ECE381(CAD), Lecture 5:

Structural Design of Hardware in VHDL

Mehdi Modarressi

Department of Electrical and Computer Engineering,

University of Tehran

Pictures and examples are taken from the slides of "VHDL: Analysis & Modeling of Digital Systems"

Levels of abstraction

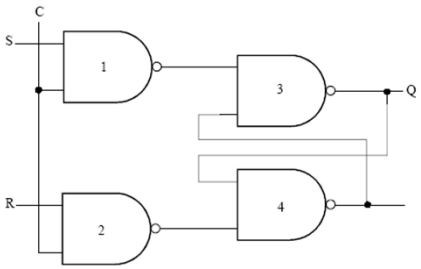
- **Structural**
- Data flow
- Behavioral

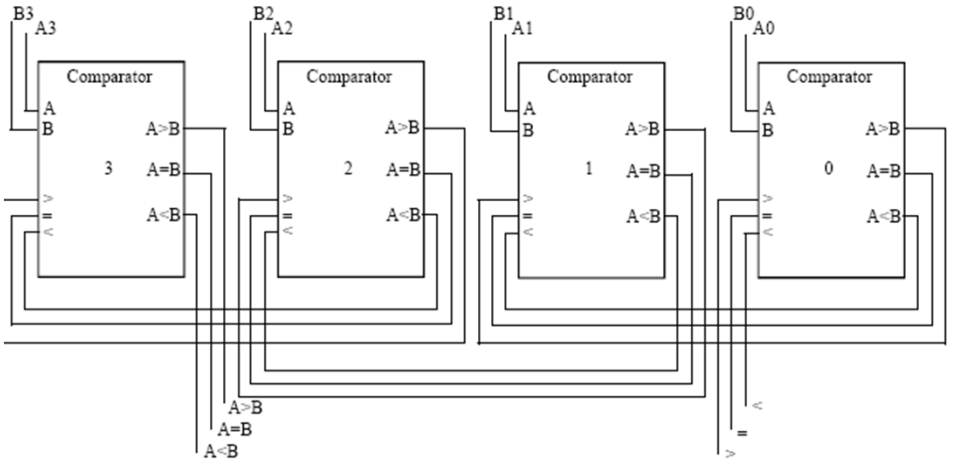
- Readings:
  1. "VHDL: Analysis & Modeling of Digital Systems": Chapter 5.1 to 5.4
  2. "VHDL by Example": pages 1to 6
  3. "Designers Guide To VHDL": chapter 5

  → Slides are based on reading 1

Structural design

- Describe the components and connections
- The *netlist* is generated by programmer
- Can be done at different levels:
  - Interconnecting simple gates
  - Interconnecting more complex components



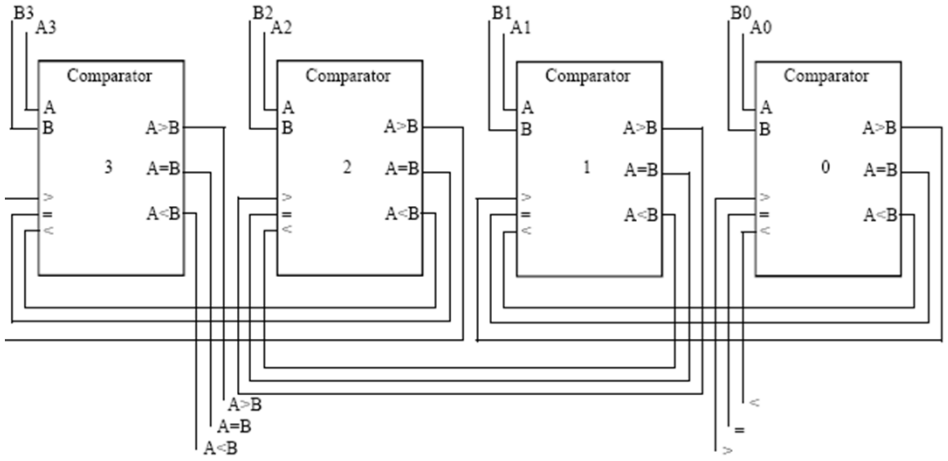


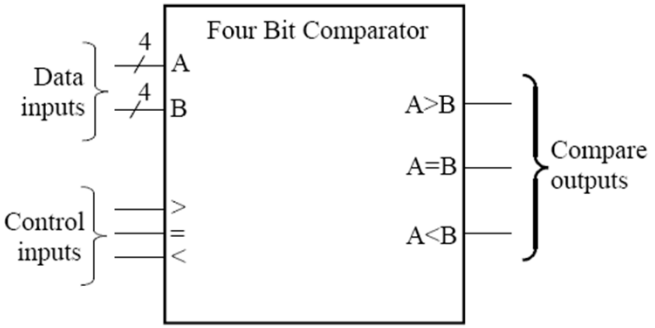
Structural design in VHDL

- Design a 4-bit comparator
- A hierarchical approach:
  - Top-down design:
    - Divide the 4-bit comparator into 4 1-bit comparators
    - Divide the 4-bit comparator into some simple gates
  - Bottom-up implementation
    - Implement simple gets
    - Interconnect gates to get a 1-bit comparator
    - Interconnect 1-bt comparators to get a 4-bit comparator

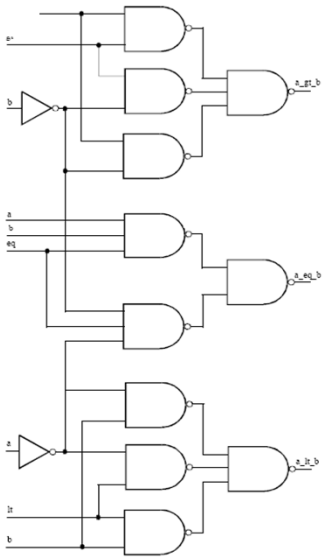
4-bit comparator top-down design

4-bit comparator





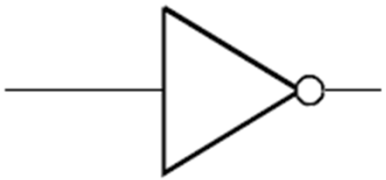
1-bit comparator- Netlist



Implementing simple gates

- We need 3 gates: NOT, 2-input and 3-input NAND
- Very basic dataflow structure to implement the gates
  - More dataflow structures in VHDL and coding techniques will be introduced in future lectures!

NOT



(a)

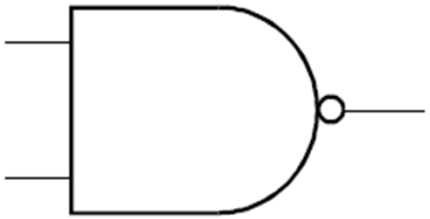
```
ENTITY inv IS
  PORT (i1 : IN BIT; o1 : OUT BIT);
END inv;
```

(b)

```
ARCHITECTURE single_delay OF inv IS
BEGIN
  o1 <= NOT i1 AFTER 4 NS;
END single_delay;
```

(c)

NAND



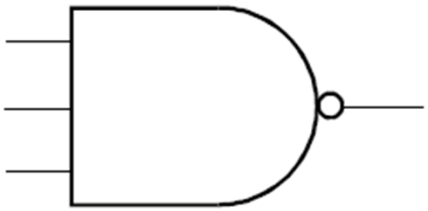
(a)

```
ENTITY nand2 IS
  PORT (i1, i2 : IN BIT; o1 : OUT BIT);
END nand2;
```

(b)

```
ARCHITECTURE single_delay OF nand2 IS
BEGIN
  o1 <= i1 NAND i2 AFTER 5 NS;
END single_delay;
```

(c)



(a)

```
ENTITY nand3 IS
  PORT (i1, i2, i3 : IN BIT; o1 : OUT BIT);
END nand3;
```

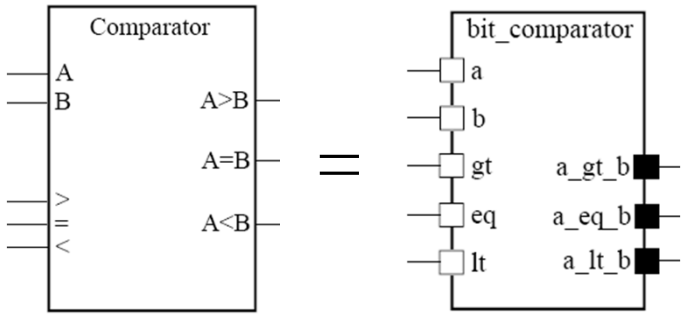
(b)

```
ARCHITECTURE single_delay OF nand3 IS
BEGIN
  o1 <= NOT ( i1 AND i2 AND i3 ) AFTER 6 NS;
END single_delay;
```

(c)

1-bit comparator

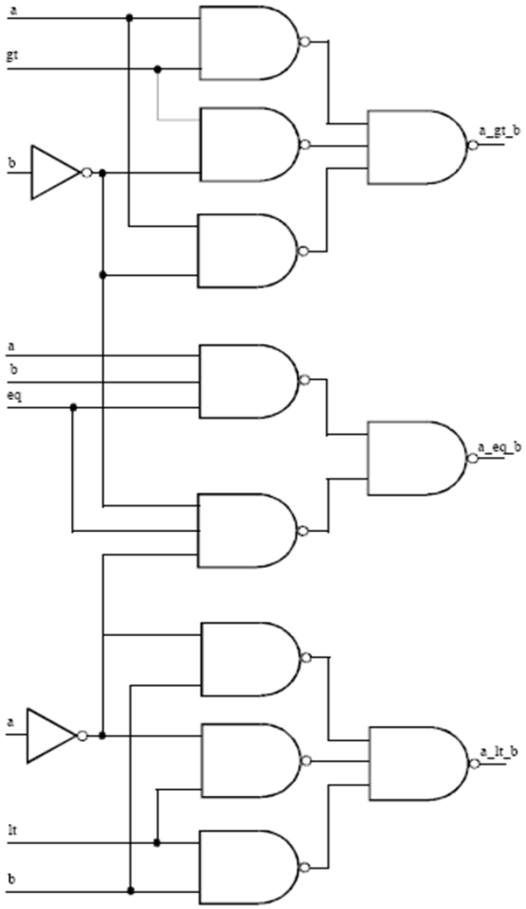
- Entity declaration of 1-bit comparator



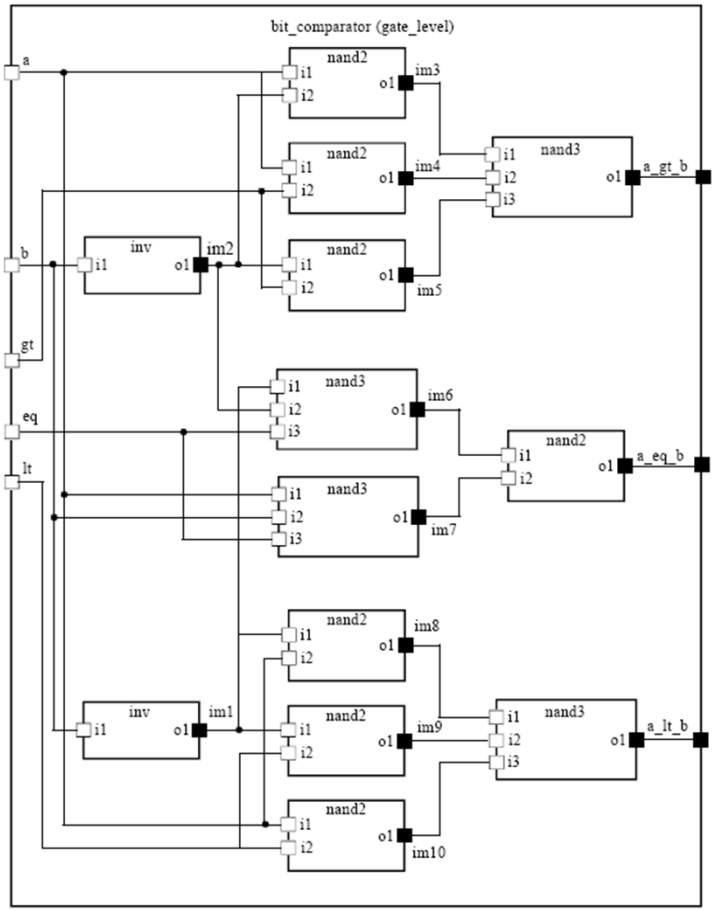

```
ENTITY bit_comparator IS
  PORT (a, b,                    -- data inputs
        gt,                      -- previous greater than
        eq,                      -- previous equal
        lt : IN BIT;             -- previous less than
        a_gt_b,                  -- greater
        a_eq_b,                  -- equal
        a_lt_b : OUT BIT);       -- less than
END bit_comparator;
```

1-bit comparator

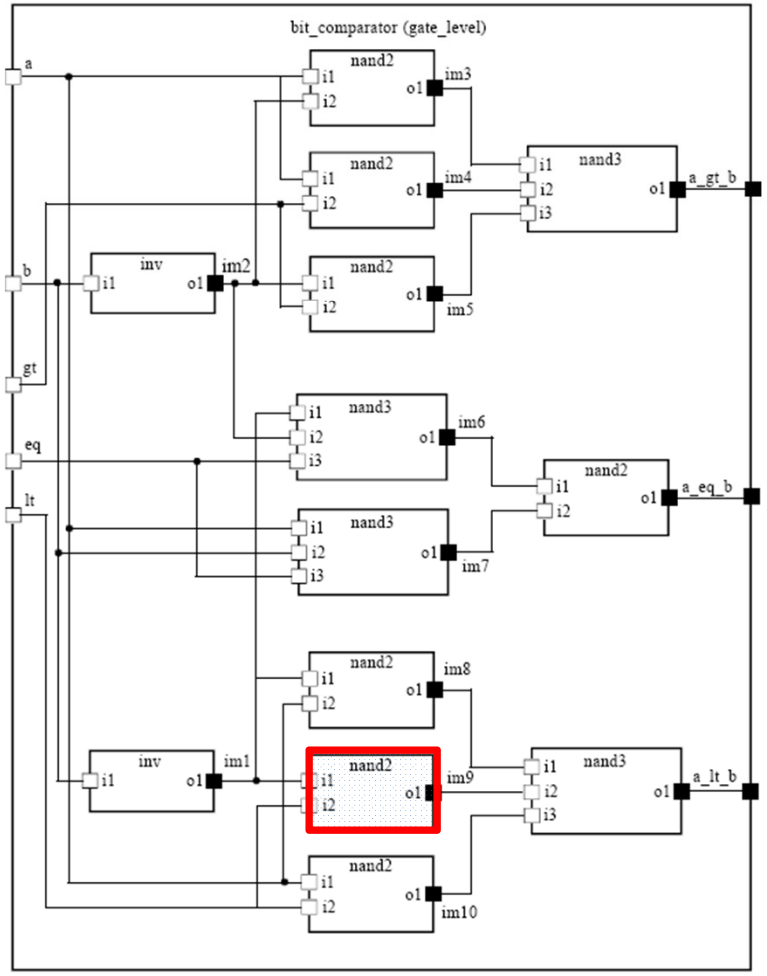
The netlist of 1-bit comparator



We have designed all required components



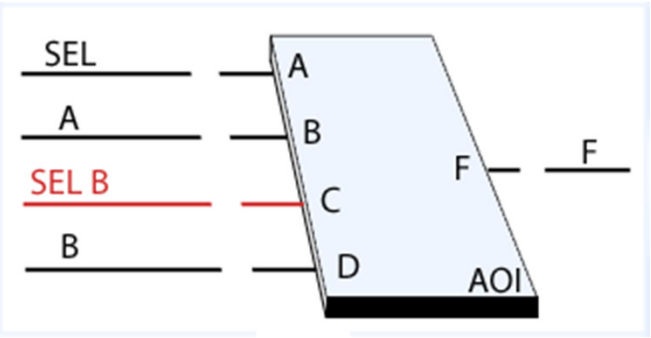
1-bit comparator

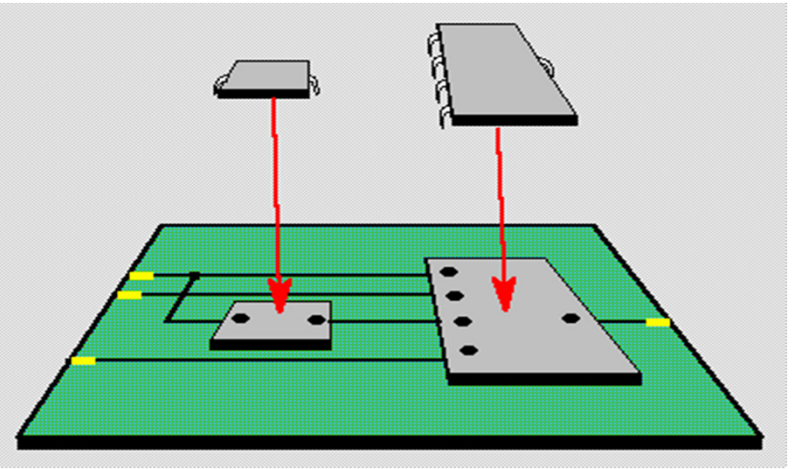



```
ARCHITECTURE gate_level OF bit_comparator IS
  COMPONENT n1 PORT (i1: IN BIT; o1: OUT BIT);
    END COMPONENT;
  COMPONENT n2 PORT (i1, i2: IN BIT; o1: OUT BIT);
    END COMPONENT;
  COMPONENT n3 PORT (i1, i2, i3: IN BIT; o1: OUT BIT);
    END COMPONENT;
  FOR ALL : n1 USE ENTITY WORK.inv (single_delay);
  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;
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;
```

Structural design

- Configuration part of the architecture
  - Specify which entity should be used
- If architecture name is not specified, the most recently compiled architecture will be used
- Use keyword *ALL* to map the same entity to all instances of a component
- Use the labels to use different entities (or the same entity with different architectures) for the same component type

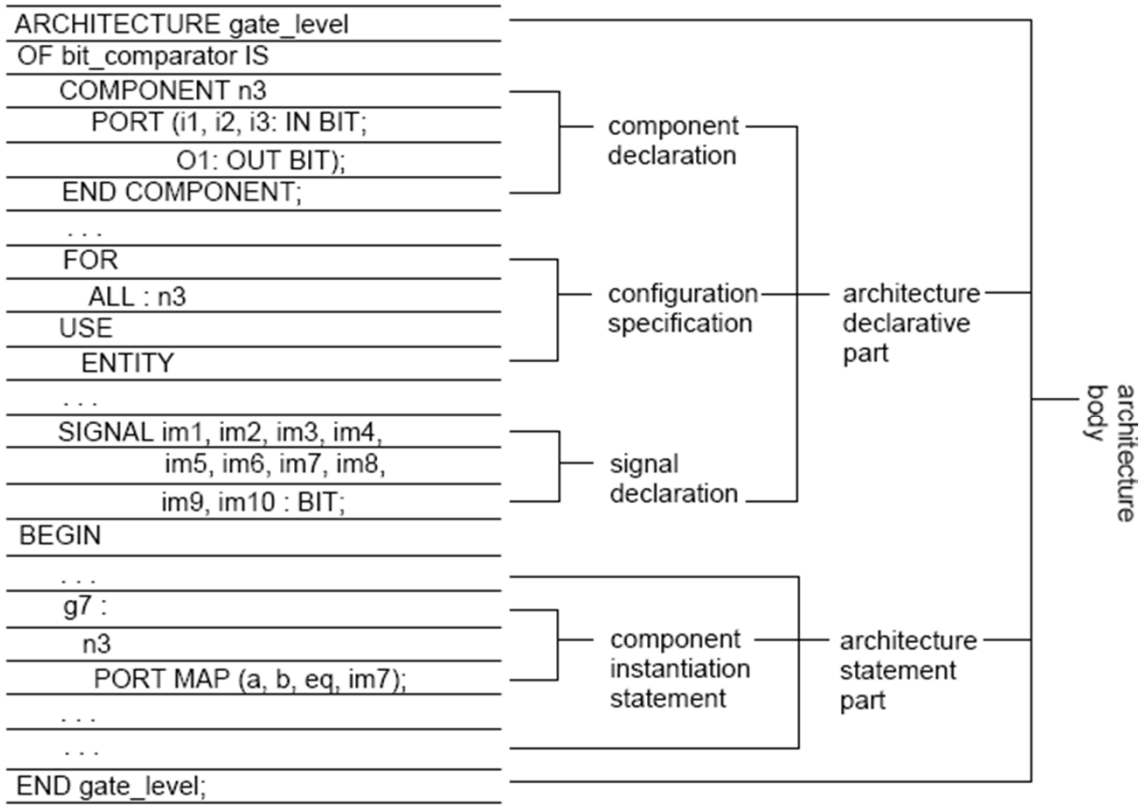




Source of pictures: http://www.doulos.com/

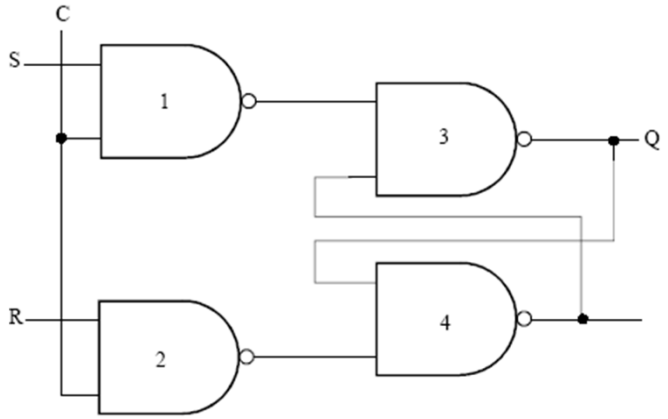
Structural design

- Define components
- Interconnect the components
- Specify which entity (e.g., the entities previously designed by the programmer) should be placed on each component



Example- SR Latch

```
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;
```



- Which architecture is functional?

```
ARCHITECTURE gate_level 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 gate_level;
```

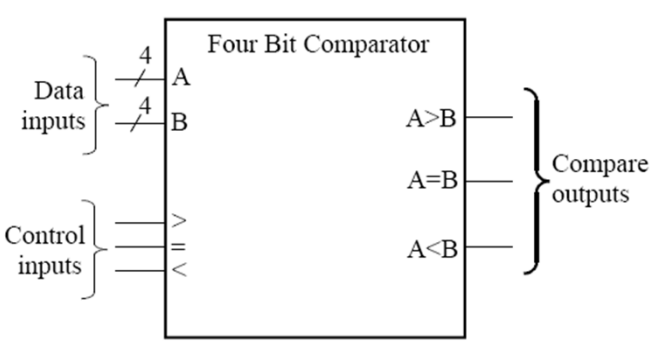
Structural design

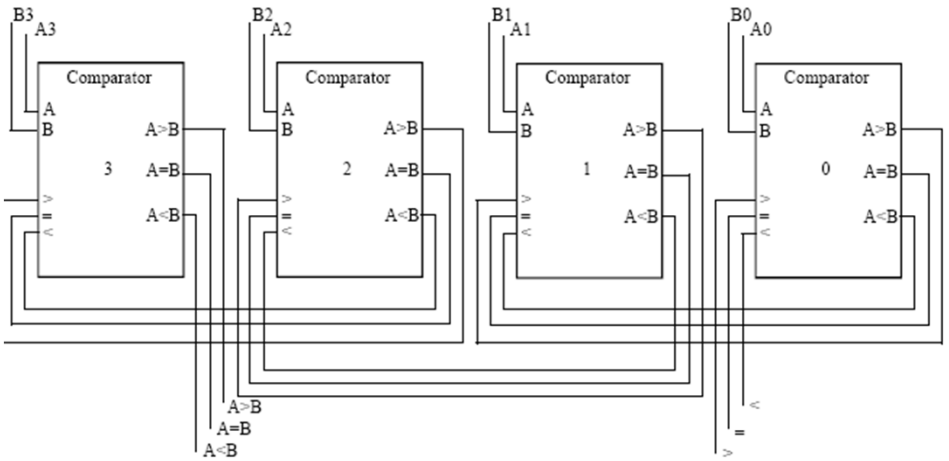
- An alternative structural description
- Direct entity instantiation
- No need to use components

```
ARCHITECTURE netlist OF bit_comparator IS
  -- Intermediate signals
  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;
```

4-bit comparator

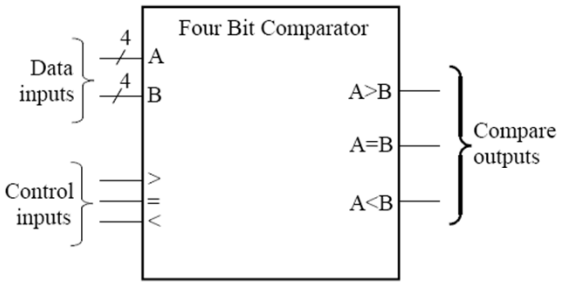
- Logical symbol and structure of a 4-bit comparator





4-bit comparator

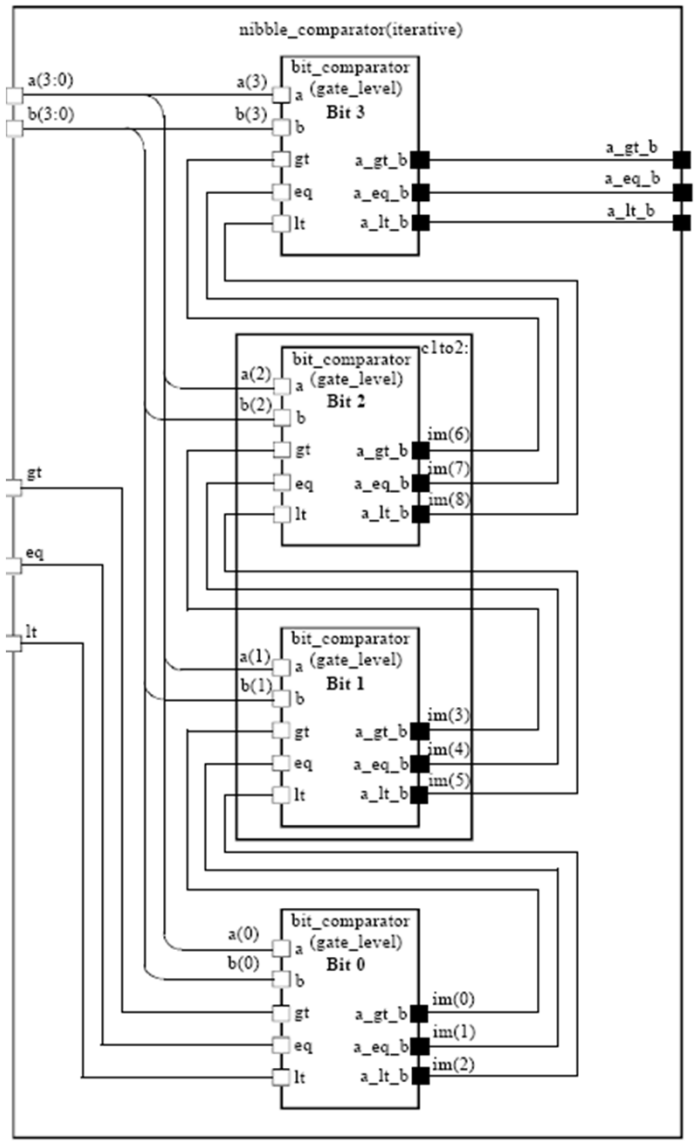
- Inputs of BIT_VECTOR type
  - VHDL standard type
  - Array of bits




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

4-bit comparator

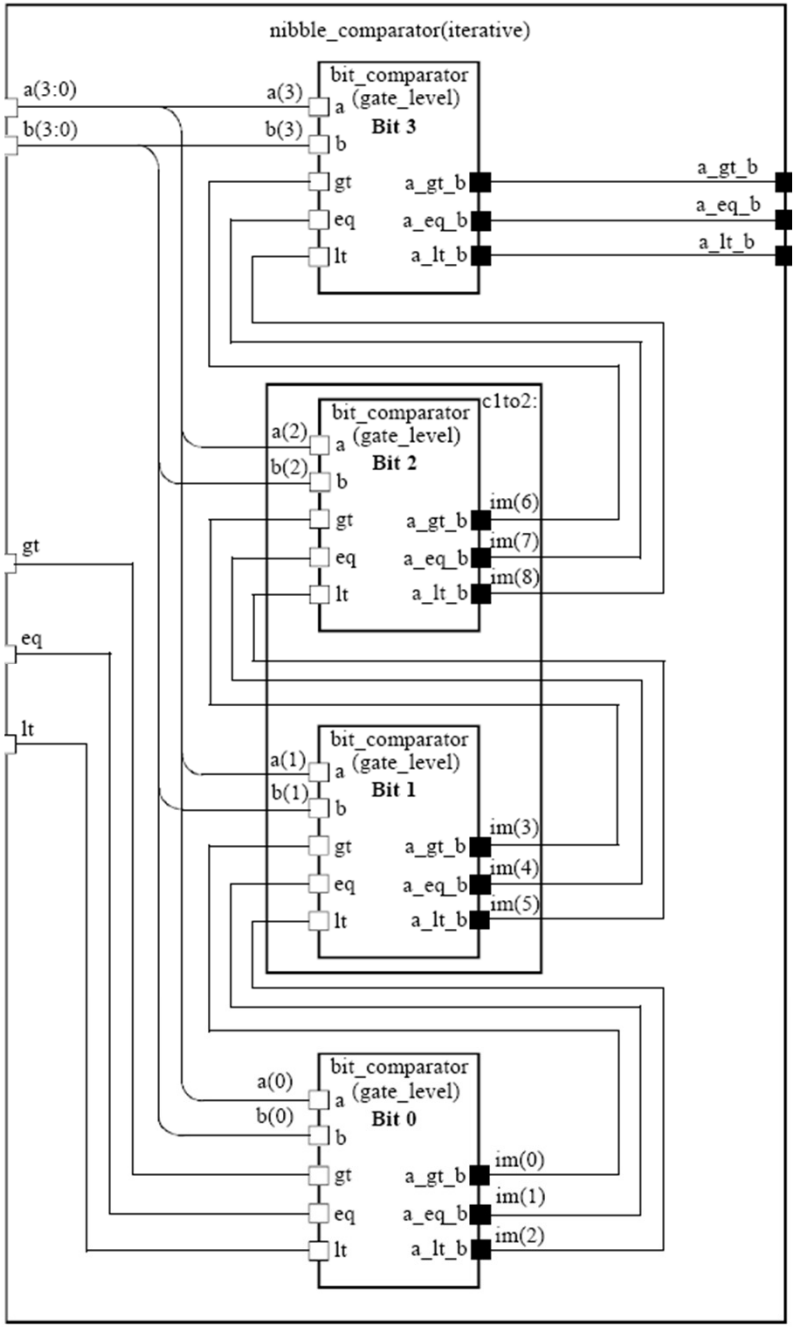
- Constructing 4-bit comparator by interconnecting the 1-bit comparator entities we have designed earlier



4-bit comparator

- Iterative instantiation by *for-generate*
  - A powerful VHDL structure
  - To exploit the intrinsic regularity of the design in the same way as the *for* loop in software does

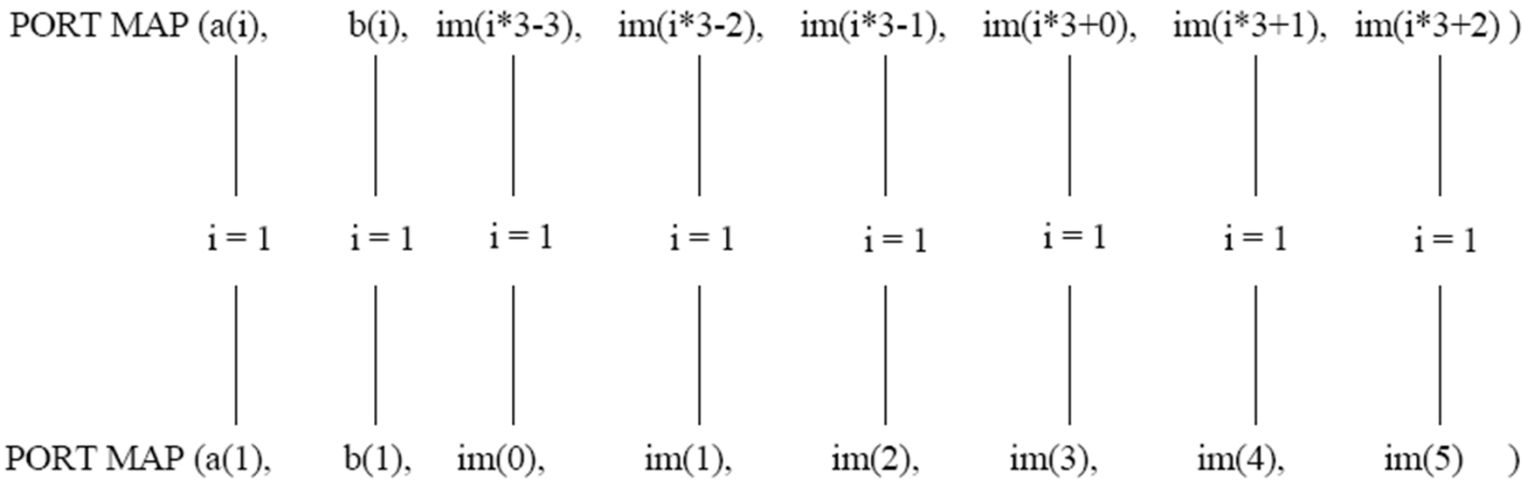
```
ARCHITECTURE iterative OF nibble_comparator IS
  COMPONENT comp1
    PORT (a, b, gt, eq, lt : IN BIT; a_gt_b, a_eq_b, a_lt_b : OUT BIT);
  END COMPONENT;
  FOR ALL : comp1 USE ENTITY WORK.bit_comparator (gate_level);
  SIGNAL im : BIT_VECTOR ( 0 TO 8);
BEGIN
  c0: comp1 PORT MAP (a(0), b(0), gt, eq, lt, im(0), im(1), im(2));
  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;
```



For-generate syntax

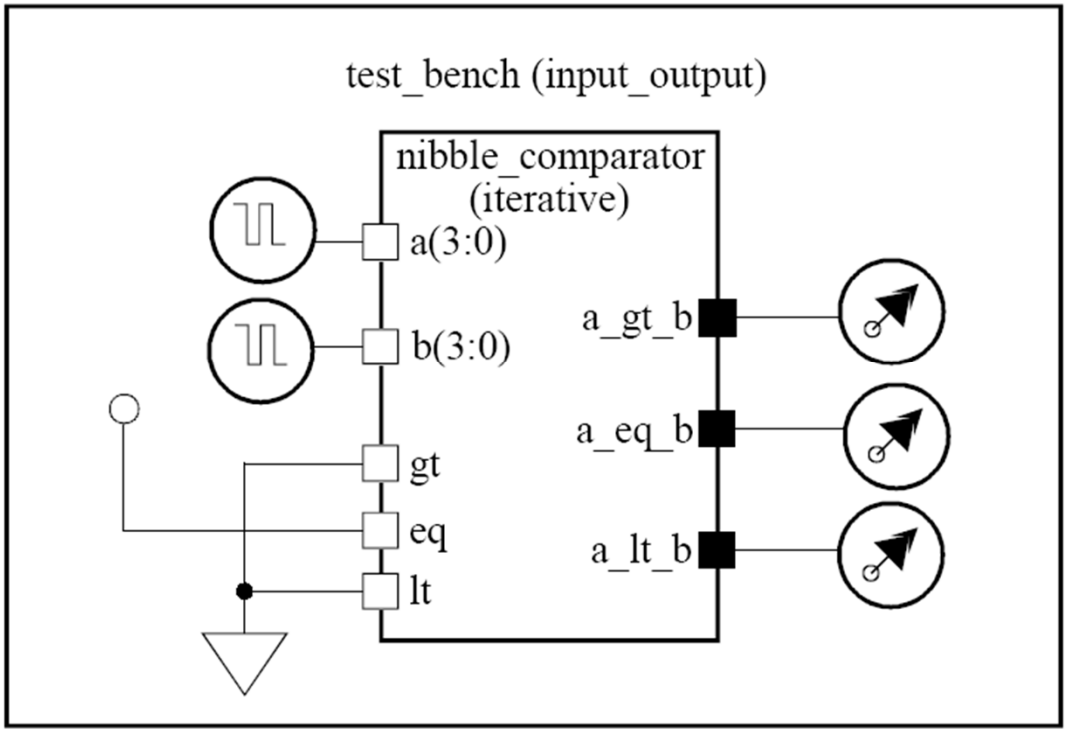
- This is a concurrent statement

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



4-bit comparator

- A test bench to verify the correctness of the design



4-bit comparator test-bench

- A test bench has no in/out ports
- Generate some internal signals to test the DUT (design under test)

```
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);
        a_gt_b, a_eq_b, a_lt_b : IN BIT;
        a_gt_b_out, a_eq_b_out, a_lt_b_out : 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 : BIT;
  SIGNAL vdd : BIT := '1';
  SIGNAL gnd : BIT := '0';
BEGIN
  a1: comp4 PORT MAP (a, b, gnd, vdd, gnd, gtr, eql, lss);
```

4-bit comparator test-bench

- Define 2 internal signals and connect them to the input of the DUT
- Use concurrent data flow statements to generate input
- Observe the output

```
ARCHITECTURE input_output OF nibble_comparator_test_bench IS
  COMPONENT comp4 PORT (a, b : IN bit_vector (3 DOWNTO 0);
        a_gt_b, a_eq_b, a_lt_b : IN BIT;
        a_gt_b_out, a_eq_b_out, a_lt_b_out : 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 : BIT;
  SIGNAL vdd : BIT := '1';
  SIGNAL gnd : BIT := '0';
BEGIN
  a1: comp4 PORT MAP (a, b, gnd, vdd, gnd, gtr, eql, lss);
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)
  "1111" AFTER 4000 NS, -- a < b (steady state, prepare for next)
  "1111" AFTER 4500 NS, -- a = b (worst case)
  "1111" AFTER 5000 NS, -- a < b (need bit 3 only, best case)
  "0000" AFTER 5500 NS, -- a = b (worst case)
  "0000" AFTER 6000 NS; -- a > b (need bit 3 only, best case)
END input_output;
```

Configuration

- Binding a component instantiation to an actual entity
- We saw how this can be done inside an architecture

```
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));
  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;
```

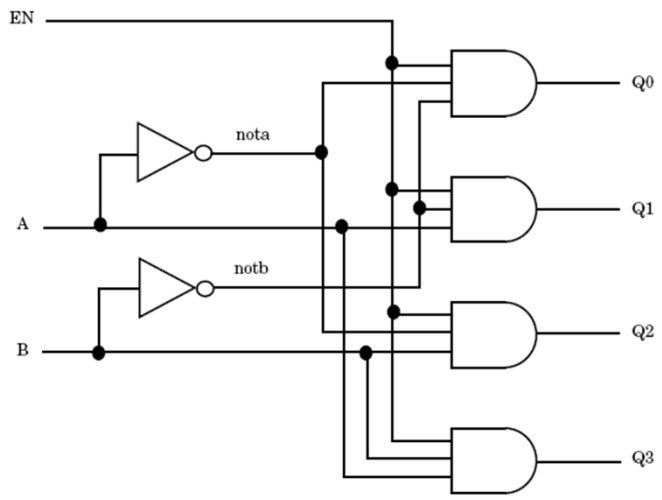
Configuration

- Component binding can be done outside an architecture as an independent structure
- Can design a general architecture and specify the lower-level entities at a later stage
- Using the *CONFIGURATION* keyword

Configuration-example

- Then, use them to construct a larger design: a decoder

```
ENTITY decode IS
 PORT( a, b, en : IN std_logic;
       q0, q1, q2, q3 : OUT std_logic);
END decode;
```




```
ARCHITECTURE structural OF decode IS
 COMPONENT inv
  PORT( a  : IN std_logic;
        b  : OUT std_logic);
 END COMPONENT;

 COMPONENT and3
  PORT( a1, a2, a3 : IN std_logic;
        o1 : OUT std_logic);
 END COMPONENT;

 SIGNAL nota, notb : std_logic;
BEGIN
 I1 : inv
  PORT MAP(a, nota);

 I2 : inv
  PORT MAP(b, notb);

 A1 : and3
  PORT MAP(nota, en, notb, Q0);

 A2 : and3
  PORT MAP(a, en, notb, Q1);

 A3 : and3

   PORT MAP(nota, en, b, Q2);

 A4 : and3
  PORT MAP(a, en, b, Q3);

END structural;
```

Configuration-example

```
CONFIGURATION      conf      OF decode IS
 FOR structural
  FOR ALL:  inv USE ENTITY WORK.inv(behave);
  END FOR;

  FOR ALL:  and3 USE ENTITY WORK.and3(behave);
  END FOR;
 END FOR;
END      conf    ;
```

This is a configuration named *decodea_con* for entity *decode*.
Use architecture *structural* as the
architecture for the topmost entity, which is *decode*.
For all component Instances of type *inv* instantiated in the *structural* architecture, use entity *inv*, architecture *behave* from the library called *WORK*. For all other component instances of type *and3*, use entity *and3*, architecture *behave* from library *WORK*

Configuration

- Configurations can be used in different ways
- To learn more, see:
  - Chapter 7 of "VHDL programming by example"
  - Chapter 6.4 of "VHDL analysis and modeling of digital systems"