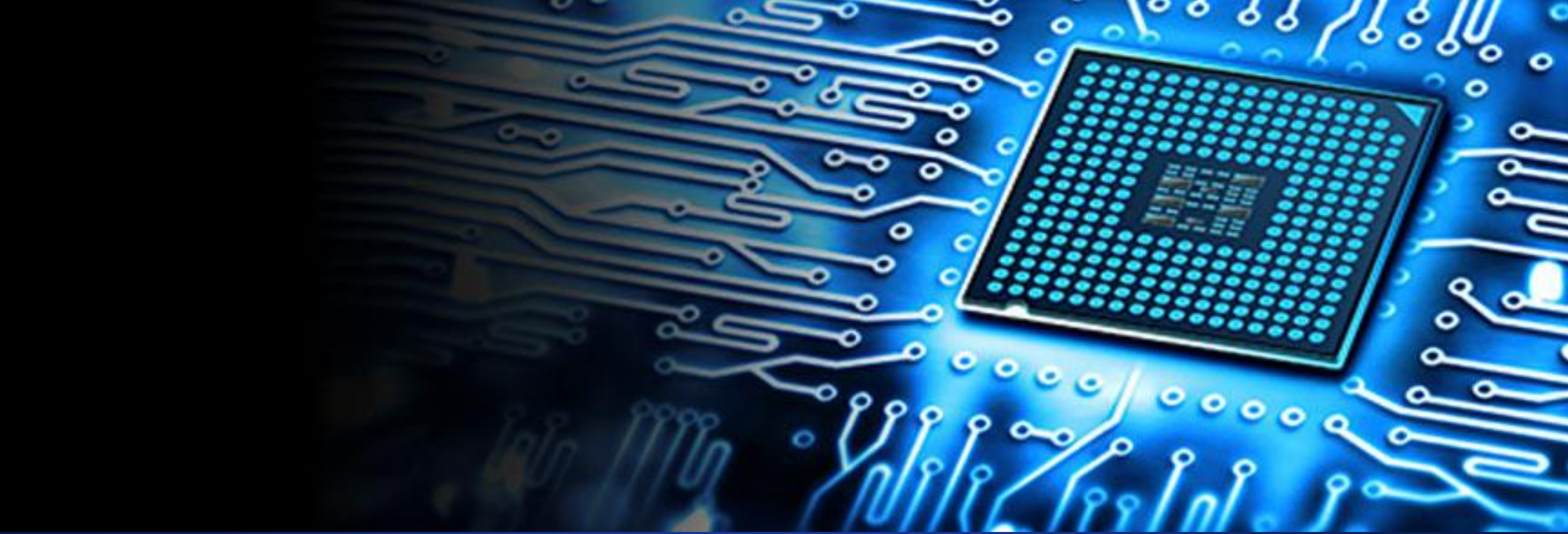


Chapter 2.4-2.7

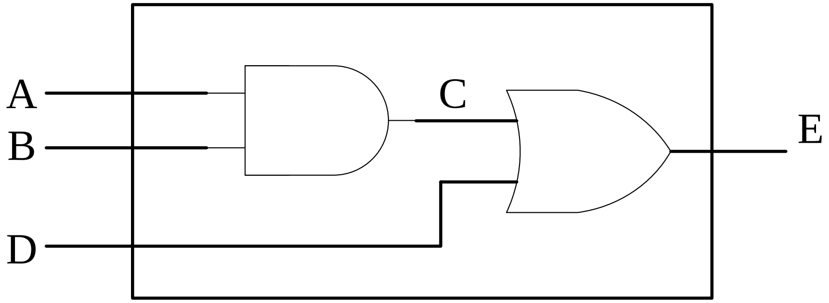
Introduction to VHDL

Version: 2023/11/21

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		

2.4 VHDL Modules（模块）



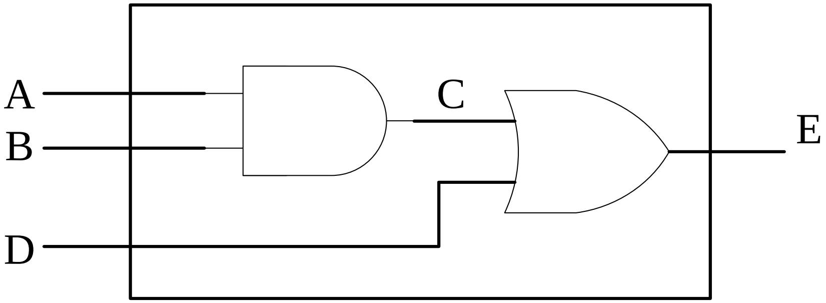
```
entity two_gates is
  port(A,B,D: in bit; E: out bit);
end two_gates;

architecture gates of two_gates is
signal C: bit;
begin
  C <= A and B ;  -- concurrent
  E <= C or D ;    -- statements
end gates;
```

Library declaration
(库声明)

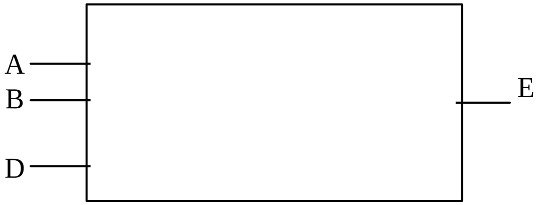
Entity
(实体)

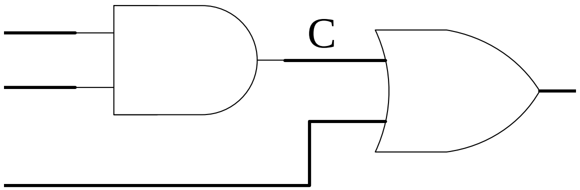
Architecture
(结构体)

2.4 VHDL Modules

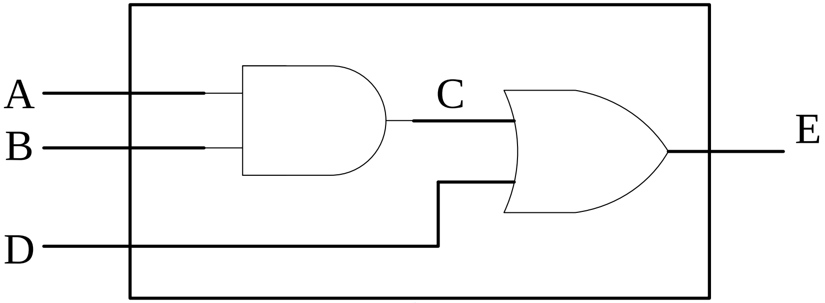


```
entity two_gates is
   port(A,B,D: in bit; E: out bit);
end two_gates;

architecture gates of two_gates is
   signal C: bit;
begin
   C <= A and B ;  -- concurrent
   E <= C or D ;     -- statements
end gates;
```





2.4 VHDL Modules

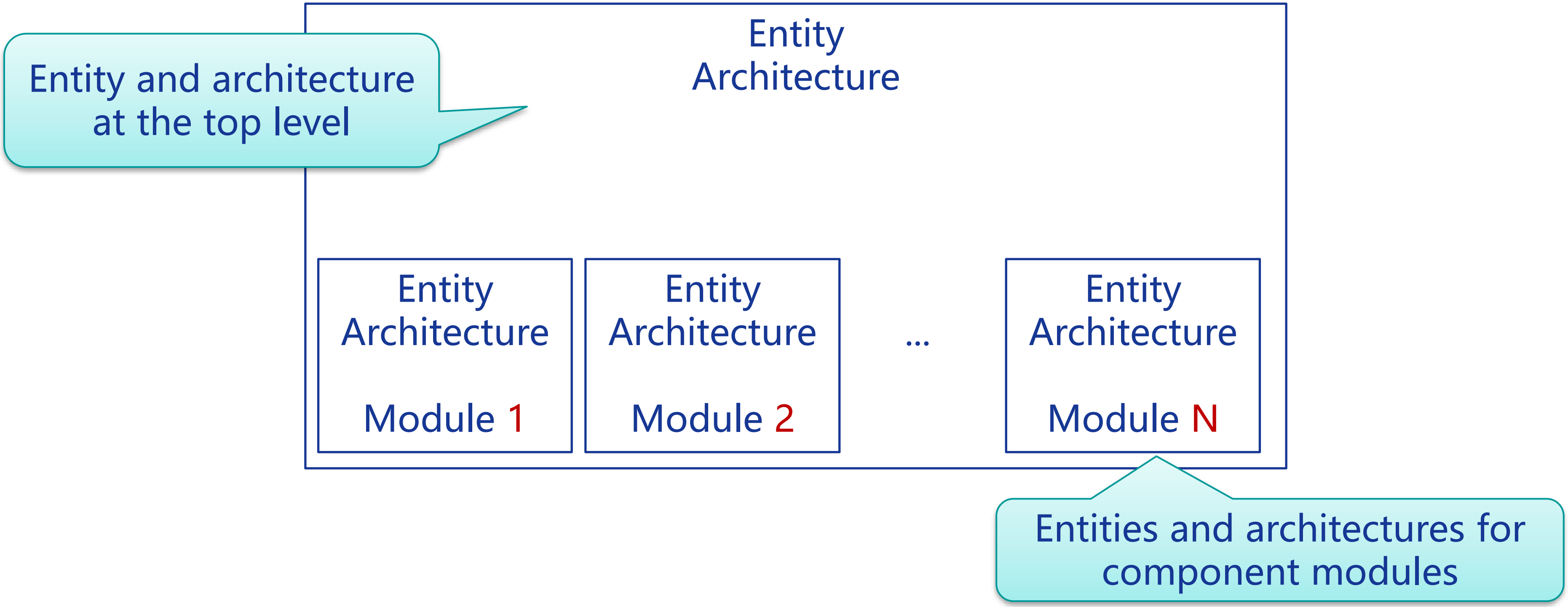


```
entity two_gates is
   port(A,B,D: in bit; E: out bit);
end two_gates;

architecture gates of two_gates is
   signal C: bit;
begin
   C <= A and B ;   -- concurrent
   E <= C or D ;     -- statements
end gates;
```

C : internal signal

Concurrent statements are placed between **begin** and **end**

2.4 VHDL Modules



Entity and architecture at the top level

Entity
Architecture

Entity
Architecture

Module 1

Entity
Architecture

Module 2

…

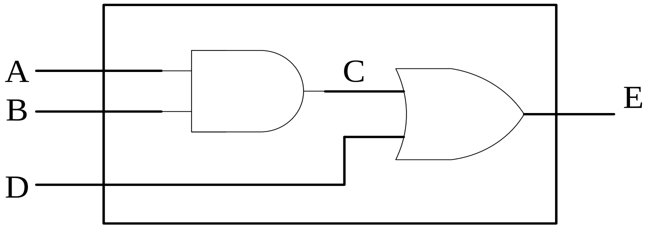
Entity
Architecture

Module N

Entities and architectures for component modules

2.4 VHDL Modules

Entity



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

Interface signals can be used to connect to other modules or to the outside world

```
entity entity-name is
      [port (interface-signal-declaration);]
end [entity] [entity-name];
```

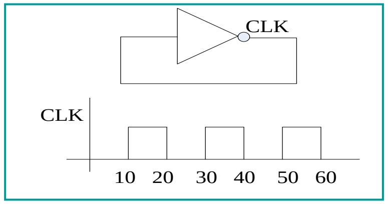
The items enclosed in [ ] are optional

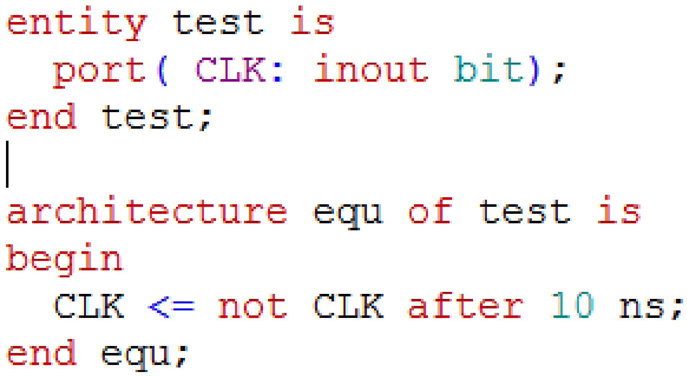
2.4 VHDL Modules

```
entity entity-name is
      [port (interface-signal-declaration);]
end [entity] [entity-name];
```

```
   list-of-interface-signals: mode type [:= initial-value]
{; list-of-interface-signals: mode type [:= initial-value]};
```

Mode	Direction of information
in	input port signals
out	output port signals
inout	bidirectional signals





```
entity test is
  port( CLK: inout bit);
end test;

architecture equ of test is
begin
  CLK <= not CLK after 10 ns;
end equ;
```

2.4 VHDL Modules

Example: entity without port

```
entity test_code_conv is
end test_code_conv;

architecture tester of test_code_conv is
signal X, CLK Z: bit;
component Code_Converter is
  port(X, CLK: in bit; Z: out bit);
end component;
begin
  clk <= not clk after 100 ns;
  X <= '0', '1' after 350 ns, '0' after 550 ns, '1' after
       750 ns, '0' after 950 ns, '1' after 1350 ns;
  CC: Code_Converter port map (X, clk, Z);
end tester;
```

Example in Chp 2.19

2.4 VHDL Modules

Type

bit, bit-vector, integer, ...

```
   list-of-interface-signals: mode type [:= initial-value]
{; list-of-interface-signals: mode type [:= initial-value]};
```

optional

Initial value

Example

A and B are initially set to 2

C and D are initially set to '0'

```
port(A, B: in integer := 2; C, D: out bit);
```

- The rules for initial values ensure that all simulations start from the same, known, state
- This means that identical simulations will give identical results even on different simulators

2.4 VHDL Modules

Initial value

Example

```
port(A, B: in integer := 2; C, D: out bit);
```

- These initial values are significant **only for simulation** and **not for synthesis**

- For synthesis, there is no hardware interpretation of an initial value
- It is not possible to initialize all signals in a circuit with a known value on power-up
- Even though it is possible to do a power-on reset in some logic technologies, it would not be desirable to do this for every wire in the circuit
- So, synthesis must ignore initial values

2.4 VHDL Modules

Architecture

```
architecture gates of two_gates is
signal C: bit;
begin
   C <= A and B ;  -- concurrent
   E <= C or D ;    -- statements
end gates;
```

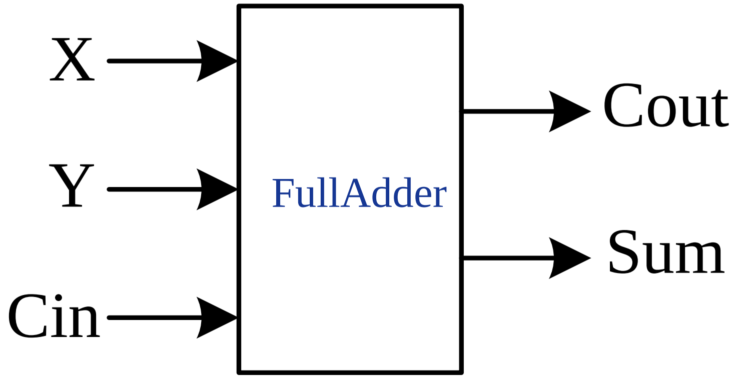
```
architecture architecture-name of entity-name is
      [declarations]
begin
      architecture body
end [architecture] [architecture-name];
```

[Declarations] (optional) declare internal signals and components (among others) used within the architecture

Architecture body contains statements that describe the operation of the module

2.4 VHDL Modules

Full adder



```
entity FullAdder is
   port (X, Y, Cin:      in      bit;      -- Inputs
              Cout, Sum:  out  bit);       -- Outputs
end  FullAdder;
```

$$Sum = X'Y'C_{in} + X'YC_{in}' + XY'C_{in} + XYC_{in} = X \oplus Y \oplus C_{in}$$

$$Cout = X'YC_{in} + XY'C_{in} + XYC_{in}' + XYC_{in} = XY + XC_{in} + YC_{in}$$

2.4 VHDL Modules

Full adder

```
entity FullAdder is
   port (X, Y, Cin: in bit;    -- Inputs
       Cout, Sum: out bit);     -- Outputs
end  FullAdder;
```

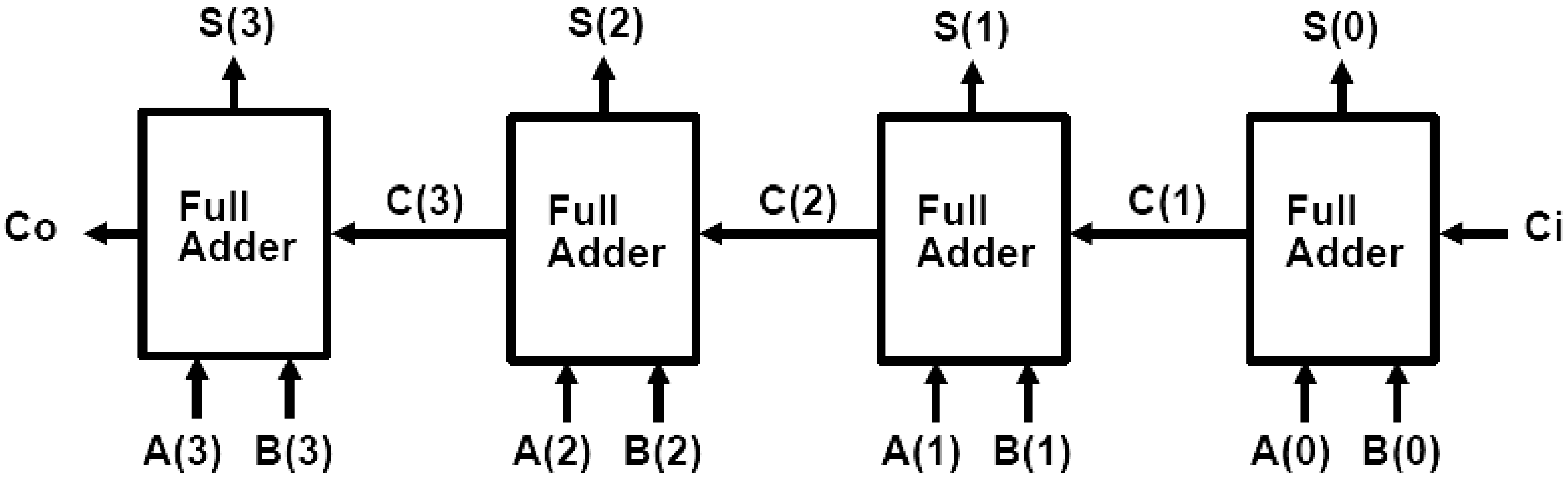
$$Sum = X'Y'C_{in} + X'YC_{in}' + XY'C_{in} + XYC_{in} = X \oplus Y \oplus C_{in}$$

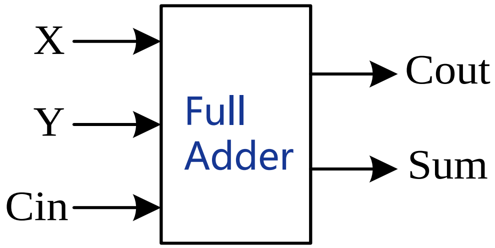
$$Cout = X'YC_{in} + XY'C_{in} + XYC_{in}' + XYC_{in} = XY + XC_{in} + YC_{in}$$

```
architecture Equations of FullAdder is
begin
   Sum <= X xor Y xor Cin after 10 ns;
   Cout <= (X and Y) or (X and Cin) or (Y and Cin) after 10 ns;
end Equations;
```

2.4.1 Four-bit full adder

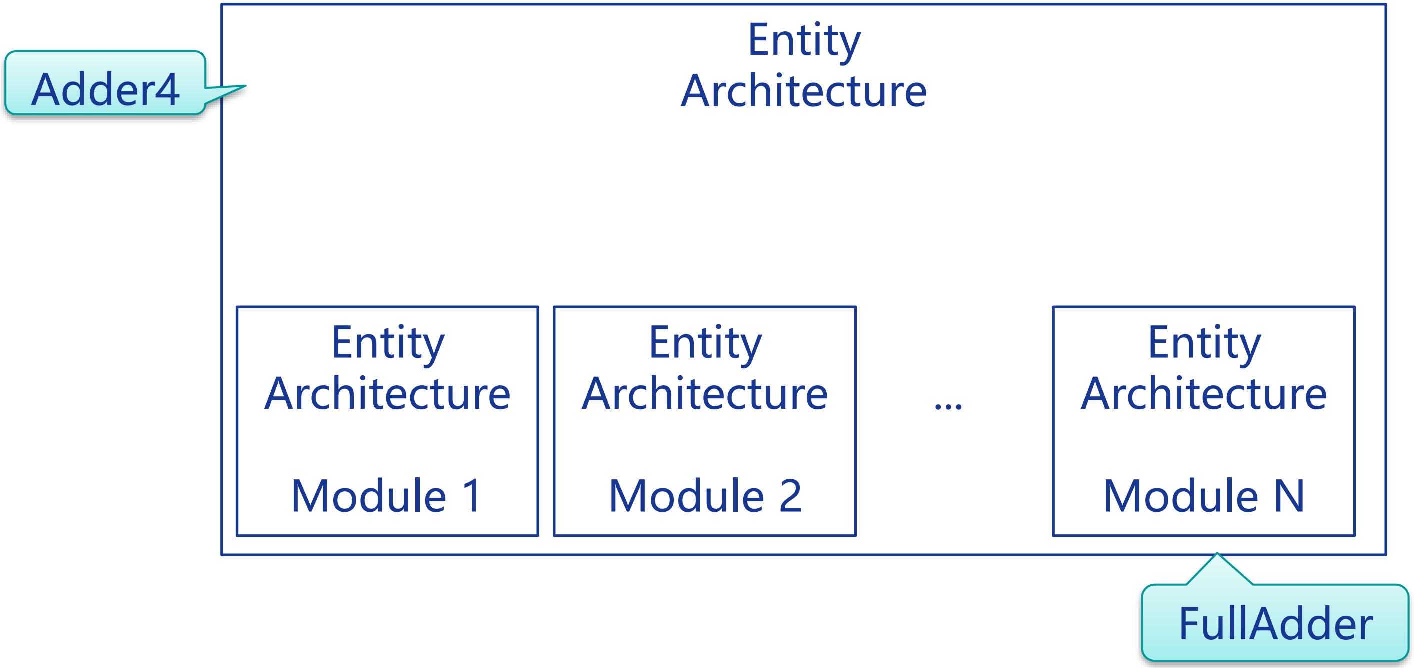
Adder4





```
entity FullAdder is
  port (X, Y, Cin: in bit; Cout, Sum: out bit);
end  FullAdder;

architecture Equations of FullAdder is
begin
   Sum <= X xor Y xor Cin after 10 ns;
   Cout <= (X and Y) or (X and Cin) or (Y and Cin) after 10 ns;
end Equations;
```

2.4 VHDL Modules



Adder4

Entity
Architecture

Entity
Architecture

Module 1

Entity
Architecture

Module 2

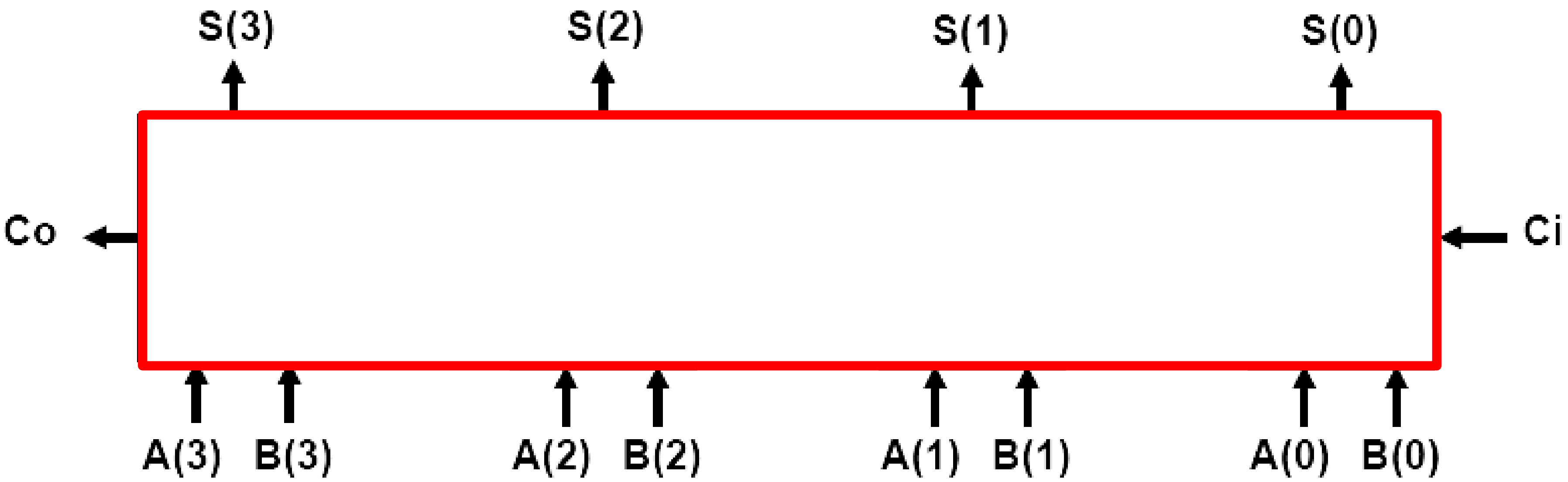
...

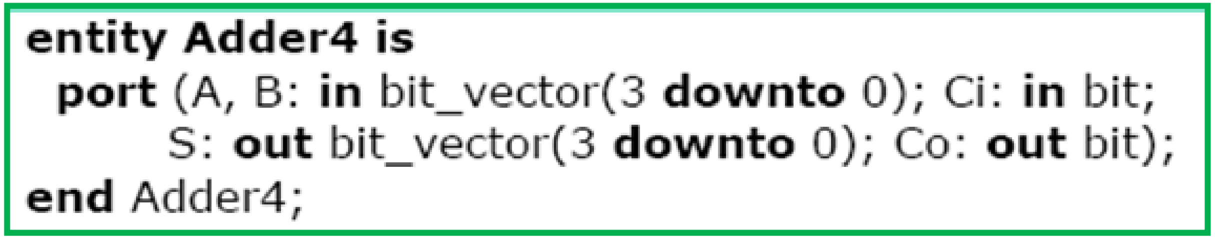
Entity
Architecture

Module N

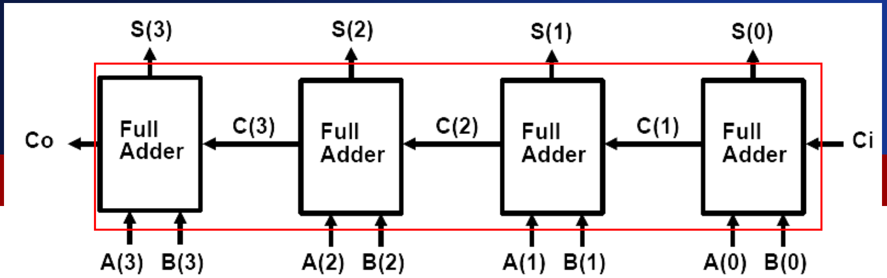
FullAdder

2.4.1 Four-bit full adder



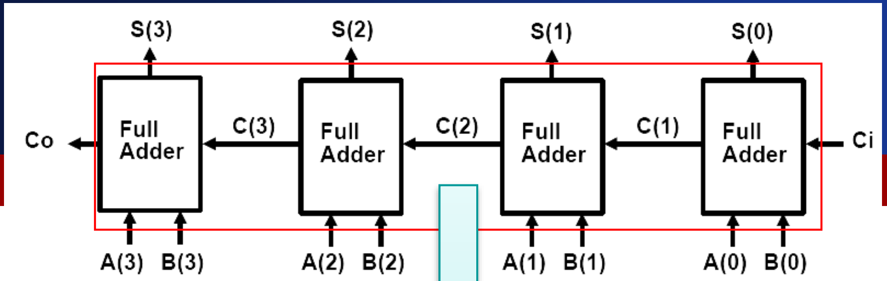


```
entity Adder4 is
  port (A, B: in bit_vector(3 downto 0); Ci: in bit;
          S: out bit_vector(3 downto 0); Co: out bit);
end Adder4;
```



```
entity FullAdder is
  port (X, Y, Cin: in bit;
      Cout, Sum: out bit);
end  FullAdder;
```

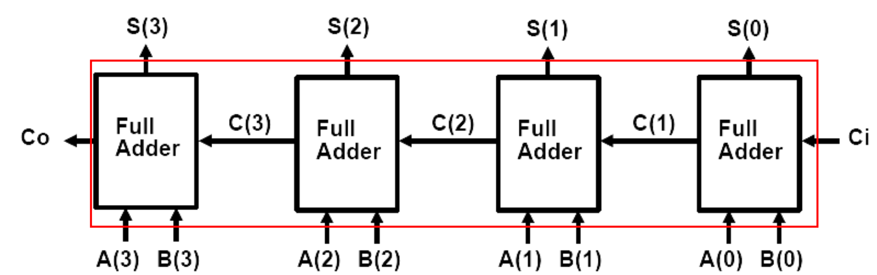
```
entity Adder4 is
  port (A, B: in bit_vector(3 downto 0); Ci: in bit;      -- Inputs
       S: out bit_vector(3 downto 0); Co: out bit);      -- Outputs
end Adder4;

architecture Structure of Adder4 is
component FullAdder
   port (X, Y, Cin: in bit;             -- Inputs
         Cout, Sum: out bit);           -- Outputs
end component;
signal C: bit_vector(3 downto 1);
begin     --instantiate four copies of the FullAdder
   FA0: FullAdder port map (A(0), B(0), Ci, C(1), S(0));
   FA1: FullAdder port map (A(1), B(1), C(1), C(2), S(1));
   FA2: FullAdder port map (A(2), B(2), C(2), C(3), S(2));
   FA3: FullAdder port map (A(3), B(3), C(3), Co, S(3));
end Structure;
```



```
entity FullAdder is
  port (X, Y, Cin: in bit;
      Cout, Sum: out bit);
end  FullAdder;
```

```
entity Adder4 is
  port (A, B: in bit_vector(3 downto 0); Ci: in bit;       -- Inputs
          S: out bit_vector(3 downto 0); Co: out bit);     -- Outputs
end Adder4;

architecture Structure of Adder4 is
component FullAdder
   port (X, Y, Cin: in bit;           -- Inputs
           Cout, Sum: out bit);       -- Outputs
end component;
signal C: bit_vector(3 downto 1);
begin     --instantiate four copies of the FullAdder
   FA0: FullAdder port map (A(0), B(0), Ci, C(1), S(0));
   FA1: FullAdder port map (A(1), B(1), C(1), C(2), S(1));
   FA2: FullAdder port map (A(2), B(2), C(2), C(3), S(2));
   FA3: FullAdder port map (A(3), B(3), C(3), Co, S(3));
end Structure;
```

3-bit internal carry signal

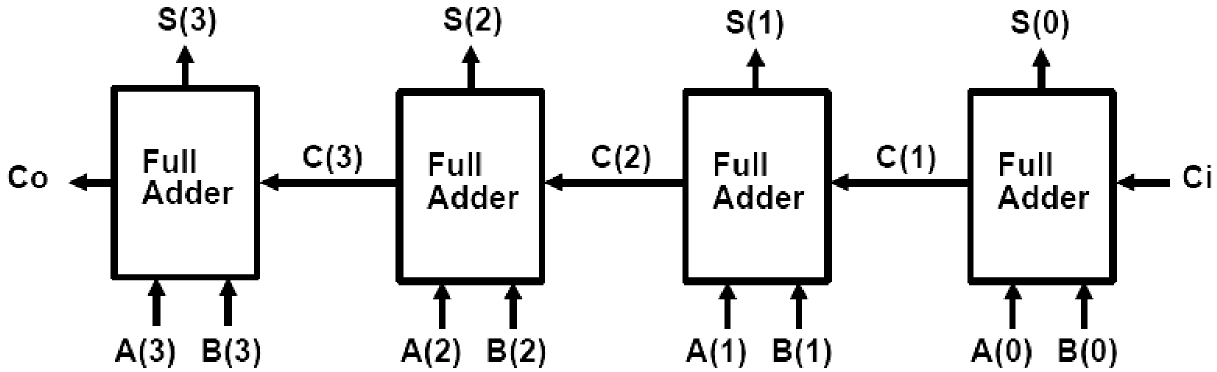


```
entity FullAdder is
  port (X, Y, Cin: in bit;
      Cout, Sum: out bit);
end  FullAdder;
```

```
entity Adder4 is
  port (A, B: in bit_vector(3 downto 0); Ci: in bit;      -- Inputs
         S: out bit_vector(3 downto 0); Co: out bit);     -- Outputs
end Adder4;

architecture Structure of Adder4 is
component FullAdder
   port (X, Y, Cin: in bit;             -- Inputs
          Cout, Sum: out bit);          -- Outputs
end component;
signal C: bit_vector(3 downto 1);
begin     --instantiate four copies of the FullAdder
   FA0: FullAdder port map (A(0), B(0), Ci, C(1), S(0));
   FA1: FullAdder port map (A(1), B(1), C(1), C(2), S(1));
   FA2: FullAdder port map (A(2), B(2), C(2), C(3), S(2));
   FA3: FullAdder port map (A(3), B(3), C(3), Co, S(3));
end Structure;
```

label

port map



```
component FullAdder
   port (X, Y, Cin: in bit;           -- Inputs
         Cout, Sum: out bit);         -- Outputs
end component;
```

```
   FA0: FullAdder port map (A(0), B(0), Ci, C(1), S(0));
   FA1: FullAdder port map (A(1), B(1), C(1), C(2), S(1));
   FA2: FullAdder port map (A(2), B(2), C(2), C(3), S(2));
   FA3: FullAdder port map (A(3), B(3), C(3), Co, S(3));
```

The order of the signals must be the same as the order of the signals in the port of the component declaration (Positional Association)

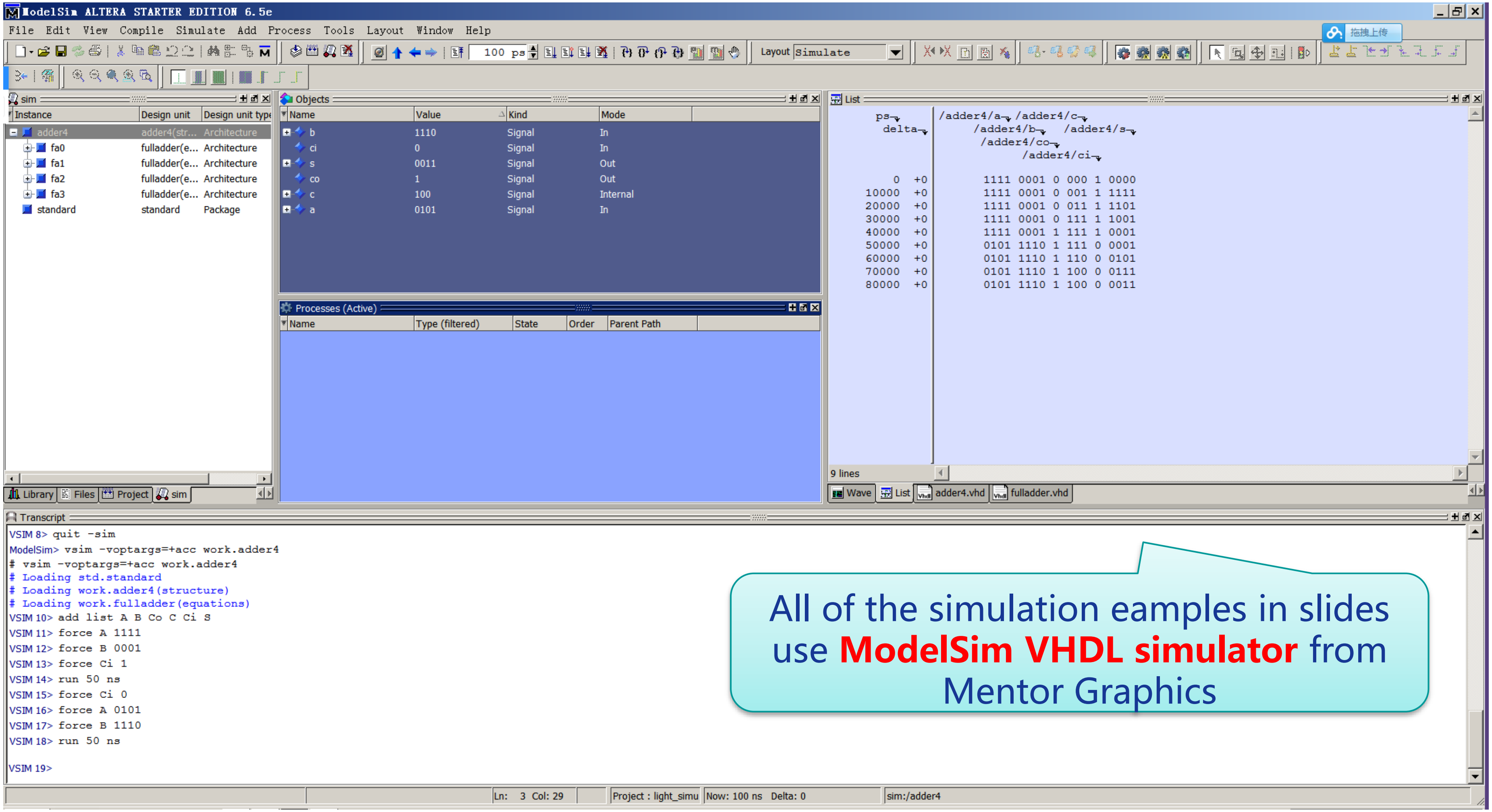
```
component component-name
          port(list-of-interface-signals-and-their-types);
end component
```

Port clause used in component declaration has the same form as the port clause used in an entity declaration

```
label: component-name port map (list-of-acutal-signal);
```

The list of actual signals must correspond one-to-one to the list of interface signals specified in the component declaration

Simulation use ModelSim


ModelSim ALTERA STARTER EDITION 6.5e
File Edit View Compile Simulate Add Process Tools Layout Window Help
100 ps
Layout Simulate
sim
Objects
Processes (Active)
List
Transcript
Library Files Project sim
Wave List adder4.vhd fulladder.vhd
9 lines

Instance	Design unit	Design unit type
adder4	adder4(str...	Architecture
fa0	fulladder(e...	Architecture
fa1	fulladder(e...	Architecture
fa2	fulladder(e...	Architecture
fa3	fulladder(e...	Architecture
standard	standard	Package

Name	Value	Kind	Mode
b	1110	Signal	In
ci	0	Signal	In
s	0011	Signal	Out
co	1	Signal	Out
c	100	Signal	Internal
a	0101	Signal	In

Name	Type (filtered)	State	Order	Parent Path

```
       ps    /adder4/a  /adder4/c
    delta         /adder4/b    /adder4/s
                   /adder4/co
                         /adder4/ci

        0  +0      1111 0001 0 000 1 0000
    10000  +0      1111 0001 0 001 1 1111
    20000  +0      1111 0001 0 011 1 1101
    30000  +0      1111 0001 0 111 1 1001
    40000  +0      1111 0001 1 111 1 0001
    50000  +0      0101 1110 1 111 0 0001
    60000  +0      0101 1110 1 110 0 0101
    70000  +0      0101 1110 1 100 0 0111
    80000  +0      0101 1110 1 100 0 0011
```

```
VSIM 8> quit -sim
ModelSim> vsim -voptargs=+acc work.adder4
# vsim -voptargs=+acc work.adder4
# Loading std.standard
# Loading work.adder4(structure)
# Loading work.fulladder(equations)
VSIM 10> add list A B Co C Ci S
VSIM 11> force A 1111
VSIM 12> force B 0001
VSIM 13> force Ci 1
VSIM 14> run 50 ns
VSIM 15> force Ci 0
VSIM 16> force A 0101
VSIM 17> force B 1110
VSIM 18> run 50 ns

VSIM 19>
```

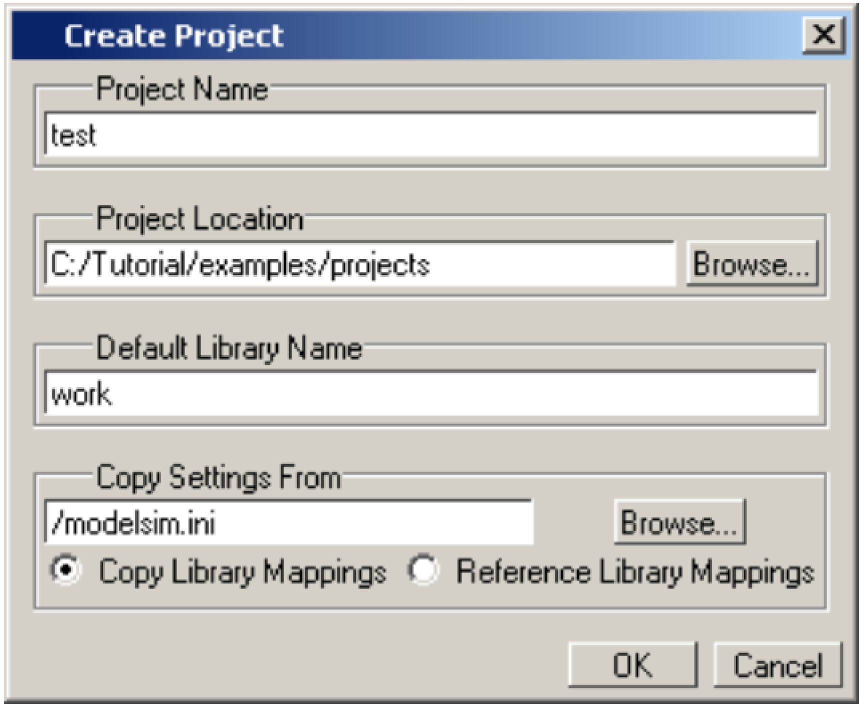
Ln: 3 Col: 29 | Project : light_simu | Now: 100 ns Delta: 0 | sim:/adder4

All of the simulation eamples in slides use **ModelSim VHDL simulator** from Mentor Graphics

Simulation use ModelSim

Create a New Project

File > New > Project



Create Project

Project Name: test

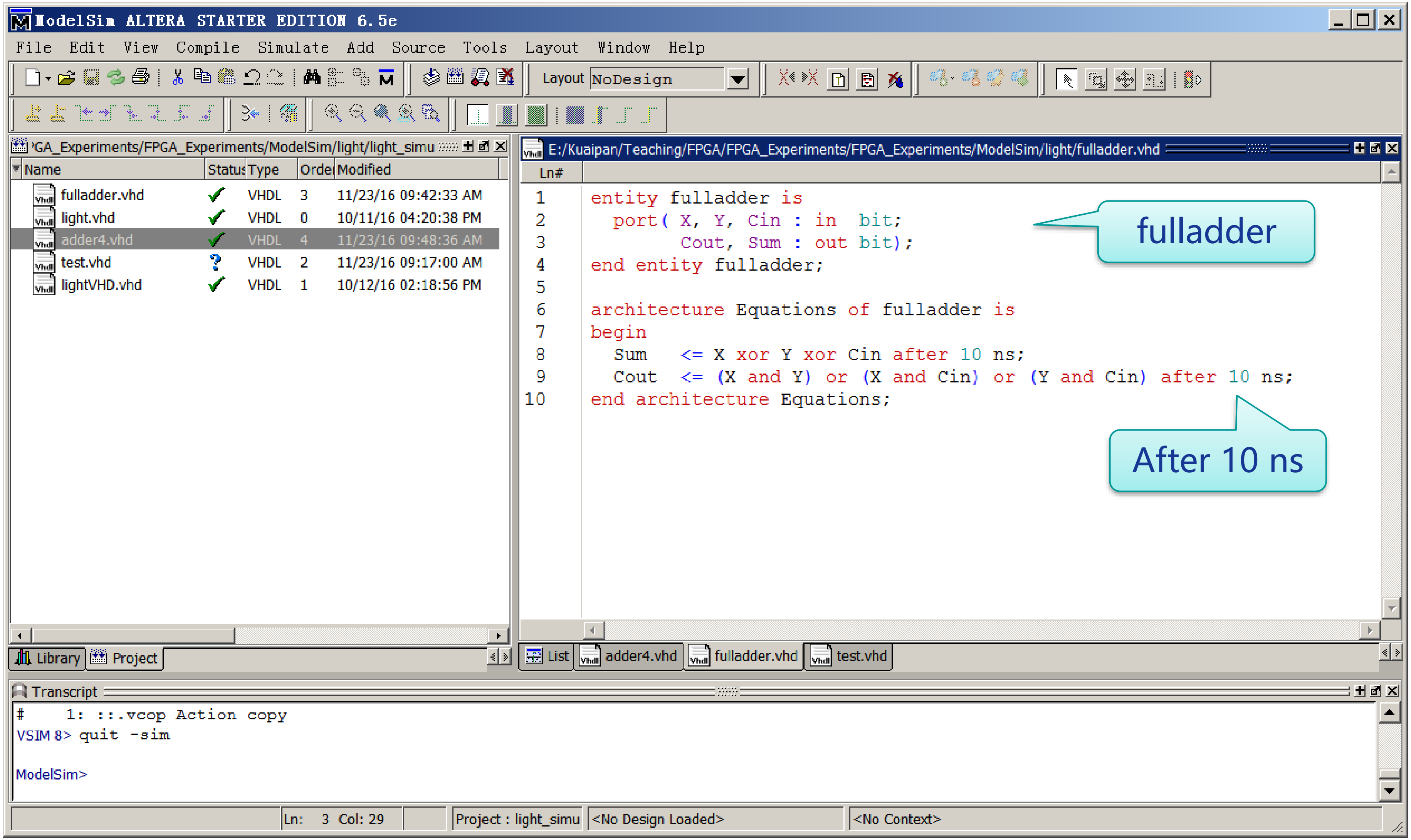
Project Location: C:/Tutorial/examples/projects

Default Library Name: work

Copy Settings From: /modelsim.ini

● Copy Library Mappings ○ Reference Library Mappings

Simulation use ModelSim


ModelSim ALTERA STARTER EDITION 6.5e
File Edit View Compile Simulate Add Source Tools Layout Window Help
Layout NoDesign
Library
Project
List
adder4.vhd
fulladder.vhd
test.vhd
Transcript

'GA_Experiments/FPGA_Experiments/ModelSim/light/light_simu

Name	Status	Type	Order	Modified
fulladder.vhd	✓	VHDL	3	11/23/16 09:42:33 AM
light.vhd	✓	VHDL	0	10/11/16 04:20:38 PM
adder4.vhd	✓	VHDL	4	11/23/16 09:48:36 AM
test.vhd	?	VHDL	2	11/23/16 09:17:00 AM
lightVHD.vhd	✓	VHDL	1	10/12/16 02:18:56 PM

E:/Kuaipan/Teaching/FPGA/FPGA_Experiments/FPGA_Experiments/ModelSim/light/fulladder.vhd

```
entity fulladder is
  port( X, Y, Cin : in  bit;
        Cout, Sum : out bit);
end entity fulladder;

architecture Equations of fulladder is
begin
  Sum   <= X xor Y xor Cin after 10 ns;
  Cout  <= (X and Y) or (X and Cin) or (Y and Cin) after 10 ns;
end architecture Equations;
```

fulladder

After 10 ns

```
#    1: ::.vcop Action copy
VSIM 8> quit -sim

ModelSim>
```

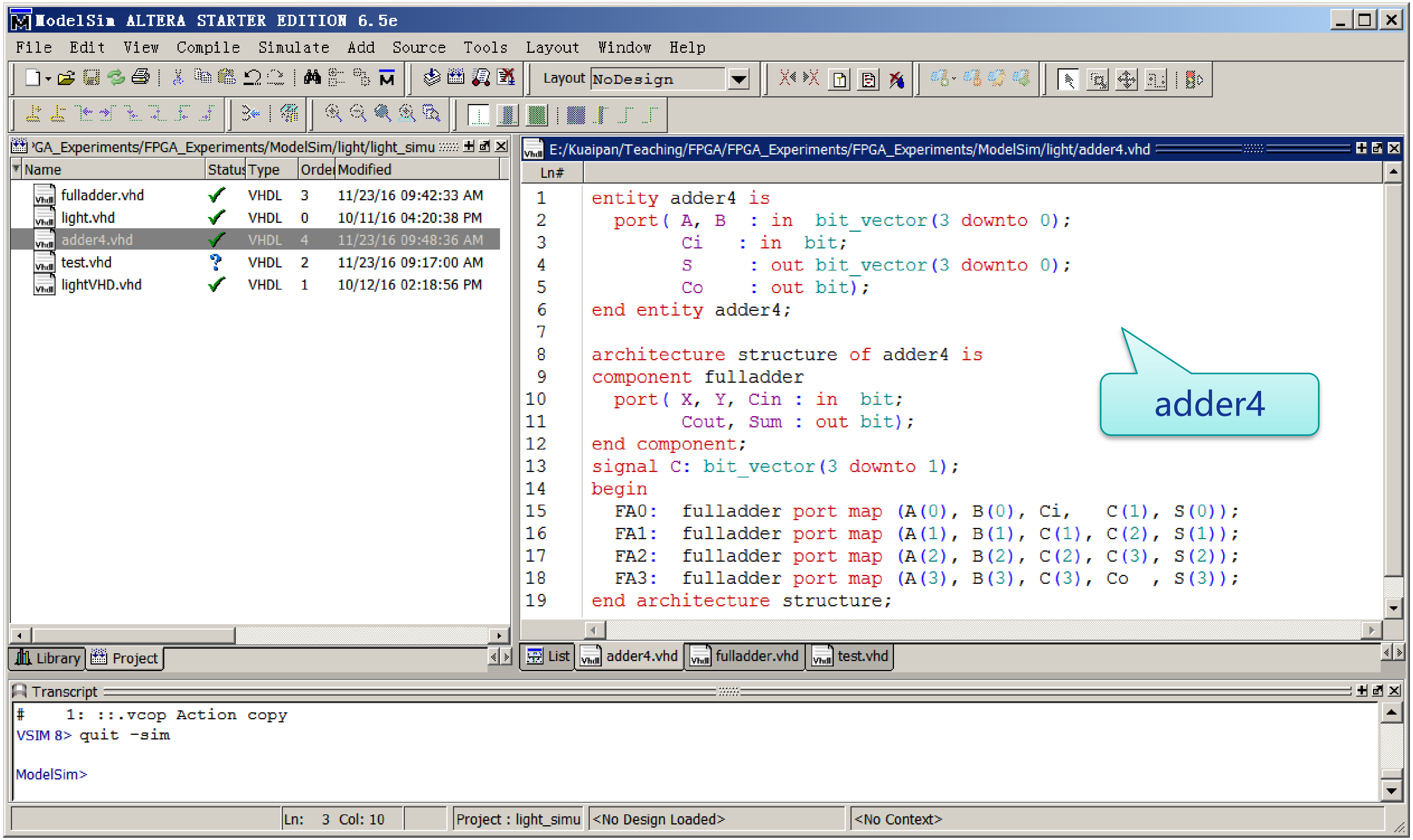
Ln: 3 Col: 29

Project : light_simu

<No Design Loaded>

<No Context>

Simulation use ModelSim


ModelSim ALTERA STARTER EDITION 6.5e
File Edit View Compile Simulate Add Source Tools Layout Window Help
Layout NoDesign
'GA_Experiments/FPGA_Experiments/ModelSim/light/light_simu
E:/Kuaipan/Teaching/FPGA/FPGA_Experiments/FPGA_Experiments/ModelSim/light/adder4.vhd
Ln#
Library
Project
List
adder4.vhd
fulladder.vhd
test.vhd
Transcript
adder4

Name	Status	Type	Order	Modified
fulladder.vhd	✓	VHDL	3	11/23/16 09:42:33 AM
light.vhd	✓	VHDL	0	10/11/16 04:20:38 PM
adder4.vhd	✓	VHDL	4	11/23/16 09:48:36 AM
test.vhd	?	VHDL	2	11/23/16 09:17:00 AM
lightVHD.vhd	✓	VHDL	1	10/12/16 02:18:56 PM

```
entity adder4 is
  port( A, B  : in  bit_vector(3 downto 0);
        Ci    : in  bit;
        S     : out bit_vector(3 downto 0);
        Co    : out bit);
end entity adder4;

architecture structure of adder4 is
component fulladder
  port( X, Y, Cin : in  bit;
        Cout, Sum : out bit);
end component;
signal C: bit_vector(3 downto 1);
begin
  FA0:  fulladder port map (A(0), B(0), Ci,   C(1), S(0));
  FA1:  fulladder port map (A(1), B(1), C(1), C(2), S(1));
  FA2:  fulladder port map (A(2), B(2), C(2), C(3), S(2));
  FA3:  fulladder port map (A(3), B(3), C(3), Co  , S(3));
end architecture structure;
```

```
#     1: ::.vcop Action copy
VSIM 8> quit -sim

ModelSim>
```

Ln: 3 Col: 10

Project : light_simu

<No Design Loaded>

<No Context>

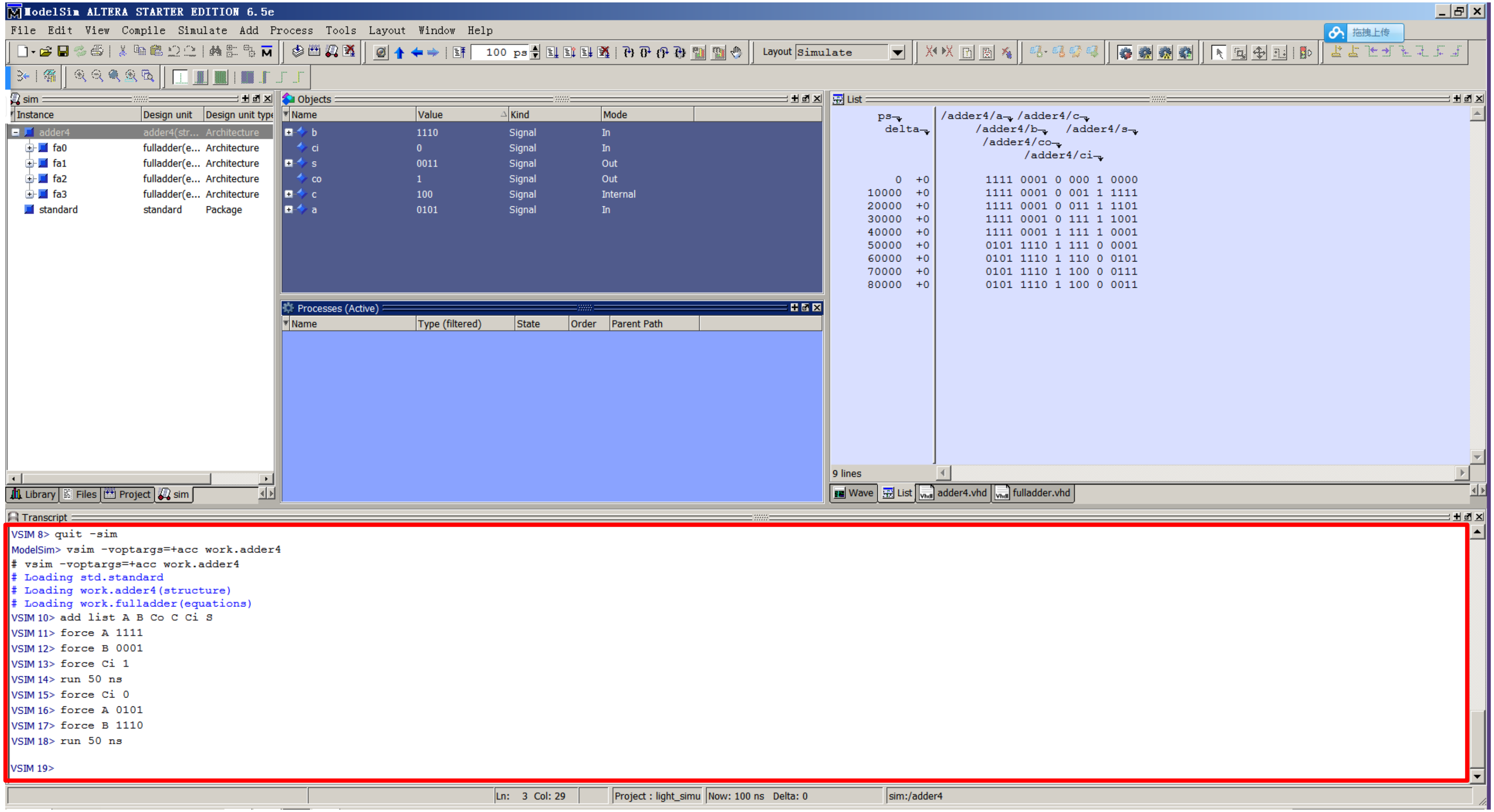
Simulation use ModelSim

Compile the Design Units

Compile > Compile

Load the Design

Simulate > Start Simulation



ModelSim ALTERA STARTER EDITION 6.5e
File Edit View Compile Simulate Add Process Tools Layout Window Help
拖拽上传
100 ps
Layout Simulate
sim
Objects
Processes (Active)
List
Transcript
Library Files Project sim
Wave List adder4.vhd fulladder.vhd
9 lines

Instance	Design unit	Design unit type
adder4	adder4(str...	Architecture
fa0	fulladder(e...	Architecture
fa1	fulladder(e...	Architecture
fa2	fulladder(e...	Architecture
fa3	fulladder(e...	Architecture
standard	standard	Package

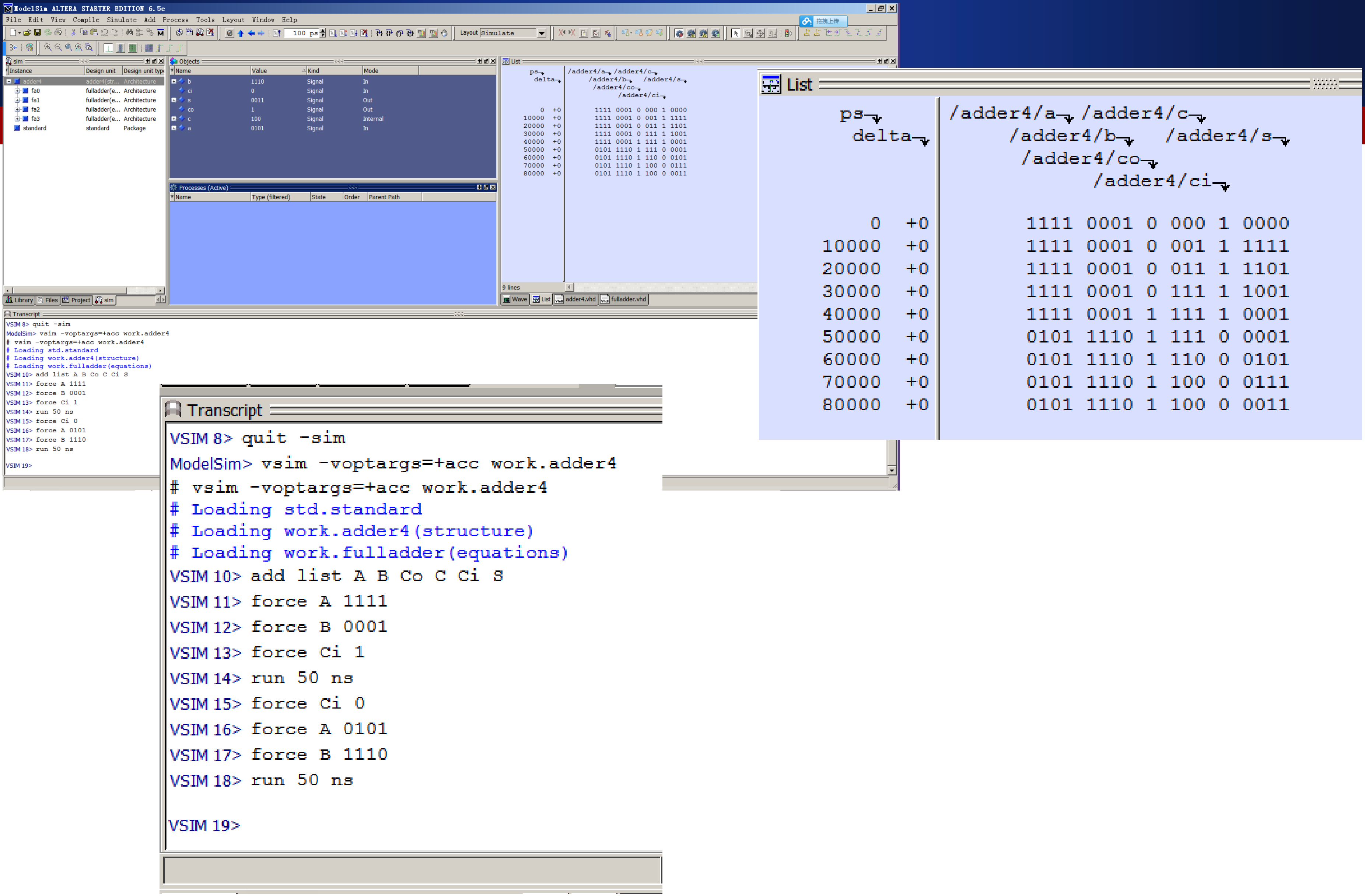
Name	Value	Kind	Mode
b	1110	Signal	In
ci	0	Signal	In
s	0011	Signal	Out
co	1	Signal	Out
c	100	Signal	Internal
a	0101	Signal	In

Name	Type (filtered)	State	Order	Parent Path

```
        ps     /adder4/a  /adder4/c
      delta        /adder4/b    /adder4/s
                    /adder4/co
                          /adder4/ci

         0  +0         1111 0001 0 000 1 0000
     10000  +0         1111 0001 0 001 1 1111
     20000  +0         1111 0001 0 011 1 1101
     30000  +0         1111 0001 0 111 1 1001
     40000  +0         1111 0001 1 111 1 0001
     50000  +0         0101 1110 1 111 0 0001
     60000  +0         0101 1110 1 110 0 0101
     70000  +0         0101 1110 1 100 0 0111
     80000  +0         0101 1110 1 100 0 0011
```

```
VSIM 8> quit -sim
ModelSim> vsim -voptargs=+acc work.adder4
# vsim -voptargs=+acc work.adder4
# Loading std.standard
# Loading work.adder4(structure)
# Loading work.fulladder(equations)
VSIM 10> add list A B Co C Ci S
VSIM 11> force A 1111
VSIM 12> force B 0001
VSIM 13> force Ci 1
VSIM 14> run 50 ns
VSIM 15> force Ci 0
VSIM 16> force A 0101
VSIM 17> force B 1110
VSIM 18> run 50 ns

VSIM 19>
```

Ln: 3 Col: 29 Project : light_simu Now: 100 ns Delta: 0 sim:/adder4


ModelSim ALTERA STARTER EDITION 6.5e
File Edit View Compile Simulate Add Process Tools Layout Window Help
Layout Simulate
sim
Objects
Processes (Active)
Library Files Project sim
Wave List adder4.vhd fulladder.vhd
9 lines
List
Transcript

Instance	Design unit	Design unit type
adder4	adder4(str...	Architecture
fa0	fulladder(e...	Architecture
fa1	fulladder(e...	Architecture
fa2	fulladder(e...	Architecture
fa3	fulladder(e...	Architecture
standard	standard	Package

Name	Value	Kind	Mode
b	1110	Signal	In
ci	0	Signal	In
s	0011	Signal	Out
co	1	Signal	Out
c	100	Signal	Internal
a	0101	Signal	In

Name	Type (filtered)	State	Order	Parent Path

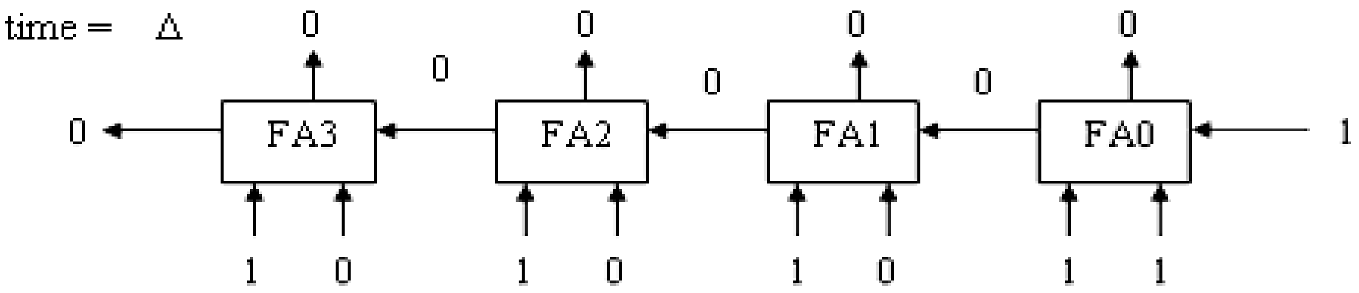
```
          ps      /adder4/a  /adder4/c
        delta         /adder4/b     /adder4/s
                       /adder4/co
                              /adder4/ci

           0   +0      1111  0001  0  000  1  0000
       10000   +0      1111  0001  0  001  1  1111
       20000   +0      1111  0001  0  011  1  1101
       30000   +0      1111  0001  0  111  1  1001
       40000   +0      1111  0001  1  111  1  0001
       50000   +0      0101  1110  1  111  0  0001
       60000   +0      0101  1110  1  110  0  0101
       70000   +0      0101  1110  1  100  0  0111
       80000   +0      0101  1110  1  100  0  0011
```

```
VSIM 8> quit -sim
ModelSim> vsim -voptargs=+acc work.adder4
# vsim -voptargs=+acc work.adder4
# Loading std.standard
# Loading work.adder4(structure)
# Loading work.fulladder(equations)
VSIM 10> add list A B Co C Ci S
VSIM 11> force A 1111
VSIM 12> force B 0001
VSIM 13> force Ci 1
VSIM 14> run 50 ns
VSIM 15> force Ci 0
VSIM 16> force A 0101
VSIM 17> force B 1110
VSIM 18> run 50 ns

VSIM 19>
```

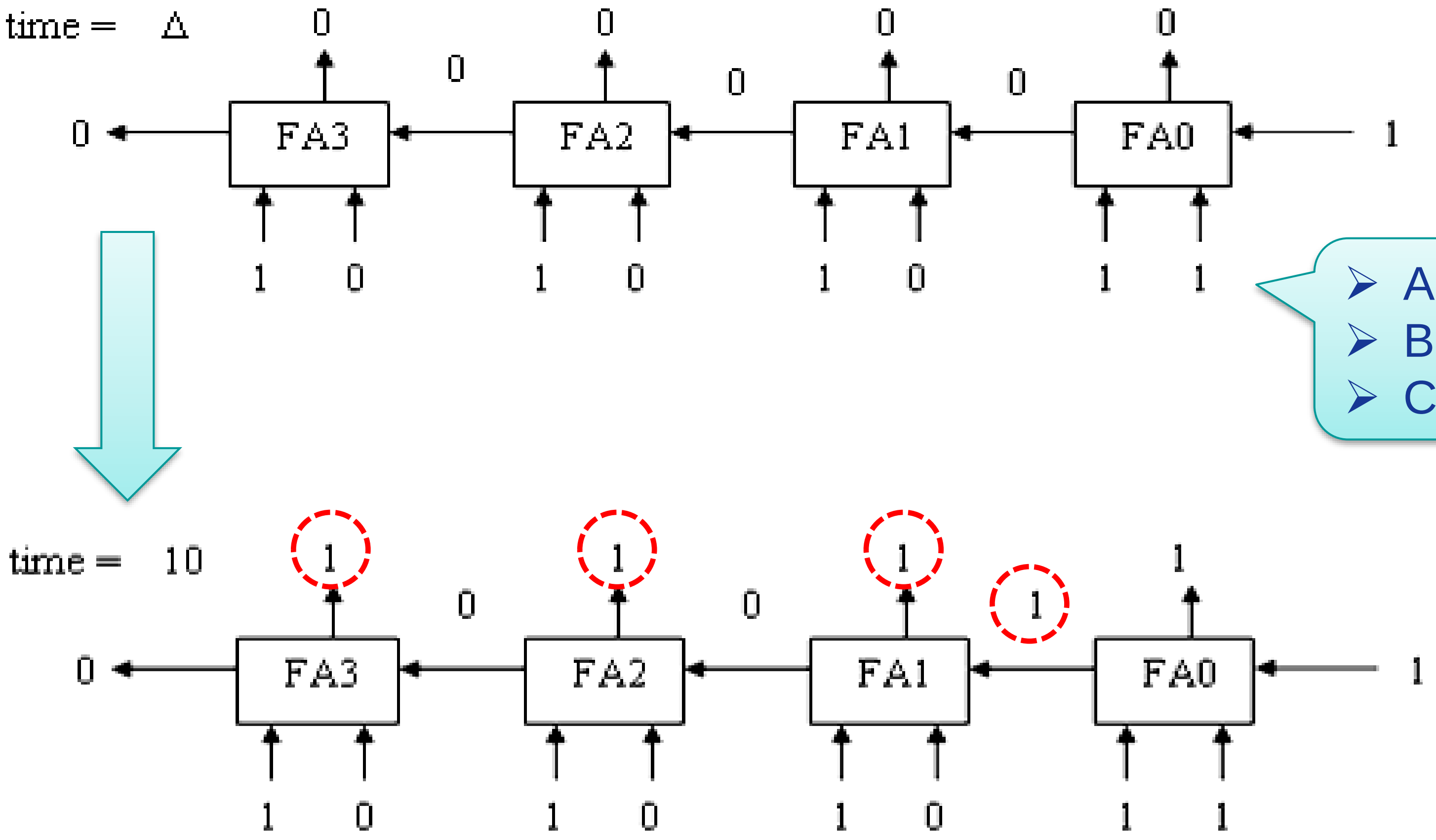
quit -sim

2.4 VHDL Modules

```
add list A B Co C Ci S  -- put these signal on the
output list
force A  1111           -- set the A inputs to 1111
force B  0001           -- set the B inputs to 0001
force Ci 1              -- set Ci to 1
run    50 ns            -- run the simulation for 50 ns
force Ci 0
force A  0101
force B  1110
run    50 ns
```

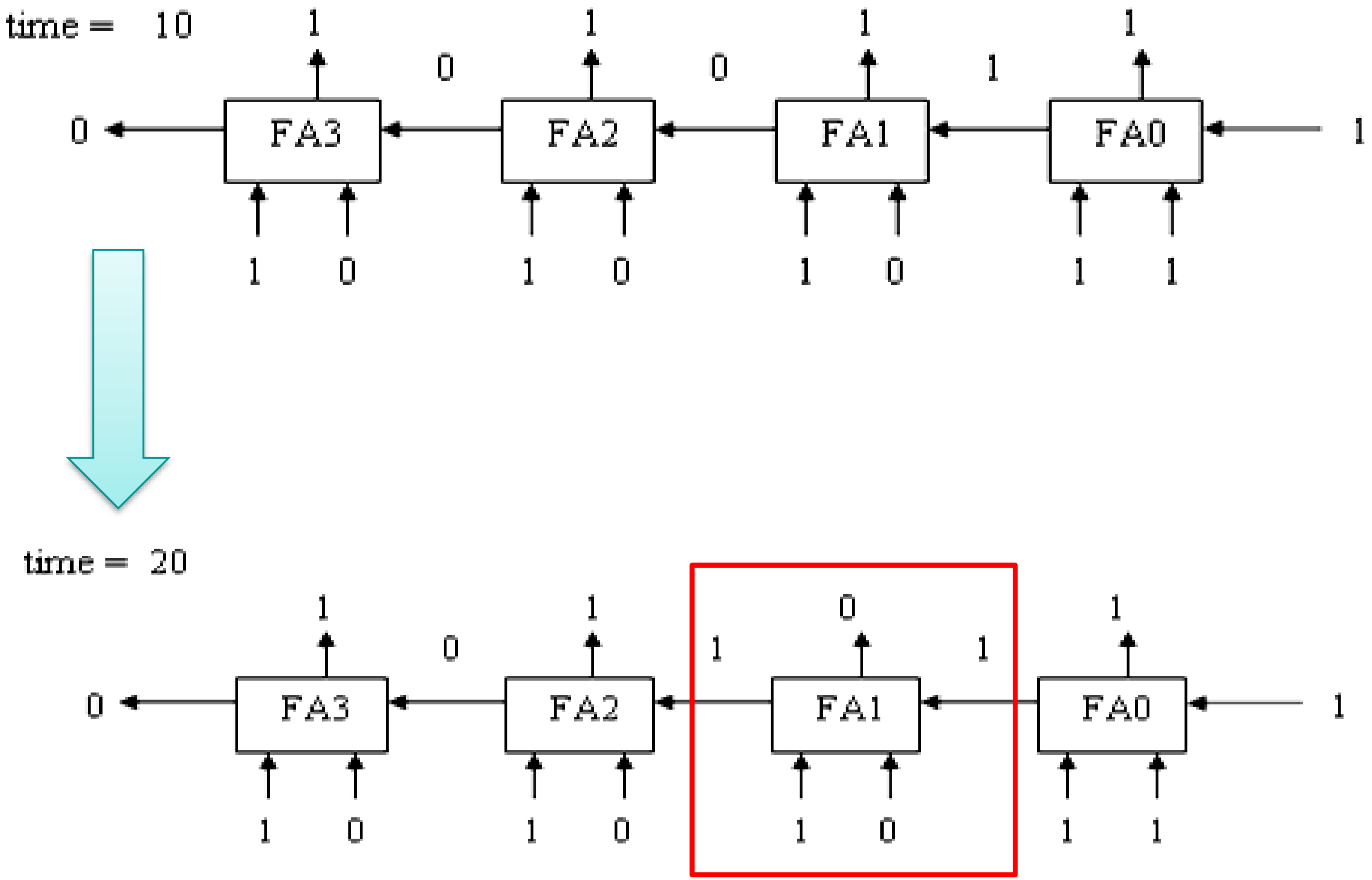


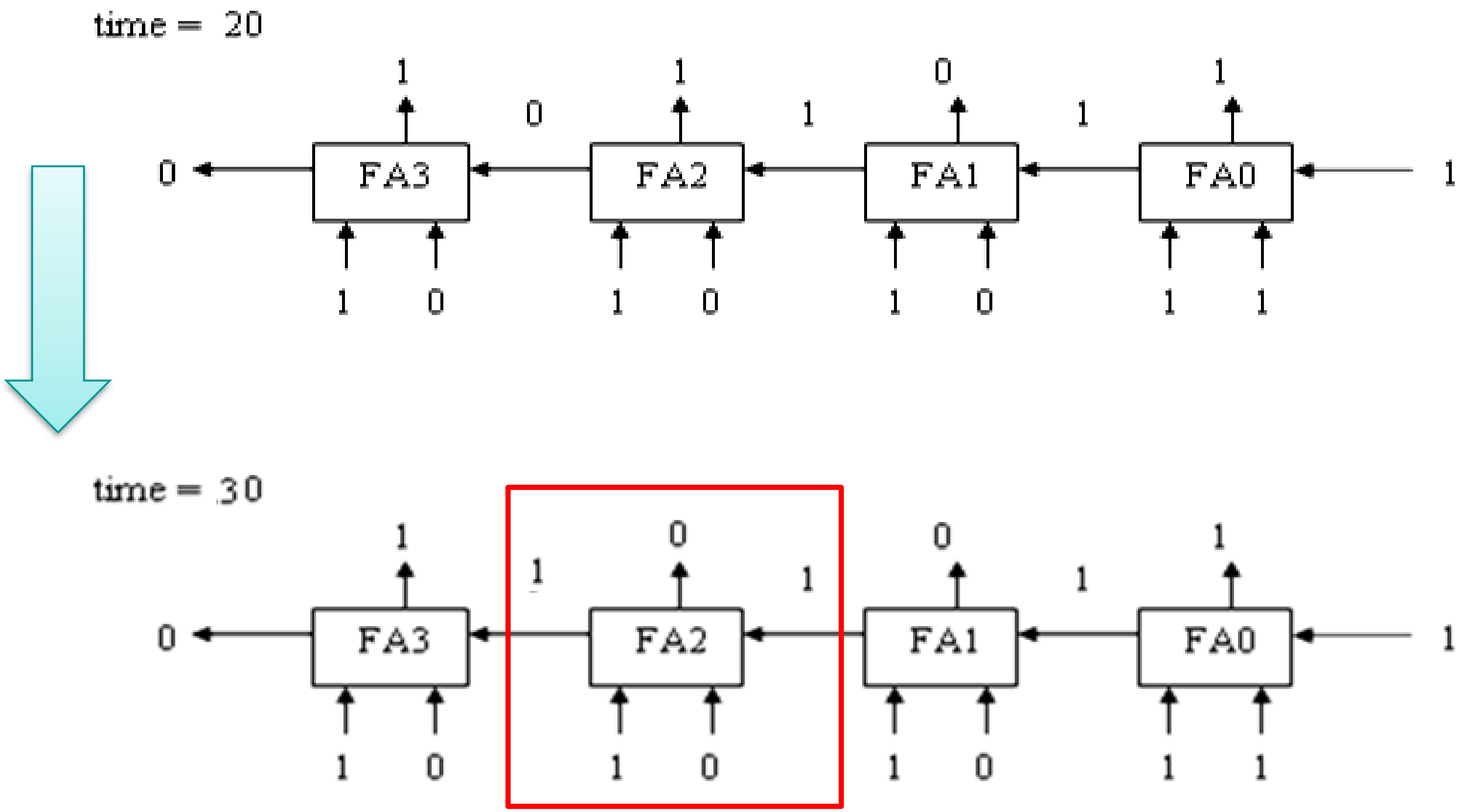
2.4 VHDL Modules

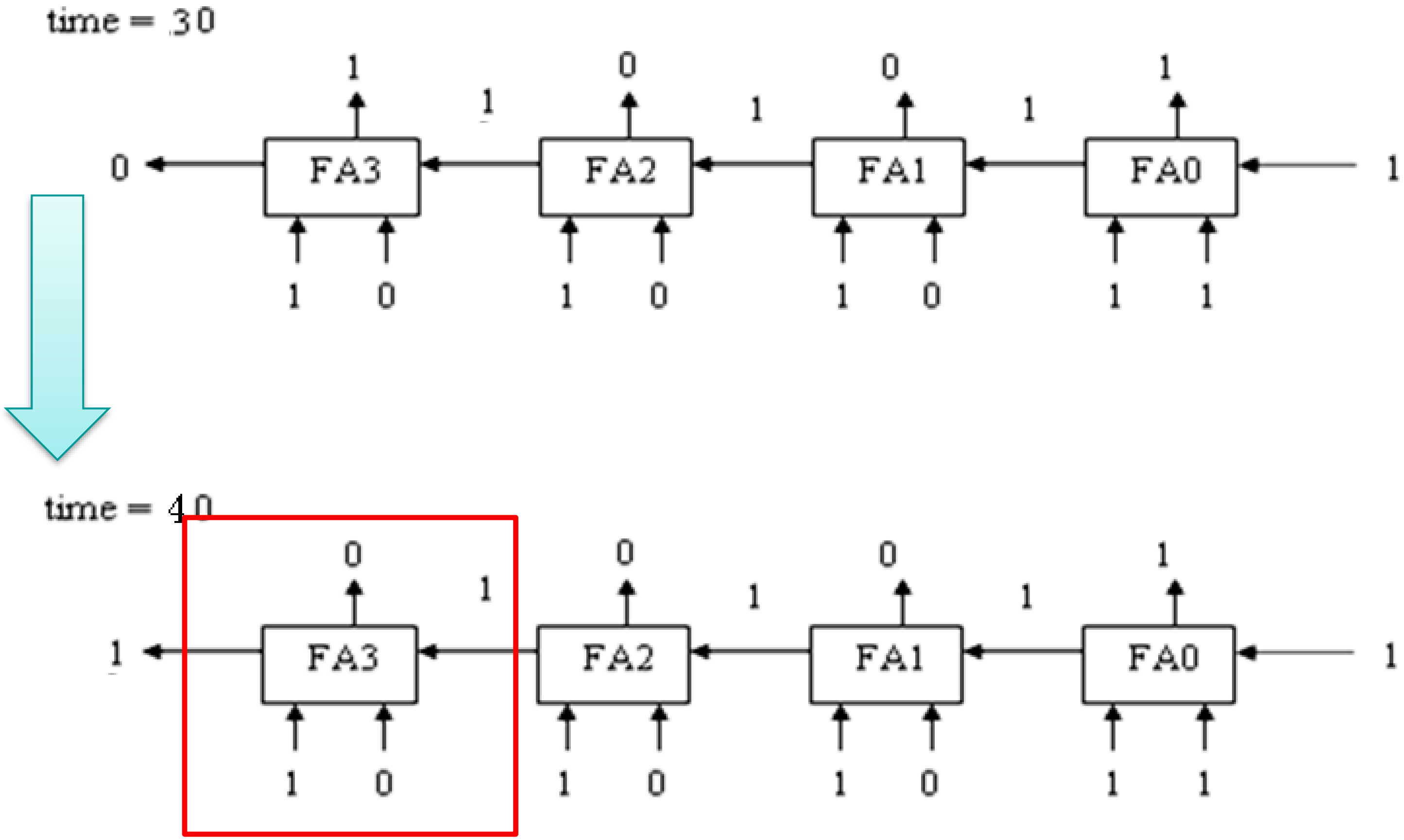


- A: 1111
- B: 0001
- Ci: 1

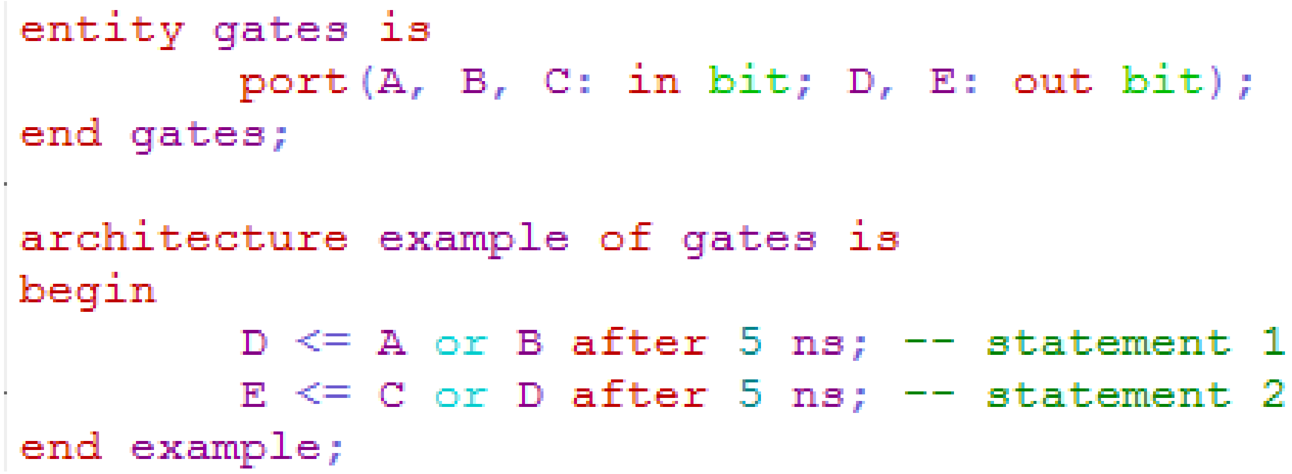
2.4 VHDL Modules



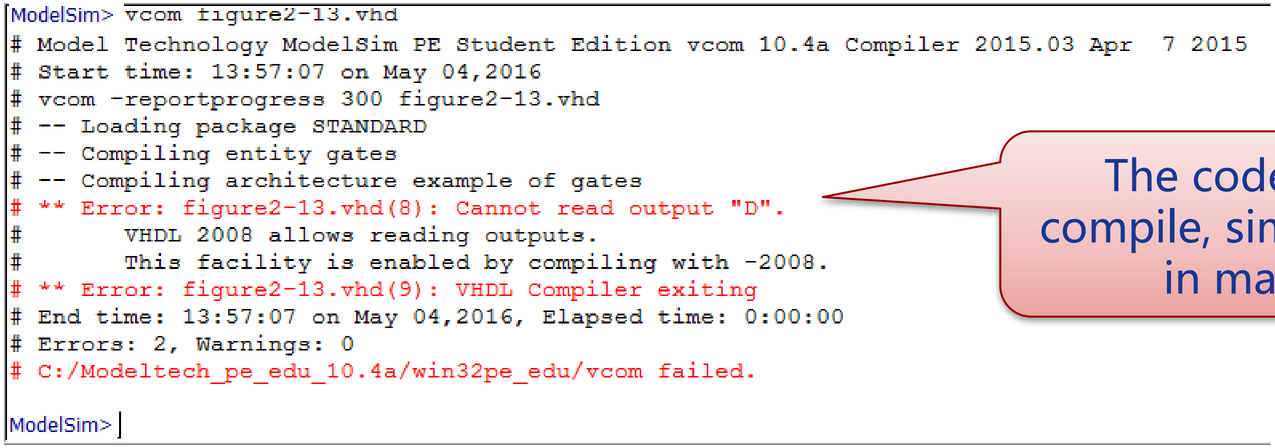

time = 20
1
1
0
1
0
1
1
1
0
FA3
FA2
FA1
FA0
1
1 0
1 0
1 0
1 1
time = 30
1
1
0
1
0
1
1
0
FA3
FA2
FA1
FA0
1
1 0
1 0
1 0
1 1


time = 30
1
0
0
1
1
1
1
0
FA3
FA2
FA1
FA0
1
1 0
1 0
1 0
1 1
time = 40
0
0
0
1
1
1
1
1
FA3
FA2
FA1
FA0
1
1 0
1 0
1 0
1 1

2.4.2 Use of "buffer" mode



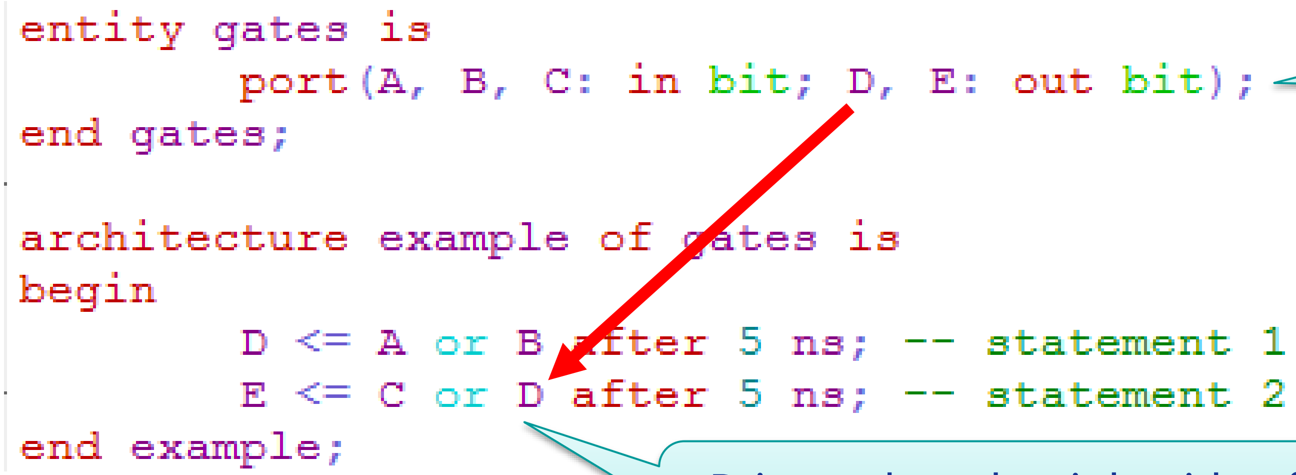
```
entity gates is
        port(A, B, C: in bit; D, E: out bit);
end gates;

architecture example of gates is
begin
        D <= A or B after 5 ns; -- statement 1
        E <= C or D after 5 ns; -- statement 2
end example;
```



```
ModelSim> vcom figure2-13.vhd
# Model Technology ModelSim PE Student Edition vcom 10.4a Compiler 2015.03 Apr  7 2015
# Start time: 13:57:07 on May 04,2016
# vcom -reportprogress 300 figure2-13.vhd
# -- Loading package STANDARD
# -- Compiling entity gates
# -- Compiling architecture example of gates
# ** Error: figure2-13.vhd(8): Cannot read output "D".
#       VHDL 2008 allows reading outputs.
#       This facility is enabled by compiling with -2008.
# ** Error: figure2-13.vhd(9): VHDL Compiler exiting
# End time: 13:57:07 on May 04,2016, Elapsed time: 0:00:00
# Errors: 2, Warnings: 0
# C:/Modeltech_pe_edu_10.4a/win32pe_edu/vcom failed.

ModelSim>
```

The code will not acutally compile, simulate, or synthesize in many tools, why?

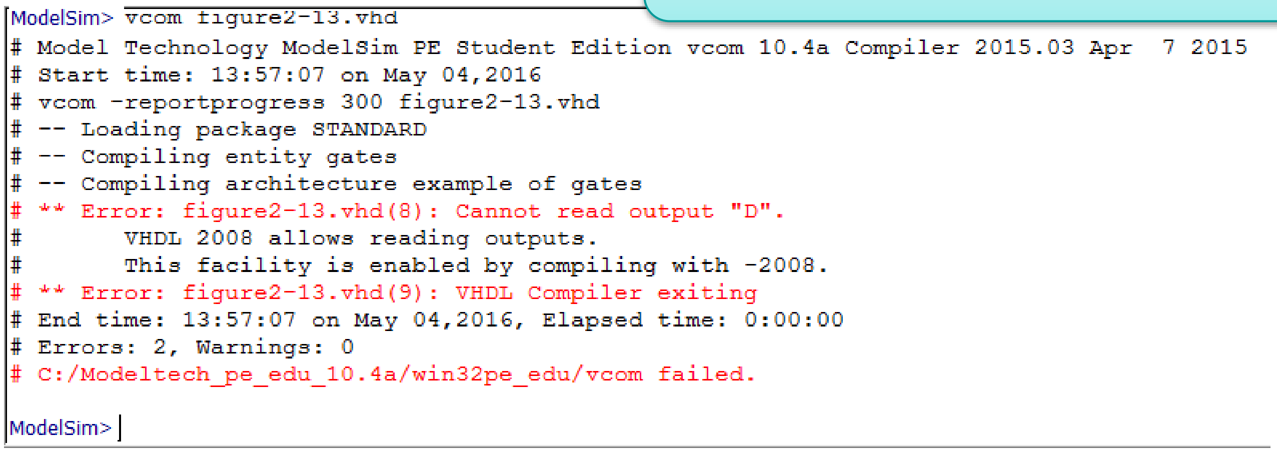
2.4.2 Use of "buffer" mode



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

architecture example of gates is
begin
        D <= A or B after 5 ns; -- statement 1
        E <= C or D after 5 ns; -- statement 2
end example;
```

D is declared only as an output

D is used on the right side of the assignment (NOT allowed)



```
ModelSim> vcom figure2-13.vhd
# Model Technology ModelSim PE Student Edition vcom 10.4a Compiler 2015.03 Apr  7 2015
# Start time: 13:57:07 on May 04,2016
# vcom -reportprogress 300 figure2-13.vhd
# -- Loading package STANDARD
# -- Compiling entity gates
# -- Compiling architecture example of gates
# ** Error: figure2-13.vhd(8): Cannot read output "D".
#       VHDL 2008 allows reading outputs.
#       This facility is enabled by compiling with -2008.
# ** Error: figure2-13.vhd(9): VHDL Compiler exiting
# End time: 13:57:07 on May 04,2016, Elapsed time: 0:00:00
# Errors: 2, Warnings: 0
# C:/Modeltech_pe_edu_10.4a/win32pe_edu/vcom failed.

ModelSim>
```

2.4.2 Use of "buffer" mode

```
entity gates is
        port(A, B, C: in bit: D: inout bit; E: out bit);
end gates;

architecture example of gates is
begin
        D <= A or B after 5 ns;          -- statement 1
        E <= C or D after 5 ns;          -- statement 2
end example
```

- Use of **inout** mode results in the synthesis tools creating a truly bidirectional signal
- D is not an external input to the circuit

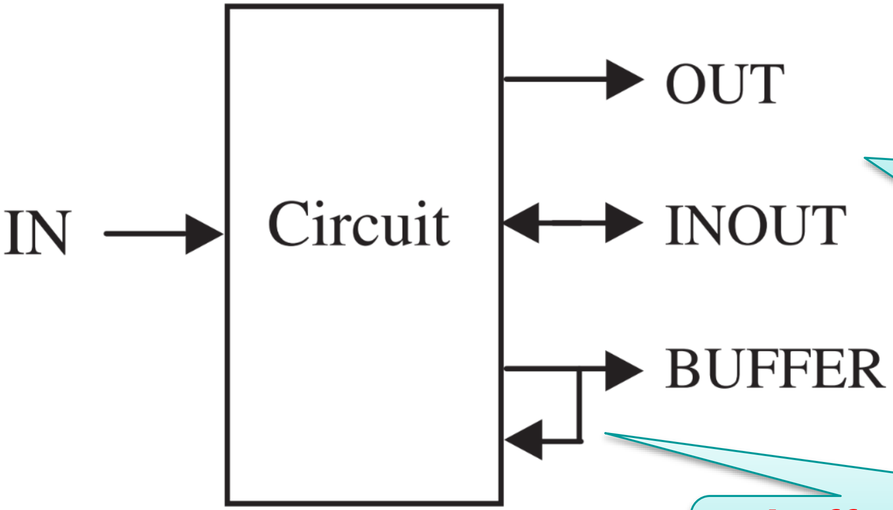
2.4.2 Use of "buffer" mode

```
entity gates is
        port(A, B, C: in bit: D: buffer bit; E: out bit);
end gates;

architecture example of gates is
begin
        D <= A or B after 5 ns;        -- statement 1
        E <= C or D after 5 ns;        -- statement 2
end example
```

- Mode **buffer** indicates a signal that is an output to the external world
- Buffer value can also be read inside the entity's architecture

- Another solution is to use an intermediate internal signal and read from that

2.4.2 Use of "buffer" mode



in and **out** are truly unidirectional pins, while **inout** is bidirectional

buffer is employed when the output signal must be used (read) internally

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		

2.5 Sequential statements and VHDL processes (进程)

- □ **Concurrent statements** are useful in modeling combinational logic
- □ Combinational logic constantly reacts to input changes
- □ In contrast, synchronous sequential logic responds to changes dependent on the clock

- □ Many input changes might be ignored since output and state changes occur only at valid conditions of the clock
- □ Modeling sequential logic requires primitives to model selective activity conditional on clock, edge-triggered devices, sequene of operations, and so on

To implement any clocked circuit (flip-flop, for example) we have to "force" VHDL to be sequential

2.5 Sequential statements and VHDL processes

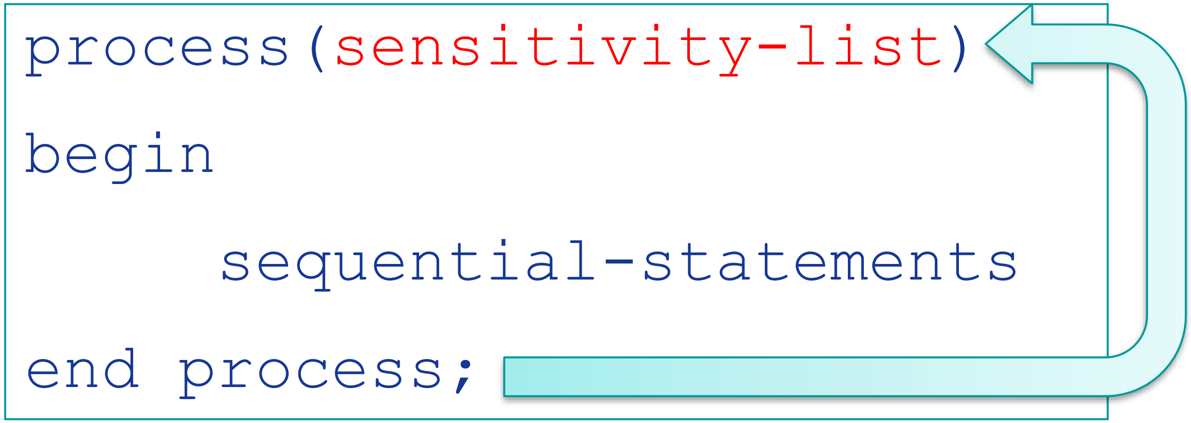
Process (进程) executes whenever any signal in sensitivity list changes

```
process(sensitivity-list)
begin
    sequential-statements
end process;
```

Whenever one of the signals in sensitivity list changes

Sequential statements in process body are executed in sequence one time

2.5 Sequential statements and VHDL processes

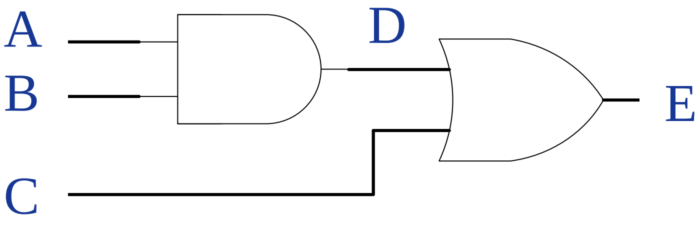


```
process(sensitivity-list)
begin
     sequential-statements
end process;
```

When a process finished executing, it goes back to the beginning and waits for a signal on the sensitivity list to change again

- VHDL code is inherently concurrent. Processes, functions, and procedures are the only sections of code that are executed sequentially
- However, as a whole, any of these blocks is still concurrent with any other statements placed outside it

2.5 Sequential statements and VHDL processes



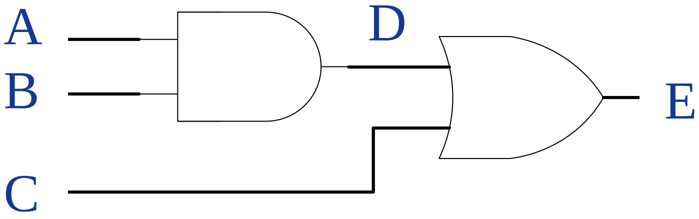
```
entity nogates is
  port( A, B, C   : in       bit;
           D      : buffer   bit;
           E      : out      bit);
end nogates;

architecture behave of nogates is
begin
  process(A, B, C)
  begin
    D <= A and B after 5 ns; -- statement 1
    E <= C or  D after 5 ns; -- statement 2
  end process;
end behave;
```

Process can be used to represent combinational logic

However, be careful!!!

2.5 Sequential statements and VHDL processes

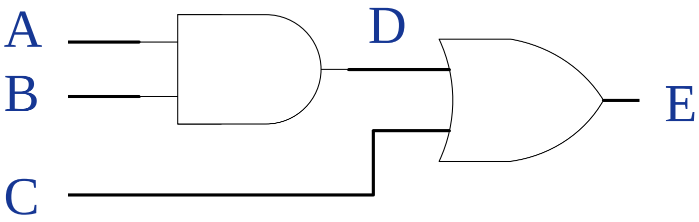
```
entity nogates is
  port( A, B, C   : in        bit;
           D      : buffer    bit;
           E      : out       bit);
end nogates;

architecture behave of nogates is
begin
  process(A, B, C)
  begin
    D <= A and B after 5 ns; -- statement 1
    E <= C or  D after 5 ns; -- statement 2
  end process;
end behave;
```



t(ns)	0	10	15	20	25
A	0				
B	1				
C	0				
D	0				
E	0				

2.5 Sequential statements and VHDL processes

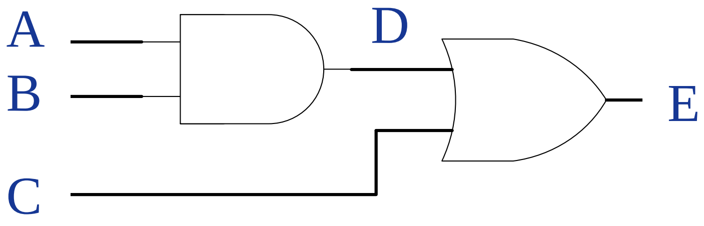
```
entity nogates is
  port( A, B, C   : in        bit;
           D      : buffer    bit;
           E      : out       bit);
end nogates;

architecture behave of nogates is
begin
  process(A, B, C)
  begin
    D <= A and B after 5 ns; -- statement 1
    E <= C or  D after 5 ns; -- statement 2
  end process;
end behave;
```



t(ns)	0	10	15	20	25
A	0 ⇨	1			
B	1	1			
C	0	0			
D	0	0			
E	0	0			

2.5 Sequential statements and VHDL processes

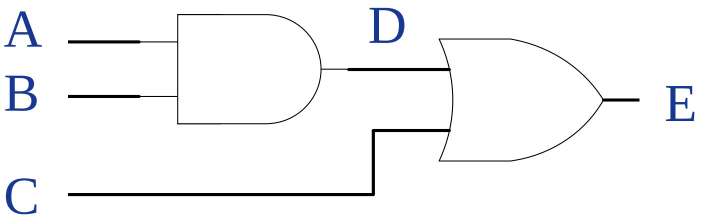
```
entity nogates is
  port( A, B, C  : in        bit;
           D     : buffer    bit;
           E     : out       bit);
end nogates;

architecture behave of nogates is
begin
  process(A, B, C)
  begin
    D <= A and B after 5 ns; -- statement 1
    E <= C or  D after 5 ns; -- statement 2
  end process;
end behave;
```



t	0	10	15	20	25
A	0	1	1		
B	1	1	1		
C	0	0	0		
D	0	0 →	1		
E	0	0	0		

2.5 Sequential statements and VHDL processes

```
entity nogates is
  port( A, B, C  : in       bit;
        D        : buffer   bit;
        E        : out      bit);
end nogates;

architecture behave of nogates is
begin
  process(A, B, C)
  begin
    D <= A and B after 5 ns; -- statement 1
    E <= C or D after 5 ns; -- statement 2
  end process;
end behave;
```



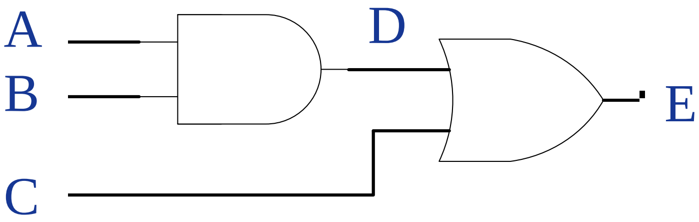
? If D is in the sensitivity list

t	0	10	15	20	25
A	0	1	1	1	1
B	1	1	1	1	1
C	0	0	0	0	0
D	0	0	1	1	1
E	0	0	0	0	0

E stays at '0'

2.5 Sequential statements and VHDL processes

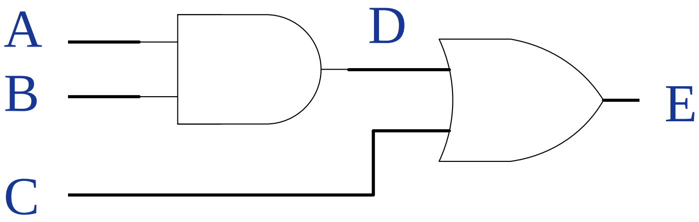
```
entity nogates is
  port( A, B, C   : in        bit;
           D      : buffer    bit;
           E      : out       bit);
end nogates;

architecture behave of nogates is
begin
  process(A, B, C, D)
  begin
    D <= A and B after 5 ns; -- statement 1
    E <= C or  D after 5 ns; -- statement 2
  end process;
end behave;
```



t	0	10	15	20	25
A	0 →	1			
B	1	1			
C	0	0			
D	0	0			
E	0	0			

2.5 Sequential statements and VHDL processes

```
entity nogates is
  port( A, B, C  : in        bit;
           D     : buffer    bit;
           E     : out       bit);
end nogates;

architecture behave of nogates is
begin
  process(A, B, C, D)
  begin
    D <= A or B after 5 ns; -- statement 1
    E <= C or D after 5 ns; -- statement 2
  end process;
end behave;
```



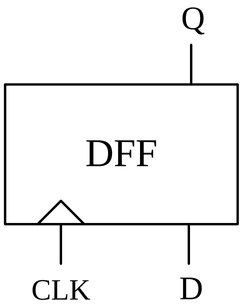
t	0	10	15	20	25
A	0	1	1	1	1
B	1	1	1	1	1
C	0	0	0	0	0
D	0	0	1	1	1
E	0	0	0 ➡	1	1

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 contants
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		

2.6 Modeling flip-flops using VHDL processes

D flip-flop



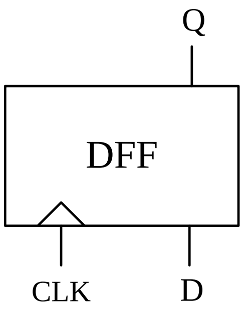
```
process (CLK)
begin
    if  CLK’event and CLK = ‘1’
        then Q <= D;
    end if;
end process;
```

Rising edge of CLK

CLK'event is **TRUE** whenever CLK changes

2.6 Modeling flip-flops using VHDL processes

D flip-flop



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

If VHDL is used only for simulation purposes, one might use a statement such as

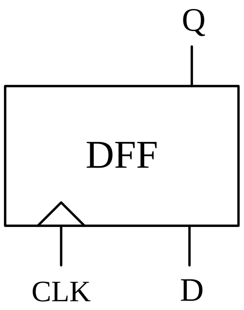
if CLK = '1' ...

and obtain action corresponding to rising edge

When VHDL code is used to synthesize hardware

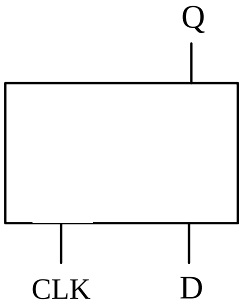
- CLK='1' results in latches
- CLK'event results in edge-triggered devices

2.6 Modeling flip-flops using VHDL processes



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

Flip-flop



```
process (CLK, D)
begin
    if CLK = ‘1’
        then Q <= D;
    end if;
end process;
```

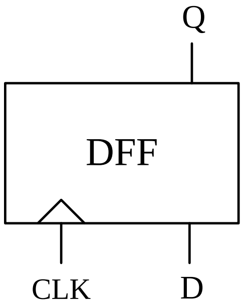
Latch

```
Q <= D;
```

Wire

2.6 Modeling flip-flops using VHDL processes

D flip-flop

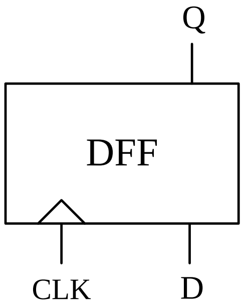


```
process (CLK)
begin
    if  CLK'event and CLK = '1'
          then Q <= D;
    end if;
end process;
```

D is not on the sensitivity list, why?

2.6 Modeling flip-flops using VHDL processes

D flip-flop

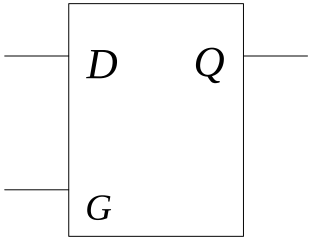


```
process (CLK)
begin
    if  CLK'event and CLK = '1'
        then Q <= D;
    end if;
end process;
```

For D flip-flop, changing D will not cause the flip-flop to change state

2.6 Modeling flip-flops using VHDL processes

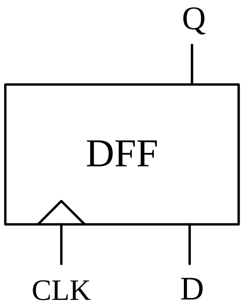
Transparent latch



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

2.6 Modeling flip-flops using VHDL processes

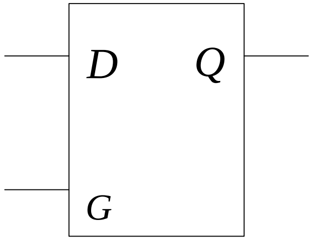
D flip-flop



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

Both G and D are on the sensitivity list, why?

Transparent latch

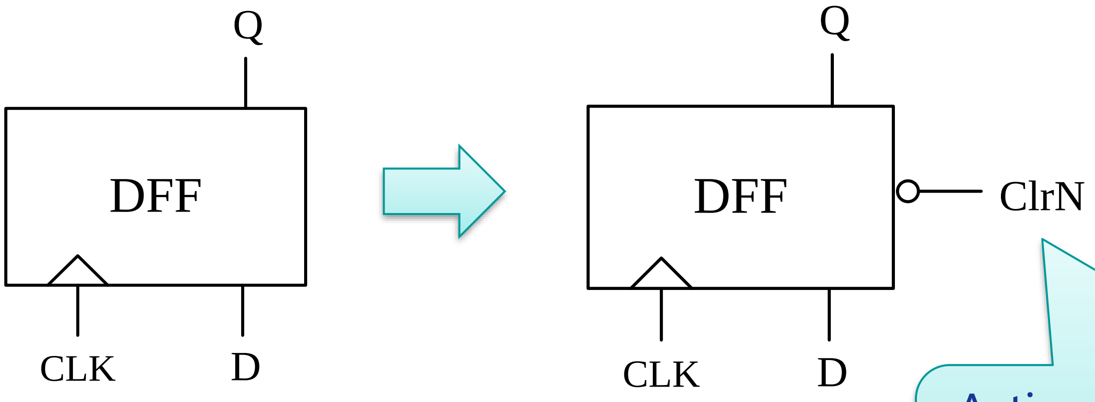


```
process (G, D)
begin
    if G = ‘1’
       then Q <= D;
    end if;
end process;
```

- Since if G = '1', a change in D causes Q to change
- If G changes to '0', the process executes, but Q does not change

2.6 Modeling flip-flops using VHDL processes

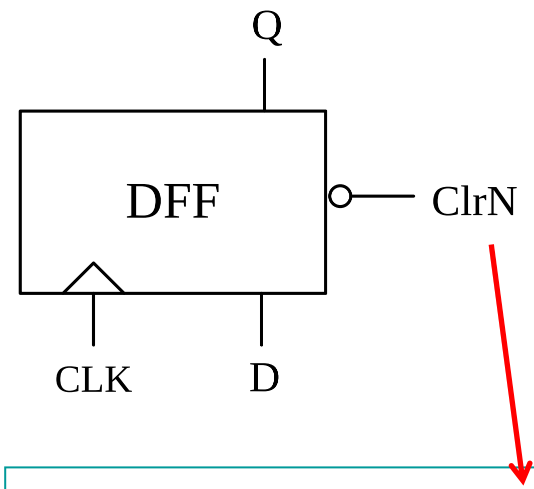
D flip-flop with asynchronous clear



Active-low asynchronous clear input resets the flip-flop independently of the clock

2.6 Modeling flip-flops using VHDL processes

D flip-flop with asynchronous clear



```
process (CLK, ClrN)
begin
    if ClrN = ‘0’ then Q <=‘0’ ;
    else
        if CLK’event and CLK = ‘1’  then Q <= D;
        end if;
    end if;
end process;
```

ClrN overrides CLK

2.6 Modeling flip-flops using VHDL processes

If statement

```
if condition then
      sequential statements1
else  sequential statements2
end if;
```

The condition is a Boolean expression which evaluates to TRUE or FALSE

if statements cannot be used as concurrent statements outside of a process

Nested ifs

```
if condition1 then
      sequential statements1
else  if condition2 then
            sequential statements2
      else
            sequential statements3
      end if;
end if;
```

2.6 Modeling flip-flops using VHDL processes

If statement

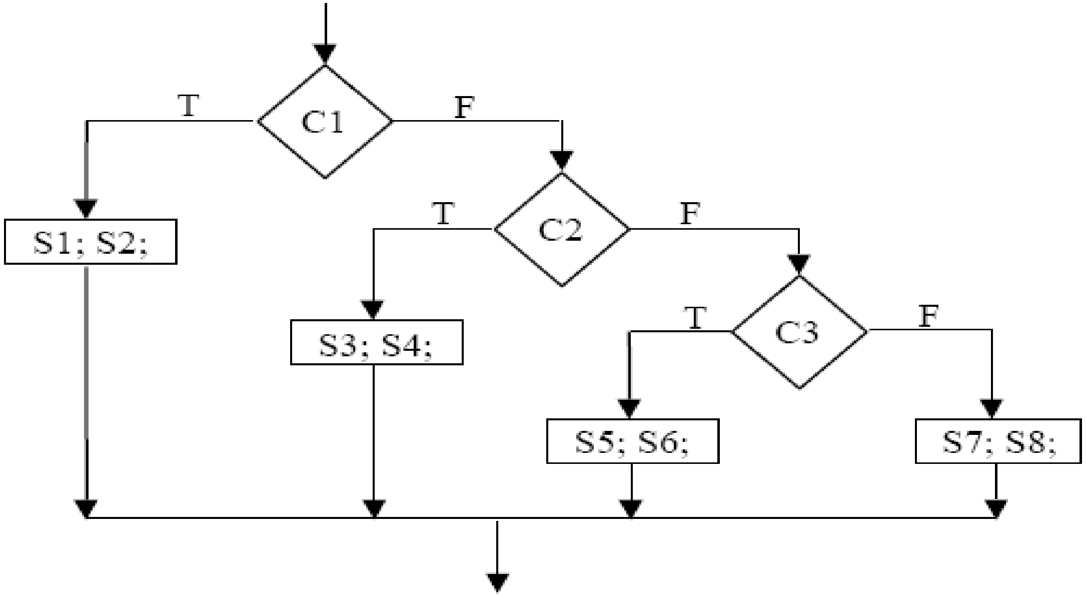
```
if condition then
      sequential statements
{elsif condition then
      sequential statements}
[else sequential statements]
end if;
```

Any number of **elsif** clauses may be included

else clause is optional

2.6 Modeling flip-flops using VHDL processes

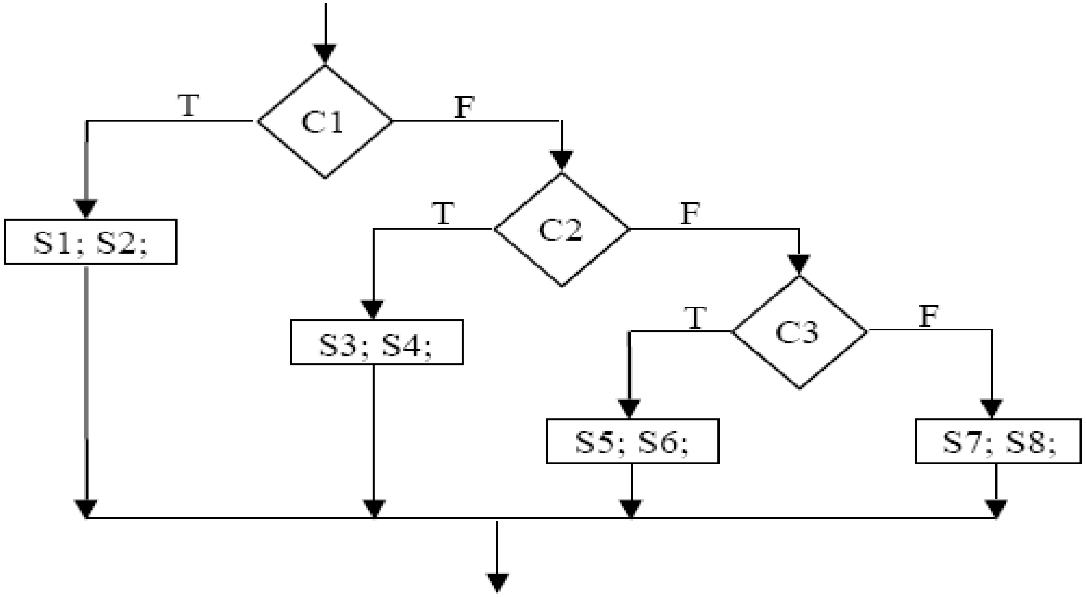
Nested ifs and elsifs



```
if (C1) then S1; S2;
  else if (C2) then S3; S4;
         else if (C3) then S5; S6;
                 else S7; S8;
             end if;
       end if;
end if;
```

2.6 Modeling flip-flops using VHDL processes

Nested ifs and elsifs

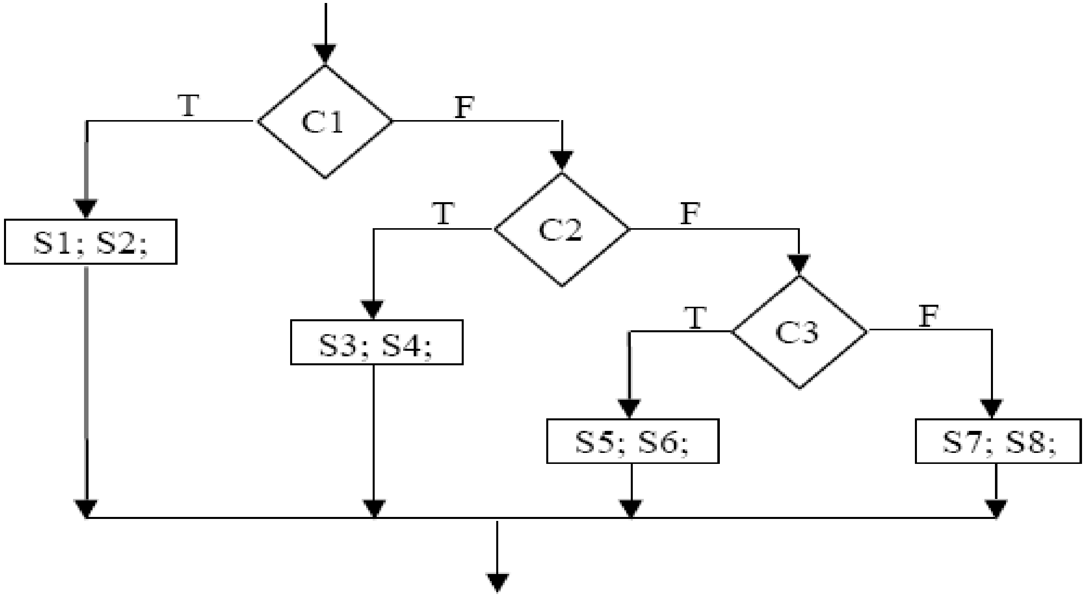


```
if (C1) then S1; S2;
  else if (C2) then S3; S4;
        else if (C3) then S5; S6;
                else S7; S8;
            end if;
      end if;
end if;
```

```
if (C1) then S1; S2;
  elsif (C2) then S3; S4;
  elsif (C3) then S5; S6;
  else S7; S8;
end if;
```

2.6 Modeling flip-flops using VHDL processes

Nested ifs and elsifs



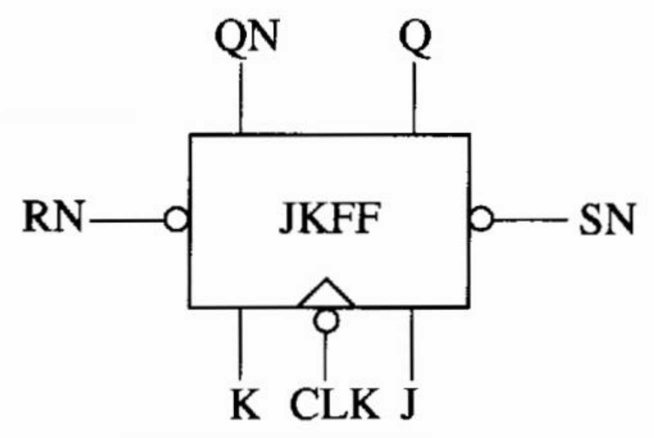
```
if (C1) then S1; S2;
 else if (C2) then S3; S4;
     else if (C3) then S5; S6;
          else S7; S8;
       end if;
    end if;
end if;
```

```
if (C1) then S1; S2;
   elsif (C2) then S3; S4;
   elsif (C3) then S5; S6;
   else S7; S8;
end if;
```

Each **if** requires a corresponding **end if**, but **elsif**s do not

2.6 Modeling flip-flops using VHDL processes

J-K flip-flop

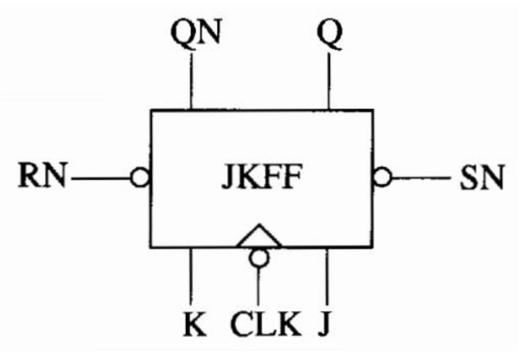


SN : active-low asynchronous preset
RN : active-low asynchronous clear

For simplicity, we will assume that the condition SN = RN = 0 does not occur

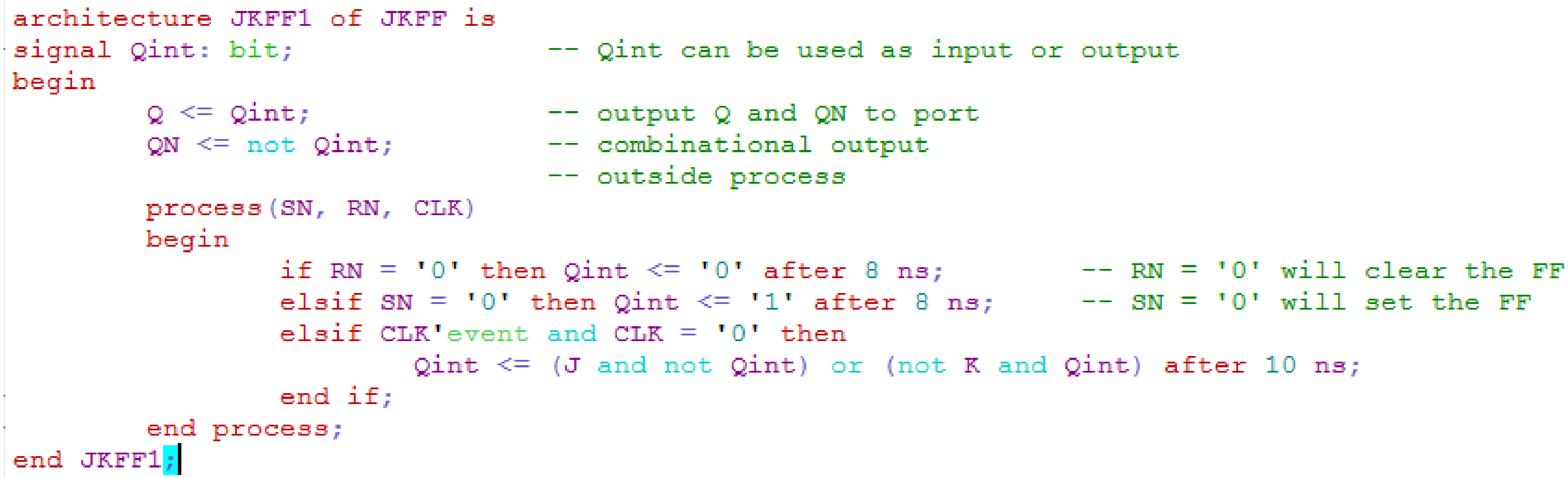
```
entity JKFF is
        port(   SN, RN, J, K, CLK: in bit;                  -- inputs
                Q, QN: out bit);
end JKFF;
```

J-K flip-flop



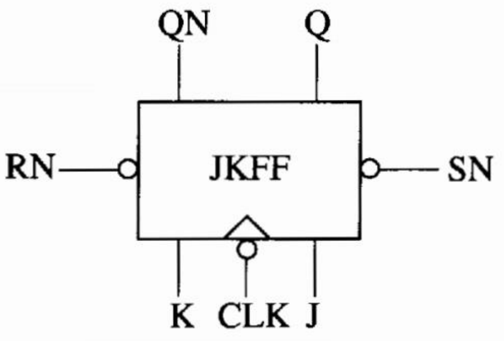
J	K	Q+
0	0	Q
0	1	0
1	0	1
1	1	Q'

$$Q^+ = JQ'+K'Q$$



```
architecture JKFF1 of JKFF is
signal Qint: bit;               -- Qint can be used as input or output
begin
        Q <= Qint;              -- output Q and QN to port
        QN <= not Qint;         -- combinational output
                                -- outside process
        process(SN, RN, CLK)
        begin
                if RN = '0' then Qint <= '0' after 8 ns;        -- RN = '0' will clear the FF
                elsif SN = '0' then Qint <= '1' after 8 ns;     -- SN = '0' will set the FF
                elsif CLK'event and CLK = '0' then
                        Qint <= (J and not Qint) or (not K and Qint) after 10 ns;
                end if;
        end process;
end JKFF1;
```

J-K flip-flop

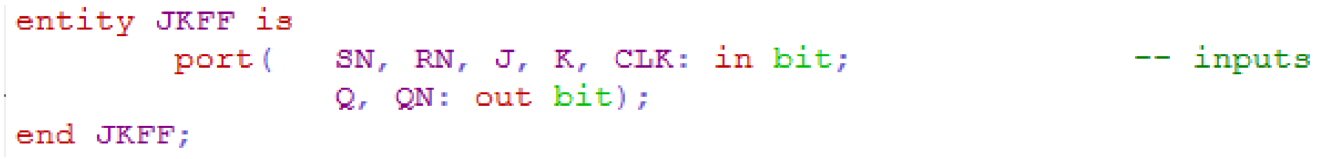


```
architecture JKFF1 of JKFF is
signal Qint: bit;               -- Qint can be used as input or output
begin
        Q <= Qint;              -- output Q and QN to port
        QN <= not Qint;         -- combinational output
                                -- outside process
        process(SN, RN, CLK)
        begin
                if RN = '0' then Qint <= '0' after 8 ns;        -- RN = '0' will clear the FF
                elsif SN = '0' then Qint <= '1' after 8 ns;     -- SN = '0' will set the FF
                elsif CLK'event and CLK = '0' then
                        Qint <= (J and not Qint) or (not K and Qint) after 10 ns;
                end if;
        end process;
end JKFF1;
```

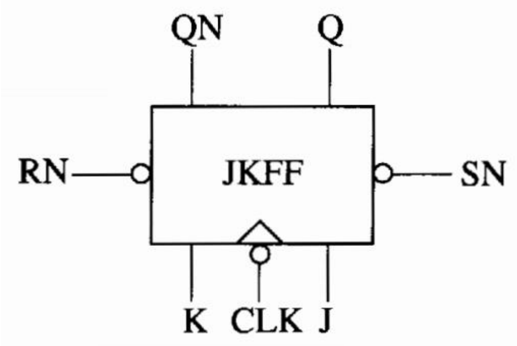
- Within architecture we define a signal **Qint** that represents the state of the flip-flop internal to the module

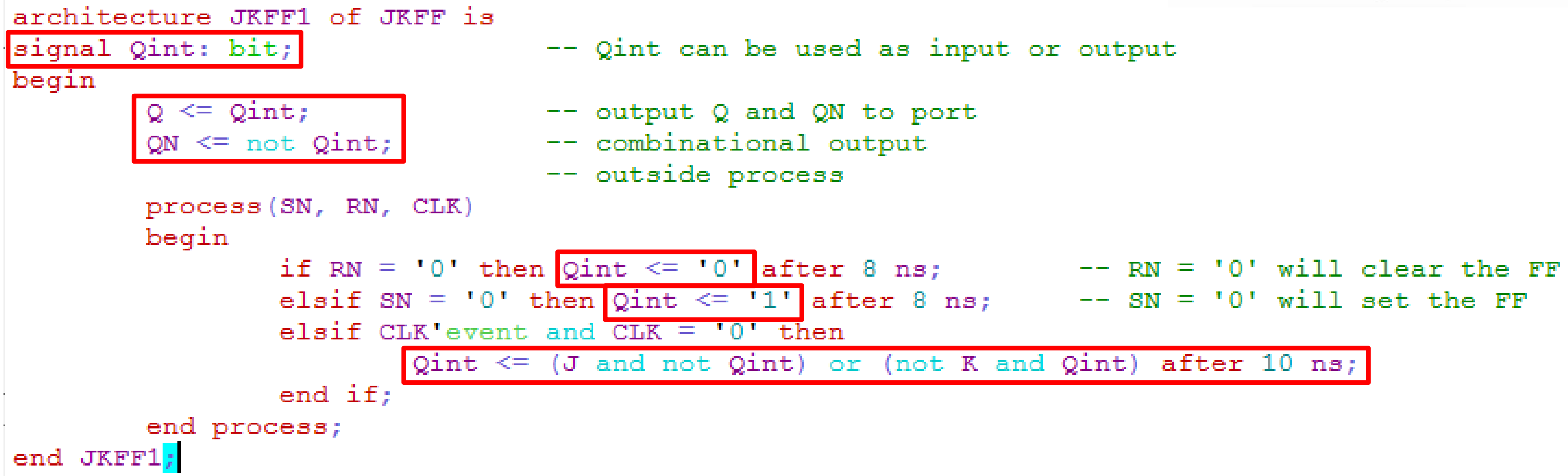
- Two concurrent statements transmit Qint to the **Q** and **QN** ouput of the flip-flop, WHY?

J-K flip-flop



```vhdl
entity JKFF is
        port(   SN, RN, J, K, CLK: in bit;              -- inputs
                Q, QN: out bit);
end JKFF;
```





```vhdl
architecture JKFF1 of JKFF is
signal Qint: bit;                   -- Qint can be used as input or output
begin
        Q <= Qint;                  -- output Q and QN to port
        QN <= not Qint;             -- combinational output
                                    -- outside process
        process(SN, RN, CLK)
        begin
                if RN = '0' then Qint <= '0' after 8 ns;         -- RN = '0' will clear the FF
                elsif SN = '0' then Qint <= '1' after 8 ns;      -- SN = '0' will set the FF
                elsif CLK'event and CLK = '0' then
                       Qint <= (J and not Qint) or (not K and Qint) after 10 ns;
                end if;
        end process;
end JKFF1;
```

We do it this way because an output signal in a port cannot appear on the right side of an assignment statement within the architecture

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		

2.7 Processes using wait statements

```
process
begin
    sequential-statements
    wait-statement
    sequential-statements
    wait-statement
    ...
end process;
```

An alternative form for a process uses **wait** statements instead of a sensitivity list

The process will wait until the specified wait condition is satisfied

Forms of wait statements

wait on sensitivity-list;

waits until one of the signals on the sensitivity-list changes

wait for time-expression;

waits until the time specified by the time-expression has lapsed

Not synthesizable

If **wait for 0 ns** is used, the wait for one delta time

wait until Boolean-expression;

Boolean-expression is evaluated whenever one of the signal in the expression changes

The process continues execution when the expression evaluates to TRUE

2.7 Processes using wait statements

```
process  (A, B, C, D)
begin
   C<= A and B after 5ns;
   E<= C or B after 5ns;
end process;
```

=

```
process
begin
   C<= A and B after 5ns;
   E<= C or B after 5ns;
   wait on A,B,C,D;
end process;
```

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

2.7 Processes using wait statements

```
process
begin
   wait until clk'event and clk='1';
      A <= E after 10ns;          -- (1)
      B <= F after 5ns;           -- (2)
      C <= G;                     -- (3)
      D <= H after 5ns;           -- (4)
end process;
```

The order in which sequential statements execute in a process is not necessarily the order in which the signal are updated

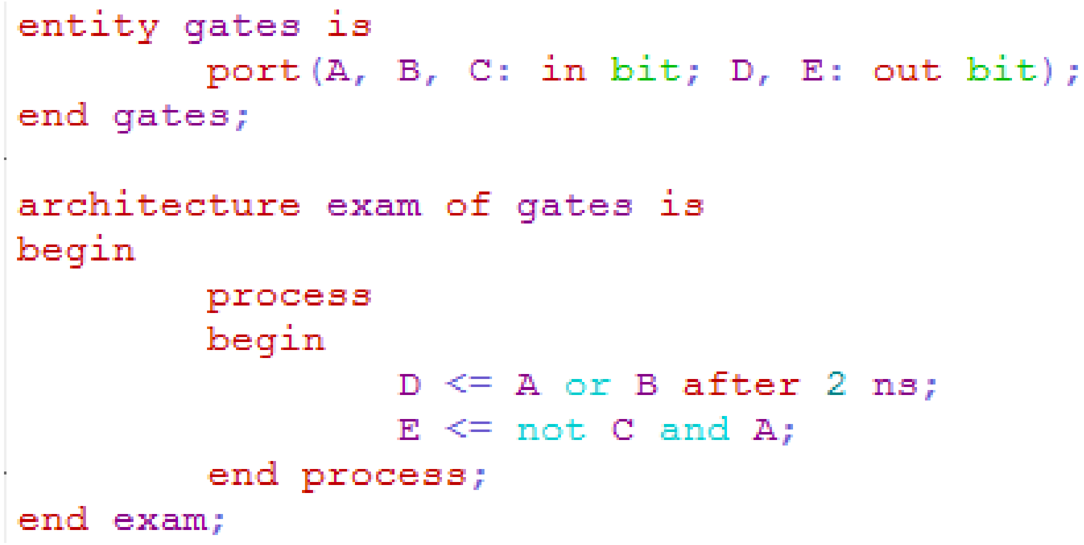
2.7 Processes using wait statements

```
process(CLK)
begin
   if CLK'event and CLK='0' then
      Q <= A;
      Q <= B;
      Q <= C;
   end if;
end process;
```

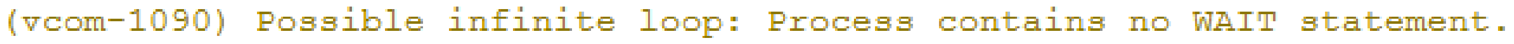
Every time CLK changes from '1' to '0', after delta time, Q will change to C

If several VHDL statements in a process update the same signal at a given time, the last value overrides

2.7 Processes using wait statements



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

architecture exam of gates is
begin
        process
        begin
                D <= A or B after 2 ns;
                E <= not C and A;
        end process;
end exam;
```



```
(vcom-1090) Possible infinite loop: Process contains no WAIT statement.
```

A process must have either a sensitivity list or wait statements

A process cannot have both wait statements and a sensitivity list