Hierarchical design and Testbench

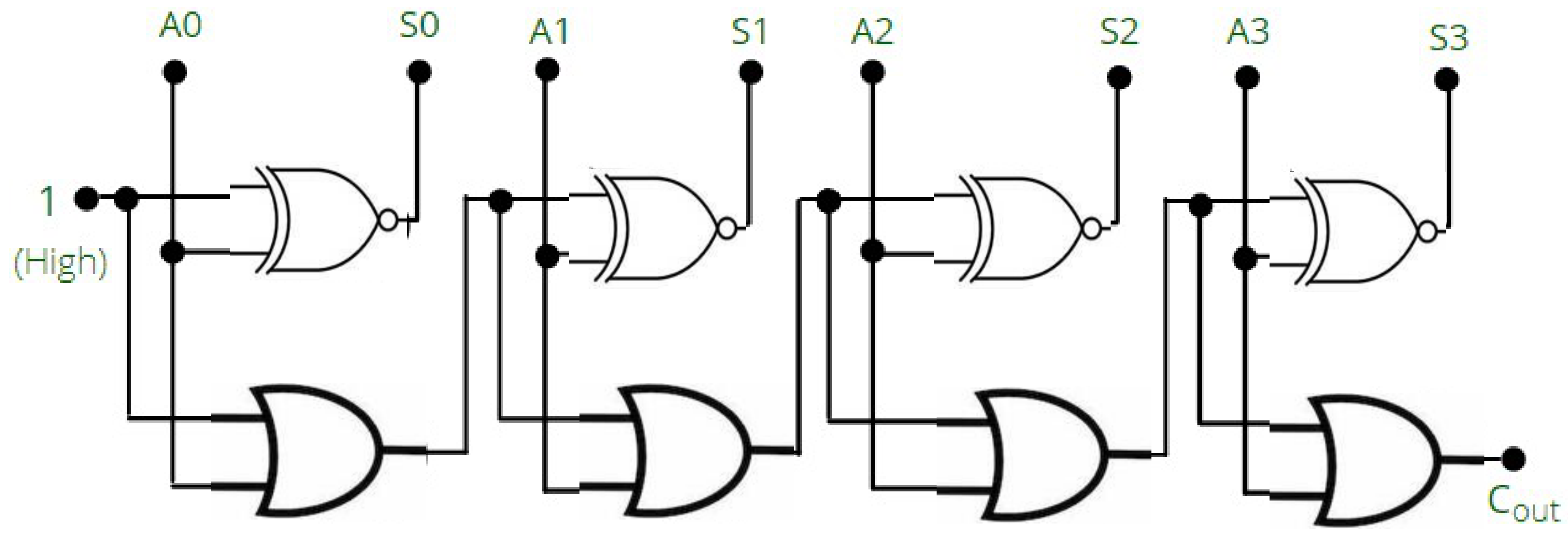
Digital Lab.2

Osama Ali

2024

osamaalisalman.khafajy@edu.bme.hu

What if I need to implement such digital circuit



4- Bit Binary Incrementer (In detail)

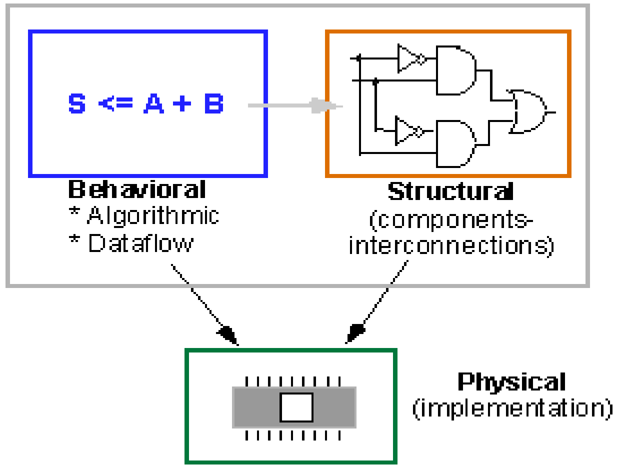
What about testing such circuit?

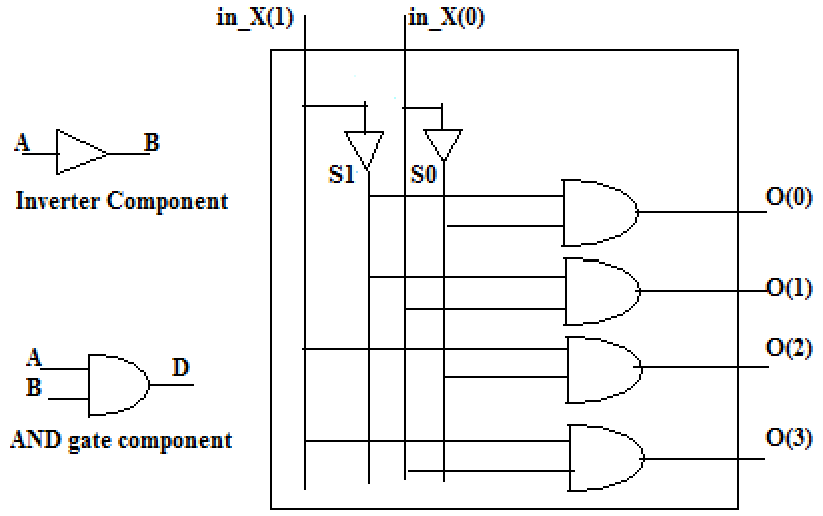
A3	A2	A1	A0	S3	S2	S1	S0	Cout
0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	1	0	0
0	0	1	0	0	0	1	1	0
0	0	1	1	0	1	0	0	0
0	1	0	0	0	1	0	1	0
0	1	0	1	0	1	1	0	0
0	1	1	0	0	1	1	1	0
0	1	1	1	1	0	0	0	0
1	0	0	0	1	0	0	1	0
1	0	0	1	1	0	1	0	0
1	0	1	0	1	0	1	1	0
1	0	1	1	1	1	0	0	0
1	1	0	0	1	1	0	1	0
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
1	1	1	1	0	0	0	0	1

Hierarchical Modeling

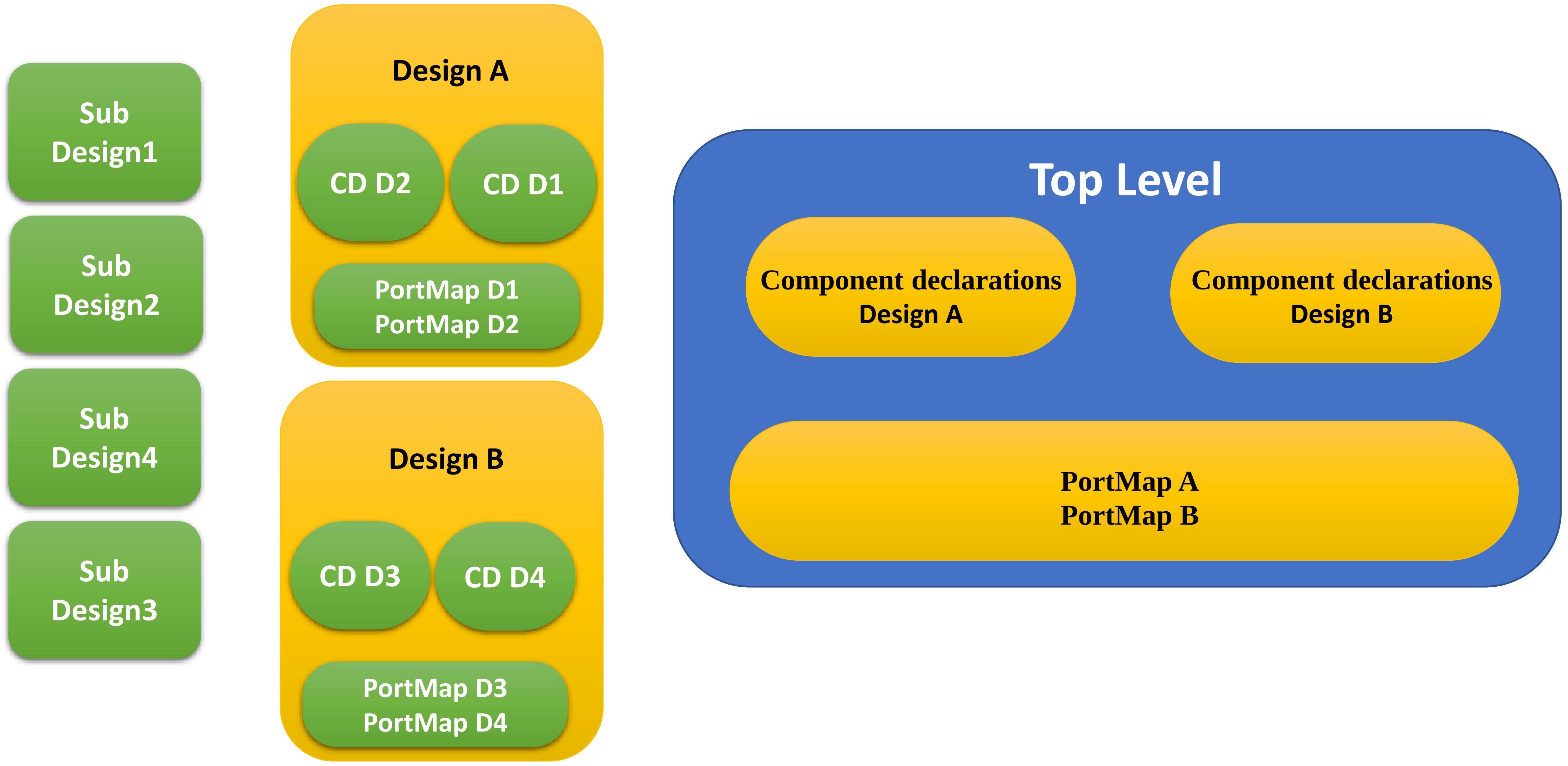
- Advantage of hierarchical design
    - Modularity
    - Abstraction
    - Simplicity
    - Collaboration
    - Scalability

- Overall, hierarchical modeling is a powerful technique for designing complex digital systems in VHDL. It provides modularity, abstraction, simplicity, collaboration, and scalability, making it an essential tool for modern digital design.



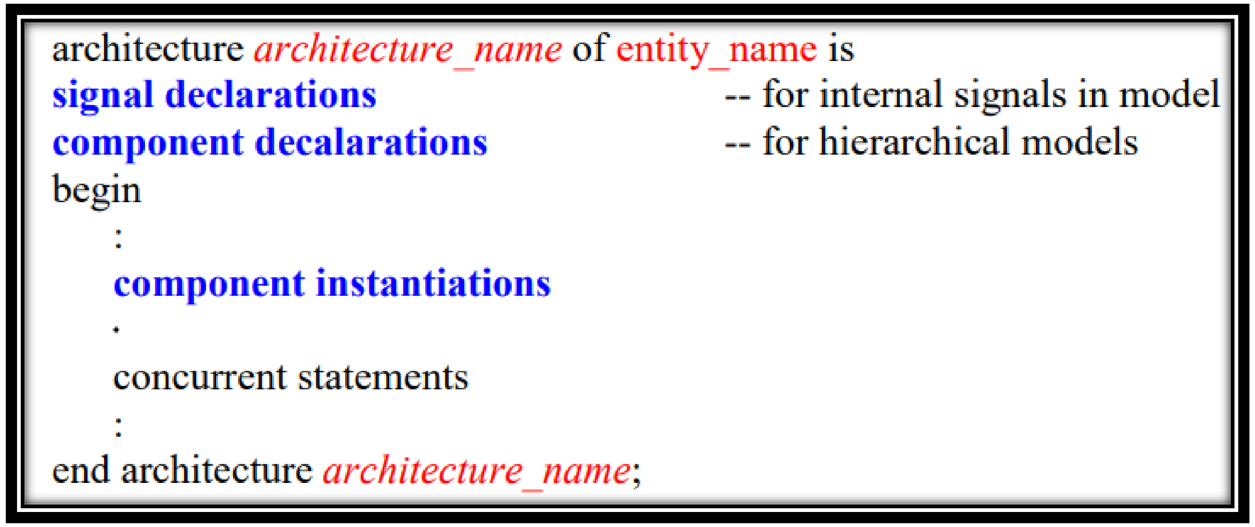


Component declarations



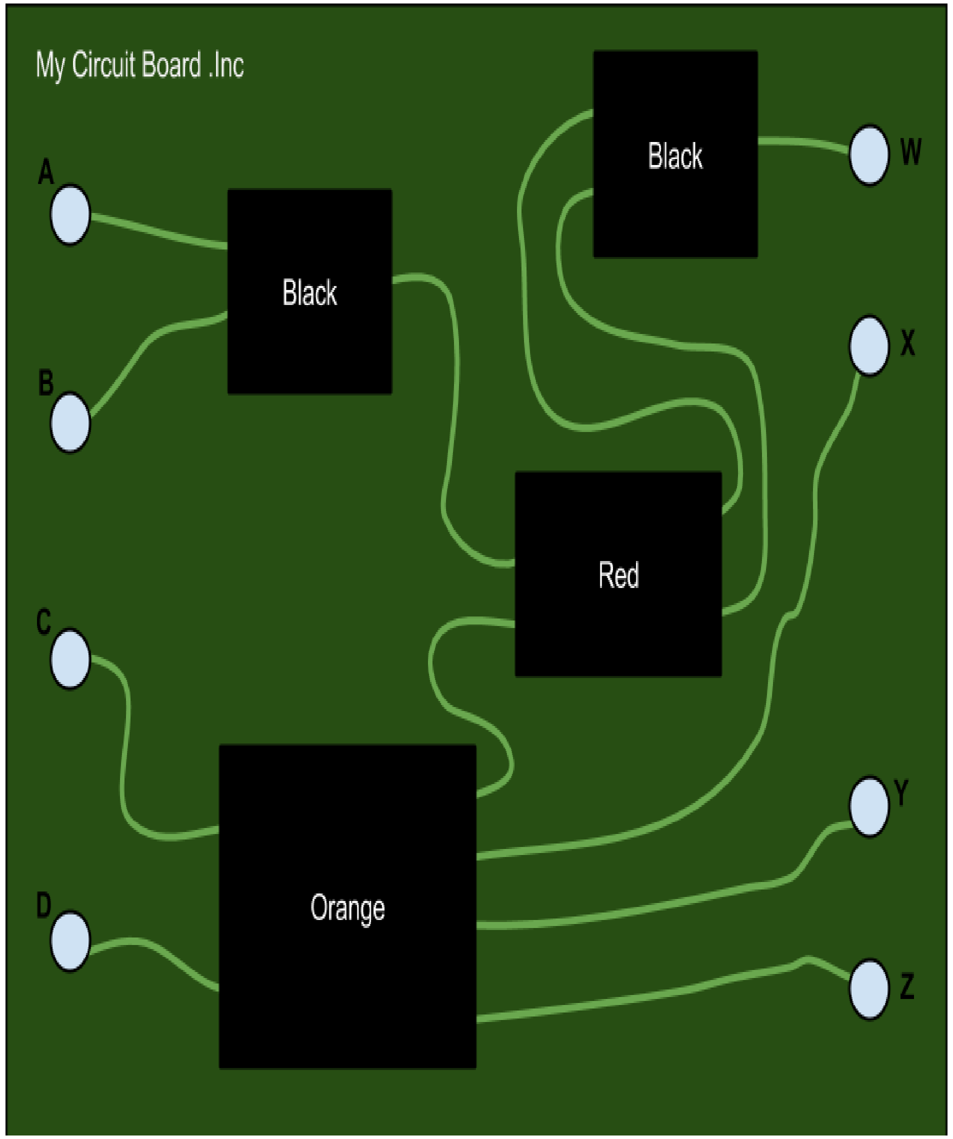
Component & Component declarations

- To incorporate hierarchy in VHDL we must add **component declarations** and **component instantiations** to the model.
- **Component Represents a precompiled Entity-Architectecture Paire**
- Instantiation is selecting a component and using it as an instance in our design
- we need to declare internal signals to interconnect the components.
    - Format for Architecture body (for internal signals & hierarchy):



```
architecture architecture_name of entity_name is
signal declarations                        -- for internal signals in model
component decalarations                    -- for hierarchical models
begin
    :
    component instantiations
    .
    concurrent statements
    :
end architecture architecture_name;
```

VHDL Code e.g.



Top Level ARCHITECTURE Definition part

```vhdl
signal S1 : STD_LOGIC;
signal S2 : STD_LOGIC;
signal S3 : STD_LOGIC;
signal S4 : STD_LOGIC;

COMPONENT black_box
   PORT(
      Data_A : IN std_logic;
      Data_B : IN std_logic;
      Data_F : OUT std_logic
      );
   END COMPONENT;

COMPONENT red_box
   PORT(
      Data_R : IN std_logic;
      Data_S : IN std_logic;
      Data_U : OUT std_logic;
      Data_V : OUT std_logic
      );
   END COMPONENT;

COMPONENT orange_box
   PORT(
      Data_H : IN std_logic;
      Data_I : IN std_logic;
      Data_L : OUT std_logic;
      Data_M : OUT std_logic;
      Data_N : OUT std_logic;
      Data_O : OUT std_logic
      );
   END COMPONENT;
```

Top Level Inside ARCHITECTURE

```vhdl
begin

   Inst_black_box_1: black_box PORT MAP(
      Data_A => A,
      Data_B => B,
      Data_F => S1
   );

   Inst_red_box: red_box PORT MAP(
      Data_R => S1,
      Data_S => S2,
      Data_U => S3,
      Data_V => S4
   );

   Inst_orange_box: orange_box PORT MAP (
      Data_H => C,
      Data_I => D,
      Data_L => S2,
      Data_M => X,
      Data_N => Y,
      Data_O => Z
   );

   Inst_black_box_2: black_box PORT MAP(
      Data_A => S3,
      Data_B => S4,
      Data_F => W
   );

end Behavioral;
```

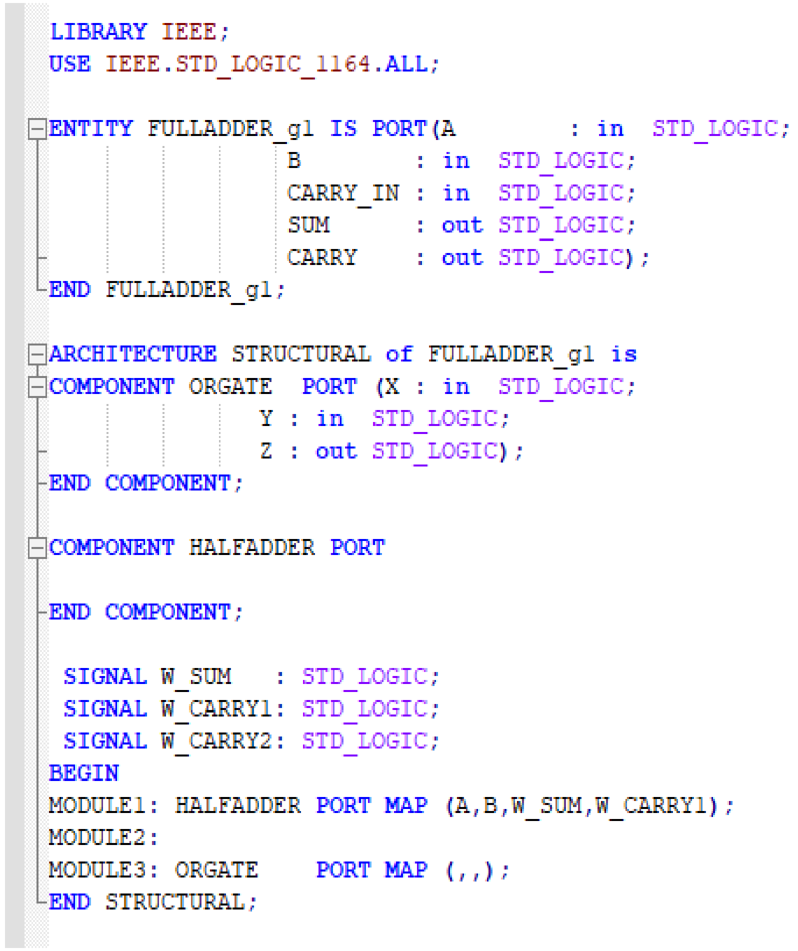
Component instantiation

- The component instantiation is the actual call to a specific use of the model.
- A single component declaration can have multiple instantiations.
- each component instantiation must include a unique name (instantiation_label along with the component (component_name) being used.
- There are two methods (and formats) for connecting signals to the port of the component:

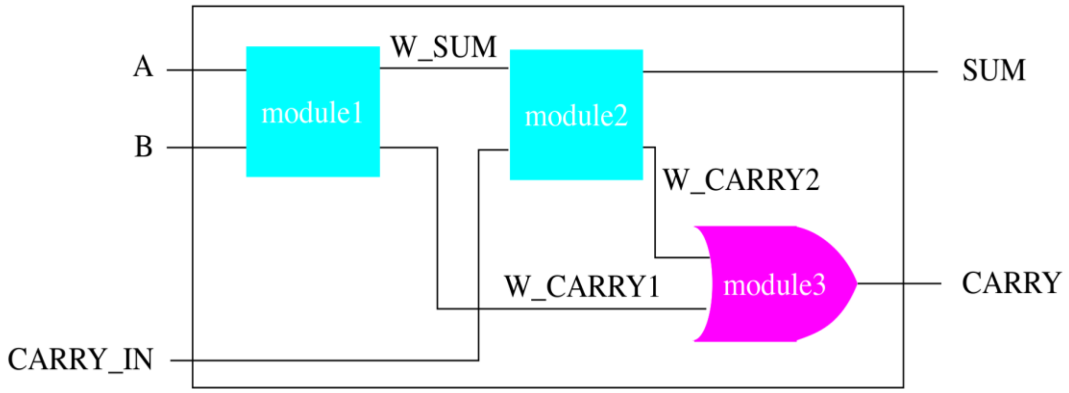
Keyword association	**Positional association**
instantiation_label: *component_name* : port map (*port_name* => *signal_name*, : *port_name* => *signal_name*);	*instantiation_label*: *component_name* : port map (*signal_name*, : *signal_name*);

FullAdder in Structural style:

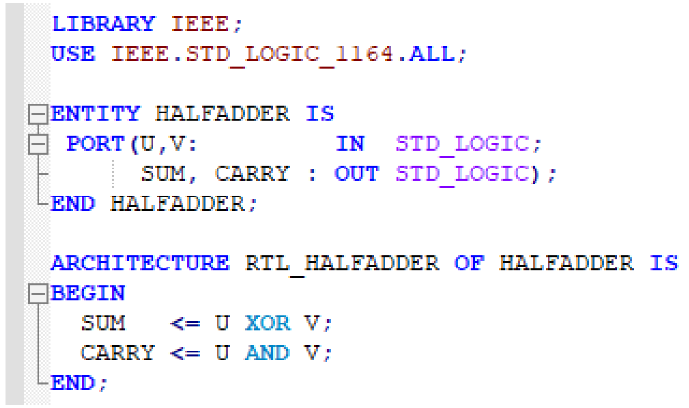
FullAdder



```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY FULLADDER_gl IS PORT(A        : in  STD_LOGIC;
                          B        : in  STD_LOGIC;
                          CARRY_IN : in  STD_LOGIC;
                          SUM      : out STD_LOGIC;
                          CARRY    : out STD_LOGIC);
END FULLADDER_gl;

ARCHITECTURE STRUCTURAL of FULLADDER_gl is
COMPONENT ORGATE  PORT (X : in  STD_LOGIC;
                      Y : in  STD_LOGIC;
                      Z : out STD_LOGIC);
END COMPONENT;

COMPONENT HALFADDER PORT

END COMPONENT;

 SIGNAL W_SUM   : STD_LOGIC;
 SIGNAL W_CARRY1: STD_LOGIC;
 SIGNAL W_CARRY2: STD_LOGIC;
BEGIN
MODULE1: HALFADDER PORT MAP (A,B,W_SUM,W_CARRY1);
MODULE2:
MODULE3: ORGATE    PORT MAP (,,);
END STRUCTURAL;
```

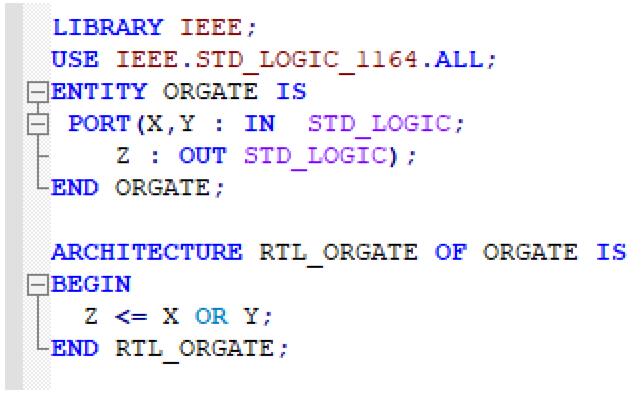


Halfadder



```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY HALFADDER IS
 PORT(U,V:          IN  STD_LOGIC;
      SUM, CARRY : OUT STD_LOGIC);
END HALFADDER;

ARCHITECTURE RTL_HALFADDER OF HALFADDER IS
BEGIN
  SUM   <= U XOR V;
  CARRY <= U AND V;
END;
```

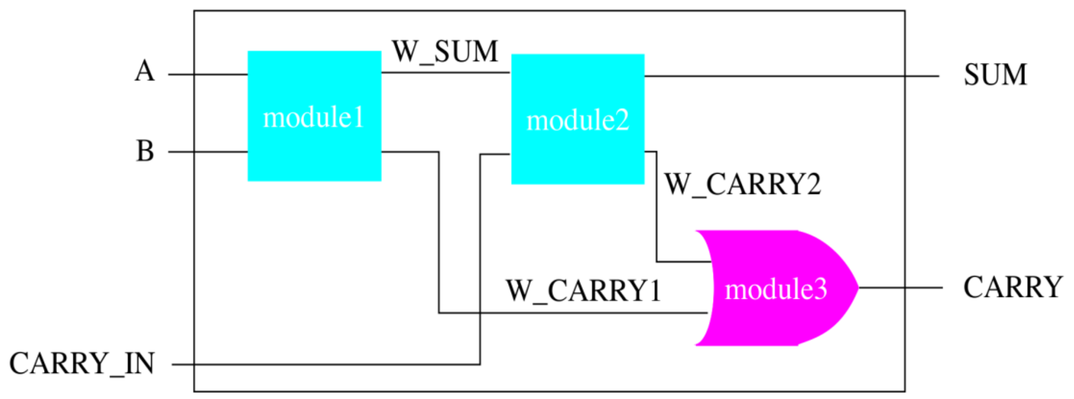
OR_gate



```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY ORGATE IS
 PORT(X,Y : IN  STD_LOGIC;
      Z : OUT STD_LOGIC);
END ORGATE;

ARCHITECTURE RTL_ORGATE OF ORGATE IS
BEGIN
  Z <= X OR Y;
END RTL_ORGATE;
```

Structural style example:

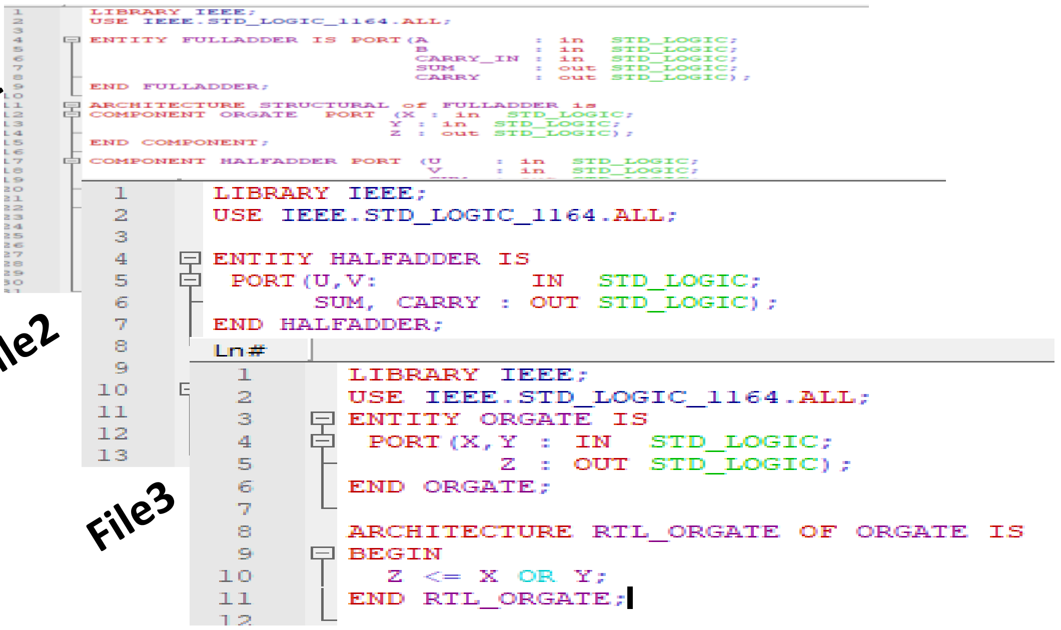
- Download those files from Model in a new folder on the desktop.
- Add all files to a new project in Modelsim
- And then compile all and simulate File1. By force the values of A, B and CIN the output should be like that



File1

File2

File3



```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY FULLADDER IS PORT(A          : in   STD_LOGIC;
                          B          : in   STD_LOGIC;
                          CARRY_IN   : in   STD_LOGIC;
                          SUM        : out  STD_LOGIC;
                          CARRY      : out  STD_LOGIC);
END FULLADDER;

ARCHITECTURE STRUCTURAL of FULLADDER is
COMPONENT ORGATE   PORT (X : in  STD_LOGIC;
                         Y : in  STD_LOGIC;
                         Z : out STD_LOGIC);
END COMPONENT;

COMPONENT HALFADDER PORT  (U     : in   STD_LOGIC;
                           V     : in   STD_LOGIC;
```

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY HALFADDER IS
 PORT(U,V:           IN   STD_LOGIC;
      SUM,  CARRY : OUT STD_LOGIC);
END HALFADDER;
```

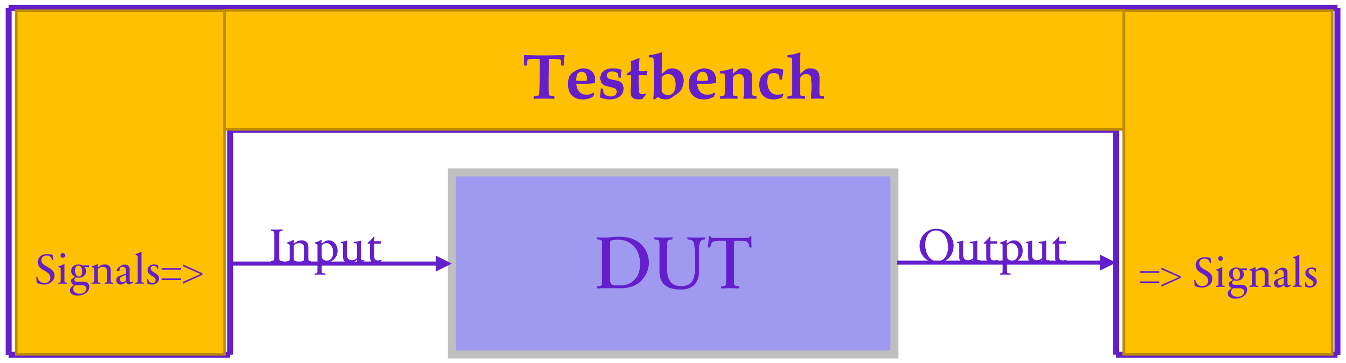
```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY ORGATE IS
 PORT(X,Y  :  IN   STD_LOGIC;
           Z  :  OUT  STD_LOGIC);
END ORGATE;

ARCHITECTURE RTL_ORGATE OF ORGATE IS
BEGIN
  Z <= X OR Y;
END RTL_ORGATE;
```

A	B	CIN	SUM	CARRY
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

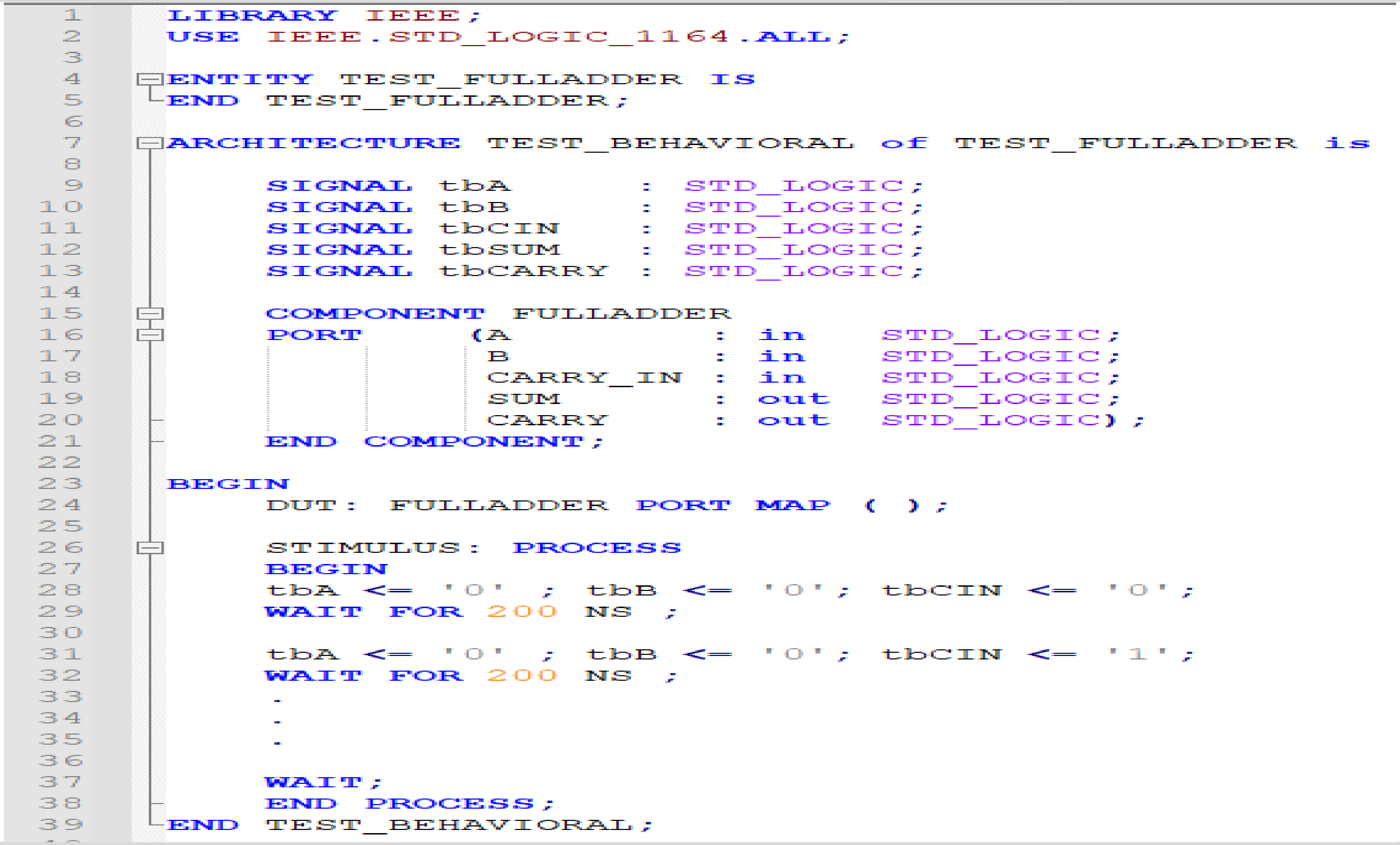
Testbench

- Testbench is an important part of VHDL design to check the functionality of Design through simulation waveform.
- Testbench provides a stimulus for **design under test** DUT or **Unit Under Test** UUT to check the output result.

- **A TestBench consists of:**
  - Entity
    - has no ports (empty entity header)
  - Architecture
    - declares, instantiates, and wires together the driver model and the model under test
    - driver model provides the stimulus and verifies model responses



Full-Adder Testbench



```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY TEST_FULLADDER IS
END TEST_FULLADDER;

ARCHITECTURE TEST_BEHAVIORAL of TEST_FULLADDER is

    SIGNAL tbA       : STD_LOGIC;
    SIGNAL tbB       : STD_LOGIC;
    SIGNAL tbCIN     : STD_LOGIC;
    SIGNAL tbSUM     : STD_LOGIC;
    SIGNAL tbCARRY   : STD_LOGIC;

    COMPONENT FULLADDER
    PORT      (A           : in   STD_LOGIC;
               B           : in   STD_LOGIC;
               CARRY_IN    : in   STD_LOGIC;
               SUM         : out  STD_LOGIC;
               CARRY       : out  STD_LOGIC) ;
    END COMPONENT;

BEGIN
    DUT: FULLADDER PORT MAP ( );

    STIMULUS: PROCESS
    BEGIN
    tbA <= '0' ; tbB <= '0'; tbCIN <= '0';
    WAIT FOR 200 NS ;

    tbA <= '0' ; tbB <= '0'; tbCIN <= '1';
    WAIT FOR 200 NS ;
    .
    .
    .

    WAIT;
    END PROCESS;
END TEST_BEHAVIORAL;
```