adding	+ - &	
5	Unary sign	+ -	
6	Multiplying	* / mod rem	
7	Miscellaneous	not abs **	Highest

Example

If A = "110", B = "111", C = "011000", D = "111011"

```
("110110" and D) = "110010"
```

= "110010"

	Operator class	Operators	Precedence
1	Binary logical	and or nand nor xor xnor	Lowest
2	Relational	= /= < <= > >=	
3	Shift	sll srl sla sra rol ror	
4	Adding	+ - &	
5	Unary sign	+ -	
6	Multiplying	* / mod rem	
7	Miscellaneous	not abs **	Highest

Example

If A = "110", B = "111", C = "011000", D = "111011"

```
"110010" = "110010"
```

The result of applying a relational operator is a Boolean (FALSE or TRUE)

	Operator class	Operators	Precedence
1	Binary logical	and or nand nor xor xnor	Lowest
2	Relational	= /= < <= > >=	
3	Shift	sll srl sla sra rol ror	
4	Adding	+ - &	
5	Unary sign	+ -	
6	Multiplying	* / mod rem	
7	Miscellaneous	not abs **	Highest

Example

If A = "110", B = "111", C = "011000", D = "111011"

TRUE

	Operator class	Operators	Precedence
1	Binary logical	and or nand nor xor xnor	Lowest
2	Relational	= /= < <= > >=	
3	Shift	sll srl sla sra rol ror	
4	Adding	+ - &	
5	Unary sign	+ -	
6	Multiplying	* / mod rem	
7	Miscellaneous	not abs **	Highest

Shift operators

A = “10010101”		
A sll 2	"01010100"	shift left logical, filled with '0'
A srl 3	"00010010"	shift right logical, filled with '0'
A sla 3	"10101111"	shift left arithmetic, filled with right bit
A sra 2	"11100101"	shift right arithmetic, filled with left bit
A rol 3	"10101100"	rotate left
A ror 5	"10101100"	rotate right

Chapter 2 Introductin to VHDL

1	Computer-aided design	**11**	**Simple synthesis examples**
2	Hardware description language	12	VHDL models for multiplexers
3	VHDL description of combinational circuits	13	VHDL libraries
4	VHDL modules	14	Modeling registers and counters using VHDL processes
5	Sequential statements and VHDL processes	15	Behavioral and structural VHDL
6	Modeling flip-flop using VHDL processes	16	Variables, signals, and constants
7	Processes using wait statemetns	17	Arrays
8	Two types of VHDL delays: Transport and inertial delays	18	Loops in VHDL
9	Complication,simulation, and synthesis of VHDL code	19	Assert and report statements
10	VHDL data types and operatos		

2.11 Simple Synthesis Examples

- Synthesis tools try to infer the hardware components needed by "looking" at the VHDL code
- In order for code to synthesize correctly, certain conversions must be followed

- When writing VHDL code, you should always keep in mind that you are designing hardware, not simply writing a computer program
- Each VHDL statement implies certain hardware requirements. Poorly written VHDL code may result in poorly designed hardware
- Even if VHDL code gives the corrected result when simulated, it may not result in hardware that works correctly when synthesized
- Timing problems may prevent the hardware from working properly even though the simulation results are correct

VHDL Code Example where Simulation and Synthesis Results in Different Outputs

```
entity Q1 is
  port(A, B : in bit;
       C    : out bit);
end Q1;

architecture circuit of Q1 is
begin
  process(A)
  begin
    C <= A or B after 5 ns;
  end process;
end circuit;
```

B is missing from the process sensitivity

If B changes now, that will not cause the process to execute

VHDL Code Example where Simulation and Synthesis Results in Different Outputs

Quartus II – E:/Kuaipan/Teaching/FPGA/Code/Chp2/Chp2_11/Q1/Q1 – Q1

File Edit View Project Assignments Processing Tools Window Help

Project Navigator

Files
Q1.vhd

Hierarchy | Files | Design Units

Tasks

Flow: Compilation | Customize...

Task	Time
Compile Design	00:00:15
Analysis & Synthesis	00:00:06
Edit Settings	
View Report	
Analysis & Elaboration	
Partition Merge	
Netlist Viewers	
RTL Viewer	
State Machine Viewer	
Technology Map Viewer (Post-Mapping)	

Q1.vhd | Compilation Report

```
1  entity Q1 is
2     port ( A, B: in bit;
3            C: out bit);
4  end Q1;
5
6  architecture circuit of Q1 is
7  begin
8     process(A)
9     begin
10       C <= A or B after 5 ns;
11    end process;
12 end circuit;
```

The sythesizer will warn you that B is missing from the sensitivity list, but will go ahead and synthesize the code properly

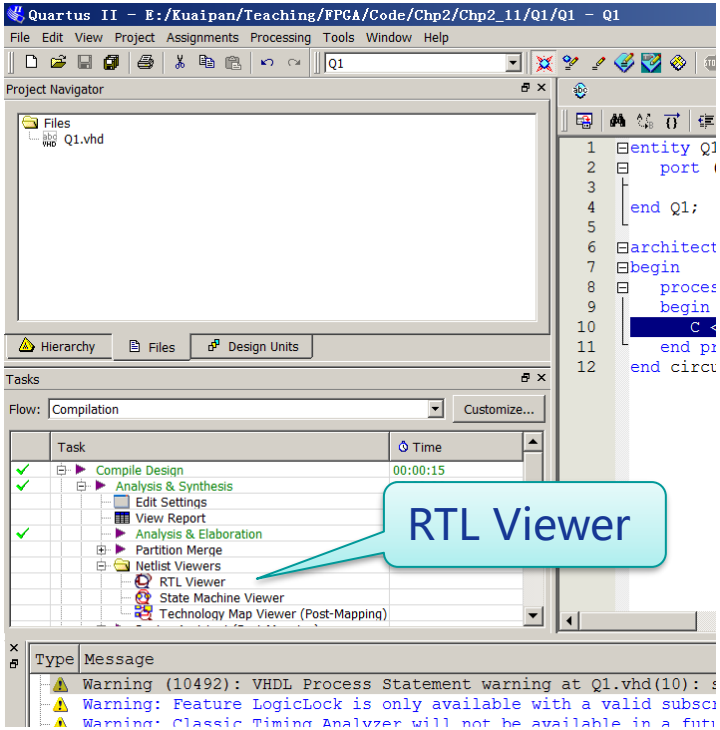
Type	Message
Warning	(10492): VHDL Process Statement warning at Q1.vhd(10): signal "B" is read inside the Process Statement but isn't in the Process Statement's sensitivity list
Warning	Feature LogicLock is only available with a valid subscription license. You can purchase a software subscription to gain full access to this feature.
Warning	Classic Timing Analyzer will not be available in a future release of the Quartus II software. Use the TimeQuest Timing Analyzer to run timing analysis on your design. Convert
Warning	Found 1 output pins without output pin load capacitance assignment
Warning	The Reserve All Unused Pins setting has not been specified, and will default to 'As output driving ground'.
Warning	Classic Timing Analyzer will not be available in a future release of the Quartus II software. Use the TimeQuest Timing Analyzer to run timing analysis on your design. Convert

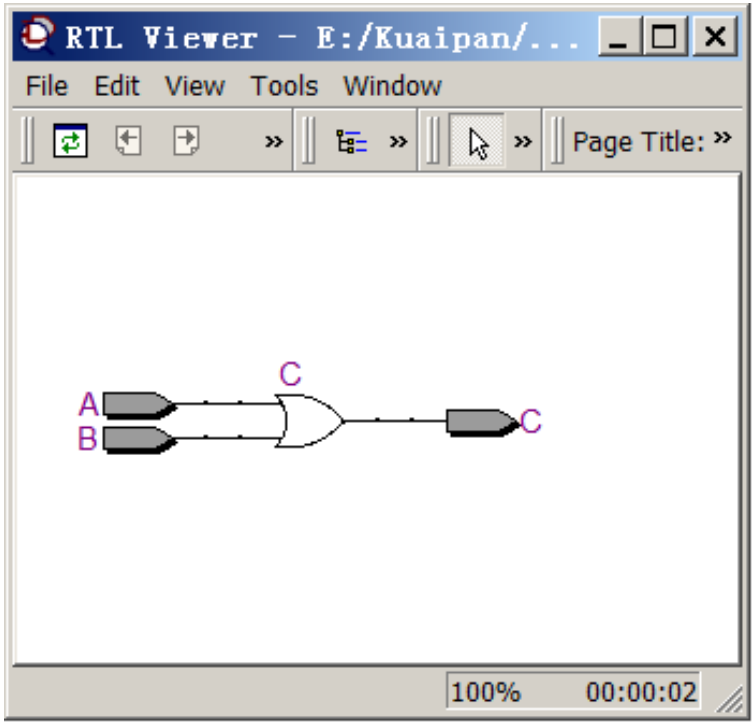
System | Processing (53) | Extra Info | Info (46) | Warning (6) | Critical Warning (1) | Error | Suppressed (6) | Flag

Message: 1 of 7 | Location in Design File | Locate

100% 00:00:15

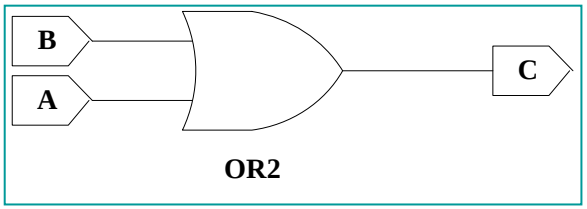
VHDL Code Example where Simulation and Synthesis Results in Different Outputs





VHDL Code Example where Simulation and Synthesis Results in Different Outputs

```
entity Q1 is
  port(A, B : in bit;
       C    : out bit);
end Q1;

architecture circuit of Q1 is
begin
  process(A)
  begin
    C <= A or B after 5 ns;
  end process;
end circuit;
```

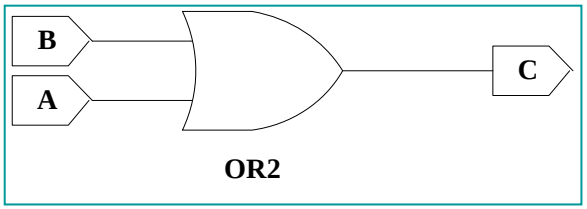


The delay will be ignored

If you want to model an exact 5-ns delay, you will have to use counters

VHDL Code Example where Simulation and Synthesis Results in Different Outputs

```
entity Q1 is
  port(A, B : in bit;
       C    : out bit);
end Q1;

architecture circuit of Q1 is
begin
  process(A)
  begin
    C <= A or B after 5 ns;
  end process;
end circuit;
```

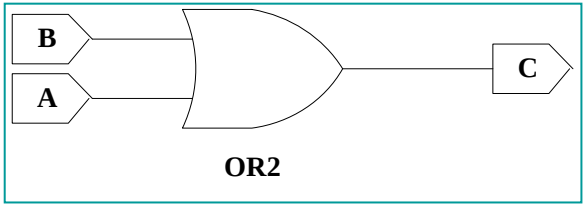


This is an example of where the synthesizer guessed a little more than what you wrote

It assumed that you probably meant an OR gate and created that circuit (accompanied by a warning)

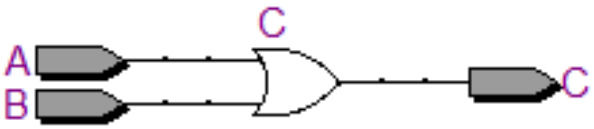
VHDL Code Example where Simulation and Synthesis Results in Different Outputs

```
entity Q1 is
  port(A, B : in bit;
       C    : out bit);
end Q1;

architecture circuit of Q1 is
begin
  process(A)
  begin
    C <= A or B after 5 ns;
  end process;
end circuit;
```



- This circuit functions differently from what simulated before synthesis
- It is important that you always check for synthesizer warnings of missing signals in the sensitivity list
- Perhaps the synthesizer helped you; perhaps it created hardware that you did not intend to

```
1   entity Q1 is
2      port ( A, B: in bit;
3             C: out bit);
4   end Q1;
5
6   architecture circuit of Q1 is
7   begin
8      process(A, B)
9      begin
10        C <= A or B after 5 ns;
11     end process;
12  end circuit;
```

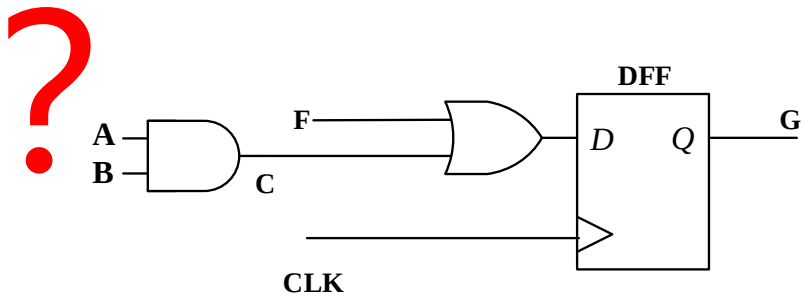


B is in the process sensitivity list now

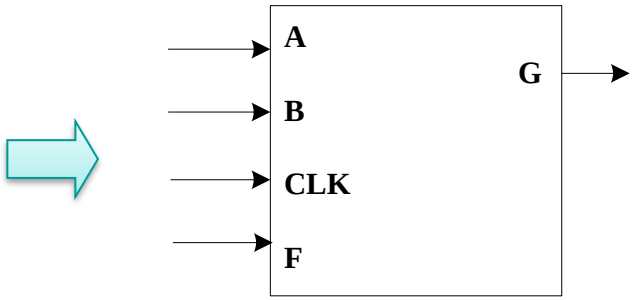
The synthesis result is the same

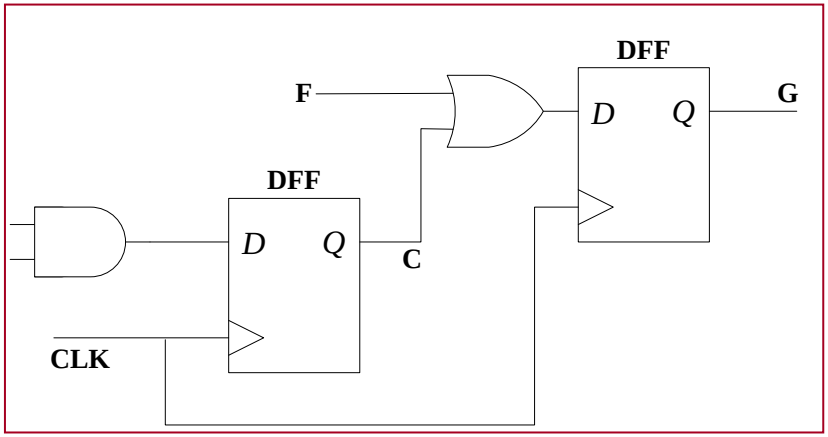
```
entity Q3 is
  port(A,B,F, CLK :  in    bit;
        G         :  out   bit);
end Q3;

architecture circuit of Q3 is
signal C: bit;
begin
  process(Clk)
  begin
    if (Clk = '1' and Clk'event) then
      C <= A and B;  -- statement 1
      G <= C or F;    -- statemnet 2
    end if;
  end process;
end circuit;
```

What hardware will you get if you synthesized this code?

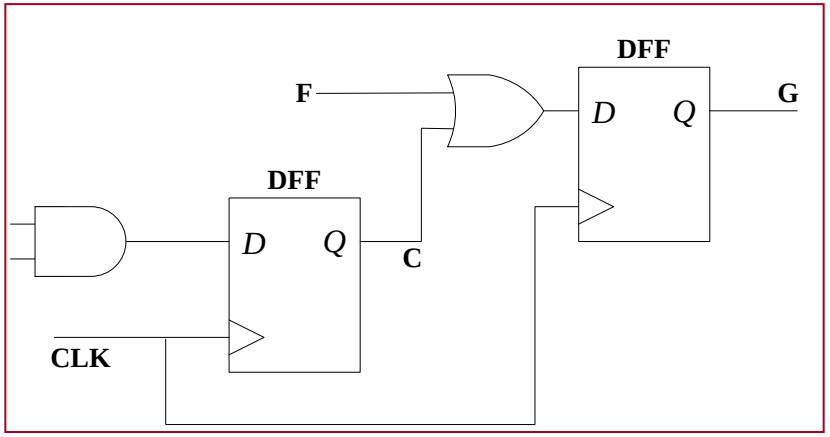


```
entity Q3 is
  port(A,B,F, CLK :  in    bit;
        G         :  out   bit);
end Q3;

architecture circuit of Q3 is
signal C: bit;
begin
  process(Clk)
  begin
    if (Clk = '1' and Clk'event) then
      C <= A and B;  -- statement 1
      G <= C or F;    -- statemnet 2
    end if;
  end process;
end circuit;
```





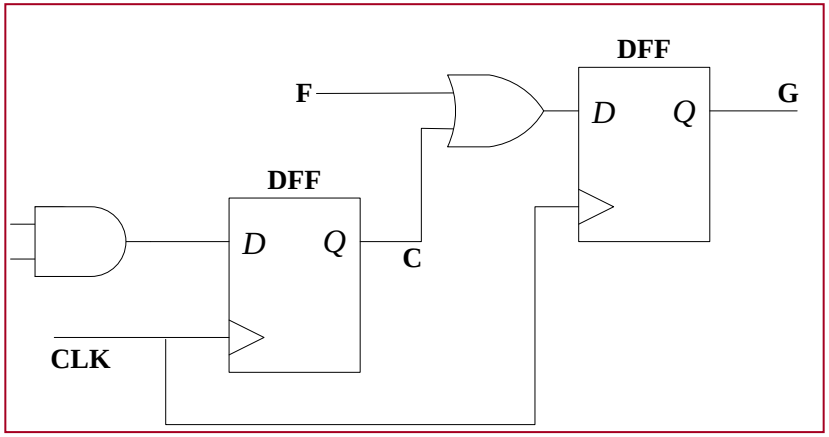
```
entity Q3 is
  port(A,B,F, CLK :  in    bit;
         G        :  out   bit);
end Q3;

architecture circuit of Q3 is
signal C: bit;
begin
  process(Clk)
  begin
    if (Clk = '1' and Clk'event) then
      C <= A and B;  -- statement 1
      G <= C or F;    -- statemnet 2
 end if;
  end process;
end circuit;
```



Ths circuit is not two cascaded gates

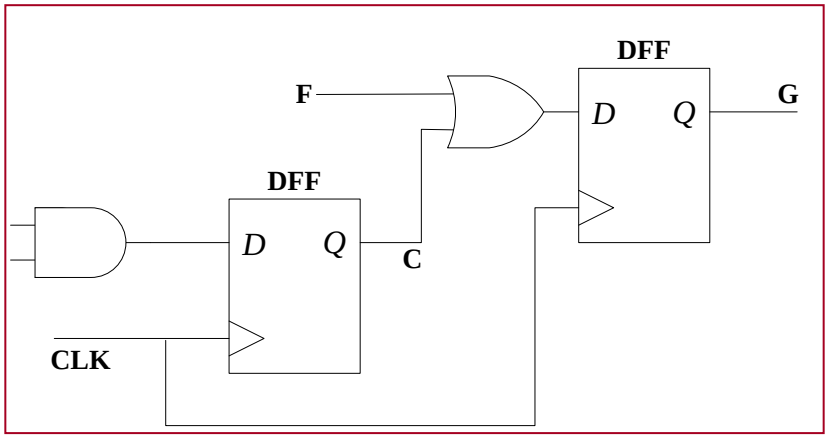
This is because the signal assignment statements are in a process

```
entity Q3 is
  port(A,B,F, CLK :  in    bit;
        G         :  out   bit);
end Q3;

architecture circuit of Q3 is
signal C: bit;
begin
  process(Clk)
  begin
    if (Clk = '1' and Clk'event) then
      C <= A and B;  -- statement 1
      G <= C or F;    -- statemnet 2
 end if;
  end process;
end circuit;
```

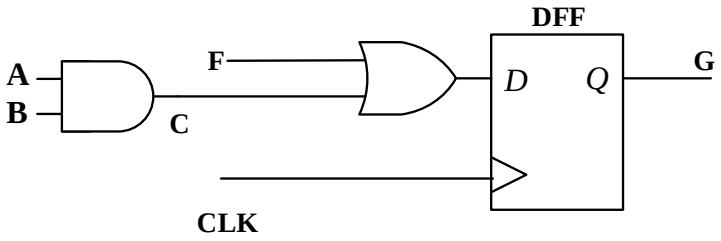


- An edge-triggered clock is implied by the use of clk'event in the clock statement preceding the signal assignment
- Since the values of C and G need to be retained after the clock edge, flip-flops are required for both C and G

```
entity Q3 is
  port(A,B,F, CLK :  in    bit;
        G         :  out   bit);
end Q3;

architecture circuit of Q3 is
signal C: bit;
begin
  process(Clk)
  begin
    if (Clk = '1' and Clk'event) then
      C <= A and B;  -- statement 1
      G <= C or F;    -- statemnet 2
 end if;
  end process;
end circuit;
```

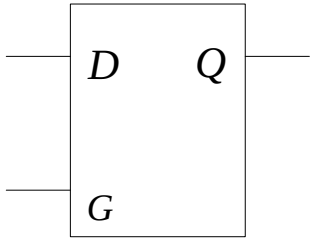


- Please note that a change in the value of C from statement 1 will not be considered during the execution of statement 2 in the pass of the process
- It will be considered only in the next pass, and the flip-flop for C makes this happen in the hardware also




```
entity Q3 is
  port(A,B,F, CLK :  in    bit;
        G         :  out   bit);
end Q3;

architecture circuit of Q3 is
signal C: bit;
begin
  C <= A and B;  -- statement 1
  process(Clk)
  begin
    if (Clk = '1' and Clk'event) then
      G <= C or F;    -- statemnet 2
    end if;
  end process;
end circuit;
```

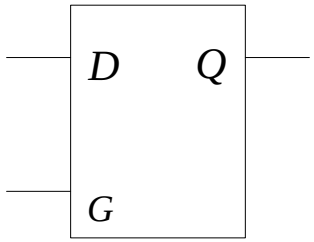
VHDL Code for a D-latch




```
process (G, D)
begin
   if G = '1' then Q <= D; end if;
end process;
```

Let us understand why this code does not represent an **AND gate** with G and D as inputs

VHDL Code for a D-latch



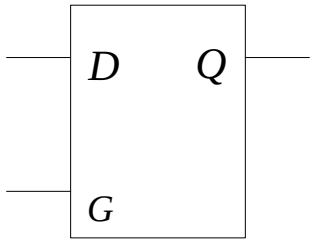

```
process (G, D)
begin
   if G = '1' then Q <= D; end if;
end process;
```

- If G = '1', an AND gate will result in the correct output to match the if statement

- What happens if currently Q = '1' and then G change to '0'?

VHDL Code for a D-latch

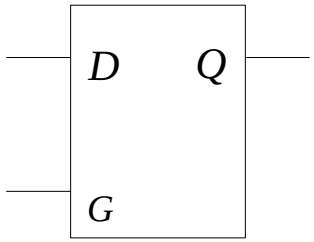
- What happens if currently Q = ‘1’ and then G change to ‘0’ ?




```
process (G, D)
begin
   if G = '1' then Q <= D; end if;
end process;
```

- When G changes to ‘0’ , and AND gate would propagate that to the output Q
- However, the device we have modeled here should not

VHDL Code for a D-latch

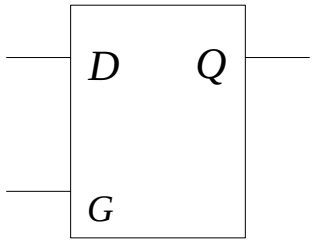



```
process (G, D)
begin
   if G = '1' then Q <= D; end if;
end process;
```

	AND gate		D-latch Code	
	D = 0	D = 1	D = 0	D = 1
G : 0 → 1	Q : 0 → 0	Q : 0 → 1	Q : 0 → 0	Q : 0 → 1
	Q : 1 → 0	Q : 1 → 1	Q : 1 → 0	Q : 1 → 1
G : 1 → 0				

Both behave the same

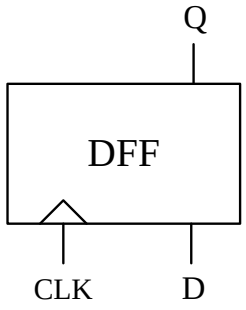
VHDL Code for a D-latch




```
process (G, D)
begin
   if G = '1' then Q <= D; end if;
end process;
```

	AND gate		D-latch Code	
	D = 0	D = 1	D = 0	D = 1
G : 0 → 1	Q : 0 → 0	Q : 0 → 1	Q : 0 → 0	Q : 0 → 1
	Q : 1 → 0	Q : 1 → 1	Q : 1 → 0	Q : 1 → 1
G : 1 → 0	Q : 0 → 0	Q : 0 → 0	Q : 0 → 0	Q : 0 → 0
	Q : 1 → 0	Q : 1 → 0	Q : 1 → 1	Q : 1 → 1

Different behaviors




```
process (CLK)
begin
   if  CLK'event and CLK = '1'
      then Q <= D;
   end if;
end process;
```

- In order to infer flip-flop or registers that change state on the rising edge of a clock signal, an **if**-clause of the form

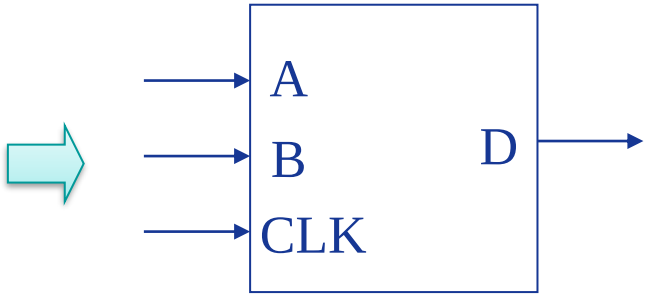
  if clock'event and clock = '1' then … end if;

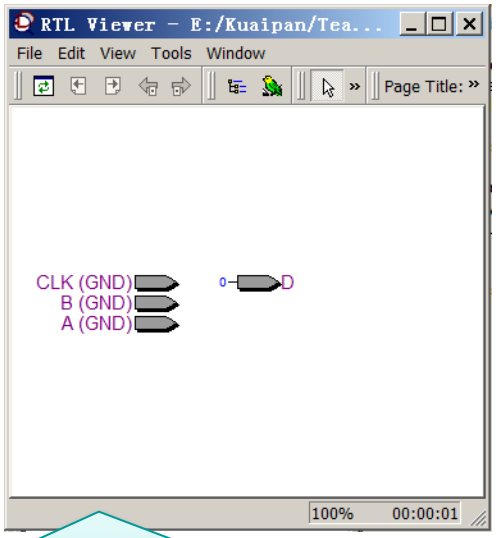
  is required by most synthesizer
- For every assignment statement between then and end if above, a signal on the left side of the assignment will cause creation of a register or flip-flop

- If you don't want to create unnecessary flip-flops, don't put the signal assignments in a clocked process
- If clock'event is omitted, the synthesizer may produce latches instead of flip-flops

Example VHDL Code That Will Not Synthesize

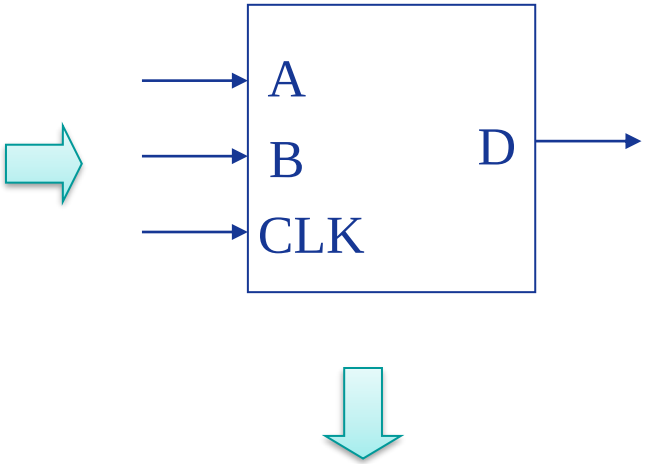
```
entity no_syn is
  port(A,B, CLK: in bit;
       D: out bit);
end no_syn;

architecture no_synthesis of no_syn is
signal C: bit;
begin
  process(Clk)
  begin
    if (Clk='1' and Clk'event) then
      C <= A and B;
    end if;
  end process;
end no_synthesis;
```





If you attempt to synthesize this code, the synthesizer will generate an empty block diagram

```
entity no_syn is
  port(A,B, CLK: in bit;
       D: out bit);
end no_syn;

architecture no_synthesis of no_syn is
signal C: bit;
begin
  process(Clk)
  begin
    if (Clk='1' and Clk'event) then
      C <= A and B;
    end if;
  end process;
end no_synthesis;
```

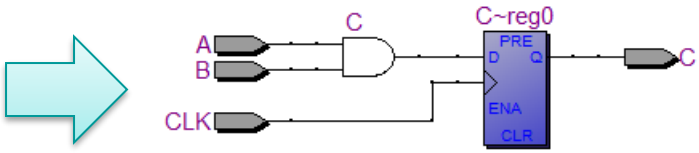


Warnings:

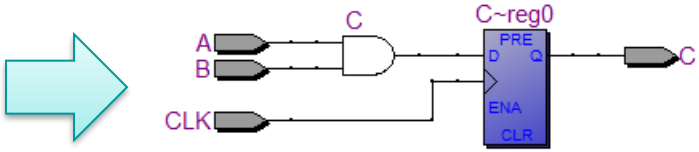
Input <CLK> is never used
Input <A> is never used
Input <B> is never used
Output <D> is never assigned

The output D is never assigned

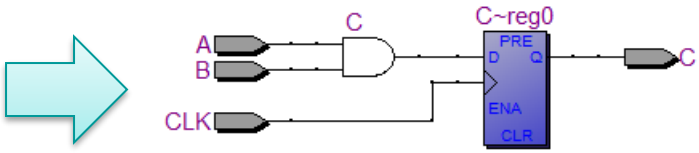
```
entity syn1 is
  port(A,B,CLK: in bit;
       C      : out bit);
end syn1;

architecture synthesis1 of syn1 is
begin
  process(Clk)
  begin
    if (Clk='1' and Clk'event) then
      C <= A and B;
    end if;
  end process;
end synthesis1;
```




```
entity syn2 is
  port(A,B,CLK: in bit;
       C      : out bit);
end syn2;

architecture synthesis2 of syn2 is
begin
  process(Clk)
  begin
    if (Clk'event and Clk='1') then
      C <= A and B;
    end if;
  end process;
end synthesis2;
```

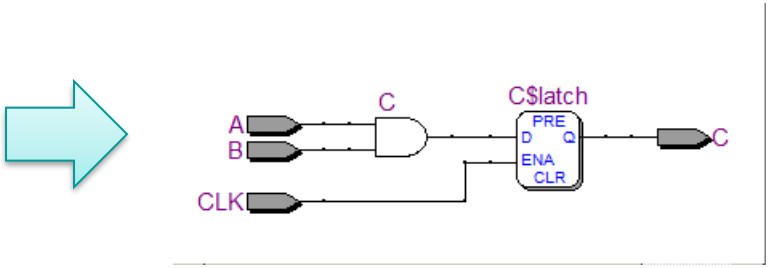


```
entity no_syn is
  port(A,B,CLK: in bit;
       C      : out bit);
end no_syn;

architecture no_synthesis of no_syn is
begin
  process(Clk)
  begin
    if Clk='1' then
      C <= A and B;
    end if;
  end process;
end no_synthesis;
```




```
entity no_syn is
  port(A,B,CLK: in bit;
       C      : out bit);
end no_syn;

architecture no_synthesis of no_syn is
begin
  process(Clk,A)
  begin
    if Clk='1' then
      C <= A and B;
    end if;
  end process;
end no_synthesis;
```

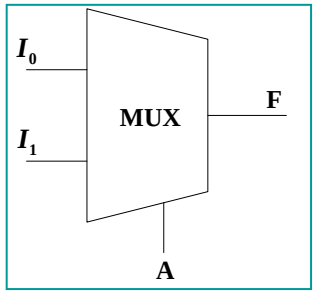
A is here



Chapter 2 Introductin to VHDL

1	Computer-aided design	11	Simple synthesis examples
2	Hardware description language	**12**	**VHDL models for multiplexers**
3	VHDL description of combinational circuits	13	VHDL libraries
4	VHDL modules	14	Modeling registers and counters using VHDL processes
5	Sequential statements and VHDL processes	15	Behavioral and structural VHDL
6	Modeling flip-flop using VHDL processes	16	Variables, signals, and constants
7	Processes using wait statemetns	17	Arrays
8	Two types of VHDL delays: Transport and inertial delays	18	Loops in VHDL
9	Complication,simulation, and synthesis of VHDL code	19	Assert and report statements
10	VHDL data types and operatos		

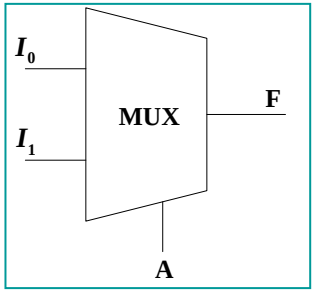
2.12 VHDL Models for Multiplexers (多路选择器)



Using **concurrent statements**	F <= (not A and I0) or (A and I1);
	Conditional signal assignment statement （条件信号赋值语句）(simple WHEN语句)
	Selective signal assignment statement （选择信号赋值语句）(selected WHEN语句)
Using **processes**	**Case statement**

2.12.1 Using Concurrent Statements

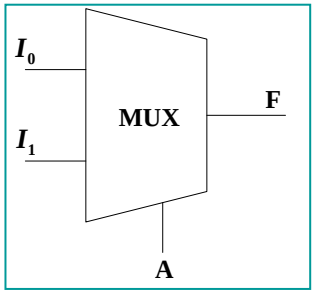
2-to-1 Multiplexer




```
F <= (not A and I0) or (A and I1);
```

2.12.1 Using Concurrent Statements

2-to-1 Multiplexer




```
F <= (not A and I0) or (A and I1);
```

```
F <= I0 when A ='0' else I1;
```

conditional signal assignment statement

This statement executes whenever A, I0, or I1 changes

I0, I1, and F can either be bits or bit-vectors

2.12.1 Using Concurrent Statements

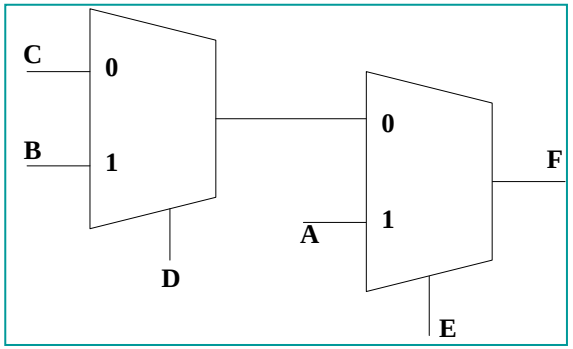
General form of conditional signal assignment statement

```
signal_name <= expression1 when condition1
               {else expression2 when condition2}
               [else expressionN];
```

This statement is executed whenever a change occurs in a signal used in one of the expressions or conditions

2.12.1 Using Concurrent Statements

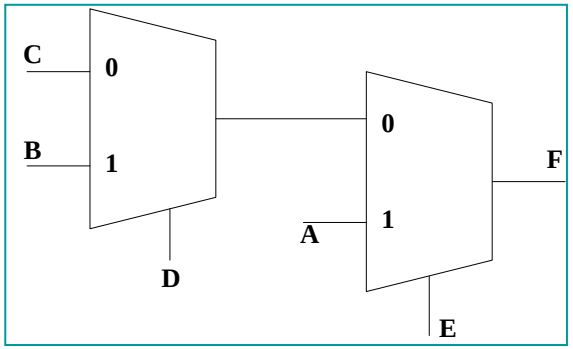
Cascaded MUXes




```
F <= __ when __
   else __ when ____
   else __;
```

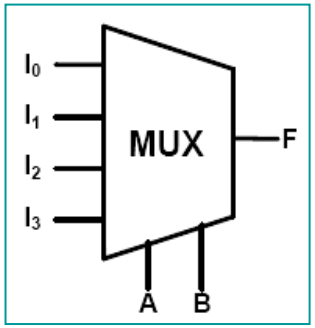
2.12.1 Using Concurrent Statements

Cascaded MUXes




```
F <= A when E = '1'
   else B when D = '1'
   else C;
```

4-to-1 MUXes



$$F = \overline{A}\,\overline{B}\,I_0 + \overline{A}\,BI_1 + A\,\overline{B}\,I_2 + ABI_3$$

```
F <= (not A and not B and I0) or (not A and B and I1) or
     (A and not B and I2) or (A and B and I3);
```

```
F <= I0  when  A & B ="00"
else  I1  when  A & B ="01"
else  I2  when  A & B ="10"
else  I3;
```

```
F <= I0  when  A ='0'  and  B='0'
else  I1  when  A ='0'  and  B='1'
else  I2  when  A ='1'  and  B='0'
else  I3;
```

```
sel <=A & B;

with  sel  select

   F <= I0 when "00",

         I1 when "01",

         I2 when "10",

         I3 when "11";
```

2.12.1 Using Concurrent Statements

General form of selected signal assignment statement

```
with expression_s select
  signal_s <= expression1 [after delay-time] when choice1,
              expression2 [after delay-time] when choice2,
               ...
             [expression_n [after delay-time] when others];
```

executes whenever a signal changes in any of the expressions

2.12.1 Using Concurrent Statements

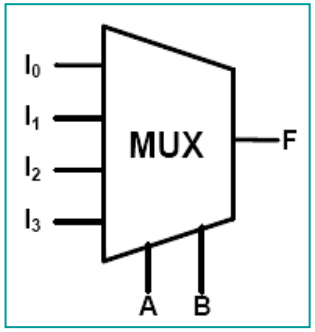
General form of selected signal assignment statement

```
with expression_s select
  signal_s <= expression1 [after delay-time] when choice1,
              expression2 [after delay-time] when choice2,
               ...
             [expression_n [after delay-time] when others];
```

- If all possible choices for the value of expression_s are given, the last line (when others) should be omitted
- Otherwise, the last line is required

2.12.2 Using Processes

4-to-1 MUXes



If a MUX model is used **inside a process**, a coucurrent statement cannot be used

```
process (Sel)
begin
  case Sel is
    when 0 =>  F <= I0;
    when 1 =>  F <= I1;
    when 2 =>  F <= I2;
    when 3 =>  F <= I3;
  end case;
end process;
```

General form of case statement

```
case expression is
   when choice1 => sequential statement1
   when choice2 => sequential statement2
   . . .
   [when others => sequential statements]
end case;
```

If all values are not explicitly given, a when others clause is required in the case statement

```
signal_name <= expression1 when condition1
               else expression2 when condition2
              [else expressionN];
```

```
with expression_s select
  signal_s <= expression1 when choice1,
              expression2 when choice2,
               ...
              [expression_n when others];
```

```
case expression is
   when choice1 => sequential statement1
   when choice2 => sequential statement2
   . . .
   [when others => sequential statements]
end case;
```