COE 405
Structural Specification of Hardware

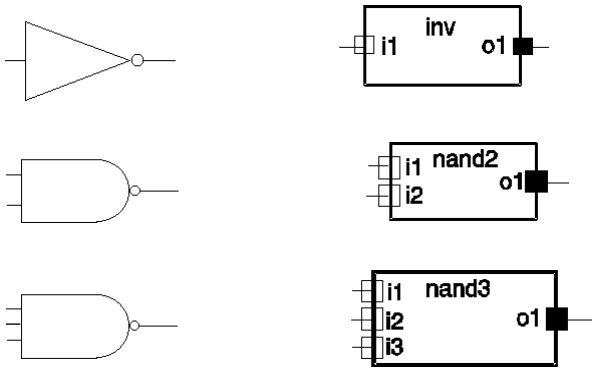
Dr. Aiman H. El-Maleh
Computer Engineering Department
King Fahd University of Petroleum & Minerals

Outline

- **Parts Library**
- **Wiring of Primitives**
  - Single-bit comparator
- **Wiring Iterative Networks**
  - 4-bit comparator
  - For…Generate Statement
  - IF…Generate Statement
- **Modeling a Test Bench**
- **Binding Alternative**
- **Top Down Wiring**
  - Sequential Comparator
  - Byte Latch
  - Byte Comparator

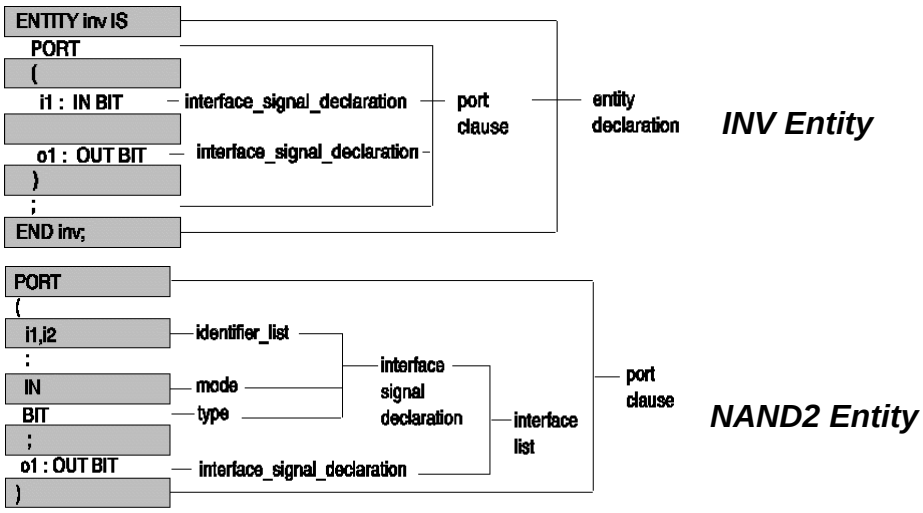
Basic Components & Graphical Notation

- Three basic components are used
- Will use as the basic components of several designs
- A graphical notation helps clarify wiring



2-3

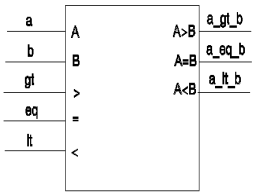
Entity Syntax



2-4

A Cascadable Single-Bit Comparator

- When a > b the a_gt_b becomes 1
- When a < b the a_lt_b becomes 1
- If a = b outputs become the same as corresponding inputs



a,b / >	00	01	11	10
0				1
1	1		1	1

a>b

a,b / =	00	01	11	10
0				
1	1		1	

a=b

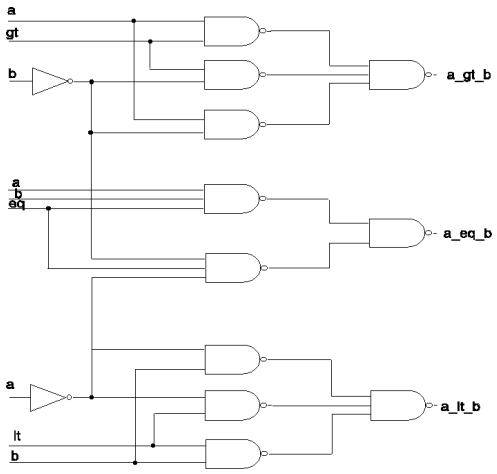
a,b / <	00	01	11	10
0		1		
1	1	1	1	

a<b

2-5

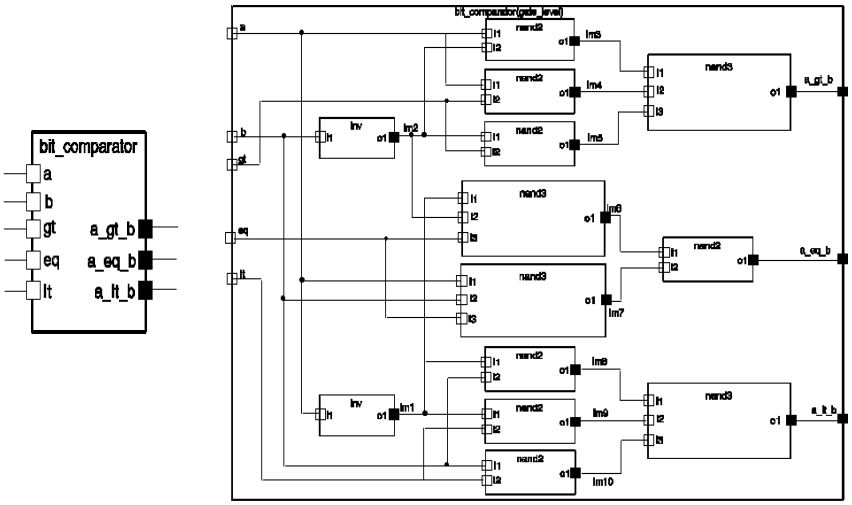
Structural Single-Bit Comparator

- Design uses basic components
- The less-than and greater-than outputs use the same logic



2-6

Structural Single-Bit Comparator: Aspect Notation



2-7

Structural Model of Single-Bit Comparator …

```
ENTITY bit_comparator IS
        PORT (a, b, gt,  eq,  lt : IN BIT;  a_gt_b,  a_eq_b, a_lt_b : OUT BIT);
END bit_comparator;
ARCHITECTURE gate_level OF bit_comparator IS
--
COMPONENT n1 PORT (i1: IN BIT; o1: OUT BIT); END COMPONENT ;
COMPONENT n2 PORT (i1,i2: IN BIT; o1:OUT BIT); END COMPONENT;
COMPONENT n3 PORT (i1, i2, i3: IN BIT; o1: OUT BIT); END COMPONENT;
--  Component Configuration
FOR ALL : n1 USE ENTITY WORK.inv (single_delay);
FOR ALL : n2 USE ENTITY WORK.nand2 (single_delay);
FOR ALL : n3 USE ENTITY WORK.nand3 (single_delay);
--Intermediate signals
SIGNAL im1,im2, im3, im4, im5, im6, im7, im8, im9, im10 : BIT;
```

2-8

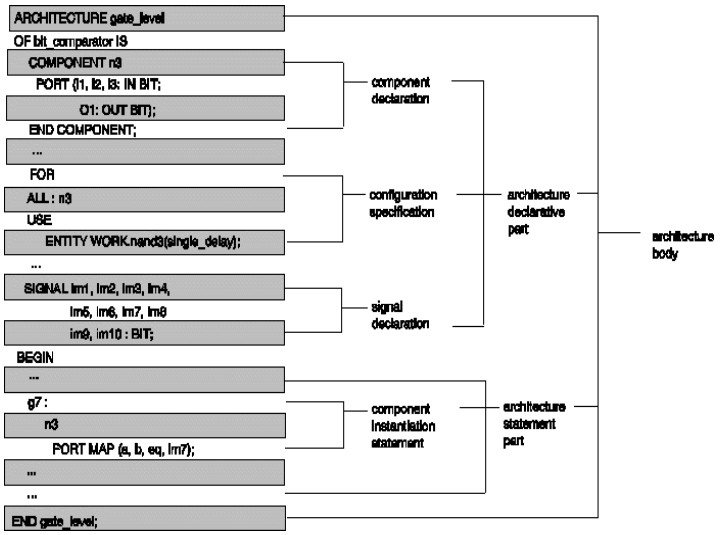
… Structural Model of Single-Bit Comparator

```
BEGIN
-- a_gt_b output
    g0 : n1 PORT MAP (a, im1);
    g1 : n1 PORT MAP (b, im2);
    g2 : n2 PORT MAP (a, im2, im3);
    g3 : n2 PORT MAP (a, gt, im4);
    g4 : n2 PORT MAP (im2, gt, im5);
    g5 : n3 PORT MAP (im3, im4, im5, a_gt_b);
--  a_eq_b output
    g6 : n3 PORT MAP (im1, im2, eq, im6);
    g7 : n3 PORT MAP (a, b, eq, im7);
    g8 : n2 PORT MAP (im6, im7, a_eq_b);
-- a_lt_b output
    g9 : n2  PORT MAP (im1, b, im8);
    g10 : n2 PORT MAP (im1, lt, im9);
    g11 : n2 PORT MAP (b, lt, im10);
    g12 : n3 PORT MAP (im8, im9, im10, a_lt_b);
END gate_level;
```


Netlist Description of Single-Bit Comparator

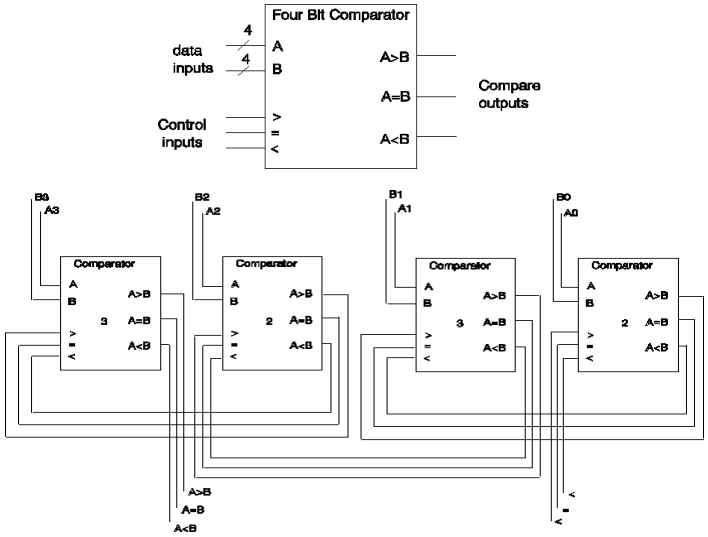
```
ARCHITECTURE netlist OF bit_comparator IS
SIGNAL im1,im2, im3, im4, im5, im6, im7, im8, im9, im10 : BIT;
BEGIN
-- a_gt_b output
    g0 : ENTITY Work.inv(single_delay)  PORT MAP (a, im1);
    g1 : ENTITY Work.inv(single_delay) PORT MAP (b, im2);
    g2 : ENTITY Work.nand2(single_delay) PORT MAP (a, im2, im3);
    g3 : ENTITY Work.nand2(single_delay) PORT MAP (a, gt, im4);
    g4 : ENTITY Work.nand2(single_delay) PORT MAP (im2, gt, im5);
    g5 : ENTITY Work.nand3(single_delay) PORT MAP (im3, im4, im5, a_gt_b);
--  a_eq_b output
    g6 : ENTITY Work.nand3(single_delay) PORT MAP (im1, im2, eq, im6);
    g7 : ENTITY Work.nand3(single_delay) PORT MAP (a, b, eq, im7);
    g8 : ENTITY Work.nand2(single_delay) PORT MAP (im6, im7, a_eq_b);
-- a_lt_b output
    g9 : ENTITY Work.nand2(single_delay) PORT MAP (im1, b, im8);
    g10 : ENTITY Work.nand2(single_delay) PORT MAP (im1, lt, im9);
    g11 : ENTITY Work.nand2(single_delay) PORT MAP (b, lt, im10);
    g12 : ENTITY Work.nand3(single_delay) PORT MAP (im8, im9, im10, a_lt_b);
END netlist;
```

Syntax Details



2-11

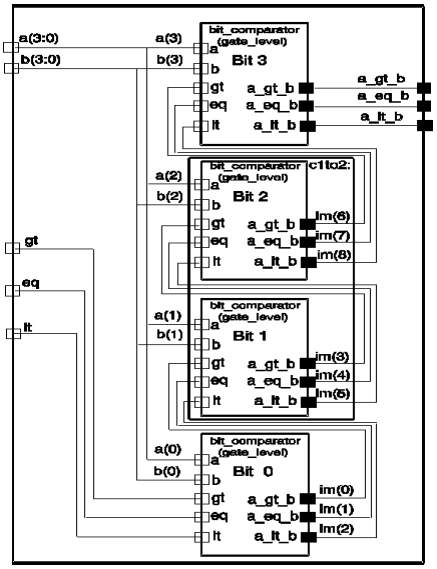
4-Bit Comparator Structural Iterative Wiring



2-12

4-Bit Comparator Aspect Notation

- Aspect notation shows VHDL wiring
- VHDL description will use same intermediate signal names



2-13

4-Bit Comparator: "For ……. Generate" Statement …

```
ENTITY nibble_comparator IS
   PORT (a, b : IN BIT_VECTOR (3 DOWNTO 0); -- a and b data inputs
   gt, eq,  lt : IN BIT;  -- previous greater, equal & less than
   a_gt_b, a_eq_b, a_lt_b : OUT BIT);  -- a > b,  a = b, a < b
END nibble_comparator;
   --
ARCHITECTURE iterative OF nibble_comparator IS

   COMPONENT comp1
        PORT (a, b, gt, eq, lt : IN BIT; a_gt_b, a_eq_b, a_lt_b : OUT BIT);
   END COMPONENT;
   FOR ALL : comp1 USE ENTITY WORK.bit_comparator (gate_level);
   SIGNAL im : BIT_VECTOR ( 0 TO 8);
BEGIN

   c0: comp1 PORT MAP (a(0), b(0), gt, eq, lt, im(0), im(1), im(2));
```

2-14

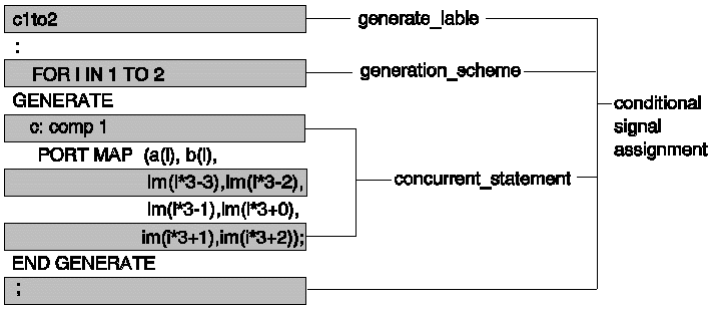
… 4-Bit Comparator: "For ……. Generate" Statement

```
c1to2: FOR i IN 1 TO 2 GENERATE

    c: comp1 PORT MAP ( a(i), b(i), im(i*3-3), im(i*3-2), im(i*3-1),
im(i*3+0), im(i*3+1), im(i*3+2) );

END GENERATE;
c3: comp1 PORT MAP (a(3), b(3), im(6), im(7), im(8), a_gt_b,
a_eq_b, a_lt_b);

END iterative;
```

- USE BIT_VECTOR for Ports a & b
- Separate first and last bit-slices from others
- Arrays FOR intermediate signals facilitate iterative wiring
- Can easily expand to an n-bit comparator

2-15

Syntax of Generate Statement



- Generate statement is a concurrent statement
- Generate statement brackets concurrent statements

2-16

4-Bit Comparator: "IF …… Generate" Statement …

```
ARCHITECTURE iterative OF nibble_comparator IS
   --
   COMPONENT comp1
        PORT (a, b, gt, eq, lt : IN BIT; a_gt_b, a_eq_b, a_lt_b : OUT BIT);
   END COMPONENT;
   --
   FOR ALL : comp1 USE ENTITY WORK.bit_comparator (gate_level);
   CONSTANT n : INTEGER := 4;
   SIGNAL im :  BIT_VECTOR ( 0 TO  (n-1)*3-1);
   --
BEGIN
   c_all: FOR i IN 0 TO n-1 GENERATE


 l: IF i = 0 GENERATE
   least: comp1 PORT MAP (a(i), b(i), gt, eq, lt, im(0), im(1), im(2) );
   END GENERATE;
```

2-17

… 4-Bit Comparator: "IF …… Generate" Statement

```
--
  m: IF i = n-1 GENERATE
       most: comp1 PORT MAP (a(i), b(i), im(i*3-3), im(i*3-2),
                                  im(i*3-1), a_gt_b, a_eq_b, a_lt_b);
      END GENERATE;
--
   r: IF i > 0 AND i < n-1 GENERATE
   rest: comp1 PORT MAP (a(i), b(i), im(i*3-3), im(i*3-2),
                           im(i*3-1), im(i*3+0), im(i*3+1), im(i*3+2) );
   END GENERATE;
--
END GENERATE; -- Outer Generate
END iterative;
```

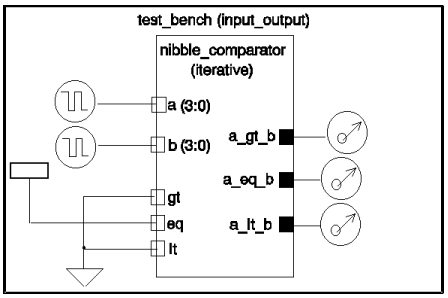
2-18

4-Bit Comparator: Alternative Architecture (Single Generate)

```
ARCHITECTURE Alt_iterative OF nibble_comparator IS
constant n: Positive :=4;
COMPONENT comp1
    PORT (a, b, gt, eq, lt : IN BIT; a_gt_b, a_eq_b, a_lt_b : OUT BIT);
END COMPONENT;
FOR ALL : comp1 USE ENTITY WORK.bit_comparator (gate_level);
SIGNAL im : BIT_VECTOR ( 0 TO 3*n+2);
BEGIN
im(0 To 2) <= gt&eq&lt;
cALL: FOR i IN 0 TO n-1 GENERATE
c: comp1 PORT MAP (a(i), b(i), im(i*3), im(i*3+1), im(i*3+2),
im(i*3+3), im(i*3+4), im(i*3+5) );
END GENERATE;
a_gt_b <= im(3*n);
a_eq_b  <= im(3*n+1);
a_lt_b  <= im(3*n+2);
END Alt_iterative ;
```


Structural Test Bench

- **A Testbench is an Entity without Ports that has a Structural Architecture**
- **The Testbench Architecture, in general, has 3 major components:**
  - Instance of the Entity Under Test (EUT)
  - Test Pattern Generator (Generates Test Inputs for the Input Ports of the EUT)
  - Response Evaluator (Compares the EUT Output Signals to the Expected Correct Output)



Testbench Example …

```
Entity nibble_comparator_test_bench IS
End nibble_comparator_test_bench ;
--
ARCHITECTURE input_output OF nibble_comparator_test_bench IS
--
COMPONENT comp4 PORT (a, b : IN bit_vector (3 DOWNTO 0);
gt, eq, lt : IN BIT;
a_gt_b, a_eq_b, a_lt_b : OUT BIT);
END COMPONENT;
--
FOR a1 : comp4 USE ENTITY WORK.nibble_comparator(iterative);
--
SIGNAL a, b : BIT_VECTOR (3 DOWNTO 0);
SIGNAL eql, lss, gtr, gnd : BIT;
SIGNAL vdd : BIT := '1';
--
BEGIN
a1: comp4 PORT MAP (a, b, gnd, vdd, gnd, gtr, eql, lss);
--
```

2-21

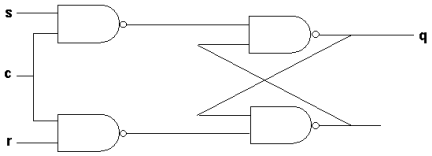
…Testbench Example

```
a2: a <= "0000",            -- a = b (steady state)
"1111" AFTER 0500 NS, -- a > b (worst case)
"1110" AFTER 1500 NS, -- a < b (worst case)
"1110" AFTER 2500 NS, -- a > b (need bit 1 info)
"1010" AFTER 3500 NS, -- a < b (need bit 2 info)
"0000" AFTER 4000 NS, -- a < b (steady state, prepare FOR next)
"1111" AFTER 4500 NS, -- a = b (worst case)
"0000" AFTER 5000 NS, -- a < b (need bit 3 only, best case)
"0000" AFTER 5500 NS, -- a = b (worst case)
"1111" AFTER 6000 NS; -- a > b (need bit 3 only, best case)
--
a3 : b <= "0000",           -- a = b (steady state)
"1110" AFTER 0500 NS, -- a > b (worst case)
"1111" AFTER 1500 NS, -- a < b (worst case)
"1100" AFTER 2500 NS, -- a > b (need bit 1 info)
"1100" AFTER 3500 NS, -- a < b (need bit 2 info)
"1101" AFTER 4000 NS, -- a < b (steady state, prepare FOR next)
"1111" AFTER 4500 NS, -- a = b (worst case)
"1110" AFTER 5000 NS, -- a < b (need bit 3 only, best case)
"0000" AFTER 5500 NS, -- a = b (worst case)
"1110" AFTER 6000 NS; -- a > b (need bit 3 only, best case)
END input_output;
```

2-22

Different Binding Schemes

USE 4 2-Input NAND gates to design an SR latch




```
Entity sr_latch IS
    Port (S, R, C : IN Bit; q : Out Bit);
END sr_latch;

ARCHITECTURE Wrong OF sr_latch IS
COMPONENT n2
    PORT (i1, i2: IN BIT; o1: OUT BIT);
END COMPONENT;
FOR ALL : n2 USE ENTITY WORK.nand2 (single_delay);

SIGNAL im1, im2, im4 : BIT;

BEGIN
g1 : n2 PORT MAP (s, c, im1);
g2 : n2 PORT MAP (r, c, im2);
g3 : n2 PORT MAP (im1, im4, q);
g4 : n2 PORT MAP (q, im2, im4); -- Error … q  declared as Output
END Wrong;
```


SR Latch Modeling …

Fix #1 *(Same Architecture -- Change mode of q to Inout)*

```
Entity sr_latch IS
Port (S, R, C : IN Bit; q : InOut Bit);
END sr_latch;
```

Fix #2 *(Same Architecture -- Change mode of q to Buffer)*

```
Entity sr_latch IS
Port (S, R, C : IN Bit; q : Buffer Bit);
END sr_latch;
```

… SR Latch Modeling …

Fix #3 *(Use Local Signal—Mode of q is maintained as Out)*

```
ARCHITECTURE Problem OF sr_latch IS
COMPONENT n2
 PORT (i1, i2: IN BIT; o1: OUT BIT);
END COMPONENT;
FOR ALL : n2 USE ENTITY WORK.nand2 (single_delay);
SIGNAL im1, im2, im3, im4 : BIT;
BEGIN
g1 : n2 PORT MAP (s, c, im1);
g2 : n2 PORT MAP (r, c, im2);
g3 : n2 PORT MAP (im1, im4, im3);
g4 : n2 PORT MAP (im3, im2, im4);
q <= im3;
END Problem;
```

- **Correct Syntax → Problem Oscillating**
- **If all delays are equal then (0,0) on (q, qbar) will oscillate**
- **Remedy by using gates of different delay values**

2-25

… SR Latch Modeling

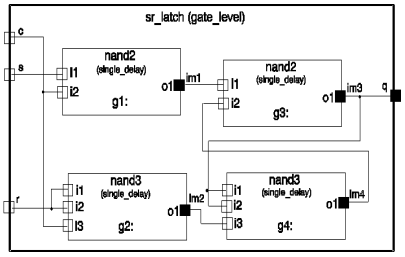
```
ARCHITECTURE fast_single_delay OF nand2 IS
BEGIN
    o1 <= i1 NAND i2 AFTER 1 NS;
END fast_single_delay;
```

```
ARCHITECTURE gate_Level OF sr_latch IS
COMPONENT n2
  PORT (i1, i2: IN BIT; o1: OUT BIT);
END COMPONENT;
FOR g1, g3 : n2 USE ENTITY WORK.nand2 (fast_single_delay);
FOR g2, g4 : n2 USE ENTITY WORK.nand2 (single_delay);
SIGNAL im1, im2, im3, im4 : BIT;
BEGIN
g1 : n2 PORT MAP (s, c, im1);
g2 : n2 PORT MAP (r, c, im2);
g3 : n2 PORT MAP (im1, im4, im3);
g4 : n2 PORT MAP (im3, im2, im4);
q <= im3;
END gate_Level;
```

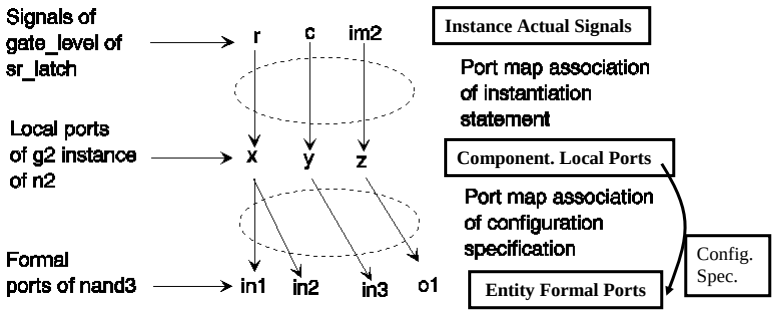
2-26

Binding 3-Input NAND Entity (*Different Delay*) to 2-Input "NAND" Component

```
ARCHITECTURE gate_level OF sr_latch IS
COMPONENT n2
    PORT (x, y: in BIT; z: out BIT);
END COMPONENT;
FOR g1, g3 : n2 USE ENTITY WORK.nand2
(single_delay) PORT MAP (x, y, z);
FOR g2, g4 : n2 USE ENTITY WORK.nand3
(single_delay) PORT MAP (x, x, y, z);
SIGNAL im1, im2, im3, im4 : BIT;
BEGIN
g1 : n2 PORT MAP (s, c, im1);
g2 : n2 PORT MAP (r, c, im2);
g3 : n2 PORT MAP (im1, im4, im3);
g4 : n2 PORT MAP (im3, im2, im4);
q <= im3;
END gate_level;
```



Port Map Association …



14

… Port Map Association

- **Association is done in two steps:**
- **Instance-To-Component**
  (Actuals → Component Local Ports)
- **Componenet-To-Entity (Configuration Port Map)**
  (Component Local Ports → Entity Formal Ports)
  - Specifying PORT MAP in Configuration Statements is Optional
  - PORT MAP of COMPONENT declaration is the default
  - To Override Component Local Port Names, USE PORT MAP in configuration declaration
  - IF No Port Map is Specified in Configuration Statement, Local Port Names of COMPONENT declaration are the Default and they must be the Same as the Formals of the Design Entity.

2-29

Default Binding

- ***Default Binding*: Component instance is Bound to an Entity which:**
  - Has the same NAME as the Component
  - Has the same number of Ports as the Component
  - *Ports* must have the *Same Name*, *Type* and be of *Compatible modes*
  - The most Recently analyzed Architecture will Be used
  - A declared **Signal** can be associated with A Formal Port of **Any Mode** as long as it has the **Same Type**
- ***Port Compatibility:* Formal => Actual**

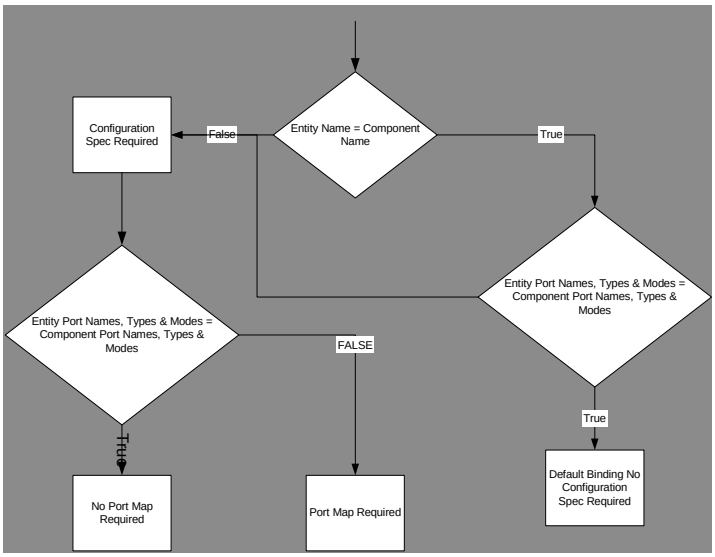
Actual`s Mode (Instance Ports)

Formal`s Mode Entity Formal Ports	**IN**	**OUT**	**INOUT**	**BUFFER**
IN	**X**		**X**	
OUT		**X**	**X**	
INOUT			**X**	
BUFFER				**X**

2-30

15

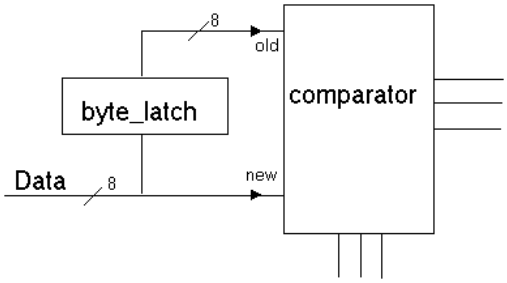
Use of Configuration Specifications



2-31

Sequential Comparator …

- **Compare consecutive sets of data on an 8-bit input bus.**
- **Data on the input bus are synchronized with a clock signal (new value on falling edge of clock)**

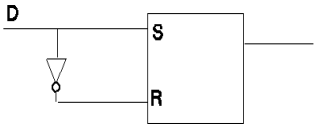


2-32

16

D-Latch

```
ENTITY d_latch IS
  PORT (d, c : IN BIT; q: OUT BIT);
END d_latch;
--
ARCHITECTURE sr_based OF d_latch IS
COMPONENT sr_latch
    PORT (s, r, C : IN BIT; q : OUT BIT);
END COMPONENT;
COMPONENT inv
    PORT (i1 : IN BIT; o1 : OUT BIT);
END COMPONENT;
SIGNAL dbar: BIT;
BEGIN
   c1 : sr_latch PORT MAP (d, dbar, c, q);
   c2 : inv PORT MAP (d, dbar);
END sr_based;
```



2-33

Byte Latch

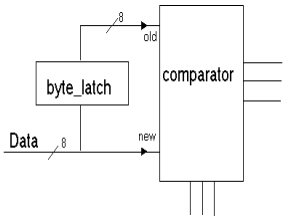
```
ENTITY byte_latch IS
PORT ( di :  IN BIT_VECTOR (7 DOWNTO 0);   clk : IN BIT; qo : OUT
    BIT_VECTOR( 7 DOWNTO 0));
END byte_latch;
--
ARCHITECTURE iterative OF byte_latch IS
COMPONENT d_latch
PORT (d, c : IN BIT; q : OUT BIT);
END COMPONENT;
BEGIN
g : FOR i IN di'RANGE GENERATE
          L7DT0 : d_latch PORT MAP (di(i), clk, qo(i));
END GENERATE;
END iterative;
```

2-34

17

… Sequential Comparator …

```
ENTITY old_new_comparator IS
PORT (i : IN BIT_VECTOR (7 DOWNTO 0);
clk : IN BIT; compare : OUT BIT);
END old_new_comparator;
--
ARCHITECTURE wiring Of old_new_comparator IS
COMPONENT byte_latch PORT (di : IN
BIT_VECTOR (7 DOWNTO 0); clk : IN BIT;
qo : OUT BIT_VECTOR (7 DOWNTO 0));
END COMPONENT;
COMPONENT byte_comparator
     PORT (a, b : BIT_VECTOR (7 DOWNTO 0);
gt, eq, lt : IN BIT; a_gt_b, a_eq_b, a_lt_b : OUT BIT);
END COMPONENT;
SIGNAL con1 : BIT_VECTOR (7 DOWNTO 0);
SIGNAL vdd : BIT := '1';
SIGNAL gnd : BIT := '0';
```



2-35

… Sequential Comparator

```
BEGIN
l : byte_latch PORT MAP (i, clk, con1);
c : byte_comparator
  PORT MAP (con1, i, gnd, vdd, gnd, OPEN, compare, OPEN);
END wiring;
```

- *Unconnected Outputs → OPEN*
- *Unconnected Inputs (Only with Default Specified) → OPEN*

2-36

18