CDA 3200 Digital Systems

Instructor: Dr. Janusz Zalewski

Developed by: Dr. Dahai Guo

Spring 2012

Outline

- Overview of VHDL
- VHDL Description of Combinational Circuits
- VHDL Models for Multiplexers
- VHDL Modules
- Signals and Constants
- Arrays
- VHDL Operators
- Packages and Libraries

Overview of VHDL (1/6)

- VHDL
  - VHSIC Hardware Description Language
  - VHSIC: Very High Speed Integrated Circuit
  - Widely used in the industry

Overview of VHDL (2/6)

- Why VHDL?
  - Detailed design of the systems at the gate level has become very tedious and time consuming.
  - A hardware description language of top-down methodology is desired.

Overview of VHDL (3/6)

- The program that designs a 3-bit adder

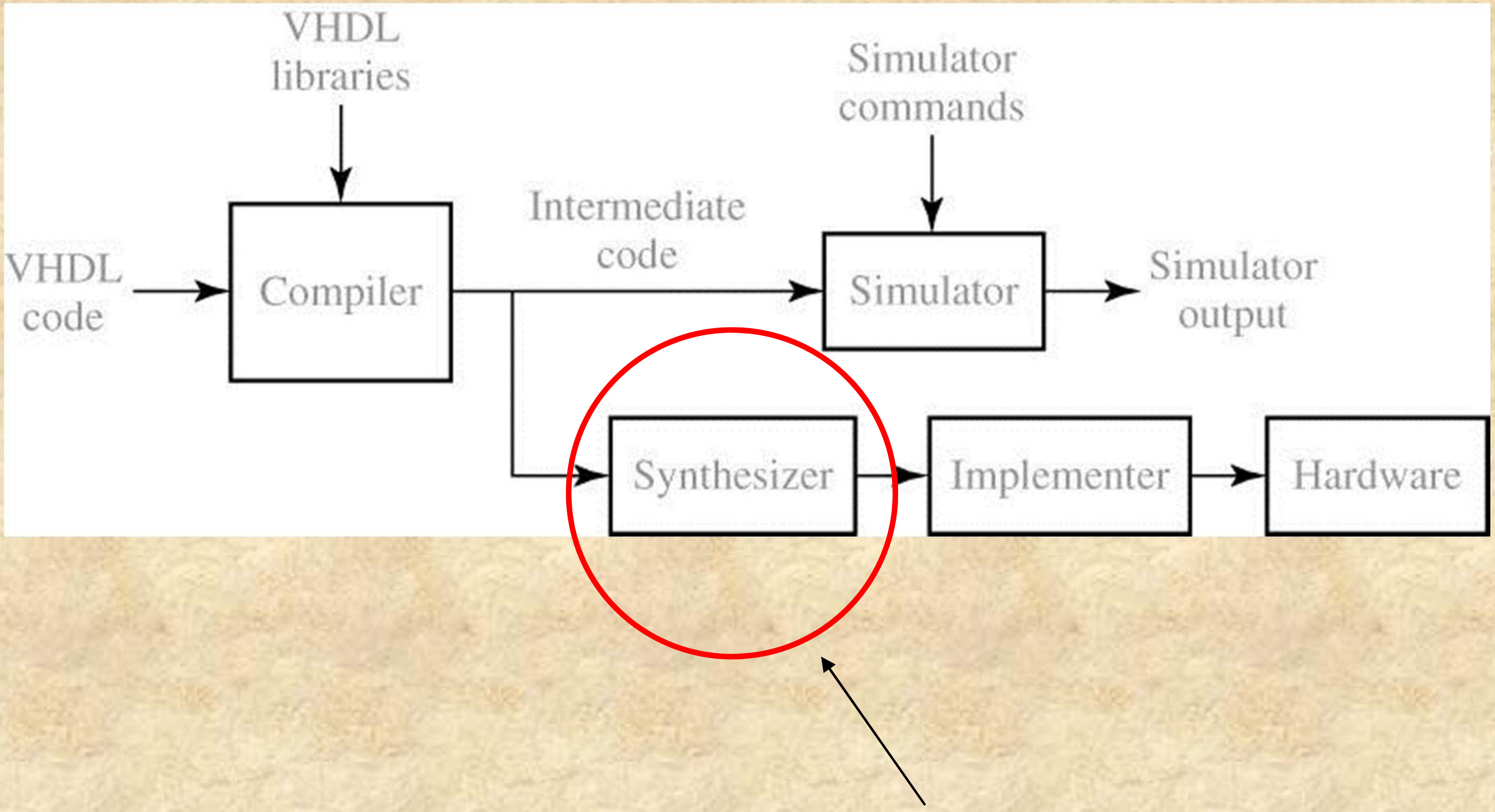
```
entity ADD is
   port (A, B  : in integer range 0 to 7;
           Z       : out integer range 0 to 15);
end ADD;

architecture ARITHMETIC of ADD is
begin
    Z <= A + B;
end ARITHMETIC;
```

Overview of VHDL (4/6)

- It seems that Boolean algebra, Karnaugh-Maps, and other techniqes (Quine-McCluskey and Petrick’s methods) are useless, if VHDL is so powerful.

Overview of VHDL (5/6)



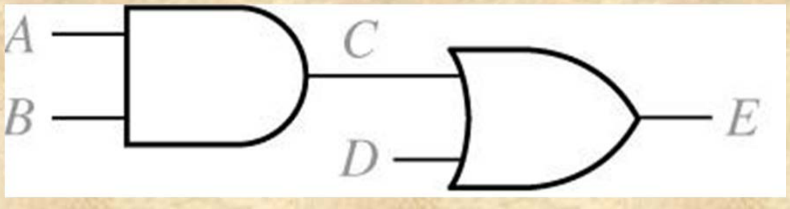
Where Boolean algebra, Karnaugh maps and other methods are integrated.

Overview of VHDL (6/6)

- Because the simplifying methods are systematic, they can be programmed into the synthesizer.
- Therefore, hardware designers can be freed of detailed work at the gate level.
- But the designers still can work at the gate level in VHDL.

VHDL Description of Combinational Circuits (1/6)

- Signal assignment operator
  1. C **<=** A and B ***after*** 5 ns;
  2. E **<=** C or D ***after*** 5 ns;



- Logically equivalent?
  1. E **<=** C or D ***after*** 5 ns;
  2. C **<=** A and B ***after*** 5 ns;

VHDL Description of Combinational Circuits (2/6)

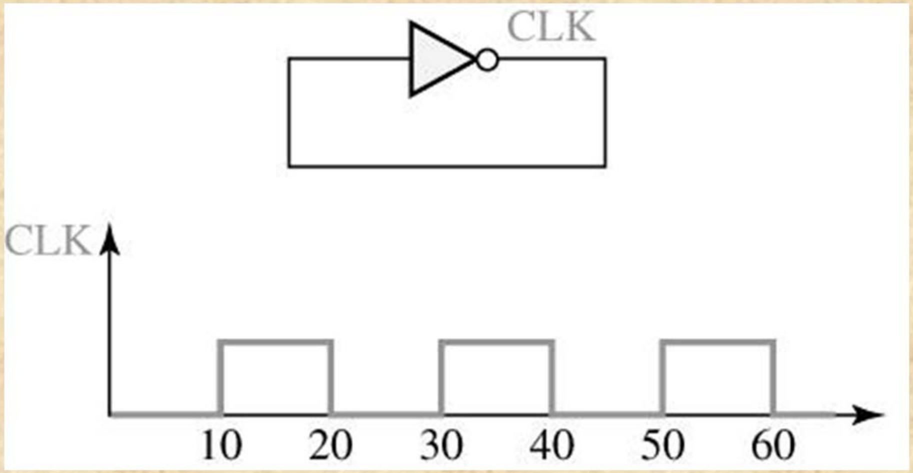
- VHDL signal assignment statements are concurrent statements.
  - The VHDL simulator monitors the right side of each concurrent statement;
  - Any time a signal changes, the expression on the right side is immediately re-evaluated;
  - The new value is assigned to the signal on the left side after the delay.
  - The order of the concurrent statements does not matter.

VHDL Description of Combinational Circuits (3/6)

- The delay is optional in assignment statements.
  - E <= C or D;
  - C <= A and B;
  - When no delay is specified, the simulator will assume an infinitesimal delay referred to as Δ (delta).

VHDL Description of Combinational Circuits (4/6)

- Clock
  - CLK <= not CLK after 10 ns;



No, because a run-time error during simulation will be caused.

Can we remove the delay?

VHDL Description of Combinational Circuits (5/6)

- Bit vectors
  - B <= “0110”;
  - A <= “1100”;
  - C <= A and B; $\leftrightarrow$

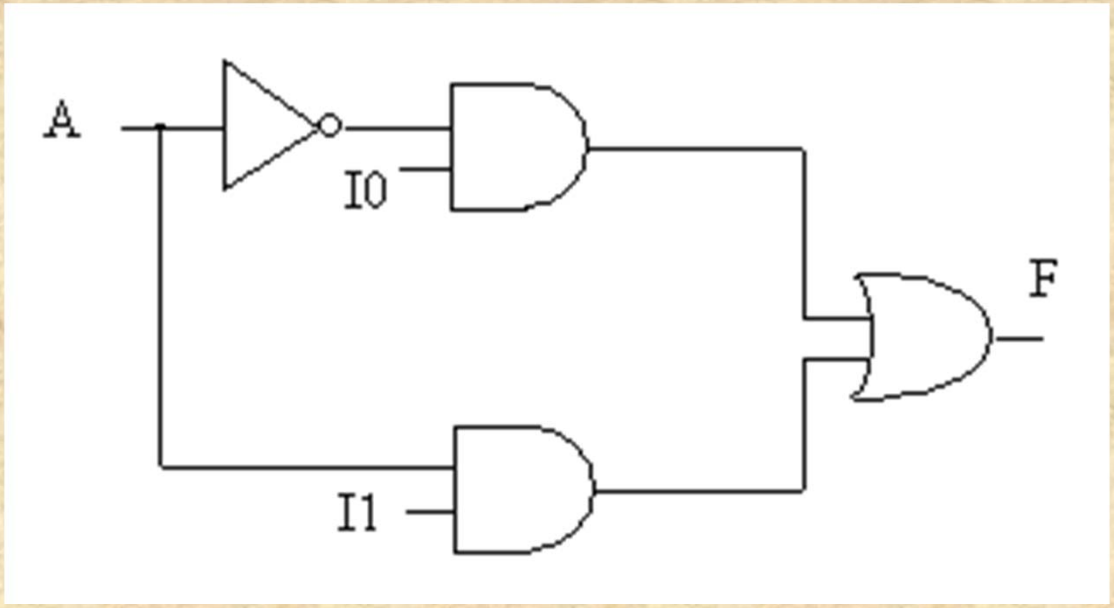
```
C(3) <= A(3) and B(3);
C(2) <= A(2) and B(2);
C(1) <= A(1) and B(1);
C(0) <= A(0) and B(0);
```

VHDL Description of Combinational Circuits (6/6)

- Comments are preceded by --
- An identifier, such as a variable, must start with a letter, and it cannot end with an underscore.
- Each statement is ended with a semicolon.
- NOT CASE SENSITIVE

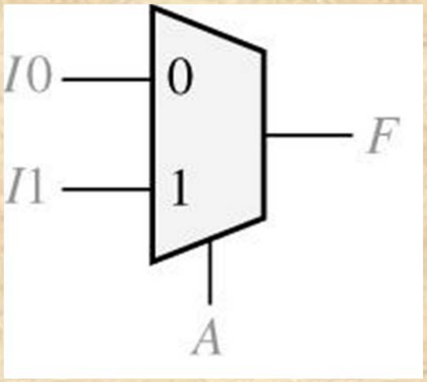
VHDL Models for Multiplexers (1/6)

- 2-to-1 multiplexer
  - F <= (not A and I0) or (A and I1);

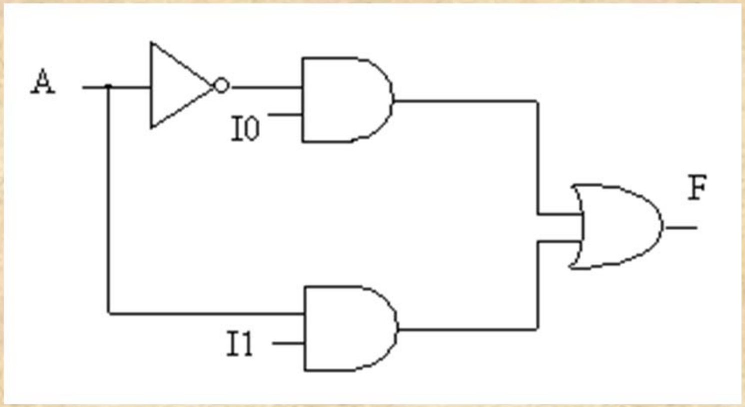


VHDL Models for Multiplexers (2/6)

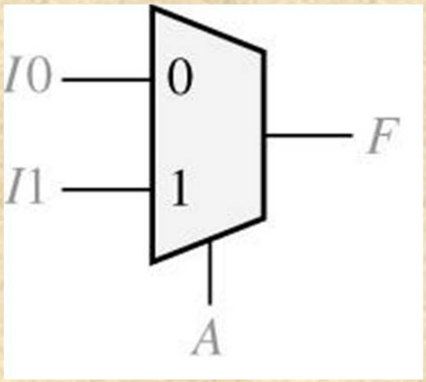
- 2-to-1 (cont)
  1. -- conditional signal assignment statement
  2. F <= I0 when A='0' else I1;



VHDL Models for Multiplexers (3/6)



Design at the gate level.



Design at the logic level.

VHDL Models for Multiplexers (4/6)

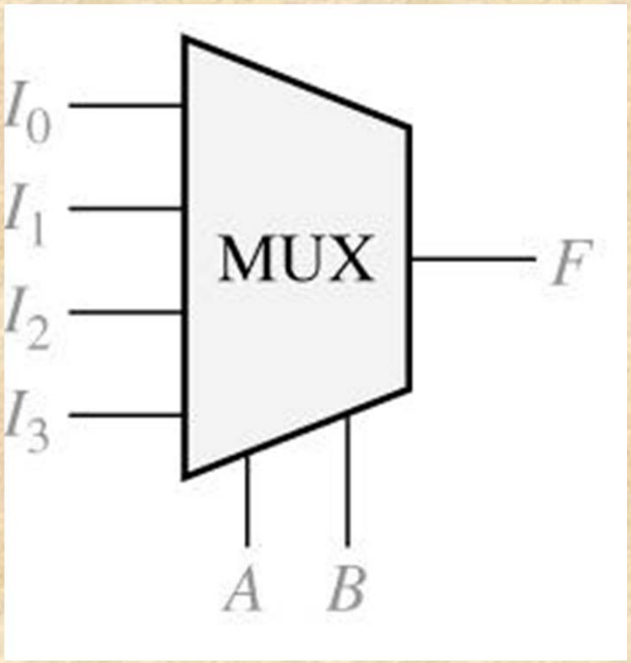
- The general form of a conditional signal assignment statement is

```
1.  signal_name <= expression1 when condition1
2.                 else expression2 when condition2
3.                          ……
4.                 [else expressionN];
```

  – This concurrent statement is executed whenever a change occurs in a signal used in the selected expression or its condition.

VHDL Models for Multiplexers (5/6)

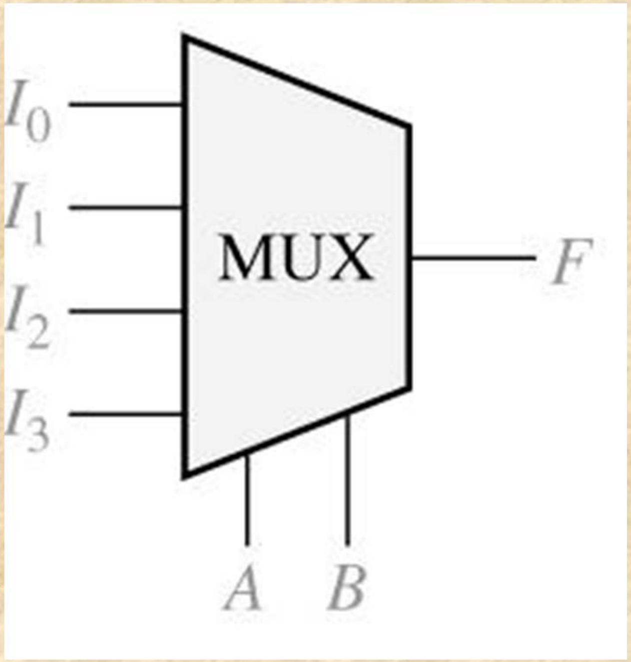
- 4-to-1



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

VHDL Models for Multiplexers (6/6)

- 4-to-1



```
sel <= A&B;
with sel select
    F <= I0 when "00",
         I1 when "01",
         I2 when "10",
         I3 when "11";
```

VHDL Modules (1/17)

- Entities and architectures
  - An entity declares the interface between a module and its environment. What are the input(s) and output(s)?
  - The architecture contains the implementation for an entity.

VHDL Modules (2/17)

- An entity does not deal with design details.
  - Example:
  - entity two_gates is
  - port (A, B, D: $\boxed{\text{in}}$ bit; E: $\boxed{\text{out}}$ bit);
  - end two_gates;

•Key words that are used to define an entity.
•The name of the entity.
•The input and output list. ***port*** is a key word that is used to declare input/output
•A, B, D are the input signals and E is the output signal.
•Be careful where you need semicolons, commas, and colons.

VHDL Modules (3/17)

- Associated with each entity is one or more architecture declarations
  - architecture gates of two_gates is
  -     signal C: bit;
  - begin
  -     C <= A and B; -- concurrent
  -     E <= C or D;   -- statements
  - end gates;

•Key words used to define an architecture.
•Every architecture must be associated with an entity.
•The architecture's name.
•Signal used within this architecture.
•Design details

VHDL Modules (4/17)

- Full adder
  - entity FullAdder is
  - port(X, Y, Cin: in bit; -- inputs
  - Cout, Sum: out bit); -- outputs
  - end FullAdder;

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

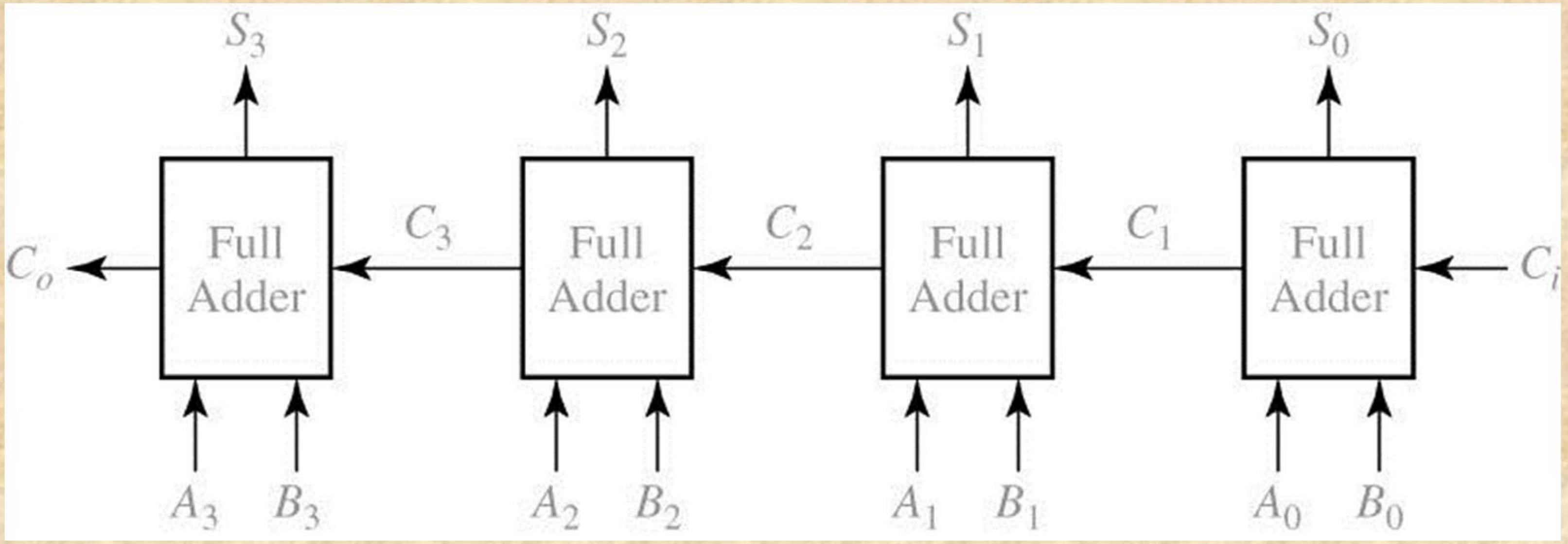
VHDL Modules (5/17)

- Full adder
  - architecture Equations of ***FullAdder*** is
  - begin
  - Sum <= X xor Y xor Cin after 10 ns;
  - Cout <= $\underline{\text{(X and Y) or (X and Cin) or (Y and Cin)}}$ after 10 ns;
  - end Equations;

VHDL does not specify an order of precedence for the logic operators, therefore parentheses are required.

VHDL Modules (6/17)

- 4-bit fuller adder



```
1.  entity Adder4 is
2.      port (A, B: in bit_vector(3 downto 0); Ci: in bit  -- Inputs
3.            S: out bit_vector(3 downto 0); Co: out bit)  -- Outputs
4.  Adder4
5.  architecture Structure of adder4 is
6.  begin

1.    S(0) <= A(0) xor B(0) xor Ci(0) after 10 ns;
2.    Co(0) <= (A(0) and B(0)) or (A(0) and Ci(0)) or (B(0) and Ci(0)) after 10 ns;
```

Find all the syntax errors in lines 1-6 and 15.

```
3.    S(1) <= A(1) xor B(1) xor Ci(1) after 10 ns;
4.    Co(1) <= (A(1) and B(1)) or (A(1) and Ci(1)) or (B(1) and Ci(1)) after 10 ns;

5.    S(2) <= A(2) xor B(2) xor Ci(2) after 10 ns;
6.    Co(2) <= (A(2) and B(2)) or (A(2) and Ci(2)) or (B(2) and Ci(2)) after 10 ns;

7.    S(3) <= A(3) xor B(3) xor Ci(3) after 10 ns;
8.    Co(3) <= (A(3) and B(3)) or (A(3) and Ci(3)) or (B(3) and Ci(3)) after 10 ns;

9.  Structure
```

```
1.  entity Adder4 is
2.      port (A, B: in bit_vector(3 downto 0); Ci: in bit;  -- Inputs
3.            S: out bit_vector(3 downto 0); Co: out bit);  -- Outputs
4.  end Adder4;
5.  architecture Structure of adder4 is
6.  begin

7.    S(0) <= A(0) xor B(0) xor Ci(0) after 10 ns;
8.    Co(0) <= (A(0) and B(0)) or (A(0) and Ci(0)) or (B(0) and Ci(0)) after 10 ns;
```

Find all the syntax errors in lines 1-6 and 15.

```
9.    S(1) <= A(1) xor B(1) xor Ci(1) after 10 ns;
10.   Co(1) <= (A(1) and B(1)) or (A(1) and Ci(1)) or (B(1) and Ci(1)) after 10 ns;

11.   S(2) <= A(2) xor B(2) xor Ci(2) after 10 ns;
12.   Co(2) <= (A(2) and B(2)) or (A(2) and Ci(2)) or (B(2) and Ci(2)) after 10 ns;

13.   S(3) <= A(3) xor B(3) xor Ci(3) after 10 ns;
14.   Co(3) <= (A(3) and B(3)) or (A(3) and Ci(3)) or (B(3) and Ci(3)) after 10 ns;

15.end Structure;
```

```vhdl
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
begin
```

In this paragraph?

```vhdl
S(0) <= A(0) xor B(0) xor Ci(0) after 10 ns;
Co(0) <= (A(0) and B(0)) or (A(0) and Ci(0)) or (B(0) and Ci(0)) after 10 ns;

S(1) <= A(1) xor B(1) xor Ci(1) after 10 ns;
Co(1) <= (A(1) and B(1)) or (A(1) and Ci(1)) or (B(1) and Ci(1)) after 10 ns;

S(2) <= A(2) xor B(2) xor Ci(2) after 10 ns;
Co(2) <= (A(2) and B(2)) or (A(2) and Ci(2)) or (B(2) and Ci(2)) after 10 ns;

S(3) <= A(3) xor B(3) xor Ci(3) after 10 ns;
Co(3) <= (A(3) and B(3)) or (A(3) and Ci(3)) or (B(3) and Ci(3)) after 10 ns;

end Structure;
```

VHDL Modules (10/17)

- 4-bit full adder
  1. S(0) <= A(0) xor B(0) xor Ci(0) after 10 ns;
  2. Co(0) <= (A(0) and B(0)) or (A(0) and Ci(0)) or (B(0) and Ci(0)) after 10 ns;
  - Ci and Co are defined as bit signals and cannot work with indices.

VHDL Modules (11/17)

- 4-bit full adder
  - architecture Structure of Adder4 is
  - <u>signal C:bit_vector(2 downto 0);</u>
  - begin
  - end Structure;

VHDL Modules (12/17)

- S(0) <= A(0) xor B(0) xor **Ci** after 10 ns;
- C(0) <= (A(0) and B(0)) or (A(0) and **Ci**) or (B(0) and **Ci**) after 10 ns;

- S(1) <= A(1) xor B(1) xor C(0) after 10 ns;
- C(1) <= (A(1) and B(1)) or (A(1) and C(0)) or (B(1) and C(0)) after 10 ns;

- S(2) <= A(2) xor B(2) xor C(1) after 10 ns;
- C(2) <= (A(2) and B(2)) or (A(2) and C(1)) or (B(2) and C(1)) after 10 ns;

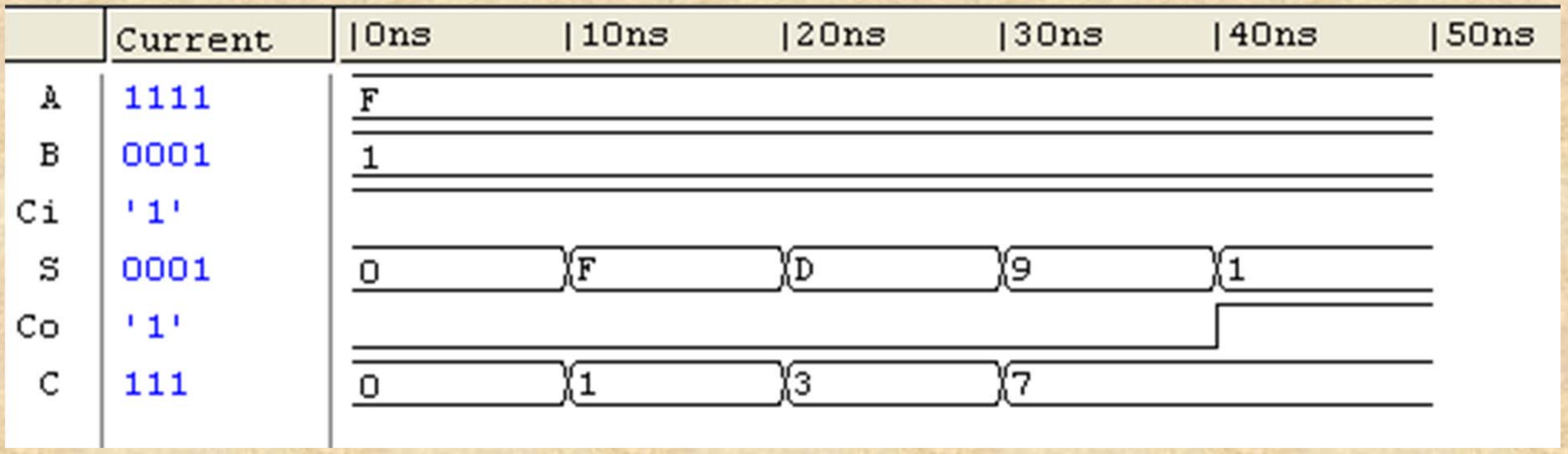
- S(3) <= A(3) xor B(3) xor C(2) after 10 ns;
- **Co** <= (A(3) and B(3)) or (A(3) and C(2)) or (B(3) and C(2)) after 10 ns;

VHDL Modules (13/17)

- 4-bit full adder
  - Simulation commands
    - *list A B Co C Ci S* -- *put these signals 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*

VHDL Modules (14/17)

- 4-bit full adder
  - Simulation output



VHDL Modules (15/17)

- Since a 4-bit full adder is a sequence of 1-bit full adders, why do not we build the 4-bit adder based on 1-bit adders in VHDL?
  - -- specify the FullAdder as a component
  - -- within the architecture of Adder4
  - architecture Structure of *Adder4* is
  - ***component FullAdder***
  - port (X, Y, Cin: in bit; -- Inputs
  - Cout, Sum: out bit); -- Outputs
  - ***end component;***
  - *-- other code*
  - end Structure;

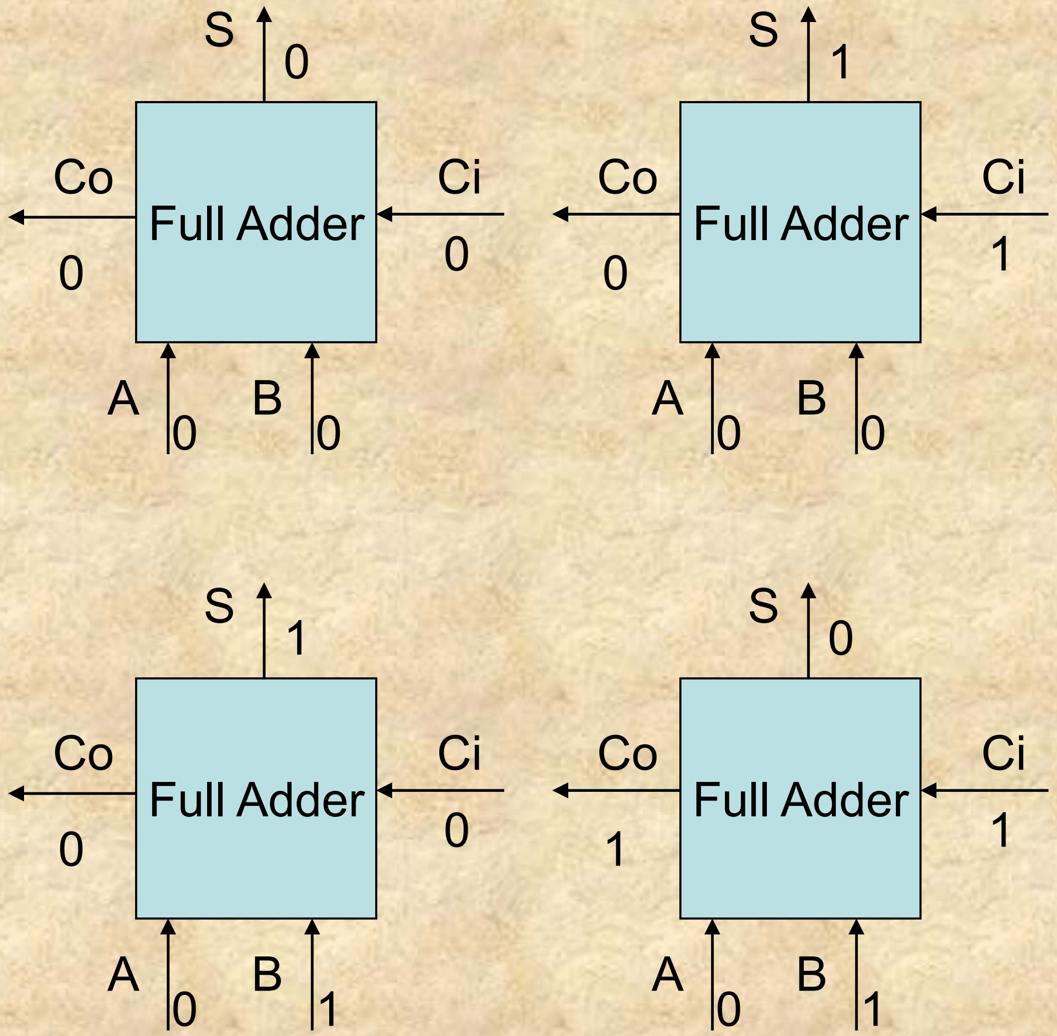
VHDL Modules (16/17)

- *-- other code*
- *signal C: bit_vector(3 downto 1);*
- *begin   --instantiate four copies of the FullAdder*
- *FA0: FullAdder port map (A(0), B(0), Ci, C(0), S(0));*
- *FA1: FullAdder port map (A(1), B(1), C(0), C(1), S(1));*
- *FA2: FullAdder port map (A(2), B(2), C(1), C(2), S(2));*
- *FA3: FullAdder port map (A(3), B(3), C(2), Co, S(3));*

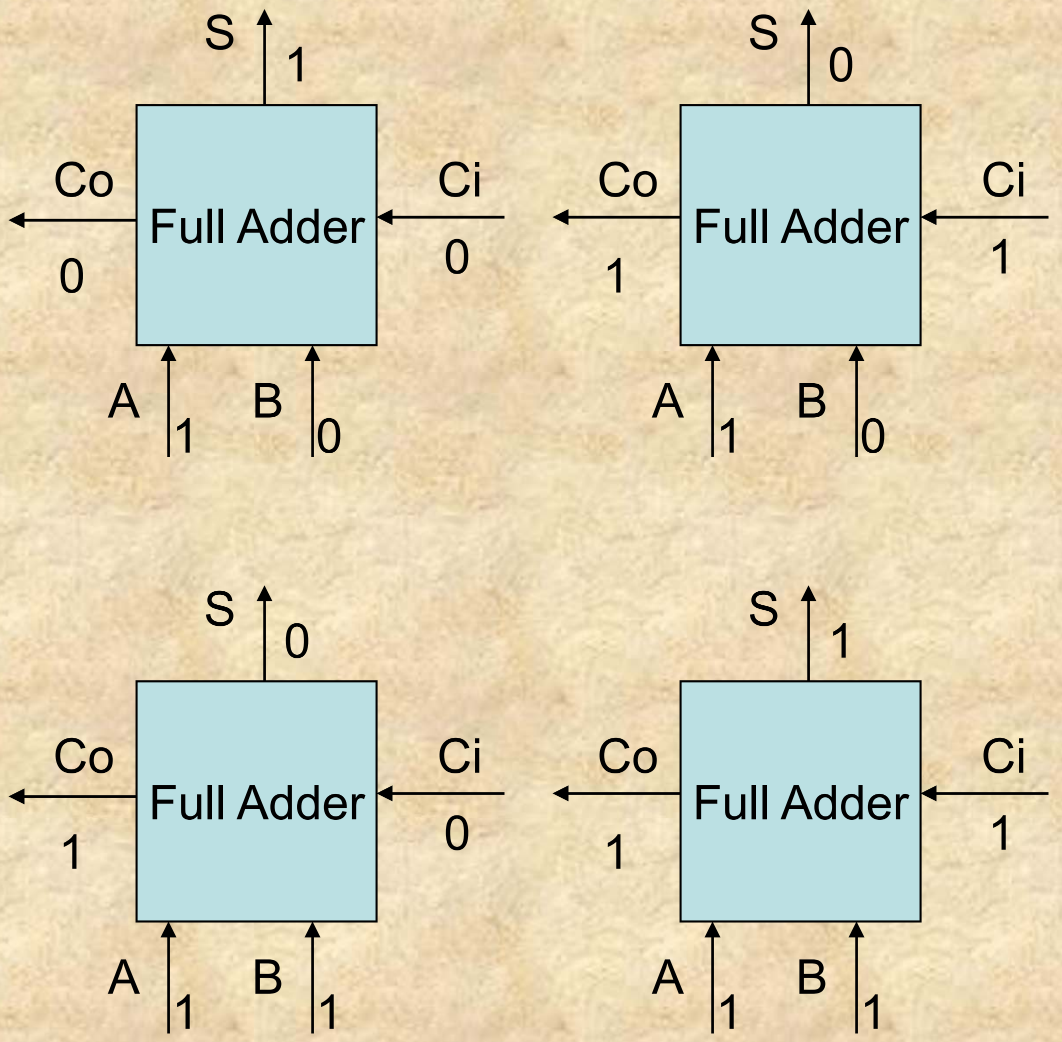
VHDL Modules (17/17)

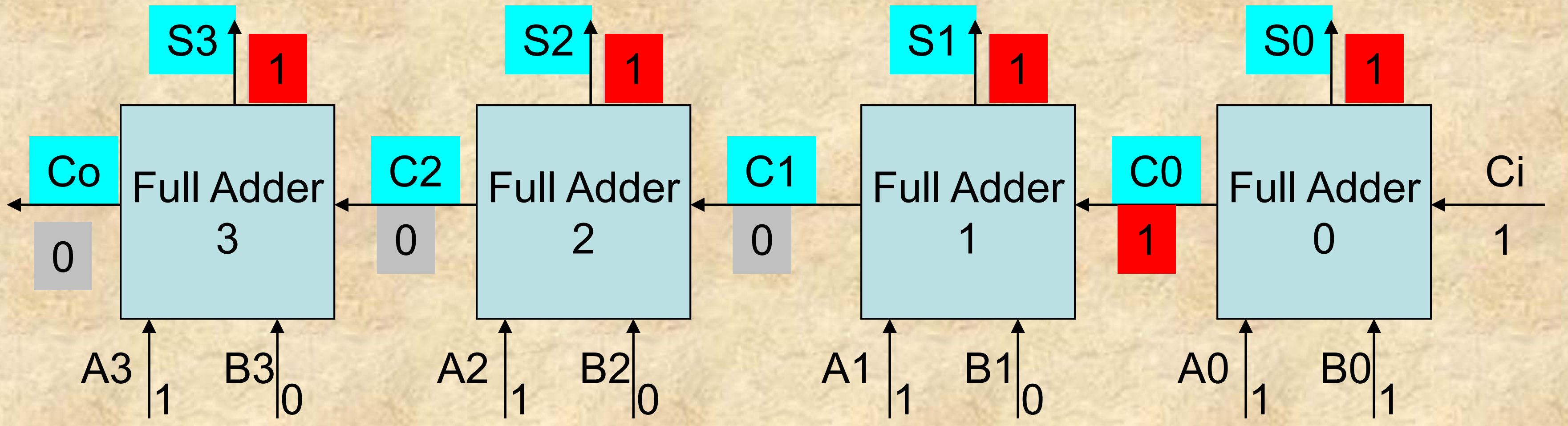
- Component instantiation statements
  - label: component-name ***port map*** (list-of-actual-signals)

4-bit Full Adder Simulation (1/7)

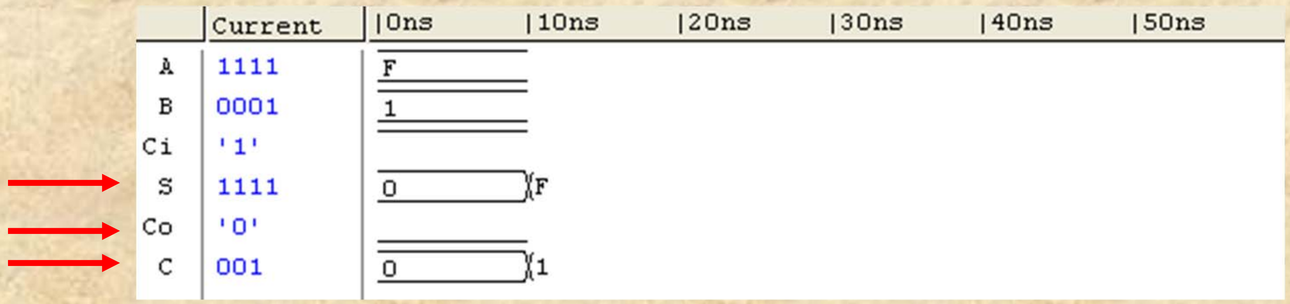


4-bit Full Adder Simulation (2/7)



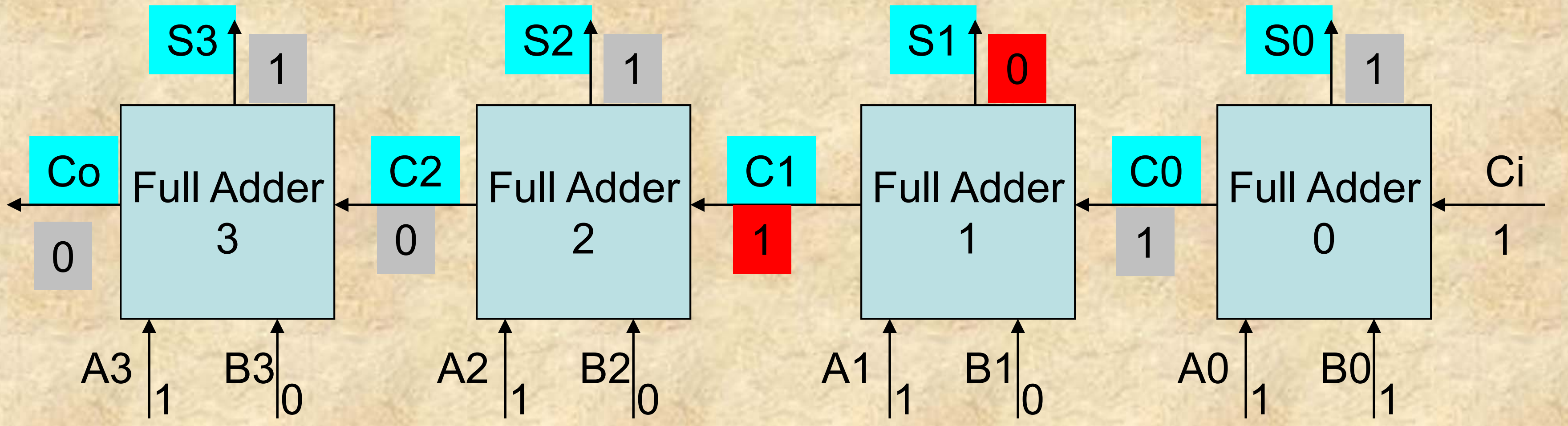


	C2	C1	C0	S3	S2	S1	S0	Co
0ns	0	0	0	0	0	0	0	0
10ns	0	0	1(1)	1	1	1	1(F)	0



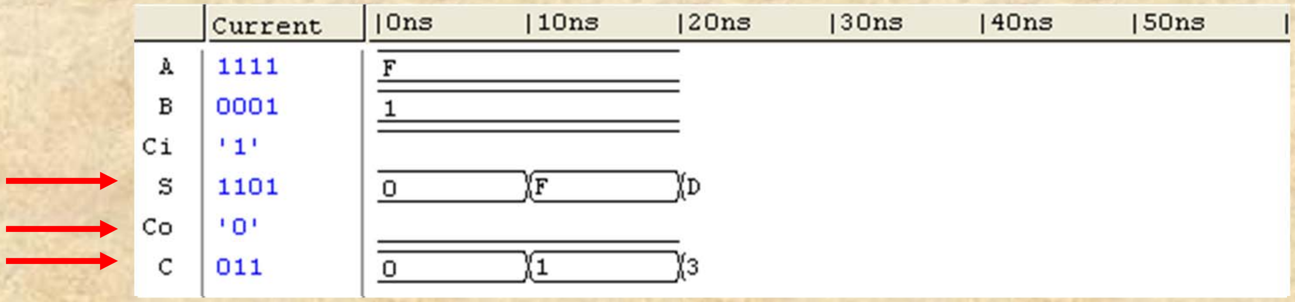
```
     Current  |0ns     |10ns    |20ns    |30ns    |40ns    |50ns
A    1111     F
B    0001     1
Ci   '1'
S    1111     0        F
Co   '0'
C    001      0        1
```

4-bit Full Adder Simulation (3/8)

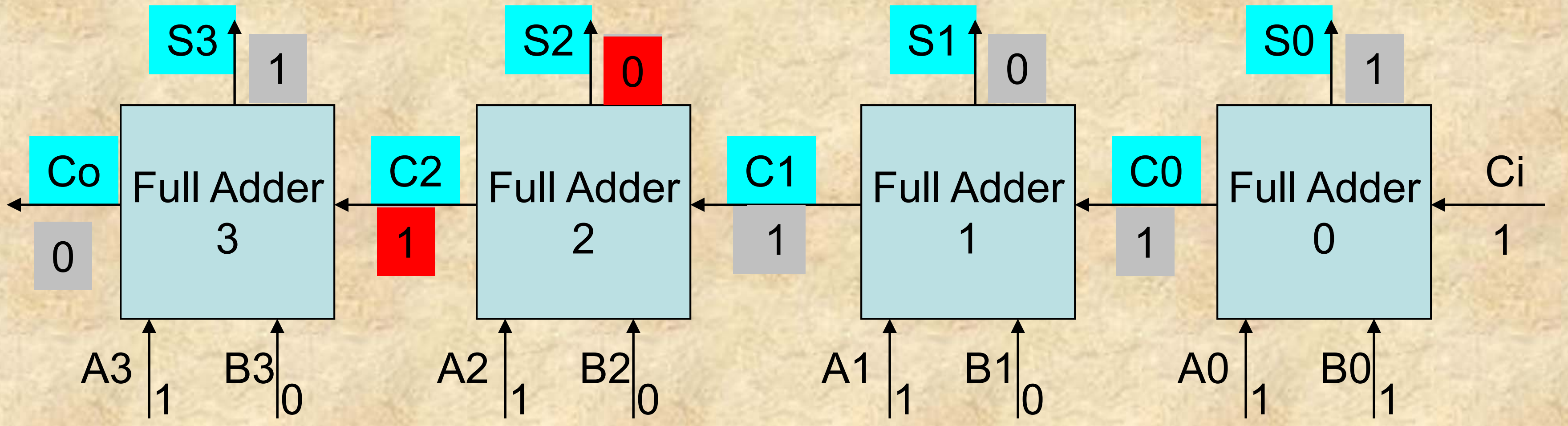


	C2	C1	C0	S3	S2	S1	S0	Co
0ns	0	0	0	0	0	0	0	0
10ns	0	0	1(1)	1	1	1	1(F)	0
20ns	0	1	1(3)	1	1	0	1(D)	0

Nothing changes to the inputs of Full Adders 0,2,3.
Only need to look at Full Adder 1.

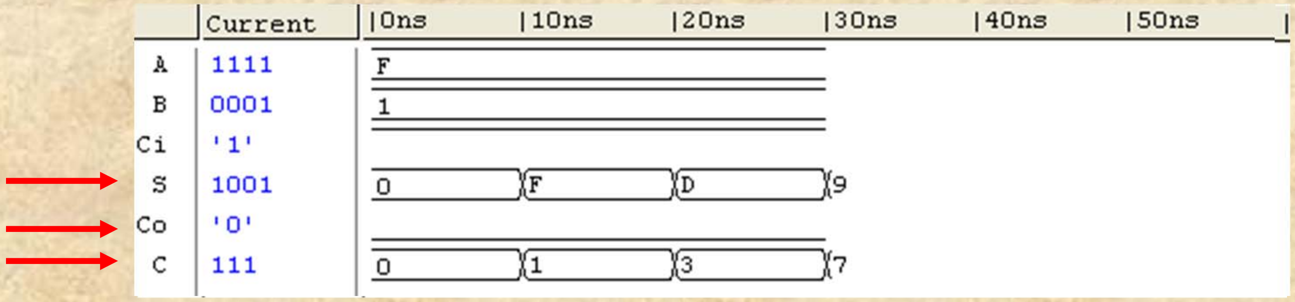


	Current	0ns	10ns	20ns	30ns	40ns	50ns
A	1111	F					
B	0001	1					
Ci	'1'						
S	1101	0	F	D			
Co	'0'						
C	011	0	1	3			

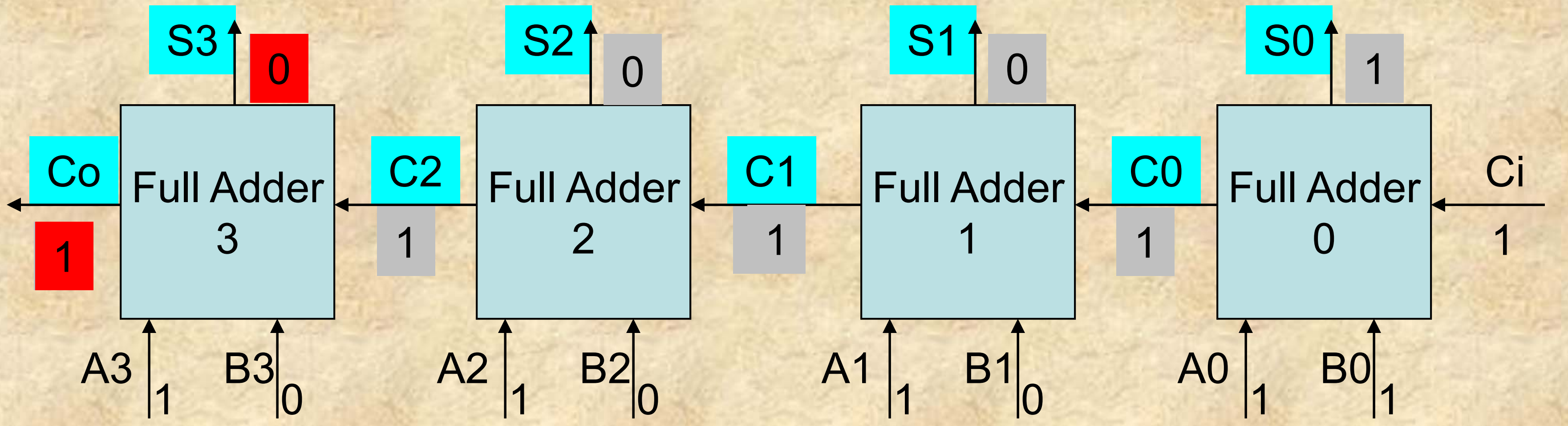


	C2	C1	C0	S3	S2	S1	S0	Co
0ns	0	0	0	0	0	0	0	0
10ns	0	0	1(1)	1	1	1	1(F)	0
20ns	0	1	1(3)	1	1	0	1(D)	0
30ns	1	1	1(7)	1	0	0	1(9)	0

Nothing changes to the inputs of Full Adders 0,1,3.
Only need to look at Full Adder 2.

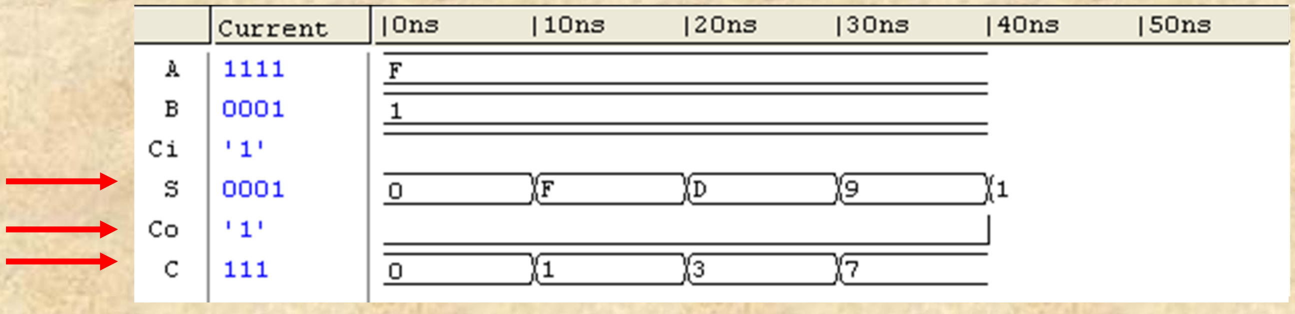

```
     Current  |0ns    |10ns   |20ns   |30ns   |40ns   |50ns
A    1111     F
B    0001     1
Ci   '1'
S    1001     0       F       D       9
Co   '0'
C    111      0       1       3       7
```

4-bit Full Adder Simulation (5/8)

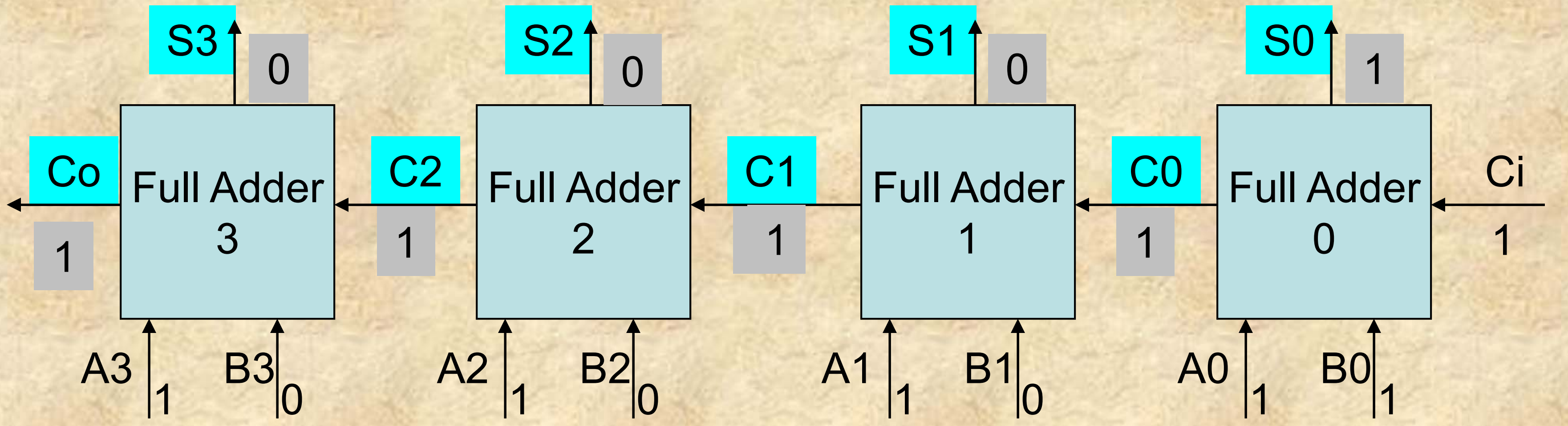


	C2	C1	C0	S3	S2	S1	S0	Co
0ns	0	0	0	0	0	0	0	0
10ns	0	0	1(1)	1	1	1	1(F)	0
20ns	0	1	1(3)	1	1	0	1(D)	0
30ns	1	1	1(7)	1	0	0	1(9)	0
40ns	1	1	1(7)	0	0	0	1(1)	1

Nothing changes to the inputs of Full Adders 0,1,2.
Only need to look at Full Adder 3

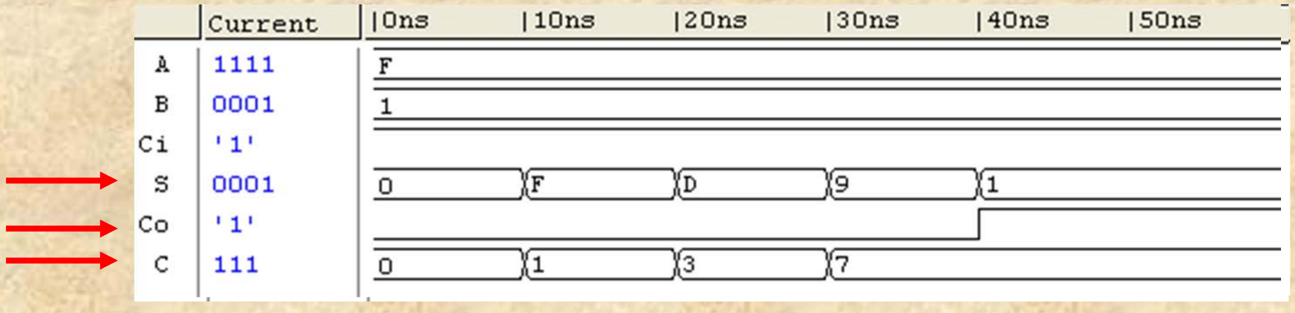

```
      Current  |0ns     |10ns    |20ns    |30ns    |40ns    |50ns
A     1111      F
B     0001      1
Ci    '1'
S     0001      0        F        D        9        1
Co    '1'
C     111       0        1        3        7
```

4-bit Full Adder
Simulation (6/8)



	C2	C1	C0	S3	S2	S1	S0	Co
0ns	0	0	0	0	0	0	0	0
10ns	0	0	1(1)	1	1	1	1(F)	0
20ns	0	1	1(3)	1	1	0	1(D)	0
30ns	1	1	1(7)	1	0	0	1(9)	0
40ns	1	1	1(7)	0	0	0	1(1)	1
50ns	1	1	1(7)	0	0	0	1(1)	1

The circuit is steady. Nothing changes from 40-50 ns.



4-bit Full Adder Simulation (7/8)

Signals and Constants (1/5)

- Signals
  - Interface: input/output
    - entity FullAdder is
    -     port(X, Y, Cin: in bit;   -- inputs
    -                   Cout, Sum: out bit); -- outputs
    - end FullAdder;
  - Internal signals
    - architecture gates of two_gates is
    -     signal C: bit;
    - begin
    -     C <= A and B; -- concurrent
    -     E <= C or D;   -- statements
    - end gates;

Signals and Constants (2/5)

- The predefined VHDL types
  - bit: ‘0’ or ‘1’
  - boolean: FALSE or TRUE
  - integer
  - positive
  - natural
  - real
  - character
  - time

Signals and Constants (2/5) (cont)

– Types	default value
– bit	‘0’
– Boolean	false
– integer	not defined
– positive	1
– natural	0
– Real	0.000
– character	not defined
– Time	0

Signals and Constants (3/5)

- Signal initialization
  - signal A: bit **:=** ‘1’; -- A <= ‘1’
  - signal A:bit; -- A<=‘0’. ‘0’ is the default value
  - signal A,B,C:bit = ‘1’; -- anything wrong?
  - signal A,B,C:bit **:=** ‘1’; -- what are A and B?
  - -- all A, B, and C will be
  - -- initialized to ‘1’

Signals and Constants (4/5)

- Constants:
  - ***constant*** limit:integer := 17;
  - ***constant*** delay1:time := 5 ns;
  - A <= B after delay1;

Signals and Constants (5/5)

- Enumeration data type
  - Defined by users
  - ***type*** state_type ***is*** (S0,S1,S2,S3,S4,S5);
  - signal state : state_type := S1;
  - state <= s3;

Arrays (1/11)

- Differences with arrays in Java/C++
  - Index values can be customized.
  - Must create an array type before declaring an array object.

Arrays (2/11)

- Differences with arrays in Java/C++ (cont)
  - Correct declaration
    - type SHORT_WORD is array (15 downto 0) of bit;
    - signal DATA_WORD : SHORT_WORD;
  - Wrong declaration
    - signal DATA_WORD : array (15 downto 0) of bit;

Arrays (3/11)

- Differences with arrays in Java/C++ (cont)
  - type SHORT_WORD is array (15 downto 0) of bit;
  - signal DATA_WORD : SHORT_WORD; -- default value “00…0”
  - signal ALT_WORD : SHORT_WORD:=“0101010101010101”;

“0101010101010101”

•The initializer for an array of bits must be double quoted.

ALT_WORD(15)

ALT_WORD(0)

Arrays (4/11)

- Differences with arrays in Java/C++ (cont)
  - A portion of the array can accessed in one statement
  - ALT_WORD(5 downto 0) <= “111111”;

Arrays (5/11)

- Differences with arrays in Java/C++ (cont)
  - The slice direction must be consistent.
    - downto
    - to

    - type SHORT_WORD is array (0 to 15) of bit;
    - signal DATA_WORD : SHORT_WORD;
    - DATA_WORD (5 downto 0) < = “111111”;

Wrong slice direction.

Arrays (6/11)

- Differences with arrays in Java/C++ (cont)
  - Unconstrained array types
  - type intvec is array (<u>natural range <></u>) of integer;
  - type matrix is array (<u>natural range <></u>
  - , <u>natural range <></u>) of integer;

Arrays (7/11)

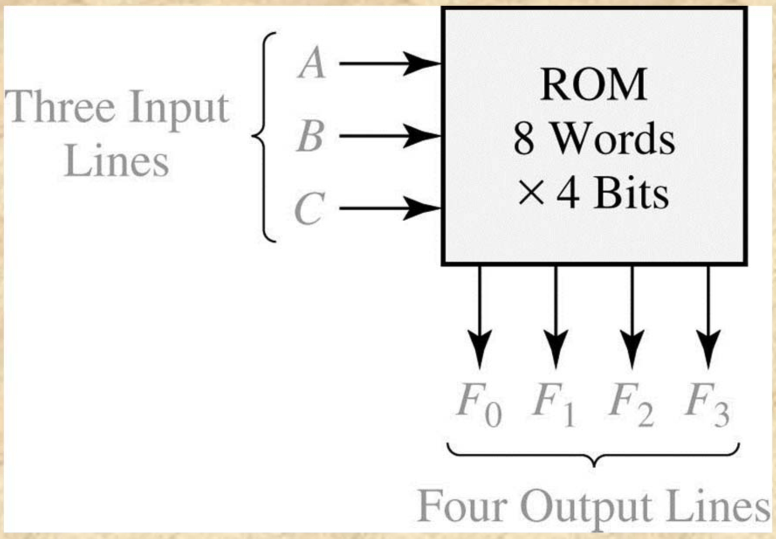
- Predefined unconstrained array types in VHDL
  - type bit_vector is array (natural range<>) of bit;
  - type String is array (positive range <>) of character;

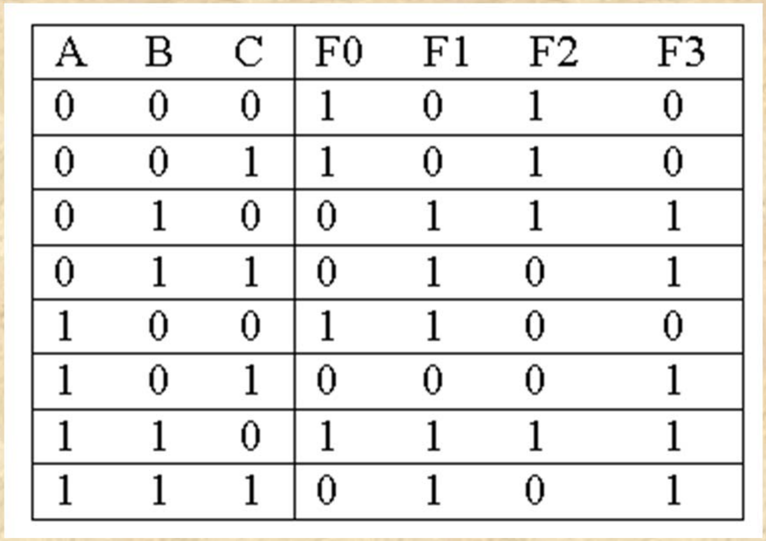
Arrays (8/11)

- Predefined unconstrained array types in VHDL
  - When declaring an array signal, the range must be specified
  - signal A: bit_vector(0 to 5) := “101011”;
  - constant string1:string(1 to 29)=“……”;

Arrays (9/11)

- Example 1: design a ROM as illustrated in Figure 9-17.





A	B	C	F0	F1	F2	F3
0	0	0	1	0	1	0
0	0	1	1	0	1	0
0	1	0	0	1	1	1
0	1	1	0	1	0	1
1	0	0	1	1	0	0
1	0	1	0	0	0	1
1	1	0	1	1	1	1
1	1	1	0	1	0	1

Arrays (10/11)

- entity ROM9_17
-     port (A,B,C: in bit; F: out bit_vector(0 to 3));
- end entity;

- or

- entity ROM9_17
-     port (A: in bit_vector(0 to 2); F: out bit_vector(0 to 3));
- end entity;

Arrays (11/11)

- architecture ROM of ROM9_17 is
-   type ROM8X4 is array (0 to 7) of bit_vector (0 to 3);
-   constant ROM1:ROM8X4 := (“1010”, “1010”, “0111”, 0101”,
-   “1100”, “0001”, “1111”, “0101”);
-   signal index: integer range 0 to 7;
- begin
-   index <= vec2int(A&B&C);   --A&B&C is a 3-bit vector
-       -- vec2int is a function that converts this vector to integer
-   F<=ROM1(index);
- end ROM;

VHDL Operators (1/2)

- Predefined VHDL operators
  1. and or nand nor xor xnor
  2. = /= < <= > >=
  3. sll srl sla sra rol ror
  4. + - &
  5. * / mod rem
  6. not abs **

low

high

Precedence

VHDL Operators (2/2)

- Predefined VHDL operators
  1. and or nand nor xor xnor
  2. = /= < <= > >=
  3. sll srl sla sra rol ror
  4. + - &
  5. * / mod rem
  6. not abs **

Packages and Libraries (1/2)

- Packages and libraries provide a convenient way of referencing frequently used functions and components
- In this class, some of your programs may need package ***bit_pack*** in library ***BITLIB***.

Packages and Libraries (2/2)

- The syntax is
  - **library BITLIB;**
  - **use BITLIB.bit_pack.all;**
- They are needed before each entity that uses library functions.