

ShanghaiTech University

EE 115B: Digital Circuits

Fall 2022

Lecture 13

Hengzhao Yang
November 15, 2022

VHDL code: sections

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity circuit1 is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           c : in  STD_LOGIC;
           f : out STD_LOGIC);
end circuit1;
```

→ Use a library

→ Use packages to provide various data types

→ Define an entity: “circuit1”

→ Define the entity ports: name, mode, and data type

VHDL code: sections

```
architecture behv1 of circuit1 is
```

→ Define an architecture
“behv1” for entity “circuit1”

```
-- this is a comment line
```

→ Comment starts with “--”

```
signal f1 : std_logic;
signal f2 : std_logic;
```

→ Declare internal signals: f1 and f2

```
begin
    f1<=a and b;
    f2<=c and NOT b;
    f<=f1 or f2;
end behv1;
```

→ Define circuit operation

VHDL

- VHDL is **NOT** case sensitive
- VHDL is **concurrent**

```
architecture behv1 of circuit1 is
```

```
-- this is a comment line
```

```
signal f1 : std_logic;
```

```
signal f2 : std_logic;
```

```
begin
```

```
    f1<=a and b;
```

```
    f2<=c and NOT b;
```

```
    f<=f1 or f2;
```

```
end behv1;
```

These statements are concurrent.
They can be written in any order and
the circuit operation is the same.

Example

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity example1 is
    Port ( a : in  STD_LOGIC;
          b : in  STD_LOGIC;
          c : out  STD_LOGIC;
          s : out  STD_LOGIC);
end example1;

architecture rtl of example1 is
begin
    c<=a and b;
    s<=a xor b
end rtl;
```

Entity

- Syntax

```
entity <entity_name> is
  port (
    <port_names> : <mode> <type>;
    <port_names> : <mode> <type>;
    ...
    -- last port has no semicolon
    <port_names> : <mode> <type>
  );
end <entity_name>;
```

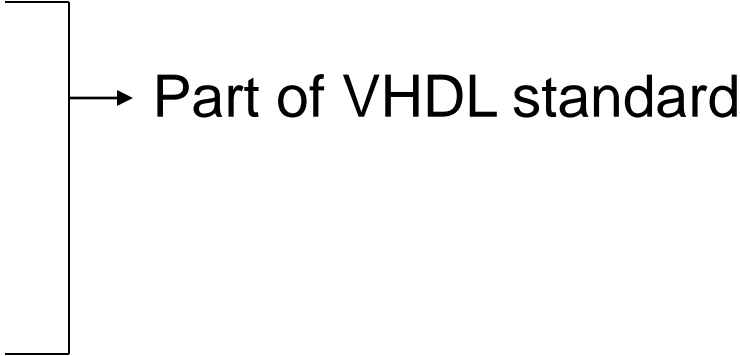
- Example

```
ENTITY model1 IS
  --VHDL is case insensitive
  PORT( a :IN std_logic;
        b :IN std_logic;
        c,d :IN std_logic;
        e :OUT std_logic);
END model1 ;
```


Entity: port mode

- <mode>: indicate the port direction
 - IN: port value may only be read
 - OUT: port value may be updated
 - BUFFER: port value may be both read and updated
 - INOUT: port value may be both read and updated
- BUFFER vs. INOUT (IEEE standard)
 - Although signals of modes INOUT and BUFFER have the same characteristics with respect to whether they may be read or updated, a signal of mode INOUT may be updated by zero or more sources, whereas a signal of mode BUFFER must be updated by at most one source.

Entity: port type

- **<type>**: indicate the signal type of the port
 - **bit**: 0 or 1
 - **bit_vector**: vector of **bit** values
 - **Boolean**: TRUE or FALSE
 - **integer**: integers

Part of VHDL standard

 - **std_logic**: 9 values for signal value and strength
 - **std_logic_vector**: vector of **std_logic** values
- 
- Require
IEEE
package**

Entity: port type

- Standard 'bit'
 - '0': binary zero
 - '1': binary one
- Example

```
ENTITY model1 IS
    PORT( a :IN bit;
          b :IN bit;
          c :IN bit;
          d :IN bit;
          e :OUT bit_vector(7 downto 0));
END model1 ;
```

Entity: port type

- IEEE `'std_logic'` (package: `'std_logic_1164'`)
 - '0': binary zero
 - '1': binary one
 - 'Z': high impedance
 - '-': don't care
 - 'U': uninitialized
 - 'X': unknown
 - 'W': weak unknown
 - 'L': weak zero
 - 'H': weak one

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY model1 IS
    PORT( a :IN std_logic;
          b :IN std_logic;
          c :IN std_logic;
          d :IN std_logic;
          e:OUT std_logic_vector(7 downto 0));
END model1 ;
```

Entity: port type

- `bit_vector`: array of `bit` objects
- `std_logic_vector`: array of `std_logic` objects
- Example

```
ENTITY model1 IS
    PORT( a :IN std_logic_vector(1 to 4);
          b :OUT std_logic_vector(7 downto 0);
          c :OUT std_logic_vector(1 to 4));
END model1 ;
```

- Results
 - `a <= "0011"`: `a(1)=0`, `a(2)=0`, `a(3)=1`, `a(4)=1`
 - `b <= "10011000"`: `b(7)=1`, `b(6)=0`, ..., `b(0)=0`
 - `c <= "1011"`: `c(1)=1`, `c(2)=0`, `c(3)=1`, `c(4)=1`

Architecture

- An architecture represents one possible implementation of its associated entity
 - **architecture declarations**: define internal signals, components, etc. to be used in architecture body
 - **architecture body**: define implementation details of input/output relationship
- Multiple architectures can exist for an entity

Architecture: overview

- Syntax

```
architecture <arch_name> of <entity_name> is  
    -- architecture declarations  
begin  
    -- architecture body  
end <arch_name>;
```

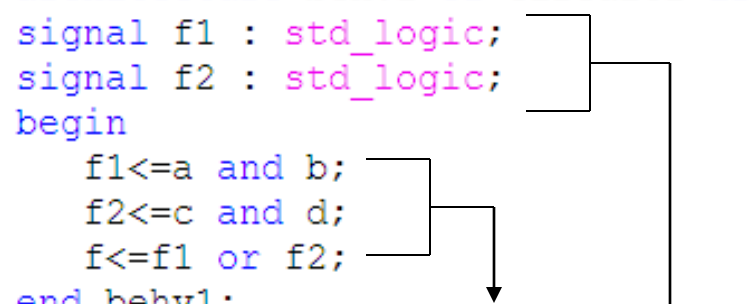
- Example

```
architecture behv1 of circuit1 is  
    signal f1 : std_logic;  
    signal f2 : std_logic;  
begin  
    f1<=a and b;  
    f2<=c and d;  
    f<=f1 or f2;  
end behv1;
```

Architecture: general form

```
architecture <arch_name> of <entity_name> is
  -- architecture declarations
  [SIGNAL declarations]
  [COMPONENT declarations]
  [CONSTANT declarations]
  [TYPE declarations]
  [ATTRIBUTE specifications]
begin
  -- architecture body
  [CONCURRENT ASSIGNMENT statements;]
  [COMPONENT instantiation statements;]
  [PROCESS statements;]
  [GENERATE statements;]
end <arch_name>;
```

```
architecture behv1 of circuit1 is
signal f1 : std_logic;
signal f2 : std_logic;
begin
  f1<=a and b;
  f2<=c and d;
  f<=f1 or f2;
end behv1;
```



CONCURRENT
ASSIGNMENT
statements

SIGNAL
declarations

Architecture declarations: **signal**

- For **internal** signals within the architecture
- Syntax: **signal** <signal_name> : <type>

```
ARCHITECTURE expr OF example1 IS
  SIGNAL u, w, x, y, z :std_logic;
  SIGNAL a, b, c :integer;
BEGIN
  x <= y AND z; -- logical expression
  w <= NOT x;
  u <= w;        -- direct assignment
  c <= a + b;    -- arithmetic expression
END expr;
```

General Form

```
architecture <arch_name> of <entity_name> is
  -- architecture declarations
  [SIGNAL declarations]
  [COMPONENT declarations]
  [CONSTANT declarations]
  [TYPE declarations]
  [ATTRIBUTE specifications]
begin
  -- architecture body
  [CONCURRENT ASSIGNMENT
   statements;]
  [COMPONENT instantiation statements;]
  [PROCESS statements;]
  [GENERATE statements;]
end <arch_name>;
```

Architecture body

- **Concurrent** assignment statements
 - Direct signal assignment
 - Selected signal assignment
 - Conditional signal assignment
 - Component statement
 - Generate statement
- **Sequential** assignment statements
 - Process statement (IF, CASE, FOR, ...)

Concurrent: direct signal assignment

- Syntax: <target> <= <expression>
 - <target> can be an internal signal or an output port
 - <expression> operates on internal signals and/or input ports

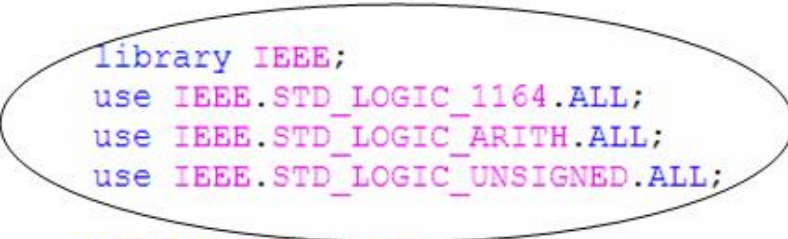
```
ARCHITECTURE expr OF example1 IS
  SIGNAL u, w, x, y, z :std_logic;
  SIGNAL a, b, c :integer;
BEGIN
  x <= y AND z; -- logical expression
  w <= NOT x;
  u <= w;        -- direct assignment
  c <= a + b;    -- arithmetic expression
END expr;
```

General Form

```
architecture <arch_name> of <entity_name> is
  -- architecture declarations
  [SIGNAL declarations]
  [COMPONENT declarations]
  [CONSTANT declarations]
  [TYPE declarations]
  [ATTRIBUTE specifications]
begin
  -- architecture body
  [CONCURRENT ASSIGNMENT
   statements;]
  [COMPONENT instantiation statements;]
  [PROCESS statements;]
  [GENERATE statements;]
end <arch_name>;
```

Operators and expressions

- VHDL standard: arithmetic and logical operators are defined for standard `integer`, `Boolean`, `bit`, `bit_vector` types.
- Logical operators using `std_logic` and `std_logic_vector` types require '`IEEE std_logic_1164`' package
- Arithmetic operators using `std_logic` and `std_logic_vector` types require '`IEEE std_logic_unsigned`' and '`IEEE std_logic_arith`' packages



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
ENTITY modell IS
    PORT( w :IN std_logic_vector(3 downto 0);
          a : IN integer;
          u :OUT std_logic_vector(3 downto 0));
END modell ;
```

```
ARCHITECTURE expr OF modell IS
BEGIN
    -- Adding an integer to an std_logic_vector returning std_logic_vector
    u <= w + a;
END expr;
```

Operators

- Order of precedence

	Class	Operator
Highest	Miscellaneous	NOT, abs, ** (exponentiation)
	Multiplying	*, /, mod (modulus), rem (remainder)
	Sign	+, -
	Adding	+, -, & (concatenation)
	Shift	sll, srl, sla, sra, rol, ror
	Relational	=, /=, <, <=, >, >=
Lowest	Logical	AND, OR, NAND, NOR, XOR, XNOR

Operators

- Order of precedence
 - Operators of highest precedence are evaluated first
 - Deepest nested parentheses are evaluated first
 - Operators of the same precedence are evaluated from left to right
 - Logical operators have same precedence
- Use parentheses to ensure correct expressions
- Example: SOP form
$$f \leq (x1 \text{ and } x2) \text{ or } (x3 \text{ and } x4)$$

Concurrent: selected signal assignment

- A signal can be assigned one of several values based on a **selection criterion**.

- Syntax

```
with <select_signal> select
<target> <= <expression> when <value>,
      <expression> when <value>,
      ....
      < expression> when others;
```

- Example

```
architecture behv of mux
begin
  with s select
    f <= a when "00",
        b when "01",
        c when "10",
        d when others;
end behv;
```

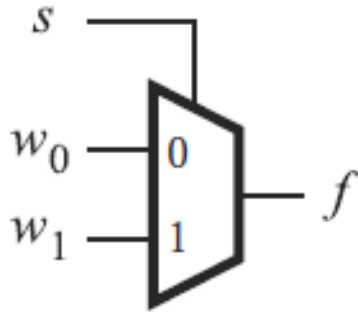
Concurrent: selected signal assignment

- Keywords: **with-select-when**
 - <select_signal> can be an internal signal or an input port
 - <target> can be an internal signal or an output port
 - <value> is one possible value of <select_signal>
 - **when others** is required if not all values of <select_signal> are covered

```
with <select_signal> select  
<target> <= <expression> when <value>,  
        <expression> when <value>,  
        ....  
        < expression> when others;
```

Example: 2-to-1 mux (ch. 6)

- Truth table



(a) Graphical symbol

s	f
0	w_0
1	w_1

(b) Truth table

Example: 2-to-1 mux

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY mux2to1 IS
    PORT ( w0, w1, s : IN    STD_LOGIC ;
          f          : OUT STD_LOGIC ) ;
END mux2to1 ;

ARCHITECTURE Behavior OF mux2to1 IS
BEGIN
    WITH s SELECT
        f <=  w0 WHEN '0',
              w1 WHEN OTHERS ;
END Behavior ;
```

Figure 6.27 VHDL code for a 2-to-1 multiplexer.

Concurrent: conditional signal assignment

- A signal can be assigned one of several values based on a **condition**.

- Syntax

```
<target> <= <expression> when <condition>  
    else <expression> when <condition>  
    else <expression> when <condition>  
    ...  
    else <expression> ;
```

- Example

```
architecture behv of mux  
begin  
    f <= a when s="00" else  
        b when s="01" else  
        c when s="10" else  
        d;  
end behv;
```

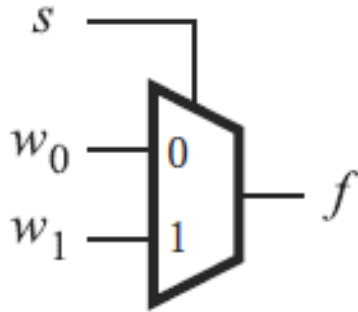
Concurrent: conditional signal assignment

- Keywords: **when-else**
 - <target> can be an internal signal or an output port
 - <expression> operates on internal signals or input ports when <condition> is true
 - Last **else** branch covers all missing conditions

```
<target> <= <expression> when <condition>  
    else <expression> when <condition>  
    else <expression> when <condition>  
    ...  
    else <expression> ;
```

Example: 2-to-1 mux (ch. 6)

- Truth table



(a) Graphical symbol

s	f
0	w_0
1	w_1

(b) Truth table

Example: 2-to-1 mux

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY mux2to1 IS
    PORT ( w0, w1, s : IN  STD_LOGIC ;
          f          : OUT STD_LOGIC ) ;
END mux2to1 ;

ARCHITECTURE Behavior OF mux2to1 IS
BEGIN
    f <= w0 WHEN s = '0' ELSE w1 ;
END Behavior ;
```

Figure 6.31 Specification of a 2-to-1 multiplexer using a conditional signal assignment.

VHDL design description

- Behavioral
 - Model a circuit based on its **functionality**
- Structural
 - Build a complex circuit based on **components**
(lower-level circuits)

Component

- An **entity** defined in one source code file can be used as a **component** in another source code file.
- Requires component declaration and component instantiation statements.

```
architecture <arch_name> of <entity_name> is
    -- architecture declarations
    [SIGNAL declarations]
    [COMPONENT declarations]
    [CONSTANT declarations]
    [TYPE declarations]
    [ATTRIBUTE specifications]
begin
    -- architecture body
    [CONCURRENT ASSIGNMENT statements;]
    [COMPONENT instantiation statements;]
    [PROCESS statements;]
    [GENERATE statements;]
end <arch_name>;
```

Component: declaration

- Syntax

```
component <component_name>
port (
    <port_name> : <mode> <type>;
    <port_name> : <mode> <type>;
    ...

    <port_name> : <mode> <type>);
end component;
```

- Example

- <component_name>
must be the name of
the entity to be used.

```
component circuit1
    Port ( A : in  STD_LOGIC;
           B : in  STD_LOGIC;
           C : in  STD_LOGIC;
           Y : out  STD_LOGIC);
end component;
```

Component: instantiation

- Syntax 1: positional association

```
<instance_name> : <component_name> PORT MAP ( <signal_name>,  
    <signal_name> .... );
```

- Syntax 2: named association

```
<instance_name> : <component_name> PORT MAP (  
    <component_port_name> => <actual_signal_name>,  
    <component_port_name> => <actual_signal_name> .... );
```

- Example

```
ex1: circuit1 port map (A=>AT,  
                        B=>BT,  
                        C=>wiresig,  
                        Y=>YT) ;
```


Example 1: 4-bit full adder

- Truth table for 1-bit full adder

c_i	x_i	y_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

(a) Truth table

$x_i y_i$					
c_i		00	01	11	10
			1		1
1		1		1	

$$s_i = x_i \oplus y_i \oplus c_i$$

$x_i y_i$					
c_i		00	01	11	10
				1	
1			1	1	1

$$c_{i+1} = x_i y_i + x_i c_i + y_i c_i$$

(b) Karnaugh maps

Example 1: 4-bit full adder

- VHDL code for 1-bit full adder

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY fulladd IS
    PORT ( Cin, x, y : IN    STD_LOGIC ;
          s, Cout    : OUT STD_LOGIC ) ;
END fulladd ;

ARCHITECTURE LogicFunc OF fulladd IS
BEGIN
    s <= x XOR y XOR Cin ;
    Cout <= (x AND y) OR (Cin AND x) OR (Cin AND y) ;
END LogicFunc ;
```

Example 1: 4-bit full adder

- General form: n -bit full adder

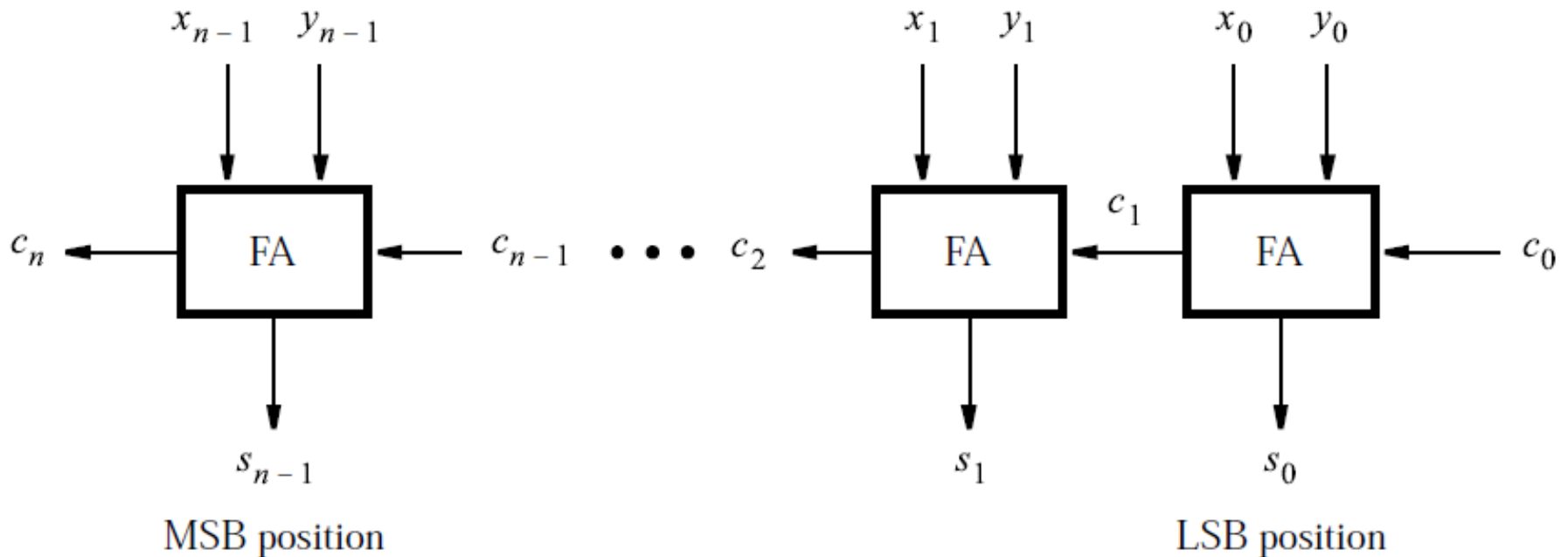


Figure 5.6 An n -bit ripple-carry adder.

Example 1: 4-bit full adder

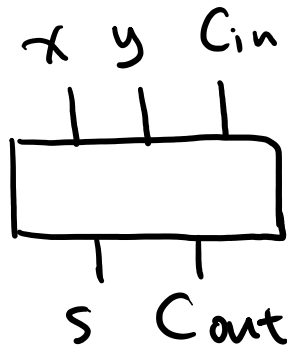
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY adder4 IS
    PORT ( Cin          : IN    STD_LOGIC ;
          x3, x2, x1, x0 : IN    STD_LOGIC ;
          y3, y2, y1, y0 : IN    STD_LOGIC ;
          s3, s2, s1, s0 : OUT   STD_LOGIC ;
          Cout           : OUT   STD_LOGIC ) ;
END adder4 ;

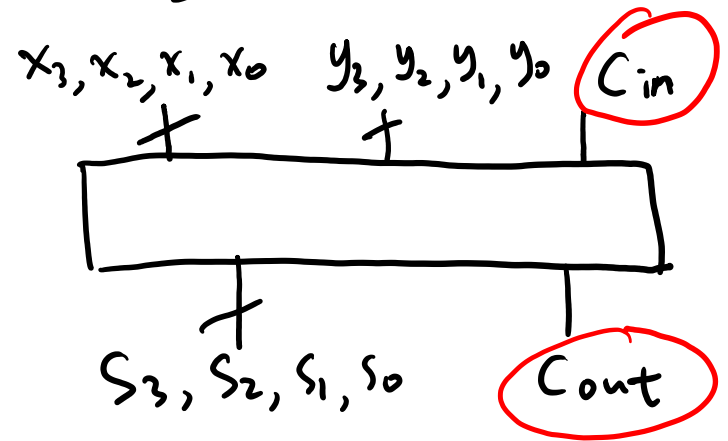
ARCHITECTURE Structure OF adder4 IS
    SIGNAL c1, c2, c3 : STD_LOGIC ;
    COMPONENT fulladd
        PORT ( Cin, x, y : IN    STD_LOGIC ;
              s, Cout   : OUT   STD_LOGIC ) ;
    END COMPONENT ;
BEGIN
    stage0: fulladd PORT MAP ( Cin, x0, y0, s0, c1 ) ;
    stage1: fulladd PORT MAP ( c1, x1, y1, s1, c2 ) ;
    stage2: fulladd PORT MAP ( c2, x2, y2, s2, c3 ) ;
    stage3: fulladd PORT MAP (
        Cin => c3, Cout => Cout, x => x3, y => y3, s => s3 ) ;
END Structure ;
```

Component vs. Entity

> Component (1-bit)



> Entity (4-bit)



$C_{in} \rightarrow C_0$

$C_{out} \rightarrow C_4$