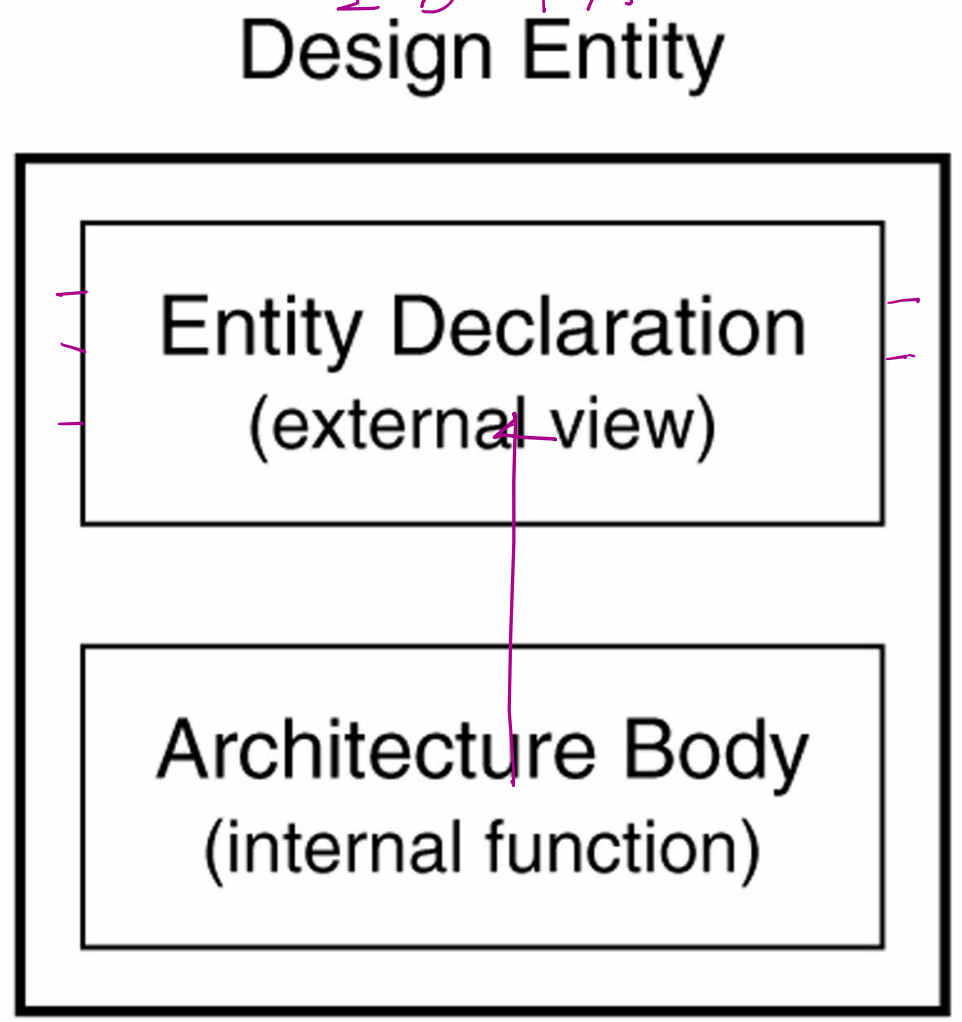
Chapter 2

Entity, Architecture, and Structural VHDL Modeling

Entity, Architecture, and Structural VHDL Modeling

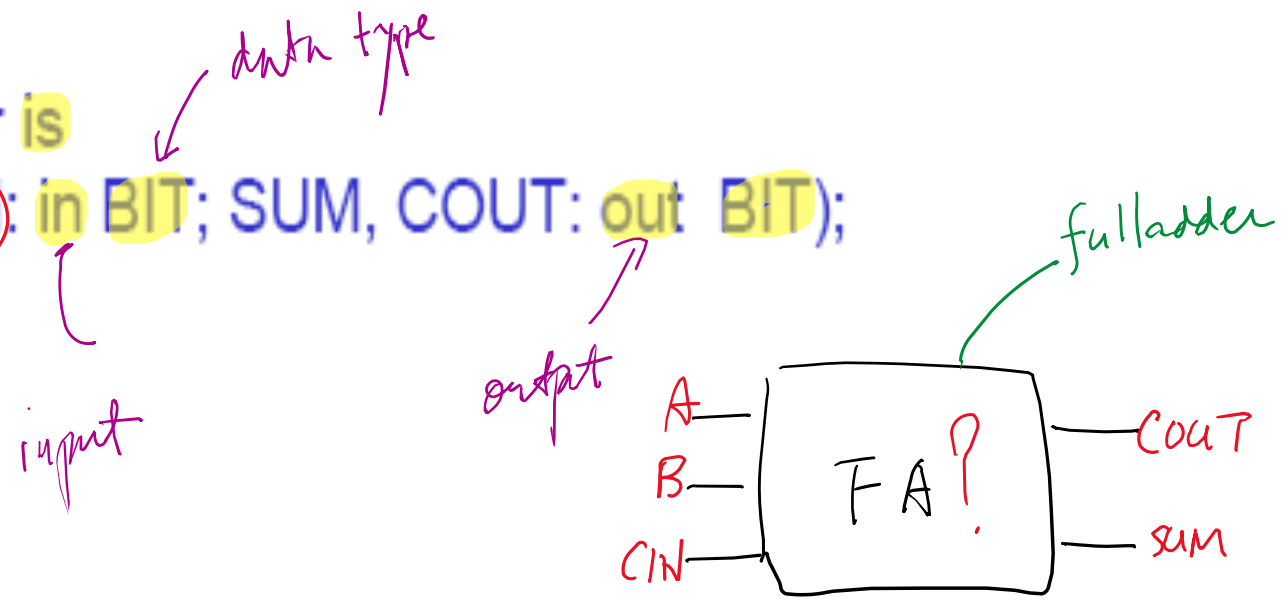
- *Describing Structure*
- *Constructing Structural VHDL Models*
- *Hierarchy, Abstraction, and Accuracy*
- *Generics*
- *Component Instantiation and Synthesis*

1-b FA


Design Entity
Entity Declaration
(external view)
Architecture Body
(internal function)

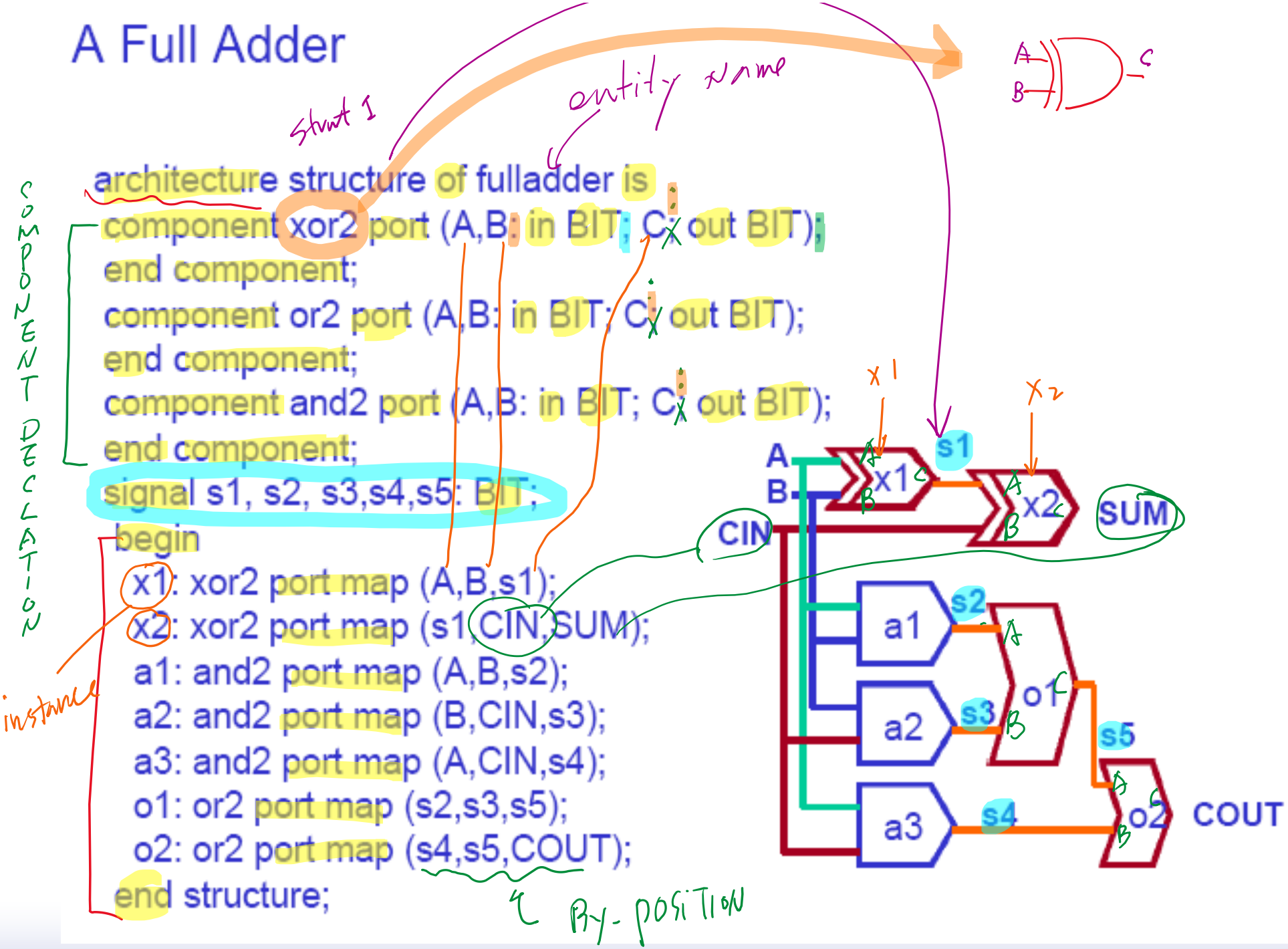
- Design entity modeled as a set of interconnected components
- Components are
  - Declared using component declaration
  - Instantiated using component instantiation statements
  - Interconnected using signals

```
entity fulladder is
  port (A,B,CIN: in BIT; SUM, COUT: out BIT);
end fulladder;
```



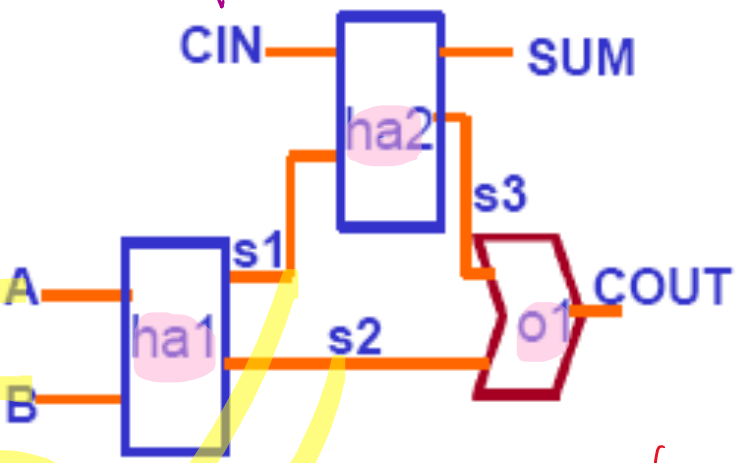
A Full Adder

```
architecture structure of fulladder is
 component xor2 port (A,B: in BIT; C: out BIT);
 end component;
 component or2 port (A,B: in BIT; C: out BIT);
 end component;
 component and2 port (A,B: in BIT; C: out BIT);
 end component;
 signal s1, s2, s3,s4,s5: BIT;
 begin
  x1: xor2 port map (A,B,s1);
  x2: xor2 port map (s1,CIN,SUM);
   a1: and2 port map (A,B,s2);
   a2: and2 port map (B,CIN,s3);
   a3: and2 port map (A,CIN,s4);
   o1: or2 port map (s2,s3,s5);
   o2: or2 port map (s4,s5,COUT);
 end structure;
```



Another Full Adder

```
architecture structure of fulladder is
  component halfadder
    port (A,B: in BIT; S, C: out BIT);
  end component;
  component or2
    port (A,B: in BIT; C; out BIT);
  end component;
  signal s1, s2, s3: BIT;
  begin
    ha1: halfadder port map (A,B,s1,s2);
    o1:   or2 port map (s2,s3,COUT);
    ha2: halfadder port map (CIN,s1,SUM,s3);
  end structure;
```

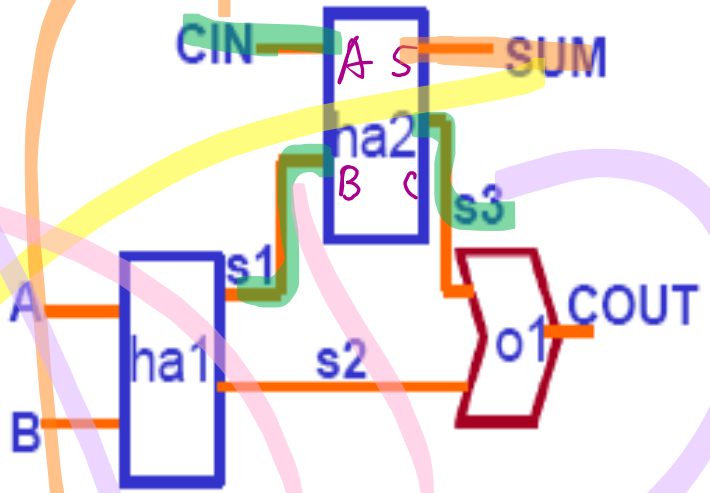


Positional association

Component instance name

Another Full Adder

```
architecture structure of fulladder is
 component halfadder
  port (A,B: in BIT; S,C: out BIT);
 end component;
 component or2
  port (A,B: in BIT; C: out BIT);
 end component;
 signal s1, s2, s3: BIT;
  begin
    ha1: halfadder port map (A,B,s1,s2);
    o1: or2 port map (s2,s3,COUT);
    ha2: halfadder port map (S=>SUM, A=>CIN, C=>s3, B=>s1);
end structure;
```



Sum⇒S

Named association

port map by name

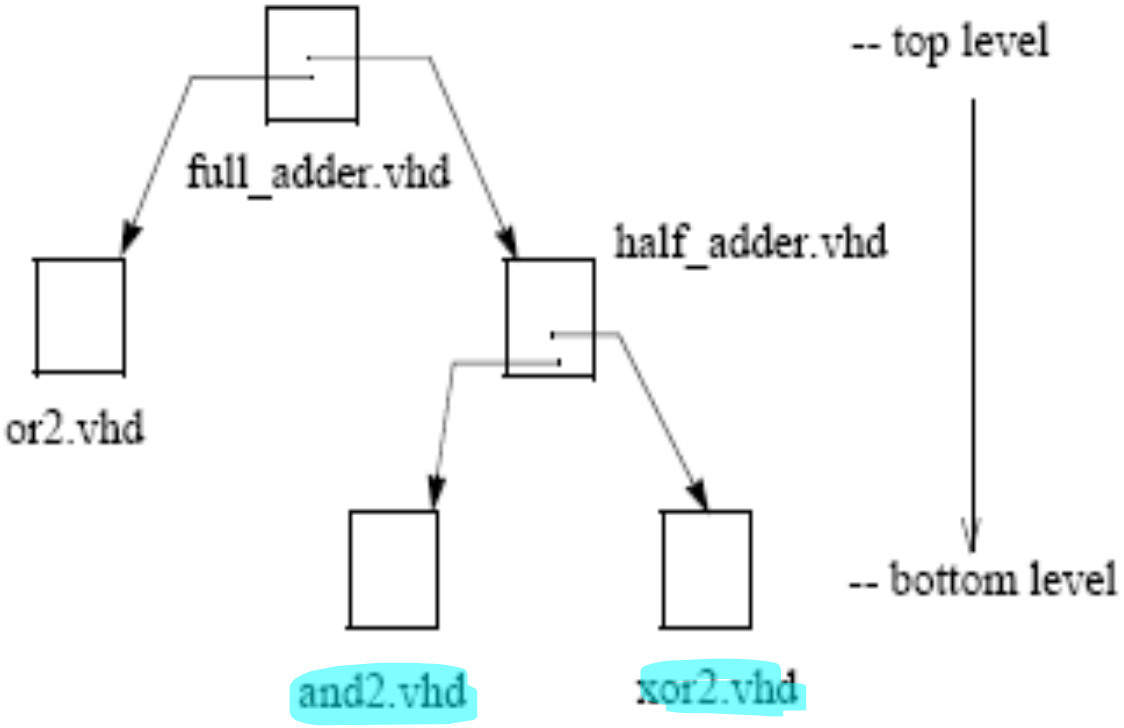
Hierarchy and Abstraction

```
architecture structural of half_adder is
component xor2 is
 port (a, b : in std_logic;
       c : out std_logic);
end component xor2;
component and2 is
port (a, b : in std_logic;
       c : out std_logic);
end component and2;
begin
EX1: xor2 port map (a => a, b => b, c => sum);
AND1: and2 port map (a=> a, b=> b, c=> carry);
end architecture structural;
```

port map by name

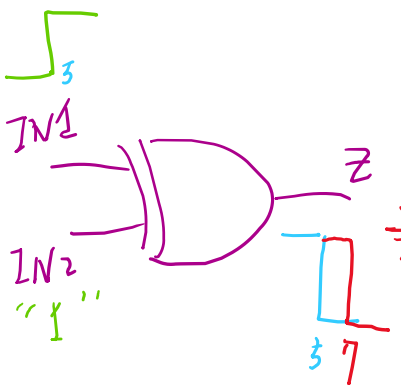
- Structural descriptions can be nested
- The half adder may itself be a structural model

Hierarchy and Abstraction



- Nested structural descriptions to produce hierarchical models
- The hierarchy is flattened prior to simulation
- Behavioral models of components at the bottom level must exist

Generics




```
library IEEE;
use IEEE.std_logic_1164.all;

entity xor2 is
generic (gate_delay : Time:= 2 ns);
port(In1, In2 : in std_logic;
     z : out std_logic);
end entity xor2;

architecture behavioral of xor2 is
begin
z <= (In1 xor In2) after gate_delay;
end architecture behavioral;
```

attribute

ps ← pico second 10^{-12}

fs ← femto second 10^{-15}

ms ← mini second 10^{-3}

us ← micro second 10^{-6}

ns ← nano second 10^{-9}

gate_delay

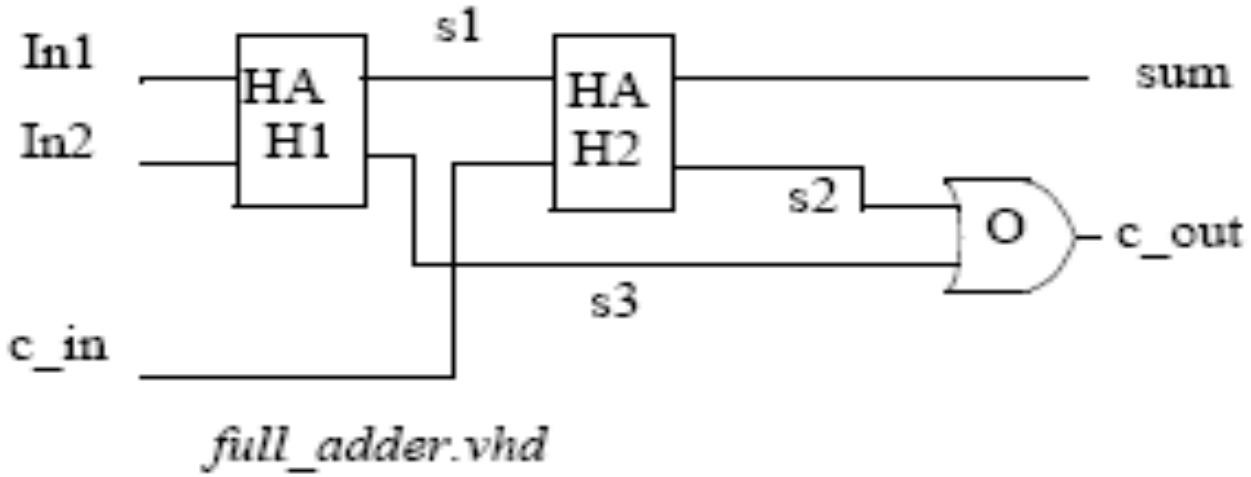
- Enables the construction of parameterized models

Generics in Hierarchical Models

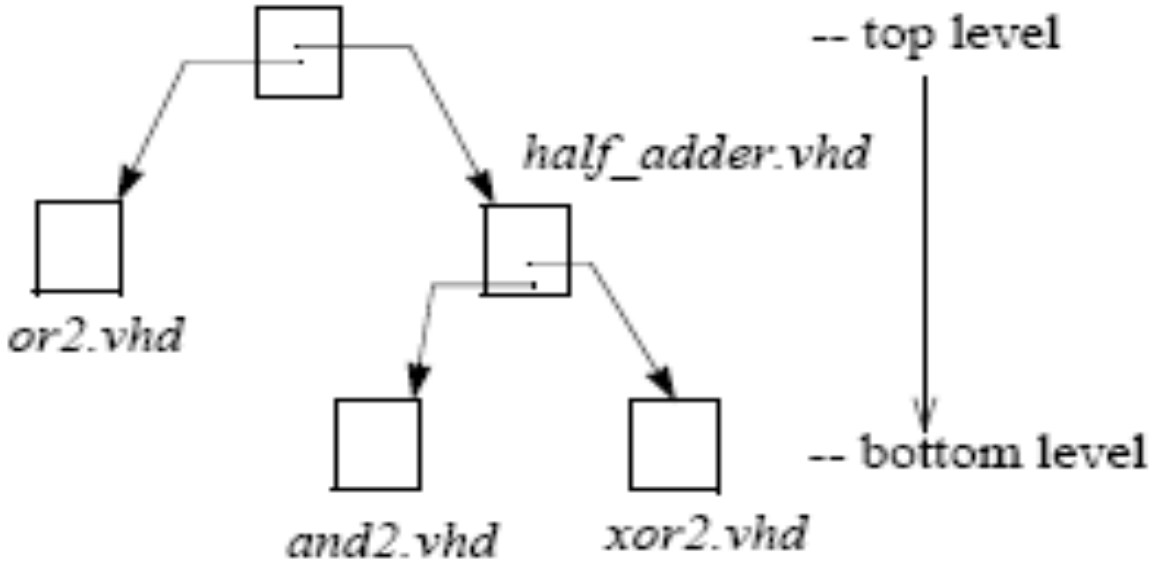
```
architecture generic_delay of half_adder is
component xor2
generic (gate_delay: Time);
port (a, b : in std_logic;
      c : out std_logic);
end component;

component and2
generic (gate_delay: Time);
port (a, b : in std_logic;
      c : out std_logic);
end component;

begin
EX1: xor2 generic map (gate_delay => 6 ns)
          port map(a => a, b => b, c => sum);
A1: and2 generic map (gate_delay => 3 ns)
          port map(a=> a, b=> b, c=> carry);
end generic_delay;
```

- Parameter values are passed through the hierarchy

Example: Full Adder



full_adder.vhd



Precedence of Generic Declarations

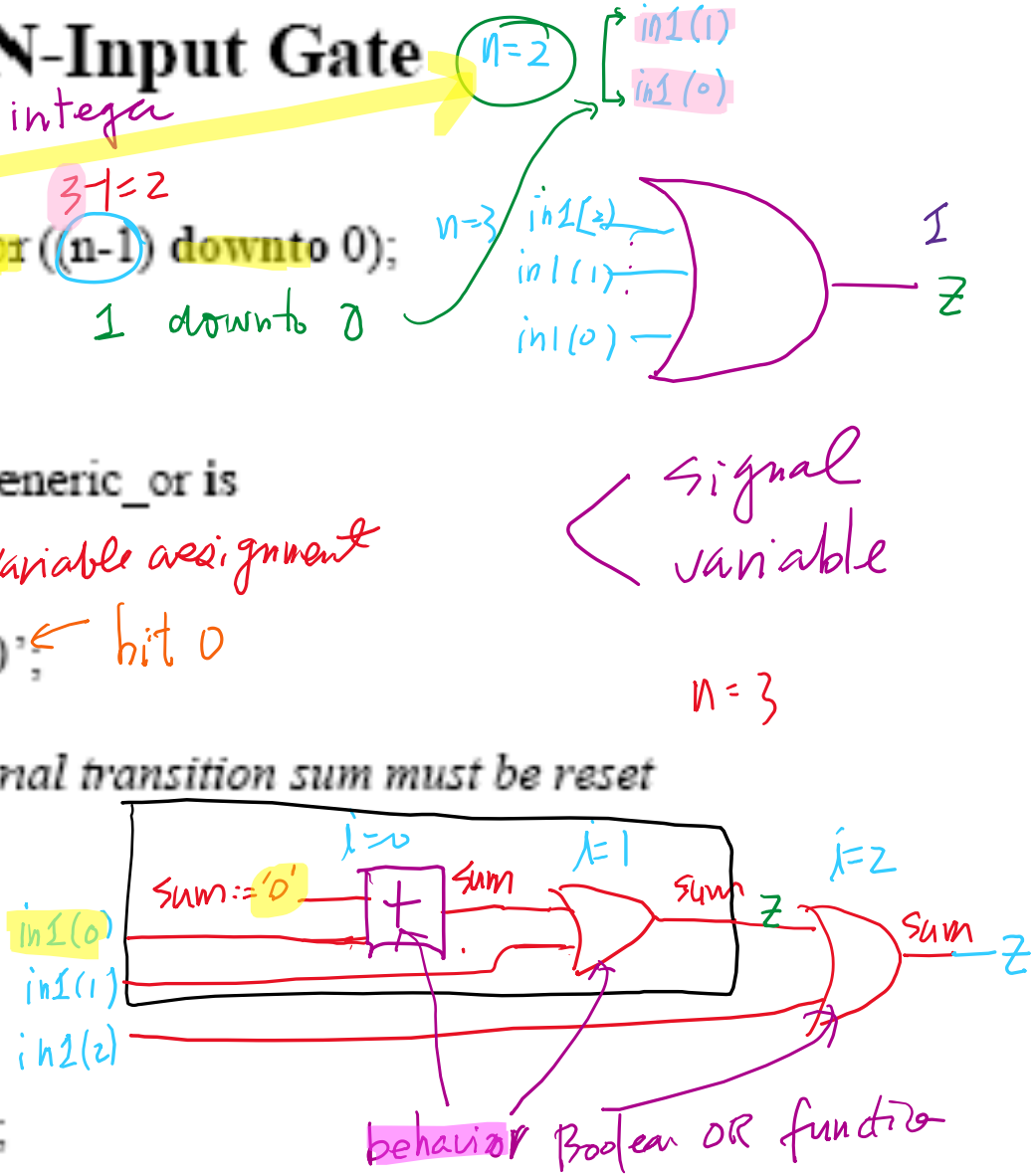
```
architecture generic_delay2 of half_adder is
component xor2
generic (gate_delay: Time);
 port(a,b : in std_logic;
        c : out std_logic);
end component;

component and2
generic (gate_delay: Time:= 6 ns);
port (a, b : in std_logic;
    c : out std_logic);
end component;

begin
EX1: xor2 generic map (gate_delay => gate_delay)
          port map(a => a, b => b, c => sum);
A1: and2 generic map (gate_delay => 4 ns)
          port map(a=> a, b=> b, c=> carry);
end generic_delay2;
```

takes precedence

- Generic map takes precedence over the component declaration

Example: N-Input Gate

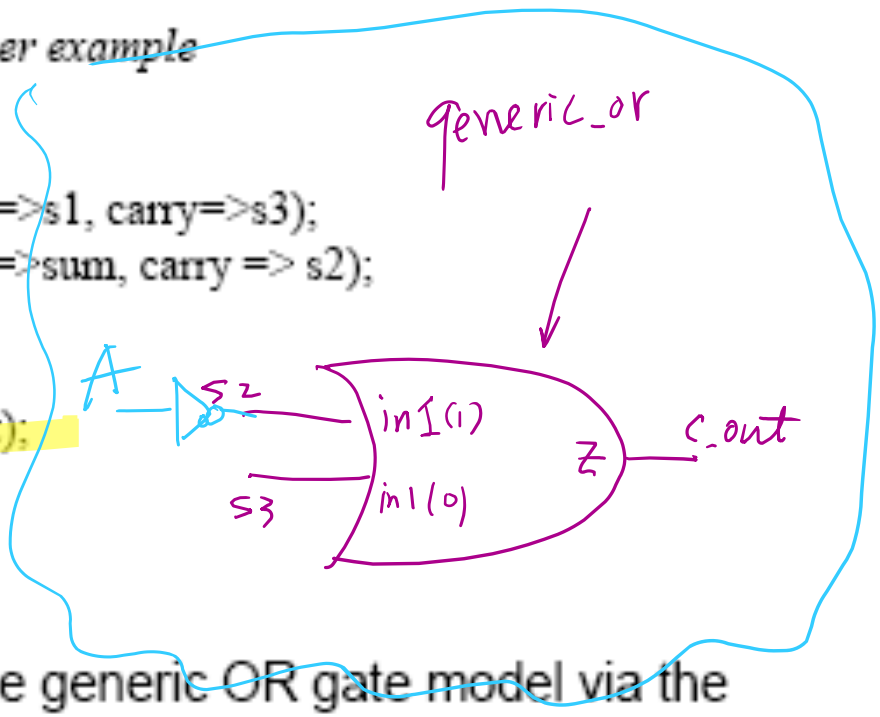
```
entity generic_or is
generic (n: positive:=2);
port (in1 : in std_logic_vector ((n-1) downto 0);
      z : out std_logic);
end entity generic_or;

architecture behavioral of generic_or is
begin
P1: process (in1)
variable sum : std_logic := '0';
begin
sum := '0'; -- on an input signal transition sum must be reset
for i in 0 to (n-1) loop
sum := sum or in1(i);
end loop;
z <= sum;
end process;
end architecture behavioral;
```



- Map the generics to create different size OR gates

Example: Using the Generic N-Input OR Gate

```
architecture structural of full_adder is
component generic_or
generic (n: positive);
port (in1 : in std_logic_vector ((n-1) downto 0);
      z : out std_logic);
end component;
...
... -- remainder of the declarative region from earlier example
...
 begin
H1: half_adder port map (a => In1, b => In2, sum=>s1, carry=>s3);
H2:half_adder port map (a => s1, b => c_in, sum =>sum, carry => s2);

O1: generic_or generic map (n => 2)
          port map (a => s2, b => s3, z => c_out);

end structural;
```



- Full adder model can be modified to use the generic OR gate model via the **generic map ()** construct

Reduced And

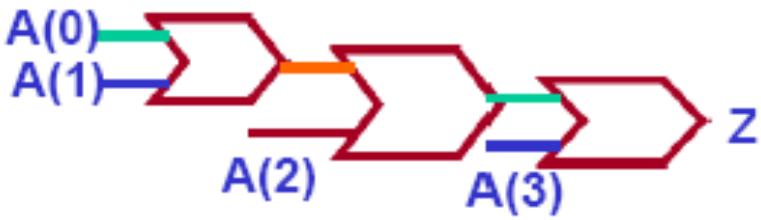
Generic - AND

```
entity RedAnd is
  generic (width : integer := 8);   -- word width
  port (A : in std_logic_vector(width-1 downto 0);
        Z : out std_logic);
end RedAnd;
architecture Behavioral of RedAnd is
begin
  reduceAnd: process (A)
    variable zv : std_logic;
  begin
    zv := A(0);
    for i in 1 to width-1 loop
      zv := zv and A(i);
    end loop;
    Z <= zv;
  end process reduceAnd;
end Behavioral;
```

process called

Reduced Or

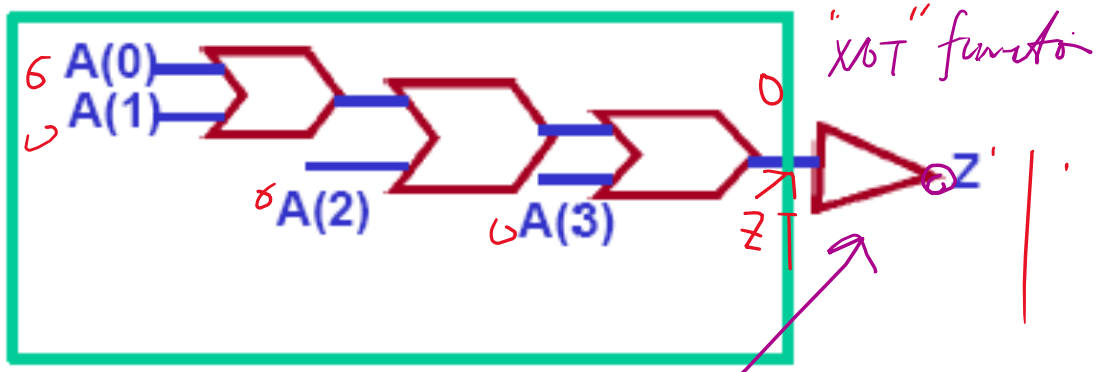
```
entity RedOr is
  generic (width : integer := 8);  -- word width
  port (A : in std_logic_vector(width-1 downto 0);
        Z : out std_logic);
end RedOr;
architecture Behavioral of RedOr is
begin
  reduceOr : process (A)
    variable zv : std_logic;
    begin
     zv := A(0);
     for i in 1 to width-1 loop
       zv := zv or A(i);
     end loop;
    Z <= zv;
  end process reduceOr;
end Behavioral;
```



All Zeroes Detection (structural model)

Generic Zero Detector

```
entity AllZeroDet is
  generic (width : integer := 8);   -- word width
  port (A:in std_logic_vector(width-1 downto 0); -- operand
        Z:out std_logic);            -- all-zeroes flag
end AllZeroDet;
architecture Structural of AllZeroDet is
  component RedOr
    generic (width : integer);
    port (A: in std_logic_vector(width-1 downto 0); Z : out std_logic);
  end component;
  signal ZT : std_logic;  -- temp.
begin
 zeroFlag : RedOr
    generic map (width)
    port map (A, ZT);
 Z <= not ZT;
end Structural;
```



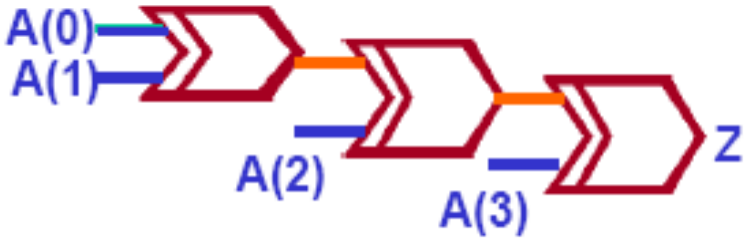
"NOT" function

Reduced Xor

Generic XOR

```
entity RedXor is
  generic (width : integer := 8);    -- word width
  port (A : in std_logic_vector(width-1 downto 0);
        Z : out std_logic);
end RedXor;
architecture Behavioral of RedXor is
begin
  reduceXor : process (A)
    variable zv : std_logic;
  begin
    zv := A(0);
    for i in 1 to width-1 loop
      zv := zv xor A(i);
    end loop;
    Z <= zv;
  end process reduceXor;
end Behavioral;
```

Boolean

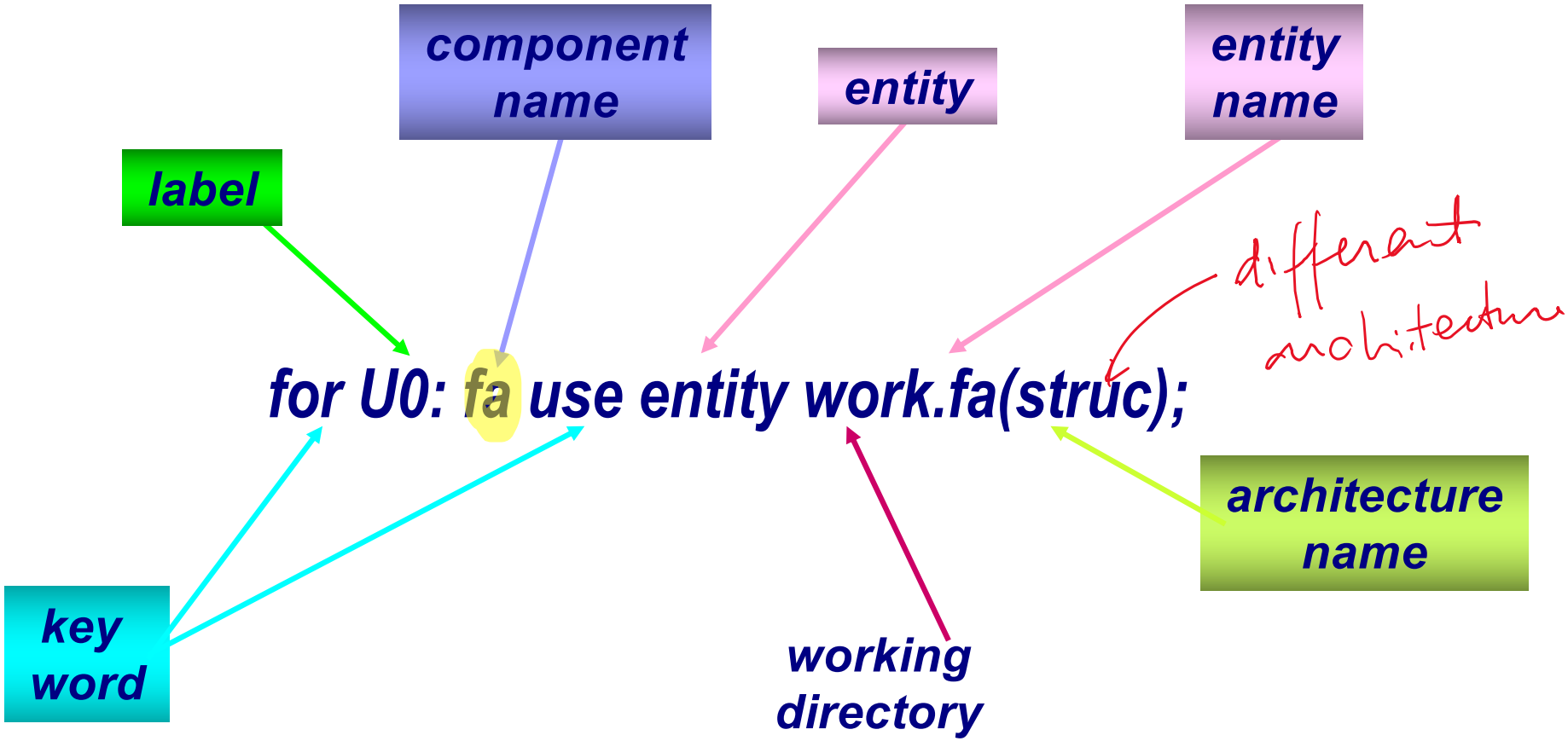


Structural Architecture Body

- Entity implementation by specifying its composition from subsystems
  - ***Structural architectural body*** – system composed only of interconnected subsystems
- Elements of structural architecture:
  - ***Signal declaration***
  - ***Component declaration***
  - ***Component instantiation***

Configuration

Component Specification - Syntax



Example: Component Instant.

```
architecture struct of ripple4 is

-- component declaration comes
here
-- component specification
for U0: fa use entity work.fa(struc);


signal c: bit_vector(2 downto 0);
begin

FA0: fa
    port map(
                a_in   => a(0),
                b_in   => b(0),
                c_in   => c_in_f,
                s_out => s_out_f(0),
                c_out => c(0)
              );
```

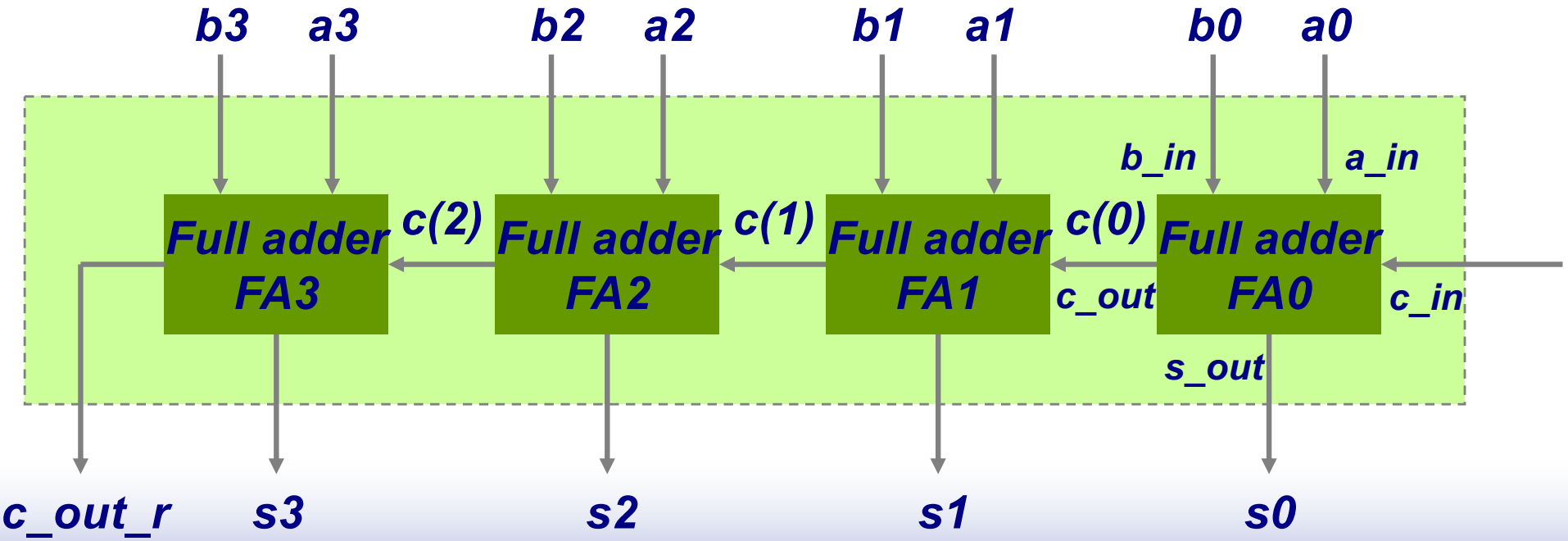
```
component fa
    port(
            a_in: in bit;
            b_in: in bit;
            c_in: in bit;
            s:      out bit;
            c_out: out bit
          );
end component;
```

label

key word

Example: 4-bit Ripple Adder - Concept

- 4-bit ripple adder composed of 4 full adders connected in series through carry bit



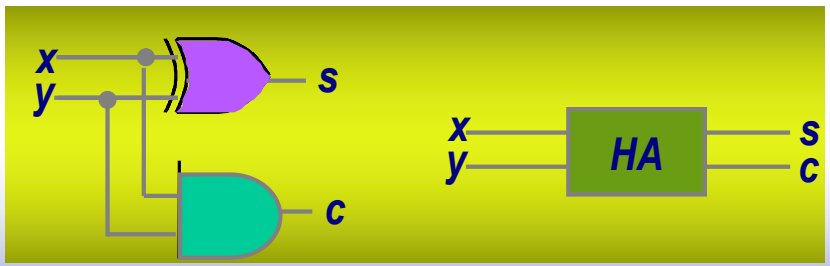
Step 1: Construction of Half Adder

- Full adder cell composed of two half adders
- Half adder: performs addition of two bits, generated sum bit and carry out

x	0	0	1	1
+ y	+ 0	+ 1	+ 0	+ 1
c s	0 0	0 1	0 1	1 0

4 cases of addition

x y	c	s
0 0	0	0
0 1	0	1
0 0	0	1
1 1	1	0



Gate level

Truth table

VHDL Code for Half Adder

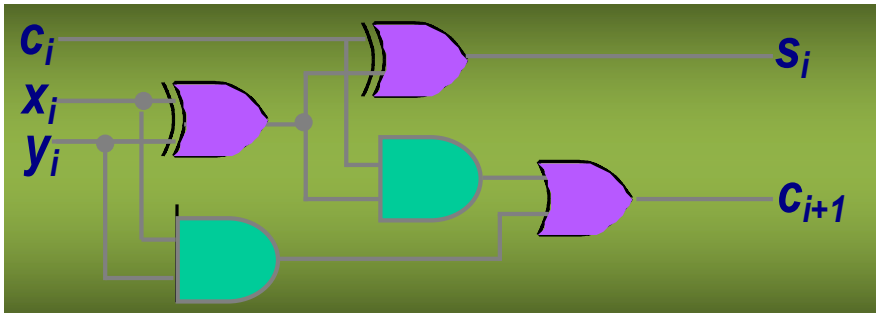
```
library IEEE;
use ieee.std_logic_1164.all;

entity ha is
   port( a, b: in bit;
         sum, c_out: out bit);
end ha;

architecture behav of ha is
   begin
      sum <= a xor b;
      c_out <= a and b;
end behav;
```

Step 2: Construction of Full Adder

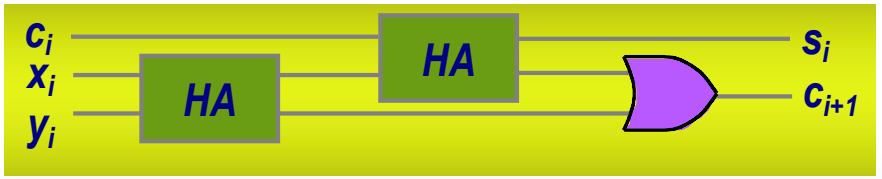
- Full adder composed from two half adders
  - Used to add inputs x, y, c_in
  - Generates outputs s and c_out

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

Truth table



Gate Structure



Composition of Half Adders (HA)

VHDL Code for Full Adder

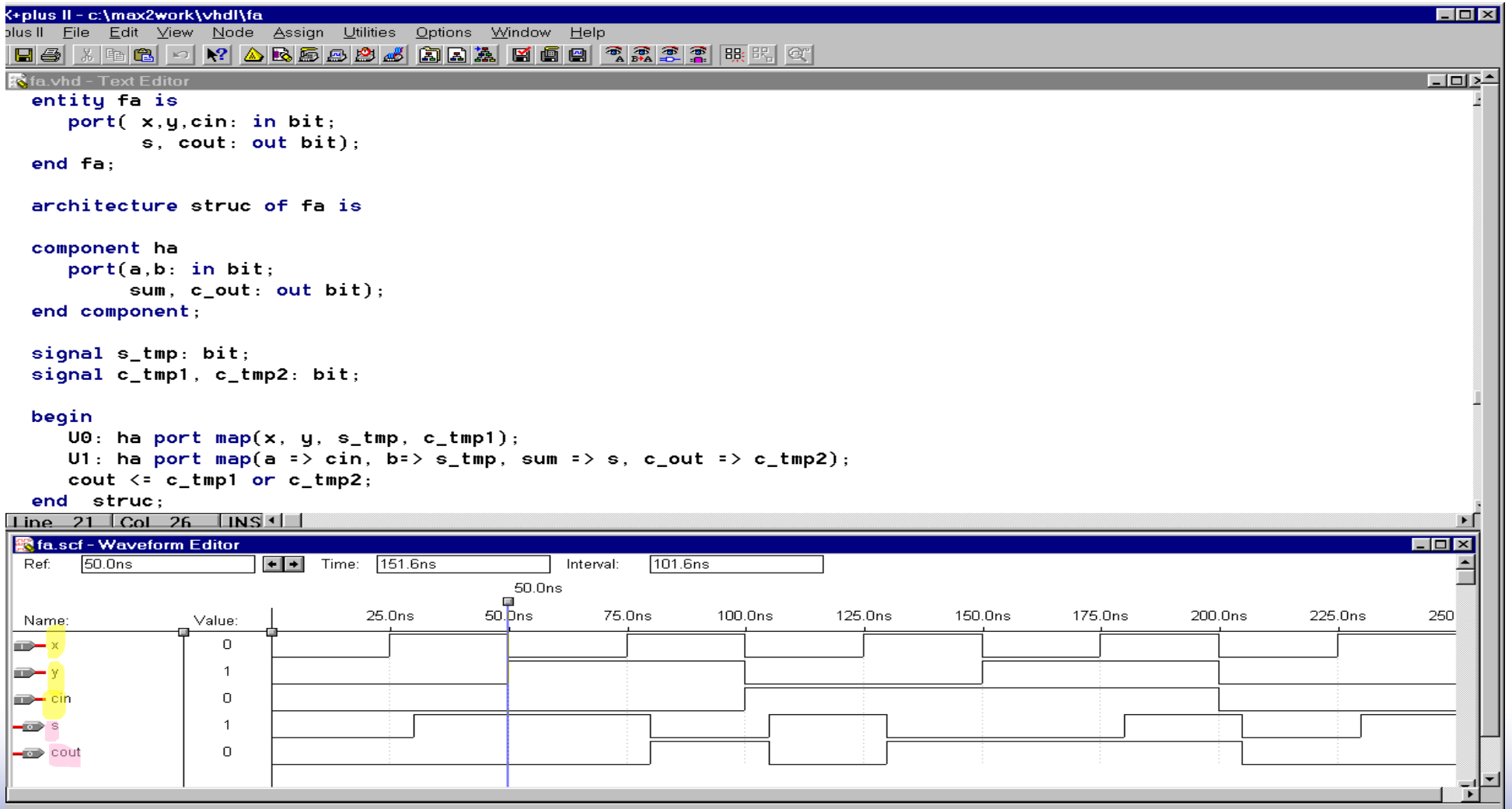
```
library IEEE;
use ieee.std_logic_1164.all;

entity fa is
    port( x, y,cin: in bit;
          s, cout: out bit);
end fa;

architecture struc of fa is
-- component declaration
component ha
    port( a, b: in bit;
          sum, c_out: out bit);
end component;

-- component specification
for U0: ha use entity work.ha(struct);
for U1: ha use entity work.ha(struct);

-- signal declaration
signal s_tmp: bit;
signal c_tmp1, c_tmp2: bit;
```

structural VHDL

```
begin
    -- component instantiation
    U0: ha port map(x, y, s_tmp,   c_tmp1);
    U1: ha port map(b  => s_tmp,
                    a => cin,
                    c_out => c_tmp2,
                    sum => s);

    -- generation of cout through signal assig.
    cout <= c_tmp1 or c_tmp2;
end struc;
```

behavioral VHDL

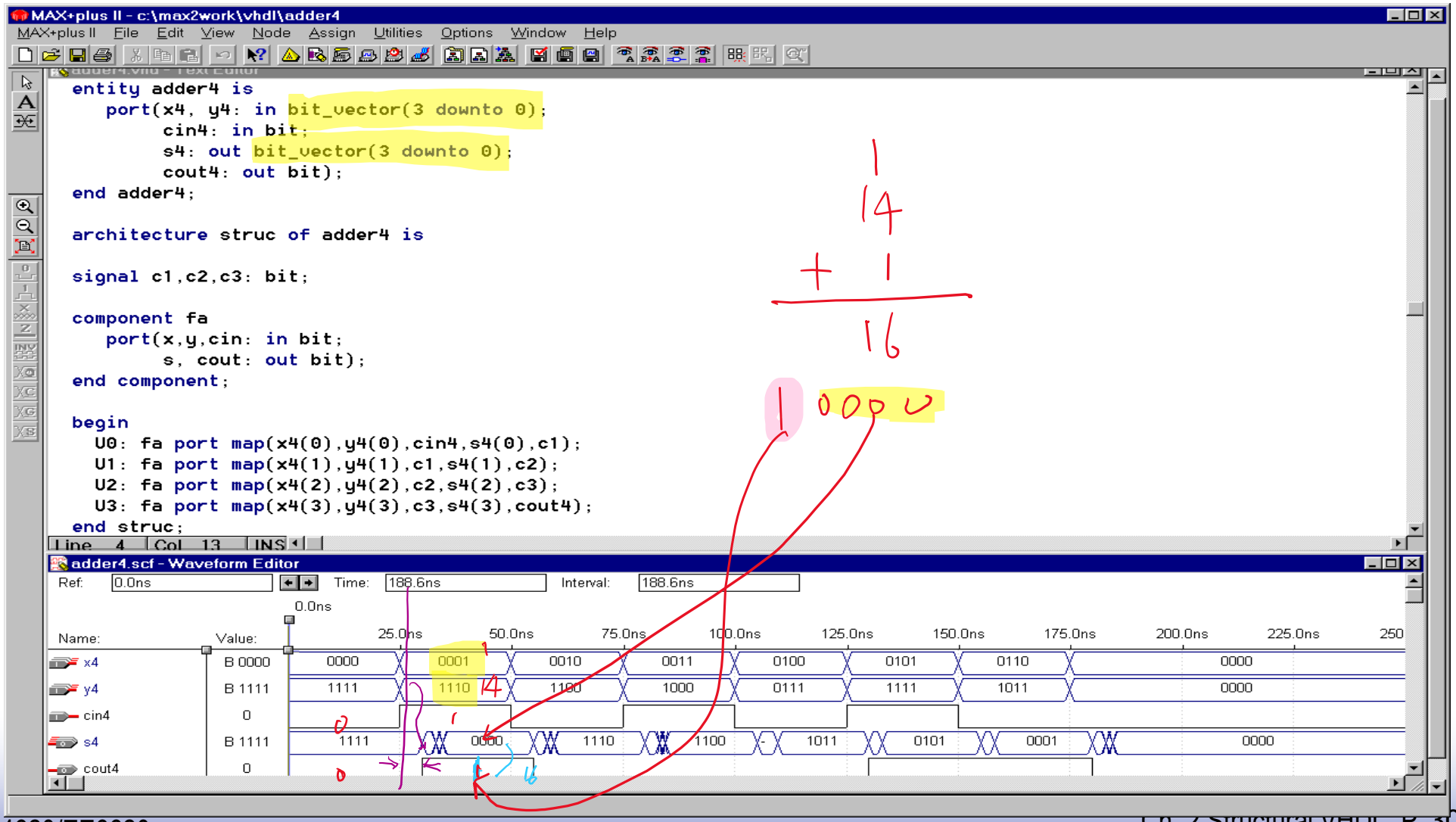
Full Adder - Simulations



Step 3: 4-bit Ripple Adder

```
library IEEE;
use ieee.std_logic_1164.all;

entity adder4 is
    port( x4, y4: in bit_vector(3 downto 0);
          cin4:    in bit;
          s4:      out bit_vector(3 downto 0);
          cout4: out bit);
end adder4;

architecture struc of adder4 is
-- component declaration
component fa
    port(x, y,cin: in bit;
          s, cout: out bit);
end component;

-- component specification
for U0: fa use entity work.fa(struc);
for U1: fa use entity work.fa(struc);
for U2: fa use entity work.fa(struc);
for U3: fa use entity work.fa(struc);

-- signal declaration
signal c1, c2, c3: bit;

begin
    U0: fa port map(x4(0), y4(0), cin4
                    s4(0), c1);
    U1: fa port map(x4(1), y4(1), c1
                    s4(1), c2);
    U2: fa port map(x4(2), y4(2), c2
                    s4(2), c3);
    U3: fa port map(x4(3), y4(3), c3
                    s4(3), cout4);
end struc;
```

4-bit Ripple Adder - Simulations



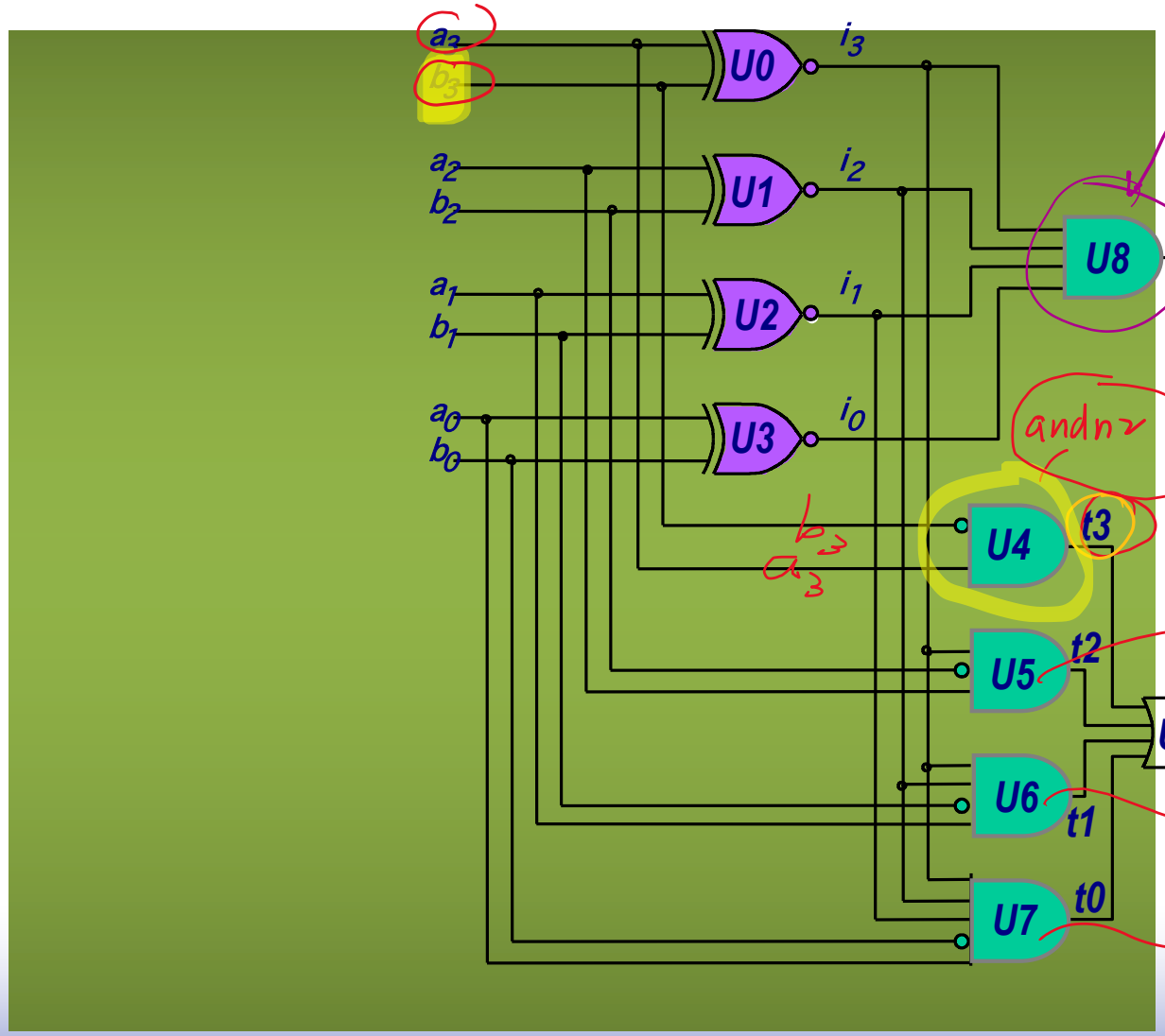
Specification

- Two 4-bit *inputs*: A and B representing unsigned binary numbers
- Three *outputs*:
  - *AeqB* - equal to 1 when A=B, otherwise 0
  - *AgtB* - equal to 1 when A > B, otherwise 0
  - *AltB*- equal to 1 when A < B, otherwise 0

Determining Output Logic Functions

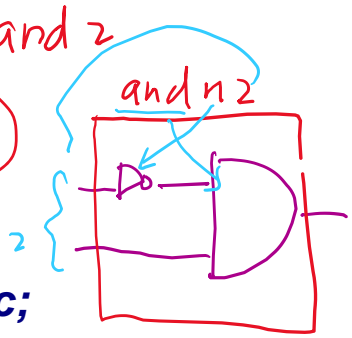
- Inputs: $A = a_3a_2a_1a_0$, $B=b_3b_2b_1b_0$
- Intermediate signals i_3, i_2, i_1, i_0
  - $i_k = 1$ iff when $a_k=b_k$
- $AeqB = i_3*i_2*i_1*i_0$
- $AgtB$
  - Bits of A and B considered from MSB to LSB
  - First bit k, where a_k /= b_k determining $AgtB$
    - a_k=0, b_k=1: $A < B$
    - a_k=1, b_k=0: $A > B$
  - $AgtB = a_3b_3' + i_3a_2b_2' + i_3i_2a_1b_1' + i_3i_2i_1a_0b_0'$
- $AltB = (AeqB + AgtB)'$

- Inputs: $A = a_3a_2a_1a_0$, $B=b_3b_2b_1b_0$
- Intermediate signals i_3, i_2, i_1, i_0
  - i_k = 1 iff when $a_k=b_k$
- $AeqB = i_3*i_2*i_1*i_0$
- $AgtB$
  - Bits of A and B considered from MSB to LSB
  - First bit k, where a_k /= b_k determining $AgtB$
    - a_k=0, b_k=1: $A < B$
    - a_k=1, b_k=0: $A > B$
  - $AgtB = a_3b_3' + i_3a_2b_2' + i_3i_2a_1b_1' + i_3i_2i_1a_0b_0'$
- $AltB = (AeqB + AgtB)'$



VHDL Code for 4-bit Comparator

```
library IEEE;
use ieee.std_logic_1164.all;

entity comp4 is
    port( a, b: in std_logic_vector(3 downto 0);
          aeqb,agtb,altb: out std_logic);
end comp4;

architecture struc of comp4 is
-- signal declaration
 i3, i2, i1, i0: std_logic;
 t3, t2, t1, t0: std_logic;
 aeqb_t, agtb_t: std_logic;

-- component declaration
component xnor2
    port( a, b: in std_logic;
          z: out std_logic);
end component;

component and4
    port( a, b: in std_logic;
          z: out std_logic);
end component;

component and4
    port( a, b: in std_logic;
          z: out std_logic);
end component;

component andn2
    port( a, b: in std_logic;
          z: out std_logic);
end component;
```




```
component andn3
    port( a, b,c: in std_logic;
          z: out std_logic);
end component;

component andn4
    port( a, b,c,d: in std_logic;
          z: out std_logic);
end component;

component andn5
    port( a, b,c,d,e: in std_logic;
          z: out std_logic);
end component;
```

VHDL Code for 4-bit Comparator, cont.

```
component or4
   port( a, b,c,d: in std_logic;
         z: out std_logic);
end component;

component nor2
   port( a, b: in std_logic;
         z: out std_logic);
end component;

-- component specification
for U0: xnor2 use entity work.xnor2(struct);
for U1: xnor2 use entity work.xnor2(struct);
for U2: xnor2 use entity work.xnor2(struct);
for U3: xnor2 use entity work.xnor2(struct);
for U4: andn2 use entity work.andn2(struct);
for U5: andn3 use entity work.andn3(struct);
for U6: andn4 use entity work.andn4(struct);
for U7: andn5 use entity work.andn5(struct);
for U8: and4 use entity work.and4(struct);
for U9: or4 use entity work.or4(struct);
for U10: nor2 use entity work.nor2(struct);
```

```
begin
   U0: xnor2 port map(a(3), b(3), i3);
   U1: xnor2 port map(a => a(2),
                      b => b(2),
                      z => i2);
   U2: xnor2 port map(a(1), b(1), i1);
   U3: xnor2 port map(a(0), b(0), i0);
   U4: andn2 port map(b(3), b(2), t3);
   U5: andn3 port map(i3, b(2), a(2), t2);
   U6: andn4 port map(i3, i2, b(1),
                      a(1), t1);
   U7: andn5 port map(i3, i2, i1, b(0),
                      a(0), t0);

   U8: and4 port map(i3, i2, i1, i0,
                     aeqb_t);
   U9: or4 port map(t3, t2, t1, t0, agtb_t);
   U10: nor2 port map(aeqb_t, agtb_t,
                      altb);

   aeqb <= aeqb_t;
   agtb <= agtb_t;
end struc;
```

Comp4 - Components

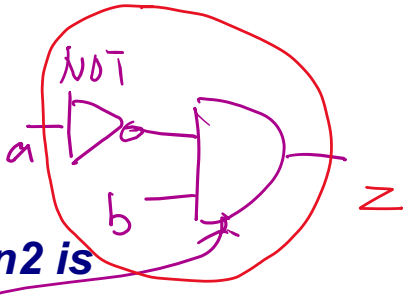
```
library IEEE;
use ieee.std_logic_1164.all;

entity xnor2 is
   port( a, b: in std_logic;
         z: out std_logic);
end xnor2;

architecture struc of xnor2 is
   begin
      z <= a xnor b;
end struc;
```

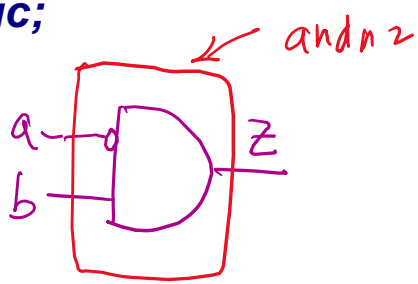
```
library IEEE;
use ieee.std_logic_1164.all;

entity nor2 is
   port( a, b: in std_logic;
         z: out std_logic);
end nor2;

architecture struc of nor2 is
   begin
      z <= a nor b;
end struc;
```

```
library IEEE;
use ieee.std_logic_1164.all;

entity or4 is
   port( a, b,c,d: in std_logic;
         z: out std_logic);
end or4;

architecture struc of or4 is
   begin
      z <= a or b or c or d;
end struc;
```

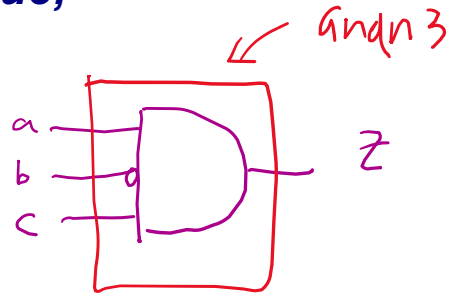
Comp4 - Components, cont.

```
library IEEE;
use ieee.std_logic_1164.all;

entity andn2 is
   port( a, b: in std_logic;
         z: out std_logic);
end andn2;

architecture struc of andn2 is
   begin
      z <= (not a) and b;
end struc;
```






```
library IEEE;
use ieee.std_logic_1164.all;

entity andn3 is
   port( a, b,c: in st_logic;
         z: out std_logic);
end andn3;

architecture struc of andn3 is
   begin
      z <= a and (not b)
           and c;
end struc;
```



Comp4 - Components, cont. 1

```
library IEEE;
use ieee.std_logic_1164.all;

entity andn4 is
   port( a, b,c,d: in st_logic;
         z: out std_logic);
end andn4;

architecture struc of andn4 is
   begin
      z <= a and b
           and (not c) and d;
end struc;
```

```
library IEEE;
use ieee.std_logic_1164.all;

entity andn5 is
   port( a, b,c,d, e: in st_logic;
         z: out std_logic);
end andn5;

architecture struc of andn5 is
   begin
      z <= a and b
           and c and (not d)
           and e;
end struc;
```