VHDL - Concurrent Code Structural modeling

Introduction - Design Hierarchy

There are three types of digital design approach:

- ✓ **Bottom-Up** approach – Implemented using **Structural modeling**:

  We first identify the building Entities that are available to us. We build bigger Entities, using these building Entities. These Entities are then used for higher-level Entities until we build the top-level Entity in the design.

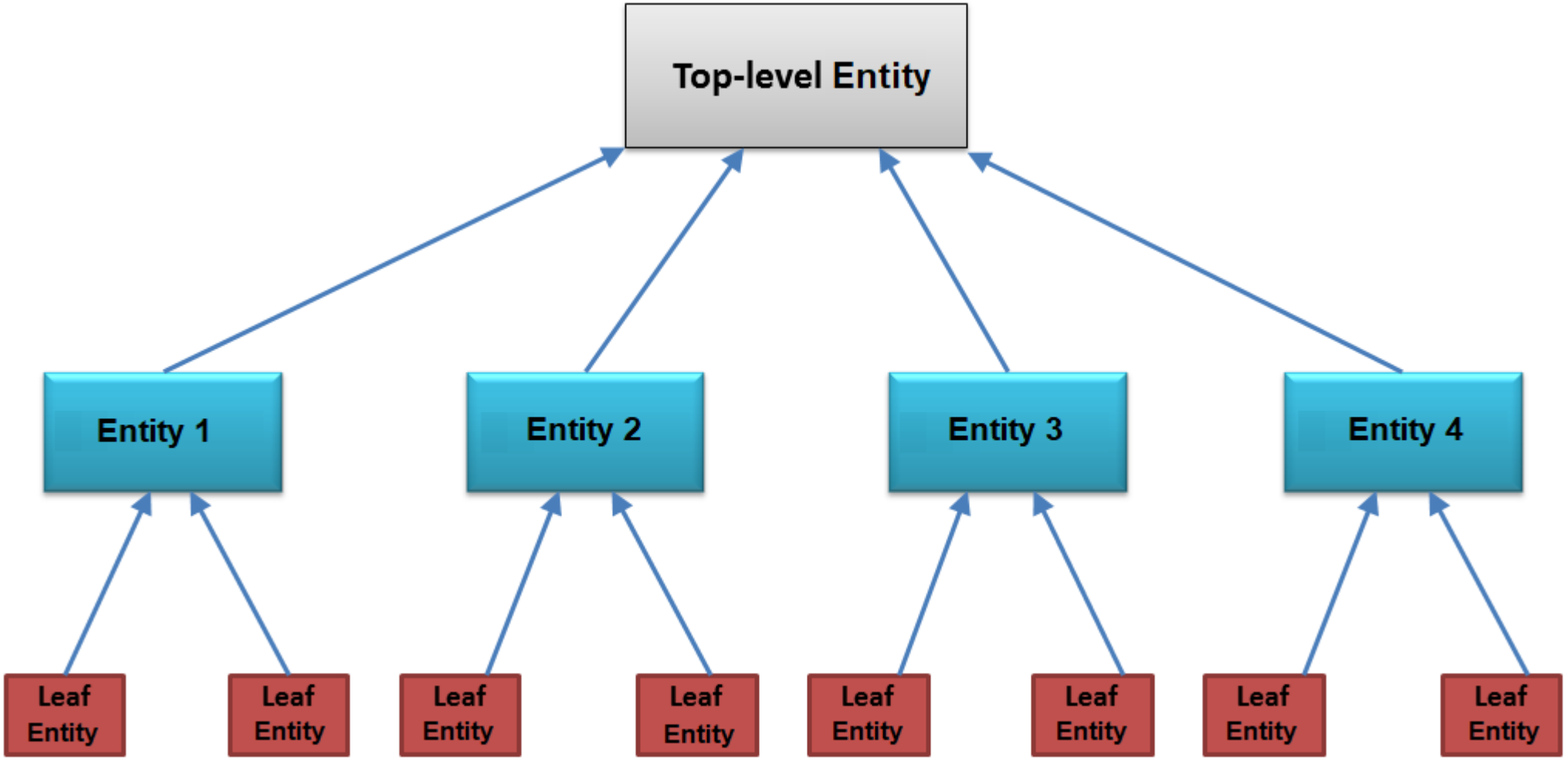
- ✓ **Top-Down** approach – Implemented using **Behavioral modeling**:

  We define the top-level Entity and identify the sub- Entities necessary to build the top-level Entity. We further subdivide the sub- Entities until we come to leaf Entities, which are the Entities that cannot further be divided.

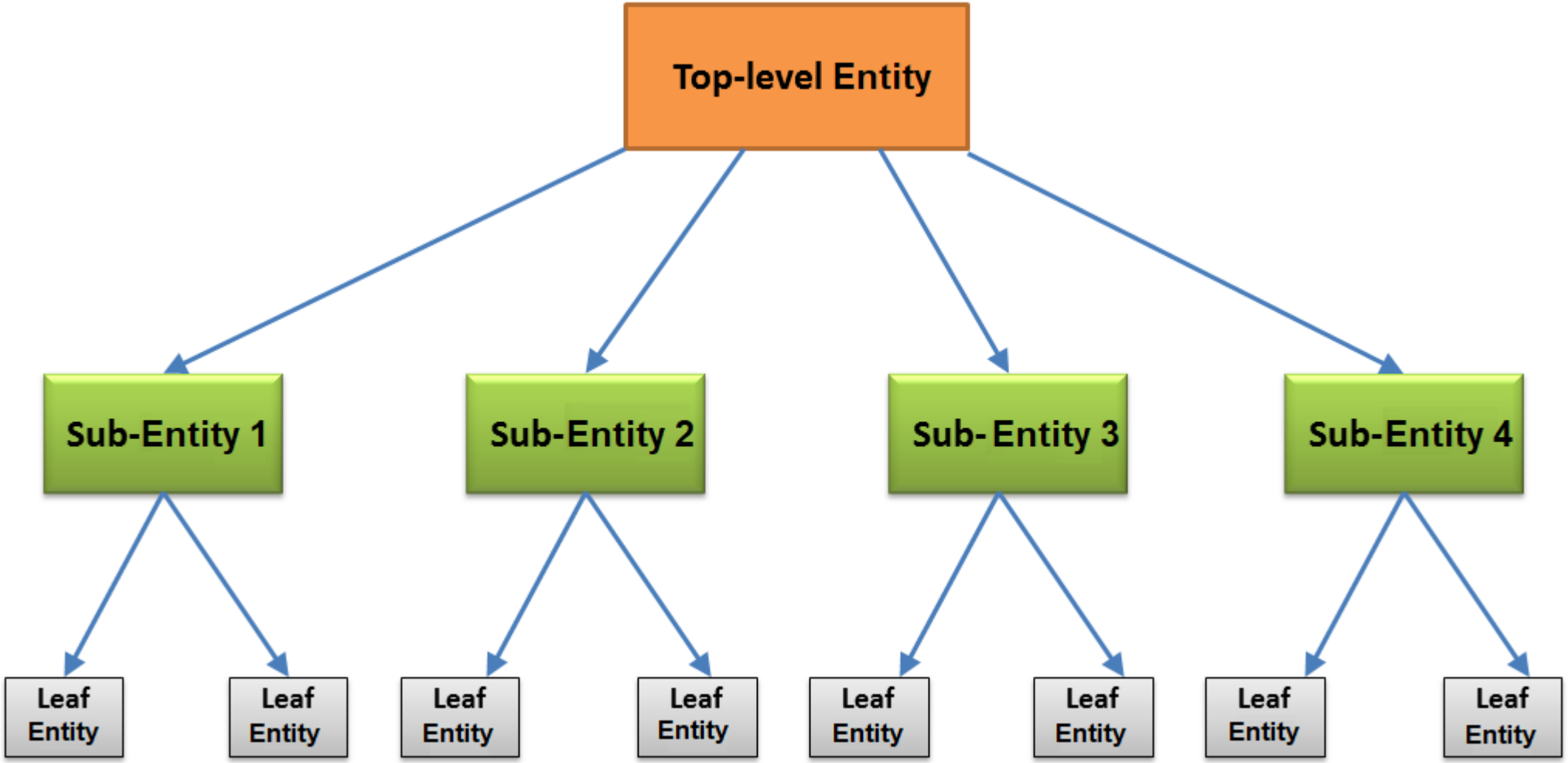
- ✓ **Mixed** approach:

  A combination of top-down and bottom-up flows (typically used).

Bottom-Up approach – Structural modeling



Top-Down approach – Behavioral modeling

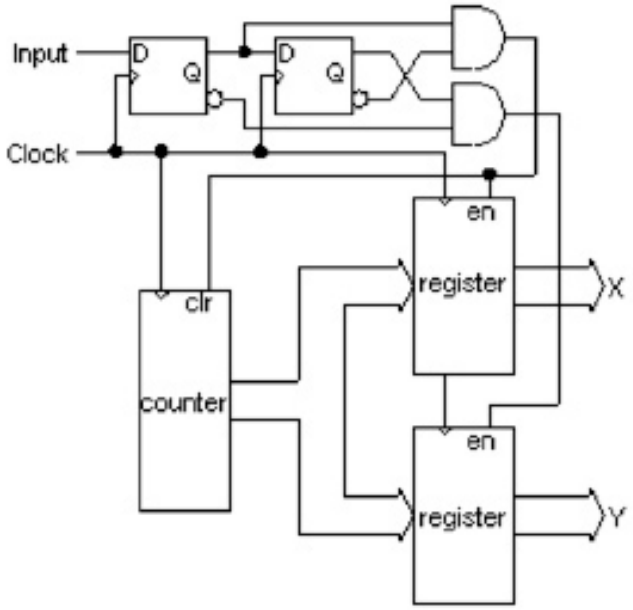


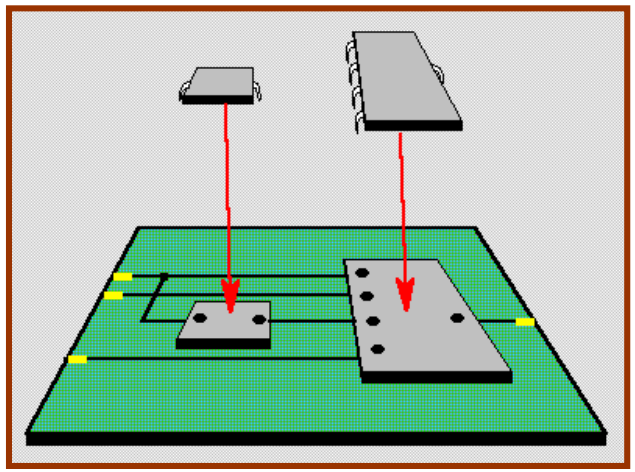
Mixed approach – Chip design

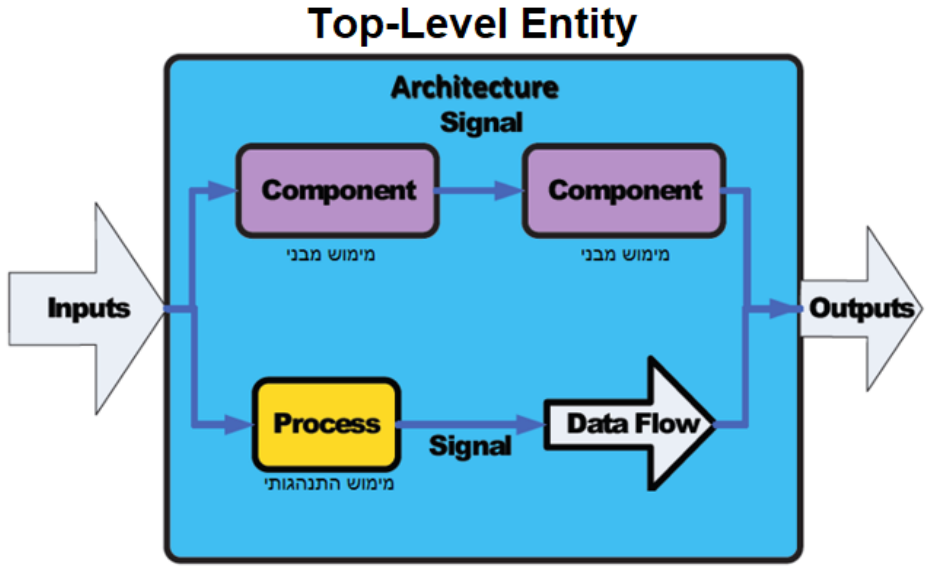
- Typically, a combination of top-down and bottom-up flows is used.
- Design architects define the specifications of the top-level block.
- Logic designers decide how the design should be structured by breaking up the functionality into blocks and sub-blocks (Top-Down).
- The Circuit designers are designing optimized circuits for leaf-level cells. They build higher-level cells by using these leaf cells (Bottom-Up).
- The design flow meets at an intermediate point where the switch-level circuit designers have created a library of leaf cells by using switches, and the logic level designers have designed from top-down until all modules are defined in terms of leaf cells.

Structural Architecture Modeling

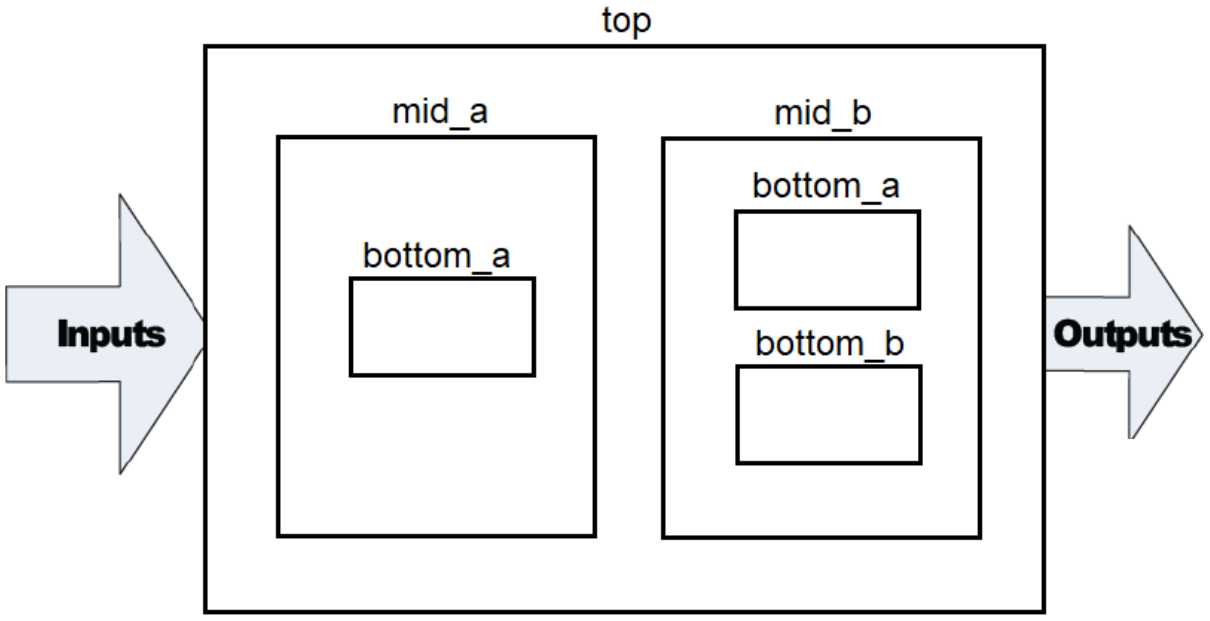
- It is a construction of hierarchical design.
- Describes connection between existing components (entities).
- Components are declared before the architecture.
- Components are connected to each other using signals.



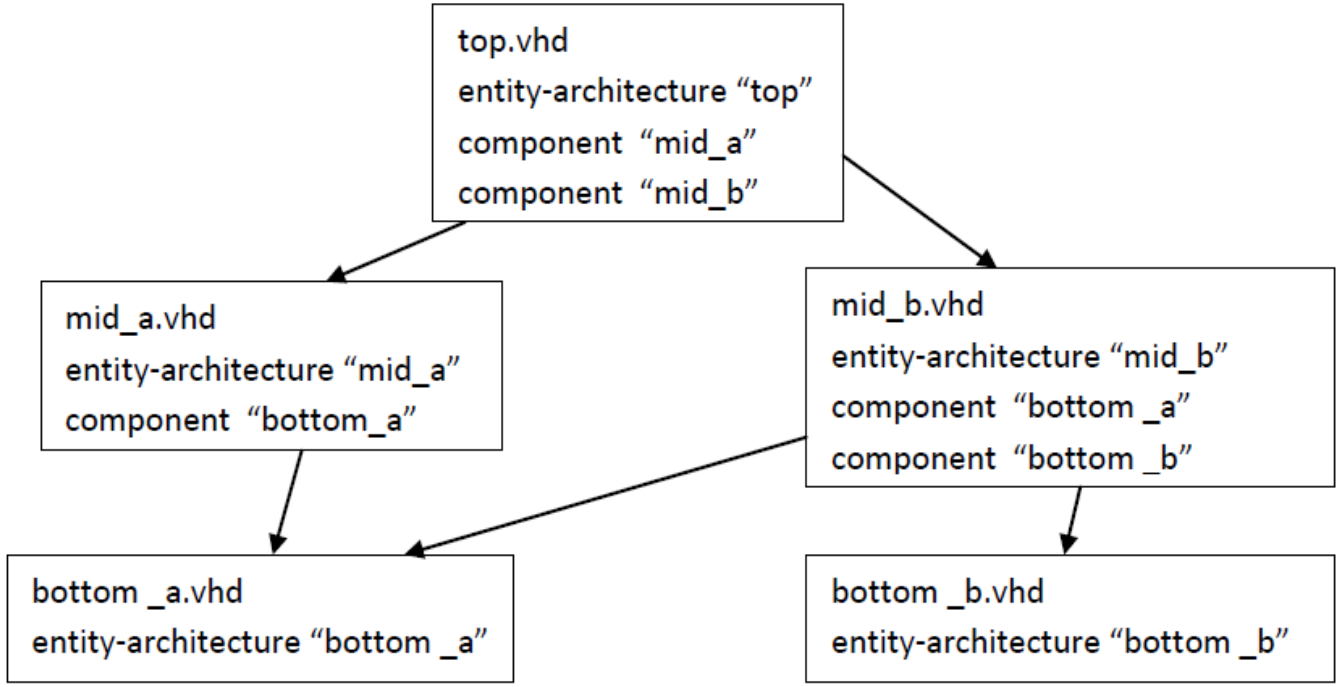




Structural Architecture Modeling



- The top level entity (called also main-code) connects all its sub-entities using **COMPONENT**s.
- The depicted design contains three hierarchical levels.
- Each Entity element in our design must be written in a separate *entity_name.vhd* file



COMPONENT

- A COMPONENT is simply a piece of conventional code (LIBRARY declarations +ENTITY +ARCHITECTURE).
- By declaring such code as being a COMPONENT, it can then be used within another circuit, thus allowing the construction of hierarchical designs.
- A COMPONENT is also another way of partitioning a code and providing code sharing and code reuse.

  An example: commonly used circuits, like flip-flops, multiplexers, adders, basic gates, etc., can be placed in a LIBRARY, so any project can make use of them without having to explicitly rewrite such codes.
- To use (instantiate) a COMPONENT, it must first be declared.

COMPONENT declaration

COMPONENT declaration – two possible locations:

- In its immediate upper level ARCHITECTURE (inside the declarative part).
- In a Package

Syntax:

- The syntax is identical to ENTITY, except the key word COMPONENT.

```
COMPONENT component_name IS
    PORT ( port_name : signal_mode signal_type;
                  ...
           port_name : signal_mode signal_type;
           port_name : signal_mode signal_type
    );
END COMPONENT;
```

COMPONENT instantiation

COMPONENT instantiation:

- After a **COMPONENT** declaration, in order to use it (instantiation) within its immediate upper level **ARCHITECTURE** (after keyword BEGIN, in the body section), we use **PORT MAP** statement.
- Using **PORT MAP** statements we interconnect between **COMPONENT**s in the same hierarchical level.
- There are two methods for COMPONENT instantiation:

  1) nominal mapping (decreasing syntax errors):

```
Instance_name  : entity_name_of_lower_level  PORT MAP (
        lower_level_port_name#1  => current_level_port_name#1 ,
                  ....
        lower_level_port_name#n  => current_level_port_name#n ,
);
```

```
U1: inverter PORT MAP (x=>a,y=>b);
```

COMPONENT instantiation

2) positioning mapping (more convenient):

```
Instance_name : entity_name_of_lower_level PORT MAP (current_level_port_list);
```

Note1: the current level port list must be in exact same order of the lower level ENTITY.

```
Instance_name  : entity_name_of_lower_level  PORT MAP (
        current_level_port_name#1 , … ,  current_level_port_name#n );
```

Example:

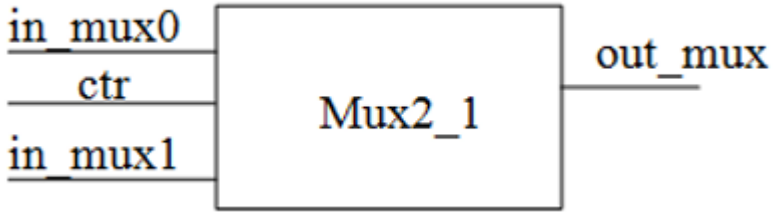
```
U1: inverter PORT MAP (x,y);
```

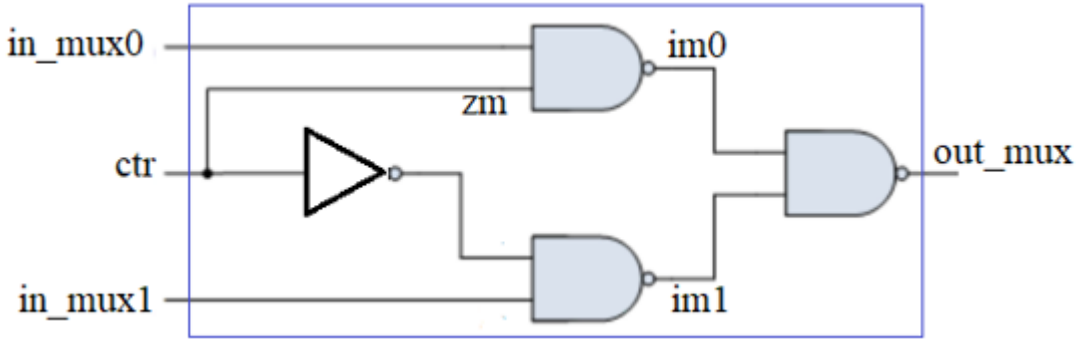
Note2: If we don't need to connect some of the COMPONENT ports, we can use with the keyword **OPEN**.

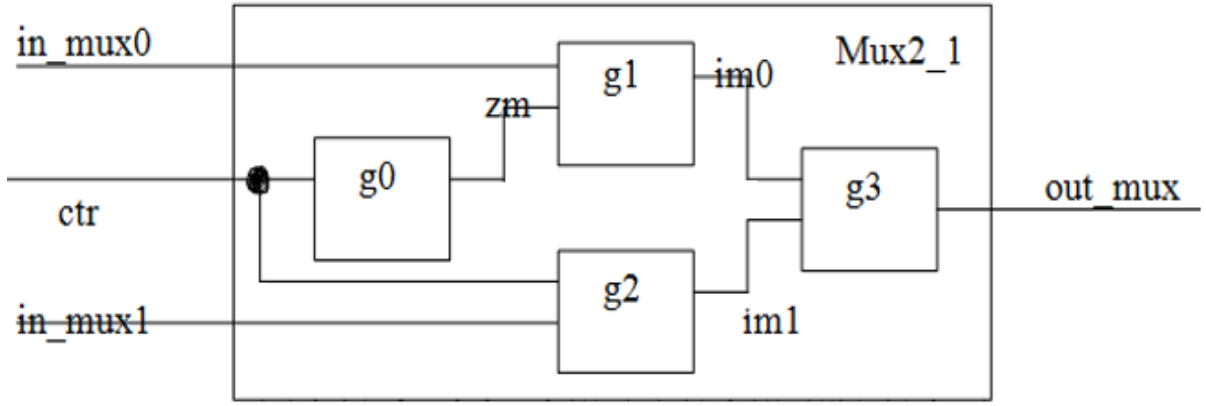
```
U1: inverter PORT MAP (x,OPEN);
```

Structural Architecture example – MUX 2-1

```
ENTITY mux2_1 IS
     PORT (  ctr : IN bit;
             in_mux0, in_mux1 : IN bit;
             out_mux : OUT bit);
END mux2_1;
```







Mux2_1.vhd
Entity-architecture of Mux2_1
Component "nd2"
component "iv"

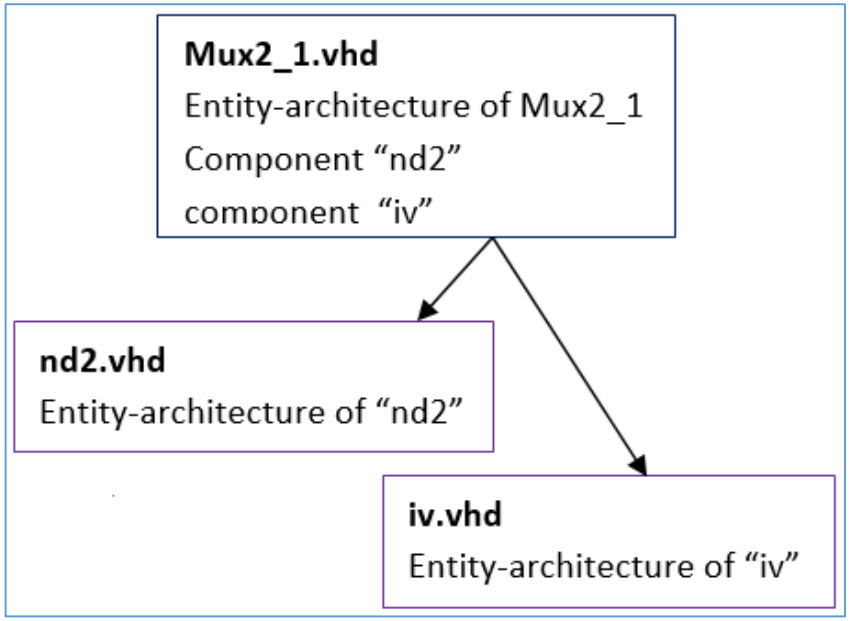
nd2.vhd
Entity-architecture of "nd2"

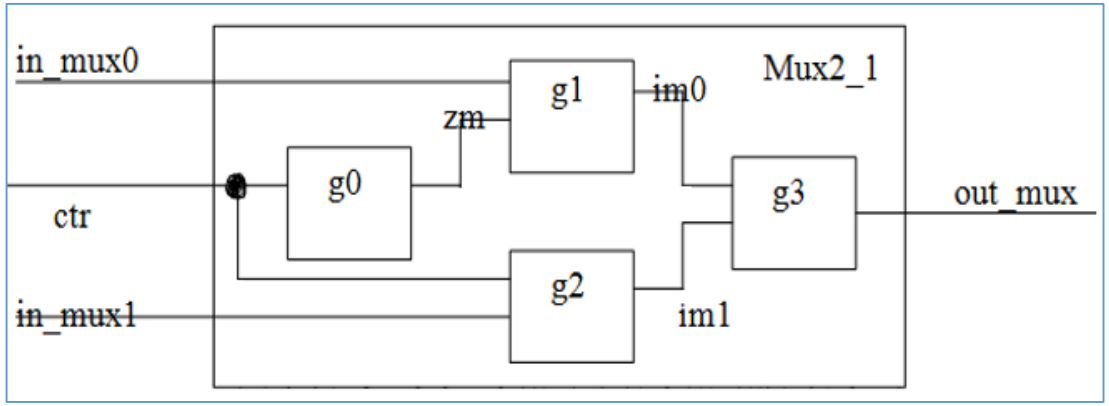
iv.vhd
Entity-architecture of "iv"

Structural Architecture example – MUX 2-1

```
ENTITY mux2_1 IS
     PORT (  ctr : IN bit;
             in_mux0, in_mux1 : IN bit;
             out_mux : OUT bit);
END mux2_1;

ARCHITECTURE str_mux2_1 OF mux2_1 IS
      COMPONENT nd2 PORT (a, b : IN bit; z : OUT bit);
      END COMPONENT;
      COMPONENT iv PORT (a : IN bit; z : OUT bit);
      END COMPONENT;
      SIGNAL im0, im1, zm : bit;
BEGIN
     g0 : iv PORT MAP (ctr, zm);
     g1 : nd2 PORT MAP (in_mux0, zm, im0);
     g2 : nd2 PORT MAP (ctr, in_mux1, im1);
     g3 : nd2 PORT MAP (im0, im1, out_mux);
END str_mux2_1;
```





GENERIC MAP

- When we use **COMPONENT** which contains **GENERIC** statement, we can pass the **GENERIC** parameters value through **GENERIC MAP** statement.
- Syntax:

```
Instance_name : component_name GENERIC MAP(param_list)PORT MAP(port_list);
```

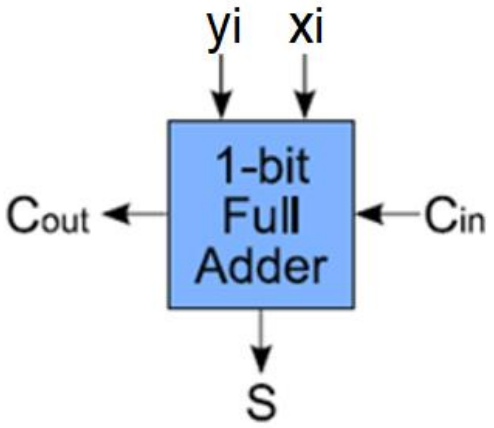
- Example:

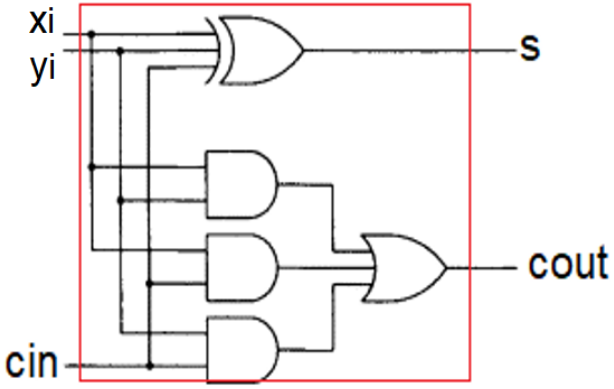
```
U1: GENERIC MAP(n)inverter PORT MAP (x=>a,y=>b);
```

- Note:

  If we don't use **GENERIC MAP** the, the **COMPONENT**'s **GENERIC** parameters will be as the defined default.

GENERIC MAP example – Ripple Adder





Adder.vhd

Entity+Arch. of "Adder"

component "FA"

FA.vhd

Entity+Arch. of "FA"