CPE 462
VHDL: Simulation and Synthesis

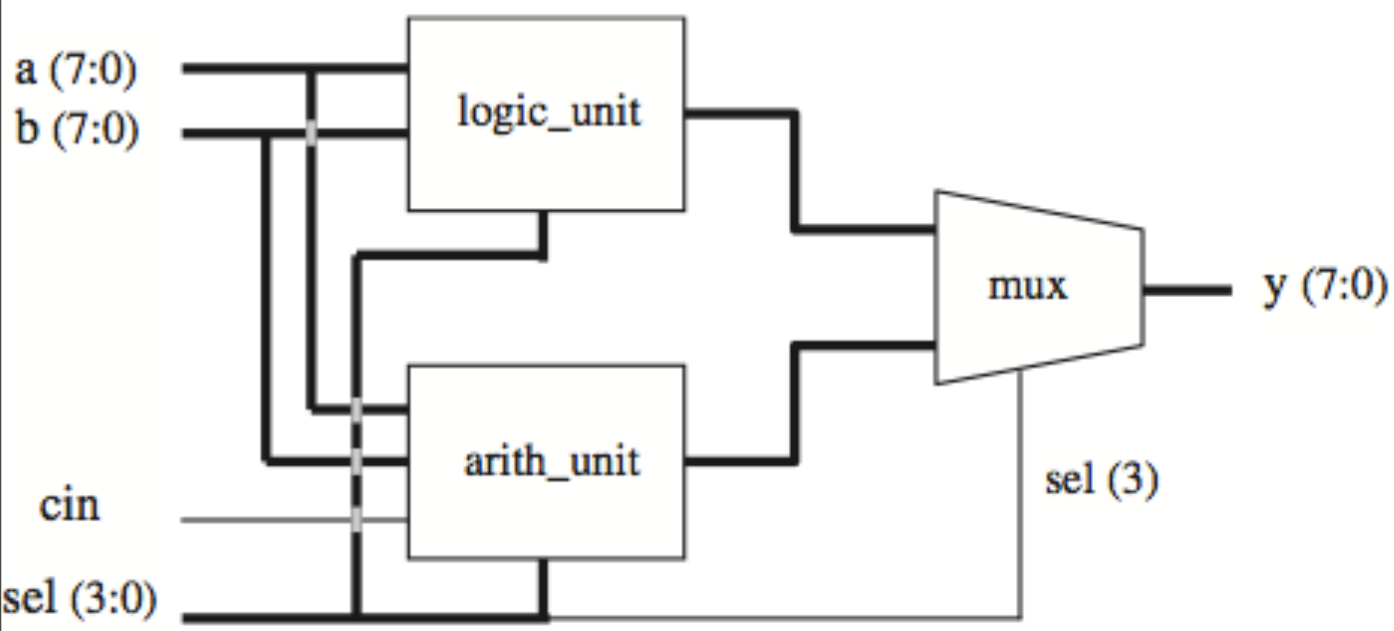
Topic #08 - a) Components

Nuno Alves (nalves@wne.edu), College of Engineering

COMPONENT

- A COMPONENT is simply a piece of conventional code
- Conventional code = LIBRARY declarations + ENTITY + ARCHITECTURE
- However, 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.
- For example, commonly used circuits (FF, multipliers, adders,...) can be placed in a LIBRARY, so any project can make use of them without having to explicitly rewrite such codes.

Motivational example

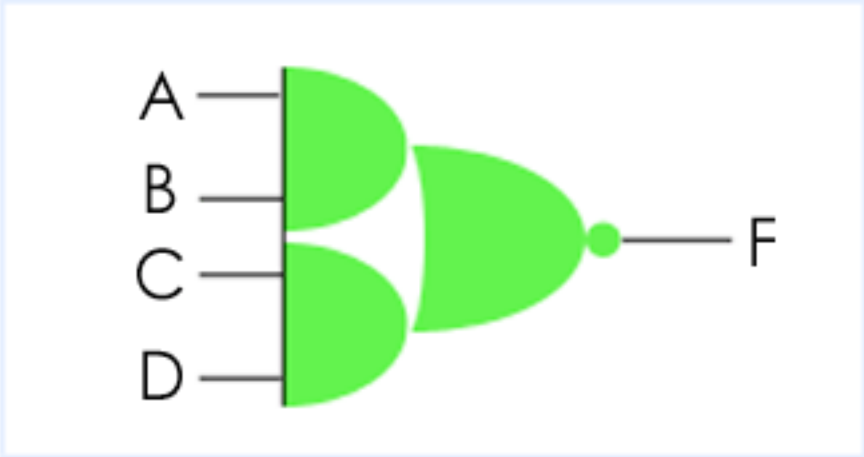


sel	Operation	Function	Unit
0000	y <= a	Transfer a	
0001	y <= a+1	Increment a	
0010	y <= a-1	Decrement a	
0011	y <= b	Transfer b	Arithmetic
0100	y <= b+1	Increment b	
0101	y <= b-1	Decrement b	
0110	y <= a+b	Add a and b	
0111	y <= a+b+cin	Add a and b with carry	
1000	y <= NOT a	Complement a	
1001	y <= NOT b	Complement b	
1010	y <= a AND b	AND	
1011	y <= a OR b	OR	Logic
1100	y <= a NAND b	NAND	
1101	y <= a NOR b	NOR	
1110	y <= a XOR b	XOR	
1111	y <= a XNOR b	XNOR	

- Instead of writing the entire code in a single VHDL architecture, I divide my design into blocks.
- Each block is a component and we can re-use components!
- Becomes much easier to collaborate in teams, if design is modular.

Let's build a modular design from scratch

AOI Gate

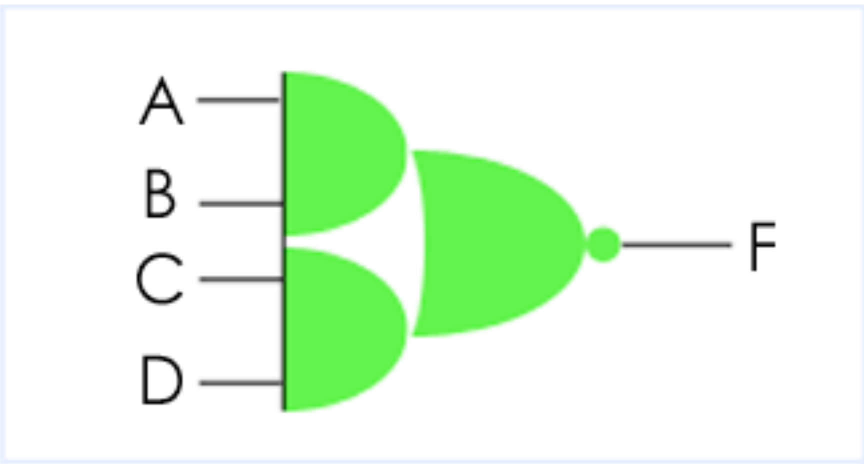


$$F = \overline{(A \wedge B) \vee (C \wedge D)}$$

2-2 AOI				
INPUT				OUTPUT
A	B	C	D	F
0	X	X	0	1
X	0	X	0	1
0	X	0	X	1
X	0	0	X	1
1	1	X	X	0
X	X	1	1	0

- We want to describe an and-or-invert (AOI) gate in VHDL.
- If we consider the AOI gate as a single chip package, it will have four input pins and one output pin
- AOI gate is particularly popular because in CMOS technology it can be implemented as a single gate, whose size is is less than if the AND, NOT, and OR functions were implemented separately.

AOI Gate



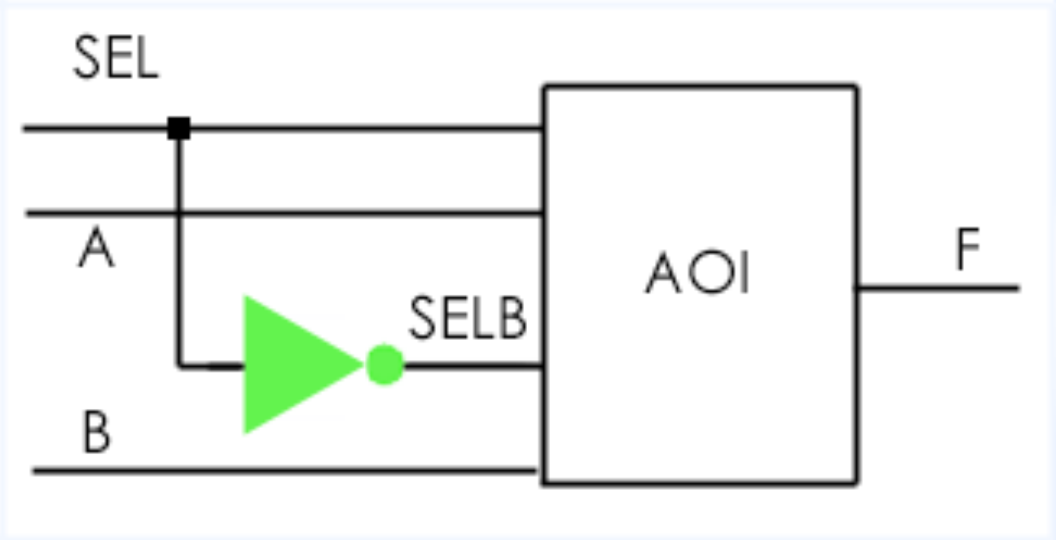
$$F = \overline{(A \land B) \lor (C \land D)}$$

2-2 AOI				
INPUT				OUTPUT
A	B	C	D	F
0	X	X	0	1
X	0	X	0	1
0	X	0	X	1
X	0	0	X	1
1	1	X	X	0
X	X	1	1	0

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity AOI is
port (
  A, B, C, D: in STD_LOGIC;
  F : out STD_LOGIC
);
end entity;

architecture V1 of AOI is
begin
  F <= not ((A and B) or (C and D));
end architecture;
```

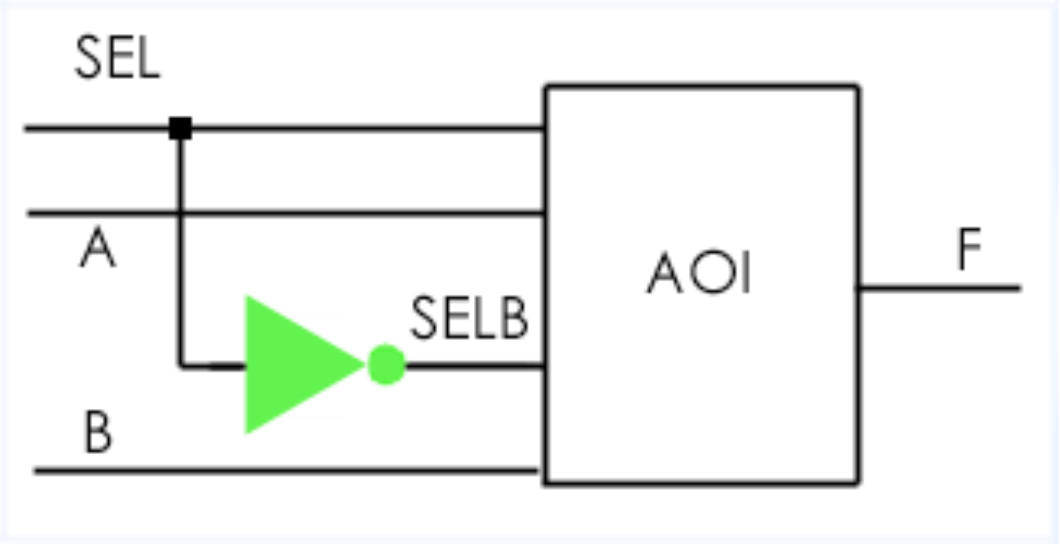
How to implement a bigger design?

- Now I want to use that AOI gate in my design.
- In particular, I want to implement the following circuit:



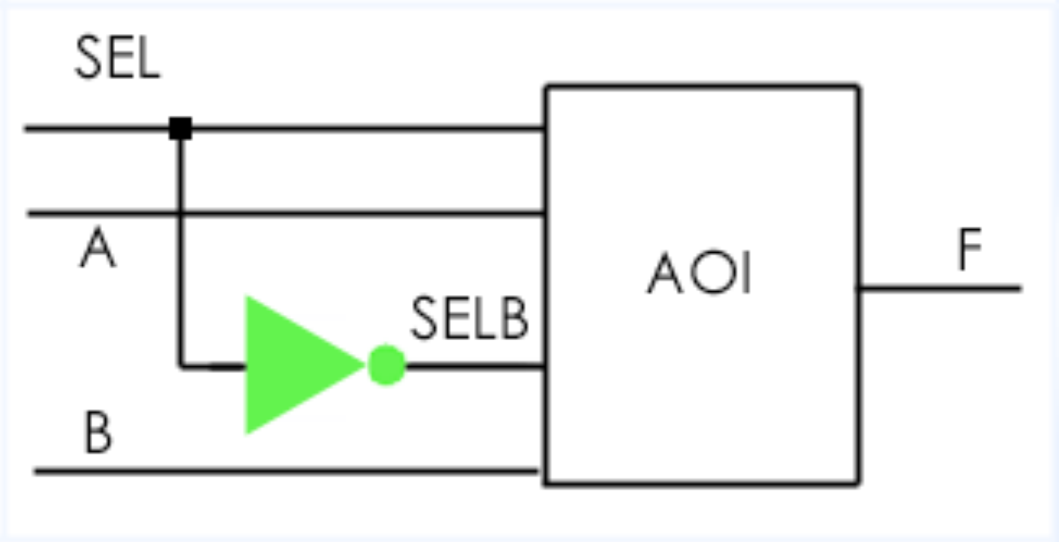
- Where the **INVERTER** and the **AOI** are two components inside another design.
- First I need to **instantiate** the components and **port map** them afterwards.

Instantiation



- I want to place two components inside another higher level design.
- AOI is a component.
- INVERTER is another component.
- Instantiating components, means we are inserting them into a bigger design.

Port Mapping



- Port Mapping means we are connecting the components to each other and to the input and output ports of the higher level design.

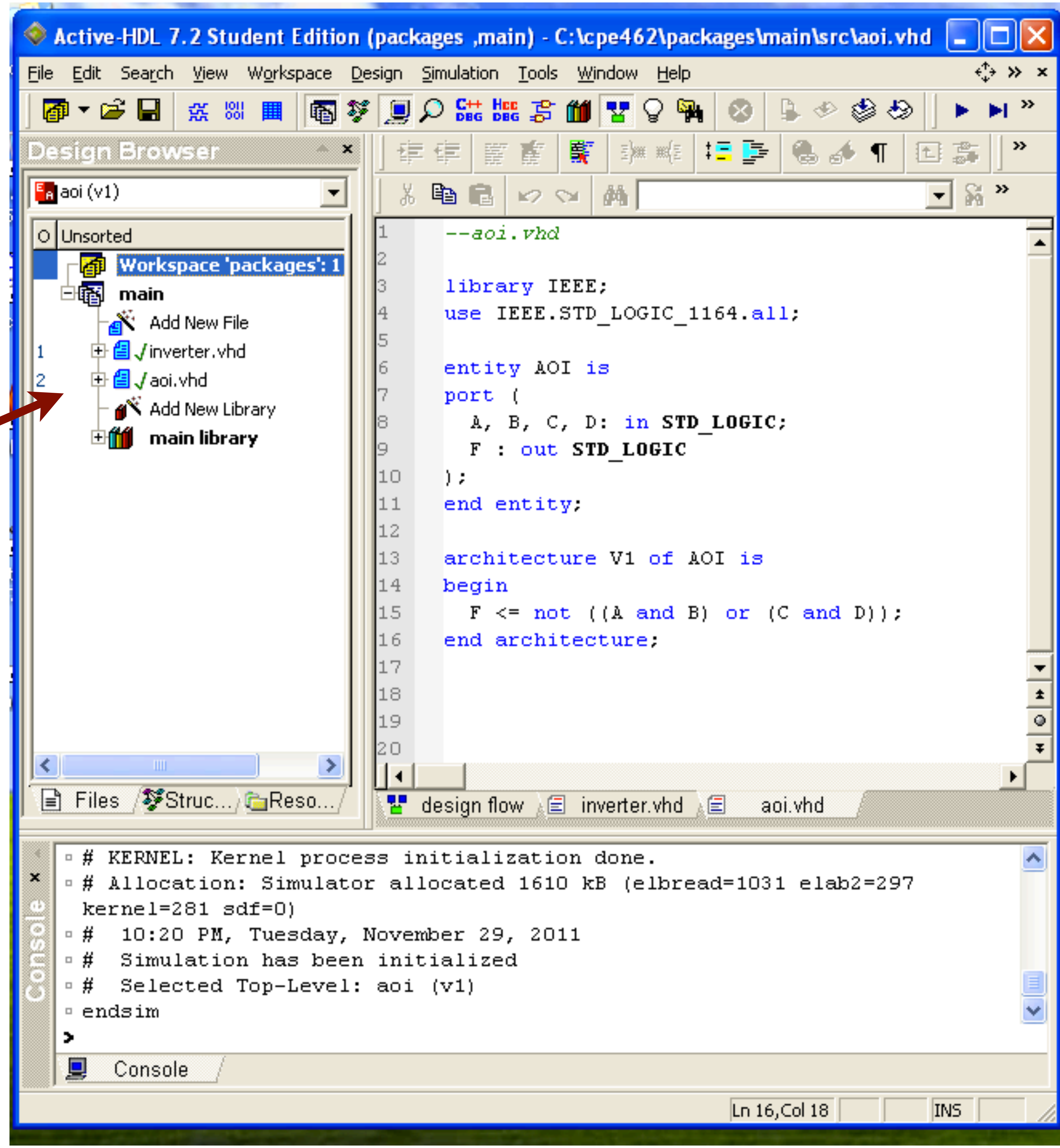
How to use components in VHDL

Step #1 - Create a separate VHDL file for each different component:

```
--inverter.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity INVERTER is
port (
  A : in STD_LOGIC;
  F : out STD_LOGIC
);
end entity;

architecture V1 of INVERTER is
begin
  F <= not (A);
end architecture;
```

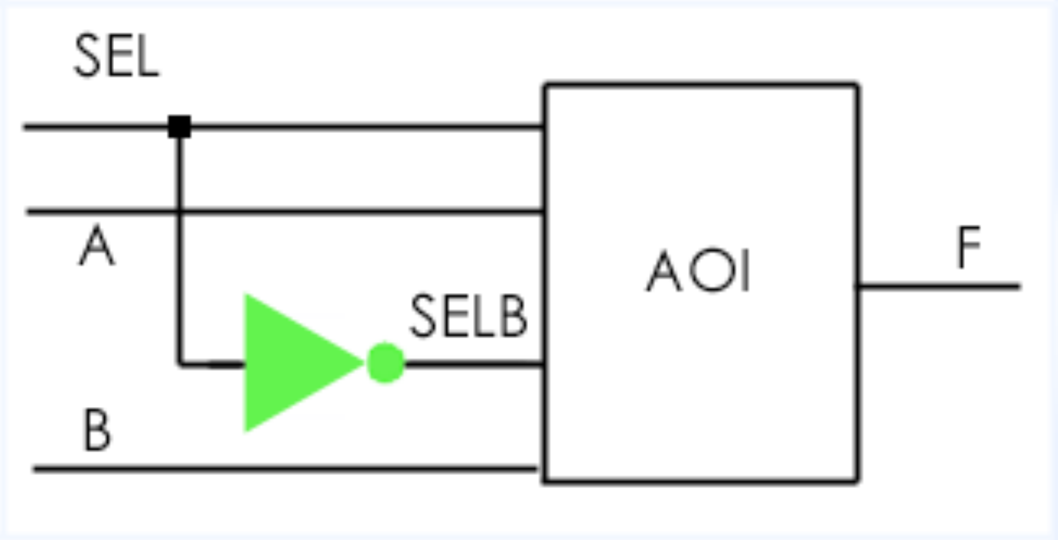
```
--aoi.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity AOI is
port (
  A, B, C, D: in STD_LOGIC;
  F : out STD_LOGIC
);
end entity;

architecture V1 of AOI is
begin
  F <= not ((A and B) or (C and D));
end architecture;
```

Create two new files in active HDL, and add your code to it.

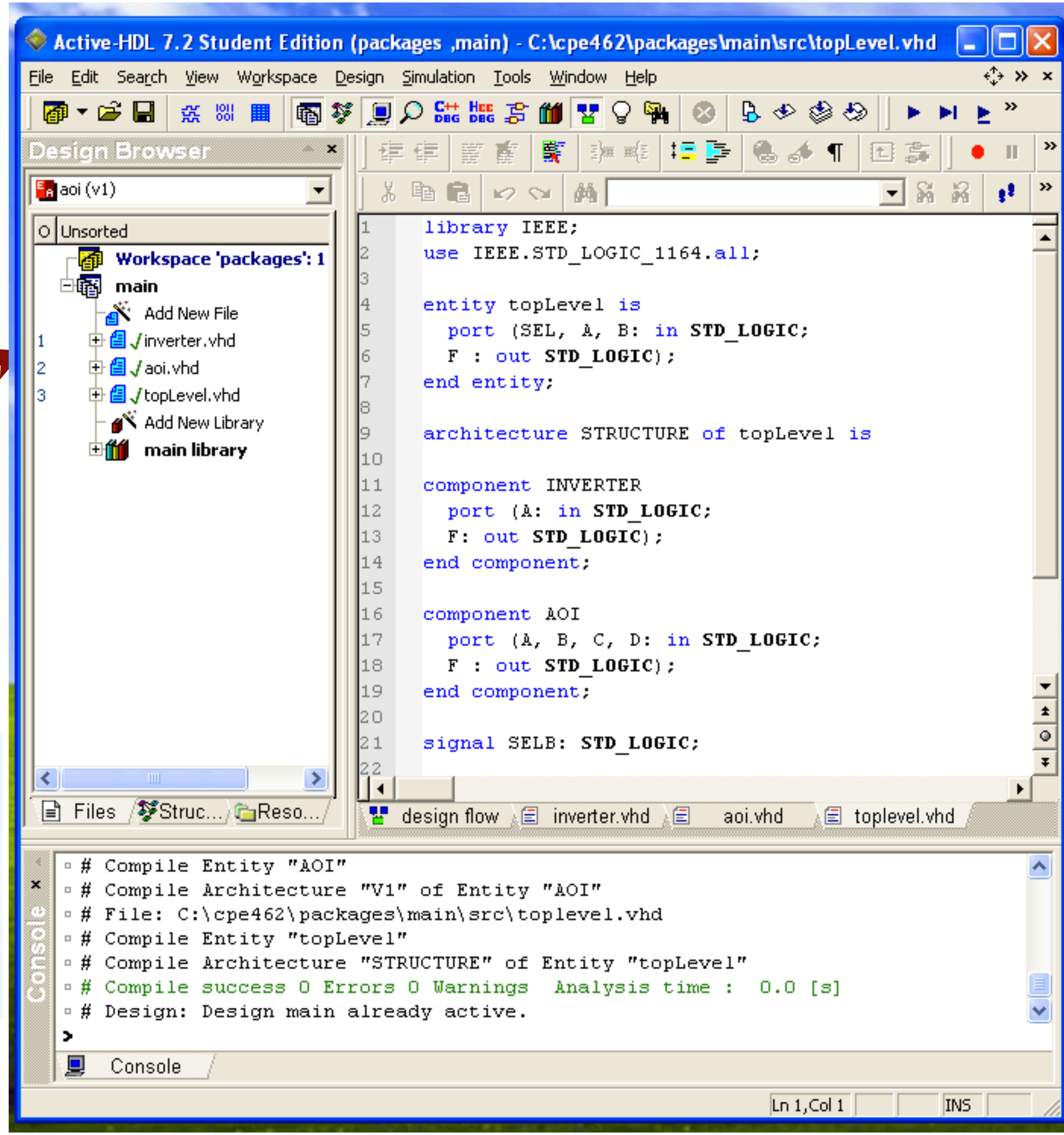


How to use components in VHDL

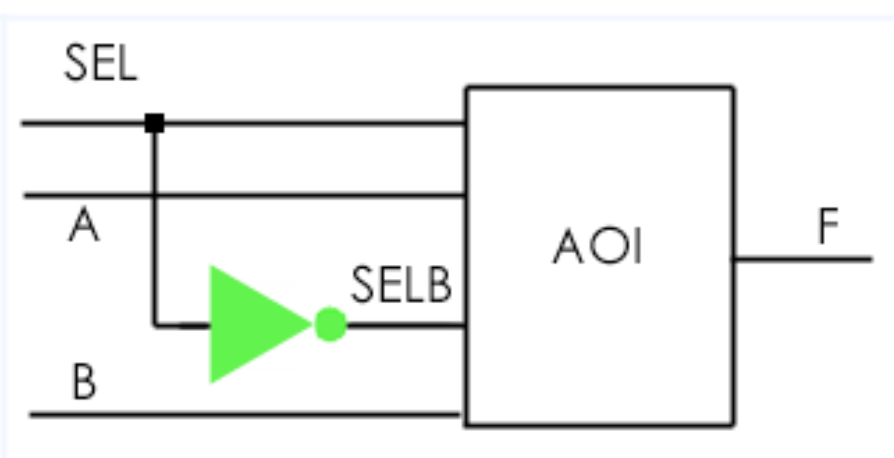
Step #2 - Create another VHDL file that uses the desired components.



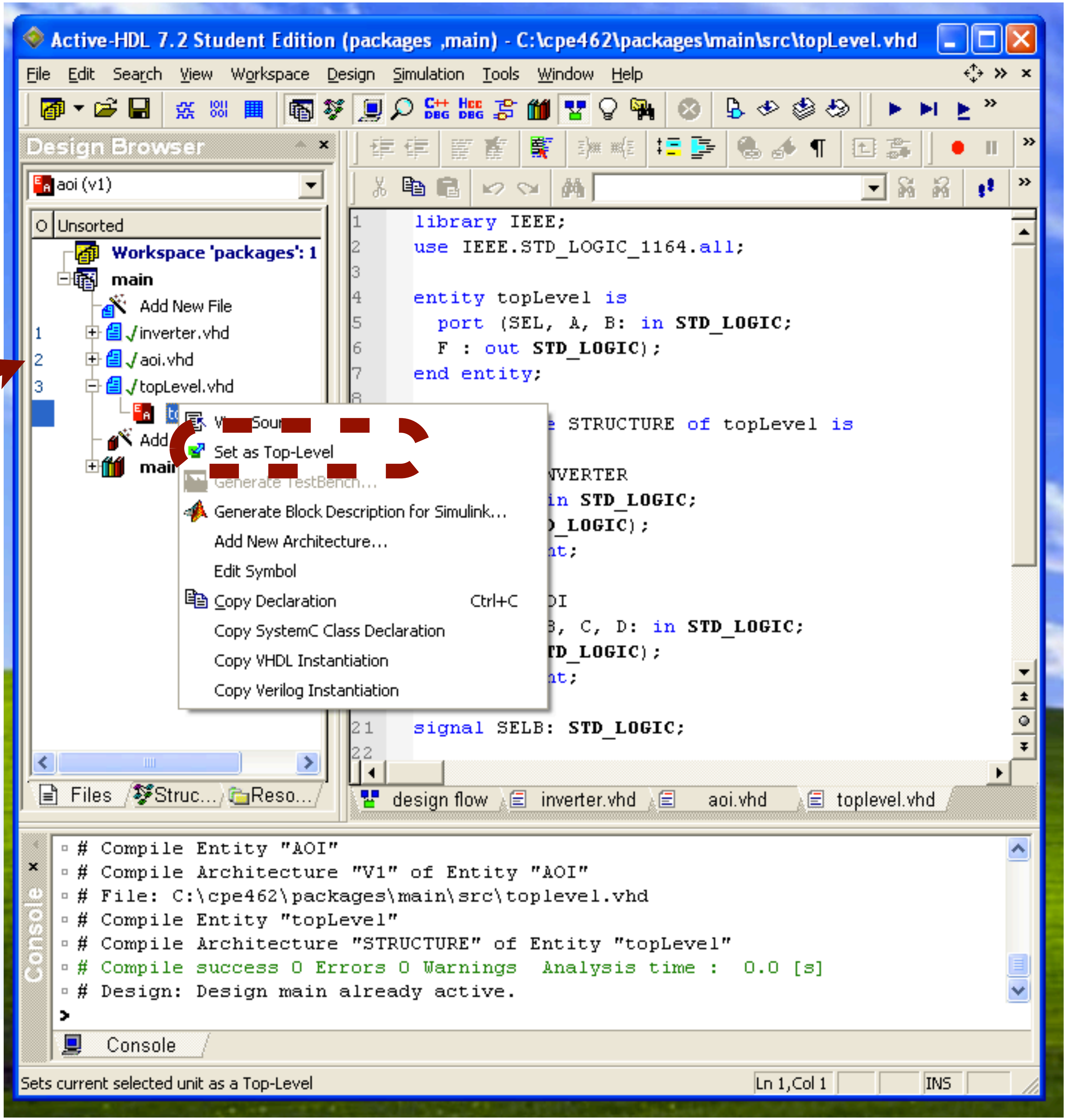

```
--topLevel.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity topLevel is
  port (SEL, A, B: in STD_LOGIC;
  F : out STD_LOGIC);
end entity;

architecture STRUCTURE of topLevel is

component INVERTER
  port (A: in STD_LOGIC;
  F: out STD_LOGIC);
end component;

component AOI
  port (A, B, C, D: in STD_LOGIC;
  F : out STD_LOGIC);
end component;

signal SELB: STD_LOGIC;

begin
  G1: INVERTER port map (SEL, SELB);
  G2: AOI port map (SEL, A, SELB, B, F);
end architecture;
```

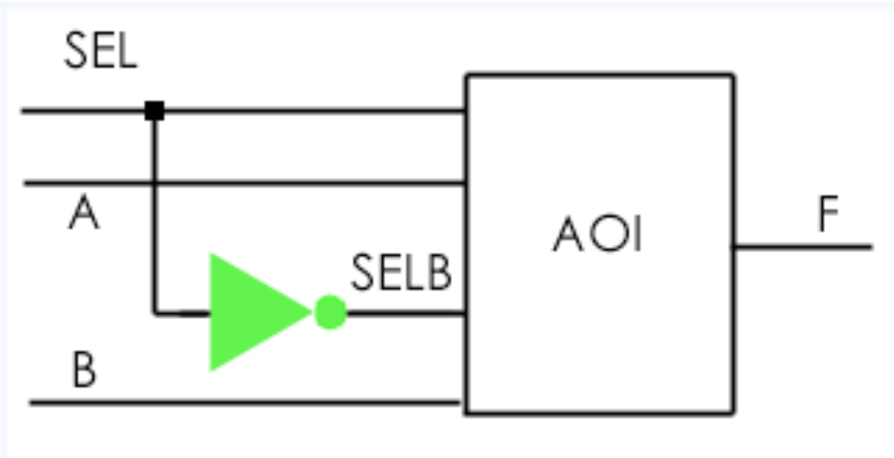


Create another file in active HDL with the topLevel circuit

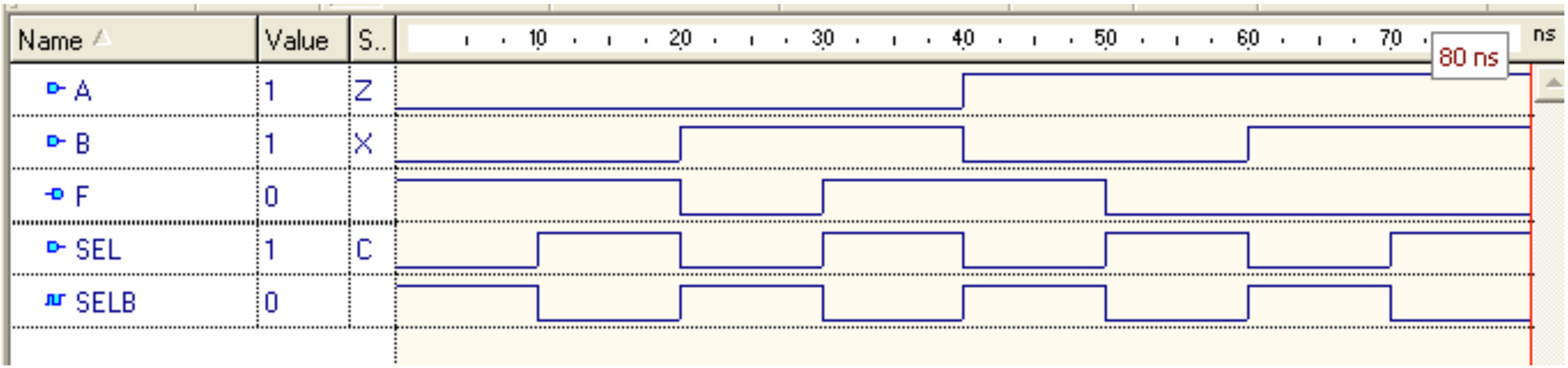


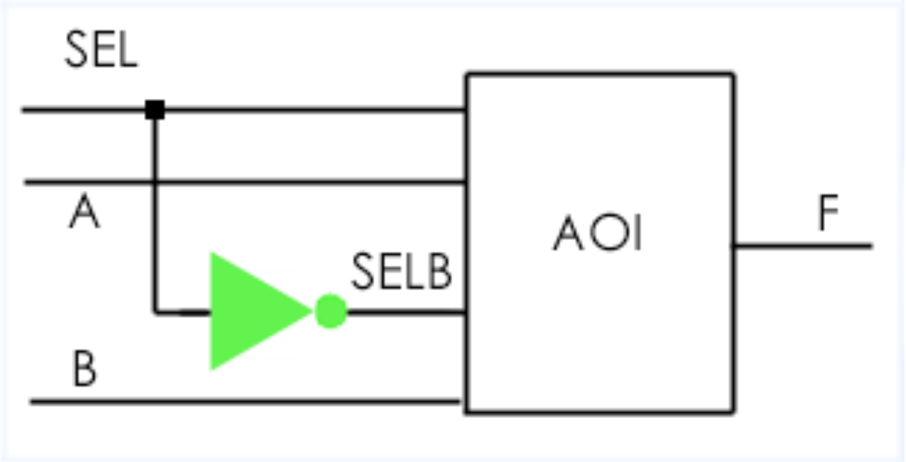
Make the topLevel.vhd file the top level!





Simulation

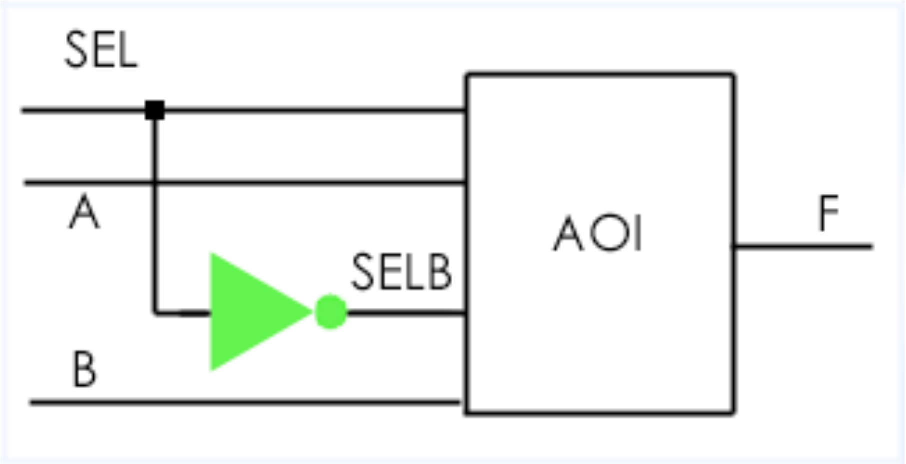




- This is an unusual 2-1 MUX
- When SEL=1, F is NOT(A)
- When SEL=0, F is NOT(B)

Detailed look at topLevel.vhd

- In the architecture, before the begin I must specify the components I am going to use. This is the **component declaration**.



- The component section looks very similar to the appropriate entity section.

```
--topLevel.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity topLevel is
  port (SEL, A, B: in STD_LOGIC;
  F : out STD_LOGIC);
end entity;

architecture STRUCTURE of topLevel is

component INVERTER
  port (A: in STD_LOGIC;
  F: out STD_LOGIC);
end component;

component AOI
  port (A, B, C, D: in STD_LOGIC;
  F : out STD_LOGIC);
end component;

signal SELB: STD_LOGIC;

begin
  G1: INVERTER port map (SEL, SELB);
  G2: AOI port map (SEL, A, SELB, B, F);
end architecture;
```

topLevel.vhd

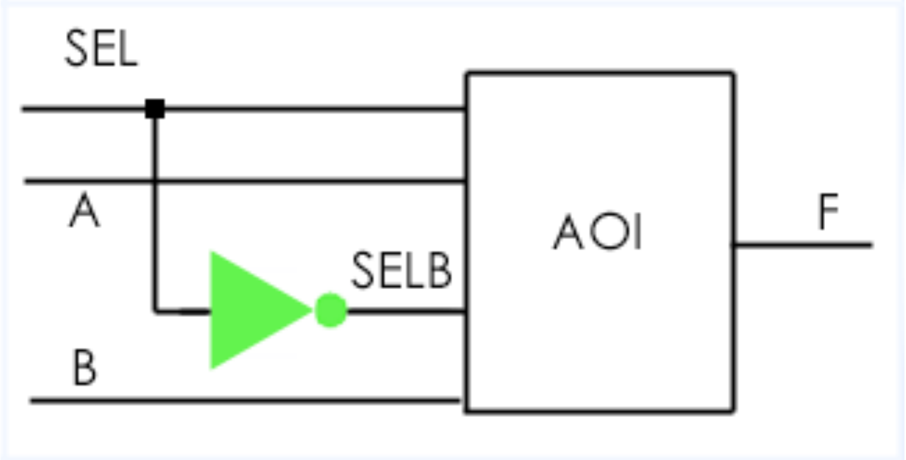
Component declaration must be identical to the entity part of the respective design!

```
--aoi.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity AOI is
port (
  A, B, C, D: in STD_LOGIC;
  F : out STD_LOGIC
);
end entity;

architecture V1 of AOI is
begin
  F <= not ((A and B) or (C and D));
end architecture;
```

```
--topLevel.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity topLevel is
  port (SEL, A, B: in STD_LOGIC;
  F : out STD_LOGIC);
end entity;

architecture STRUCTURE of topLevel is

component INVERTER
  port (A: in STD_LOGIC;
  F: out STD_LOGIC);
end component;

component AOI
  port (A, B, C, D: in STD_LOGIC;
  F : out STD_LOGIC);
end component;

signal SELB: STD_LOGIC;

begin
  G1: INVERTER port map (SEL, SELB);
  G2: AOI port map (SEL, A, SELB, B, F);
end architecture;
```

topLevel.vhd

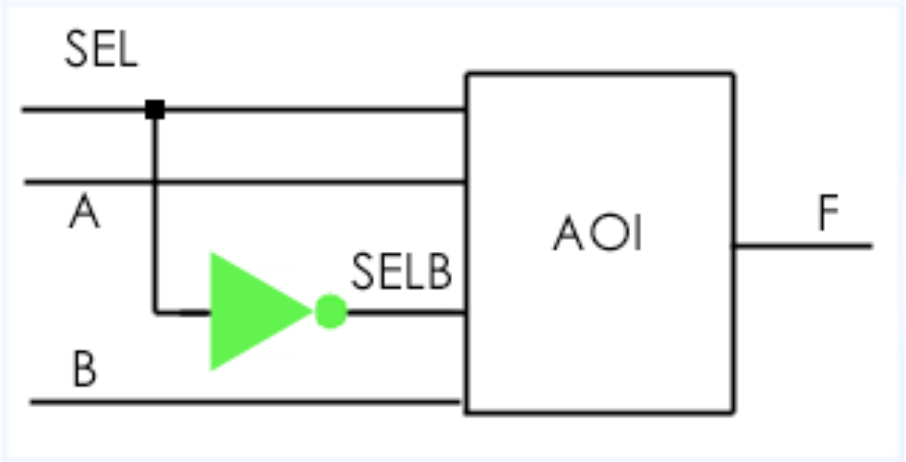
- There are two instances of **INVERTER** and **AOI** through the component instantiations at the bottom of the architecture (labelled G1 and G2).
- The instance labels (G1 and G2) identify two specific instances of the components, and are mandatory.




```
--topLevel.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity topLevel is
  port (SEL, A, B: in STD_LOGIC;
  F : out STD_LOGIC);
end entity;

architecture STRUCTURE of topLevel is

component INVERTER
  port (A: in STD_LOGIC;
  F: out STD_LOGIC);
end component;

component AOI
  port (A, B, C, D: in STD_LOGIC;
  F : out STD_LOGIC);
end component;

signal SELB: STD_LOGIC;

begin
  G1: INVERTER port map (SEL, SELB);
  G2: AOI port map (SEL, A, SELB, B, F);
end architecture;
```

Port Maps

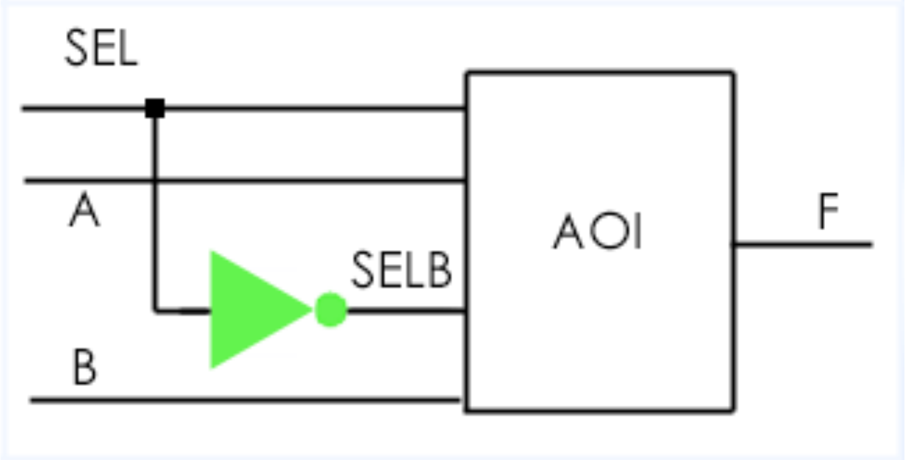
- The ports in a component declaration must usually match the ports in the entity declaration one-for-one.
- The component declaration defines the names, order, mode and types of the ports to be used when the component is instanced in the architecture body.




```
--topLevel.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity topLevel is
  port (SEL, A, B: in STD_LOGIC;
  F : out STD_LOGIC);
end entity;

architecture STRUCTURE of topLevel is

component INVERTER
  port (A: in STD_LOGIC;
  F: out STD_LOGIC);
end component;

component AOI
  port (A, B, C, D: in STD_LOGIC;
  F : out STD_LOGIC);
end component;

signal SELB: STD_LOGIC;

begin
  G1: INVERTER port map (SEL, SELB);
  G2: AOI port map (SEL, A, SELB, B, F);
end architecture;
```

Port Maps

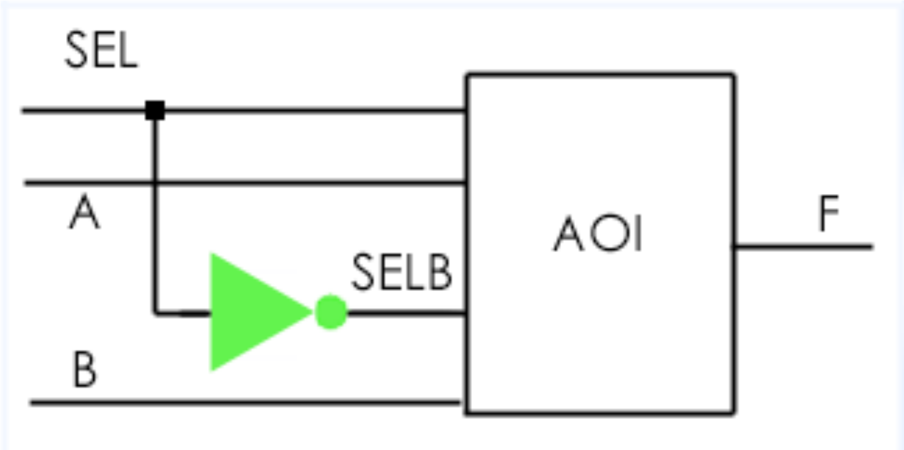
- Instancing a component implies making a local copy of the corresponding design entity
- A component is declared once within any architecture, but may be instanced any number of times.
- In this example, there is just one instance of the components **AOI** and **INVERTER**.




```
--topLevel.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity topLevel is
  port (SEL, A, B: in STD_LOGIC;
  F : out STD_LOGIC);
end entity;

architecture STRUCTURE of topLevel is

component INVERTER
  port (A: in STD_LOGIC;
  F: out STD_LOGIC);
end component;

component AOI
  port (A, B, C, D: in STD_LOGIC;
  F : out STD_LOGIC);
end component;

signal SELB: STD_LOGIC;

begin
  G1: INVERTER port map (SEL, SELB);
  G2: AOI port map (SEL, A, SELB, B, F);
end architecture;
```

Association

- Signals in an architecture are associated with ports on a component using a port map.
- A port map makes a connection between "pieces of wire" (signals) and pins on a component (ports).
- Signal SELB is associated with the F port of the INV instance and the C port of the AOI instance.

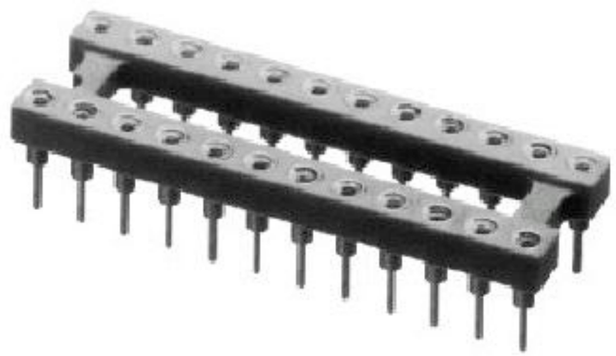


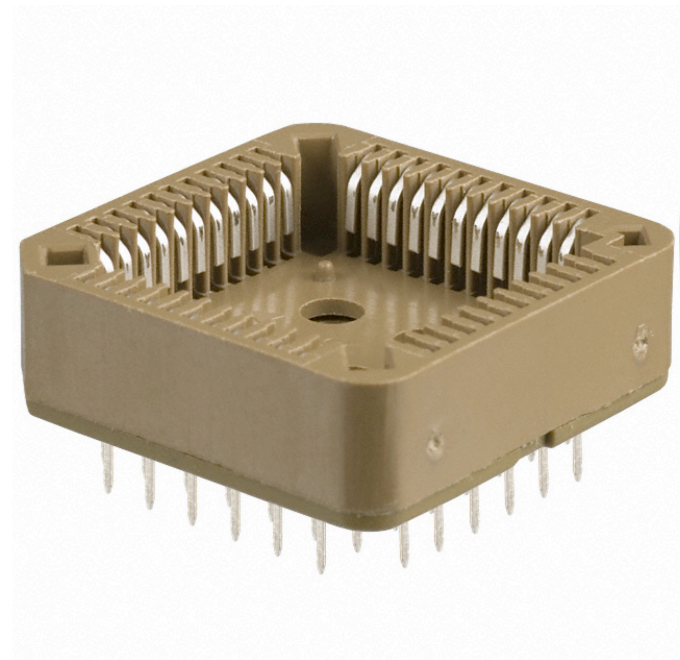

```
--topLevel.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity topLevel is
  port (SEL, A, B: in STD_LOGIC;
  F : out STD_LOGIC);
end entity;

architecture STRUCTURE of topLevel is

component INVERTER
  port (A: in STD_LOGIC;
  F: out STD_LOGIC);
end component;

component AOI
  port (A, B, C, D: in STD_LOGIC;
  F : out STD_LOGIC);
end component;

signal SELB: STD_LOGIC;

begin
  G1: INVERTER port map (SEL, SELB);
  G2: AOI port map (SEL, A, SELB, B, F);
end architecture;
```

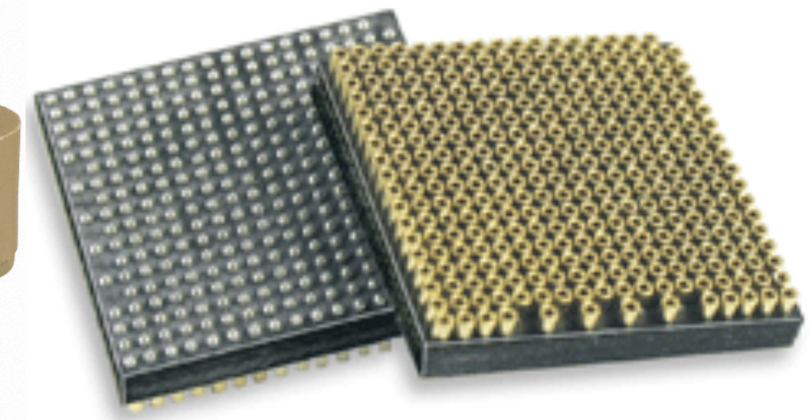
Printed Circuit Board (PCB) representation of components

Sockets and Chips

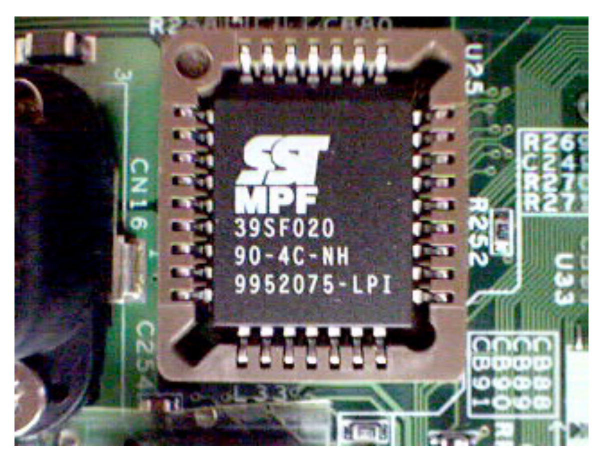
Sockets



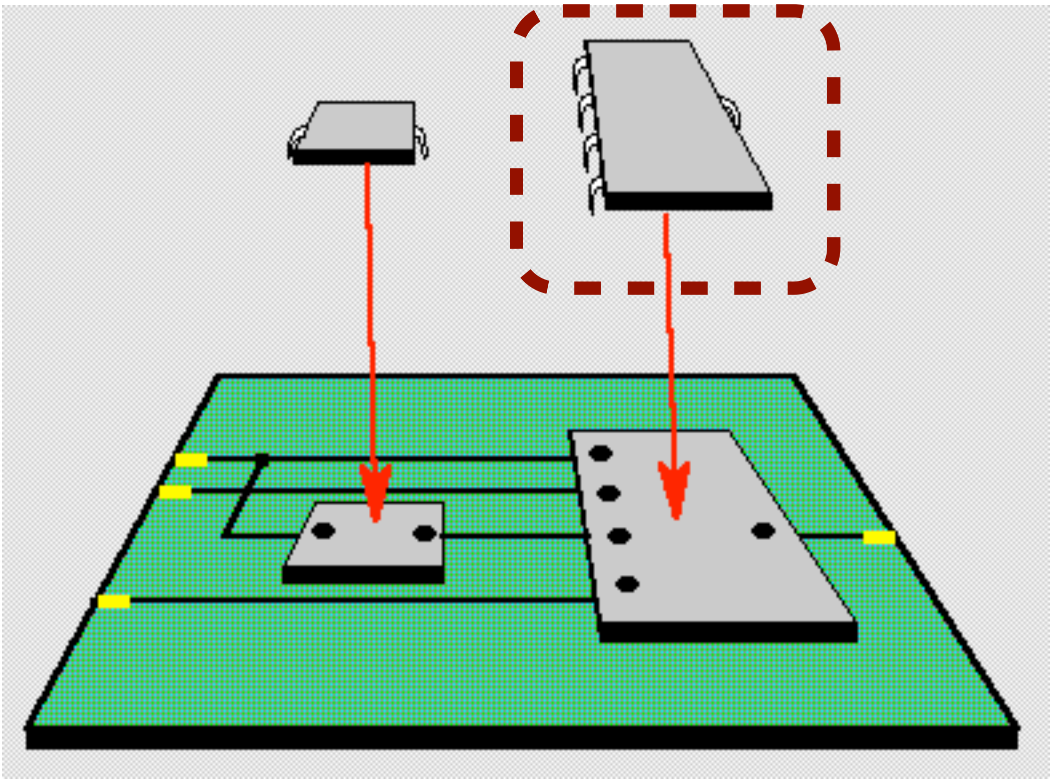


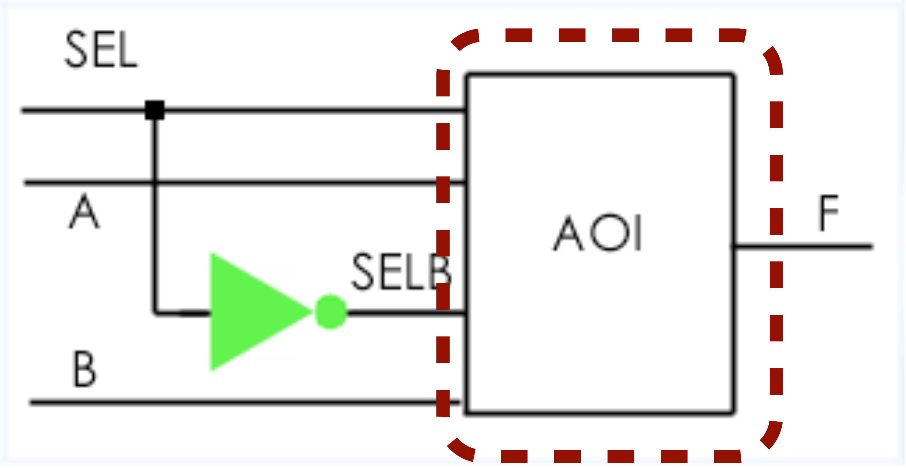


Chips



Components declaration are the list of chips we will use in our PCB

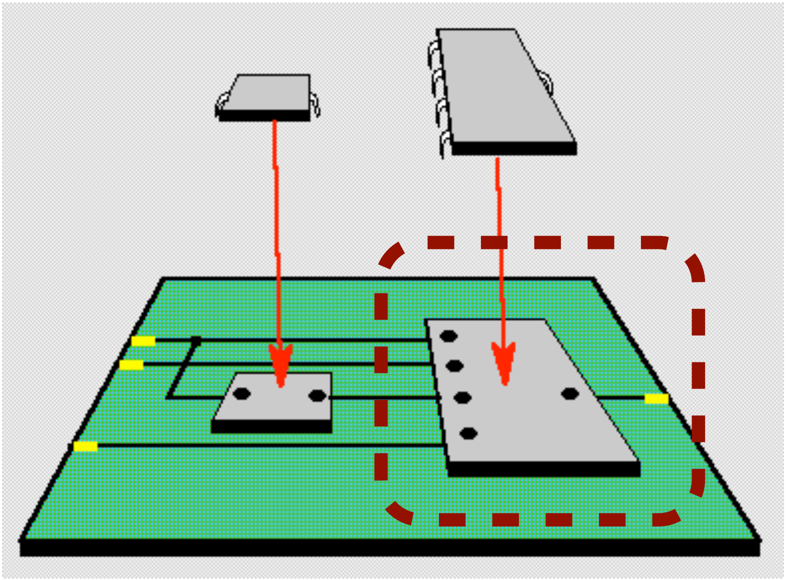




- Instantiating components in VHDL enables us to create a design hierarchy
- It's just like plugging chips into a PCB.
- In our case, the **topLevel** is the PCB, the AOI gate and INVERTER gate are two chips.

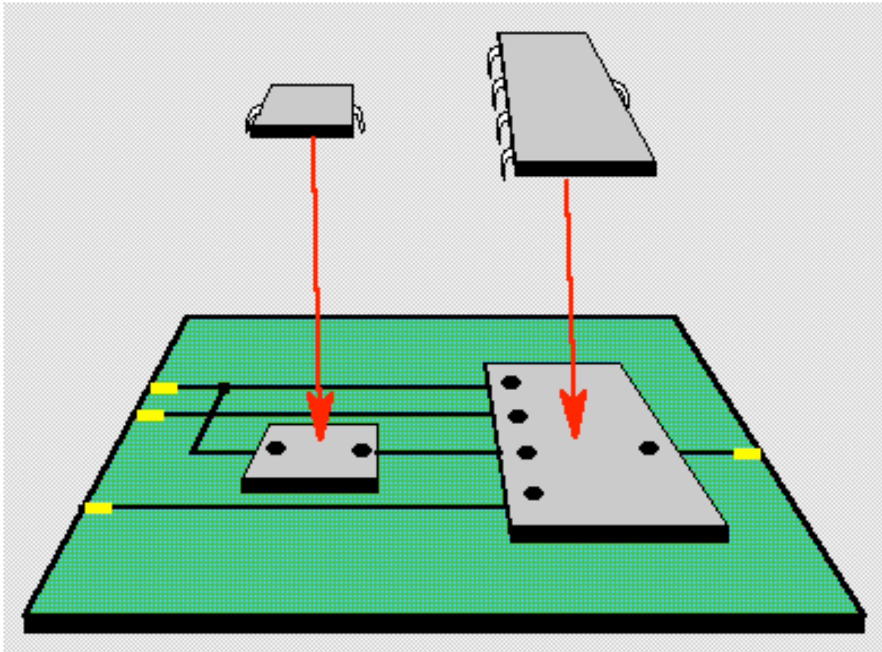
```
--this is a component declaration
component AOI
  port (A, B, C, D: in STD_LOGIC;
  F : out STD_LOGIC);
end component;
```

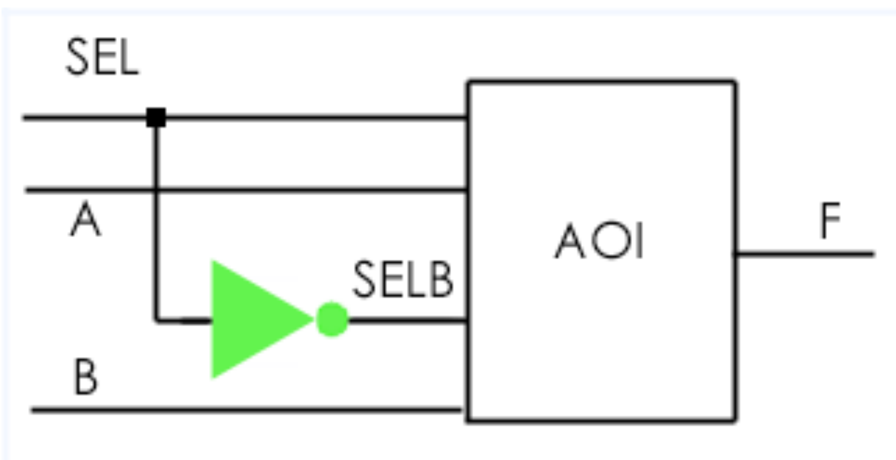
Component instances are the sockets

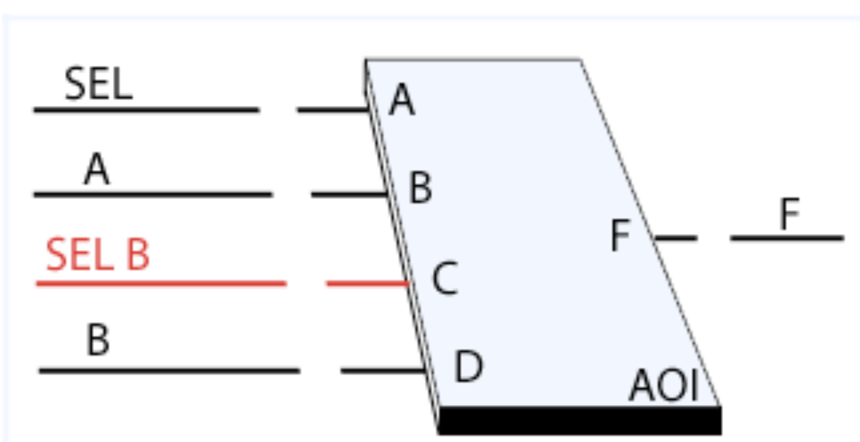


```
--this is a component declaration
component AOI
  port (A, B, C, D: in STD_LOGIC;
  F : out STD_LOGIC);
end component;

(...)

--this is a component instantiation
G2: AOI port map (SEL, A, SELB, B, F);
```

- Instantiation is essentially a VHDL term for soldering a chip socket into a PCB.
- Instantiation also assumes that the socket contains the chip referenced by the same name as it is plugged into the PCB.








```
--topLevel.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity topLevel is
  port (SEL, A, B: in STD_LOGIC;
  F : out STD_LOGIC);
end entity;

architecture STRUCTURE of topLevel is

component INVERTER
  port (A: in STD_LOGIC;
  F: out STD_LOGIC);
end component;

component AOI
  port (A, B, C, D: in STD_LOGIC;
  F : out STD_LOGIC);
end component;

signal SELB: STD_LOGIC;

begin
  G1: INVERTER port map (SEL, SELB);
  G2: AOI port map (SEL, A, SELB, B, F);
end architecture;
```

Summary of a component declaration

COMPONENT declaration:

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

```
--this is a component declaration
component AOI
  port (A, B, C, D: in STD_LOGIC;
  F : out STD_LOGIC);
end component;
```

The names of the ports must be specified, along with their modes (IN, OUT, BUFFER, or INOUT) and data types (STD_LOGIC_VECTOR, INTEGER, BOOLEAN, etc.)

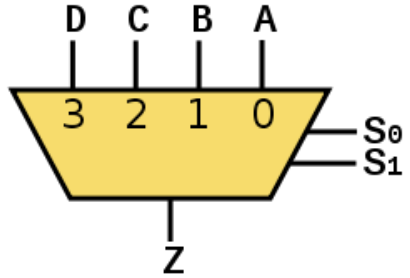
Summary of a component instantiation

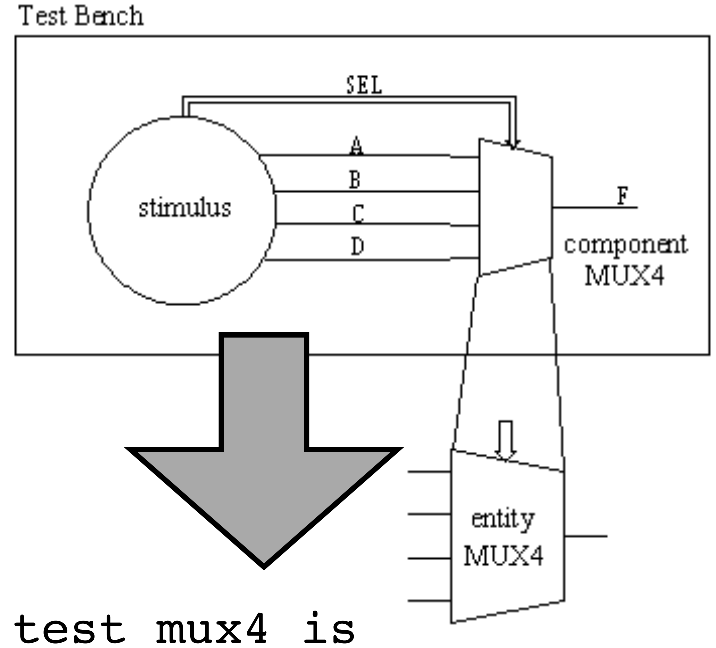
COMPONENT instantiation:

```
label: component_name PORT MAP (port_list);
```

```
--this is a component instantiation
G2: AOI port map (SEL, A, SELB, B, F);
```

A test-bench is just a PCB that tests your component!



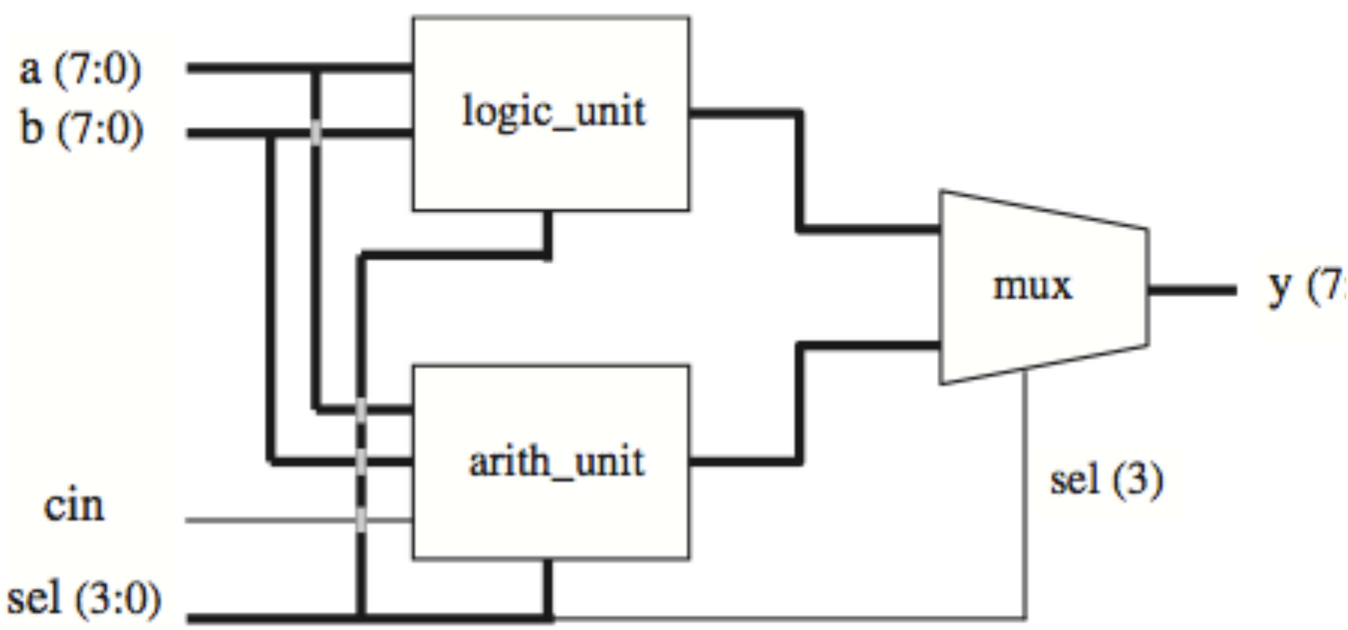



```
entity mux is
    port(a,b,c,d,s0,s1 : in bit;
    z : out bit);
end entity;

architecture myarch of mux is
begin
    z <=
    (a AND NOT(s0) AND NOT(s1))
    OR
    (b AND s0 AND NOT(s1))
    OR
    (c AND NOT(s0) AND s1)
    OR
    (d AND s0 AND s1);
end architecture ;
```

```
entity  test_mux4 is
end;

architecture bench of test_mux4 is
  component mux
    port (a, b, c, d, s0, s1: in bit;
    z :out bit);
  end component;

  signal a, b, c, d, z, s0, s1: bit;

begin
   s0 <= '0', '1' after 20 ns;
   s1 <= '0', '1' after 10ns;
   a  <= '0', '1' after 5 ns;
   b  <= '0', '1' after 20 ns, '0' after 30 ns;
   c  <= '0', '1' after 10 ns, '0' after 20 ns;
   d  <= '0', '1' after 15 ns, '1' after 25 ns;

    m: mux port map (a, b, c, d, s0, s1, z);

end bench;
```

More complex example: ALU made of components

General Structure

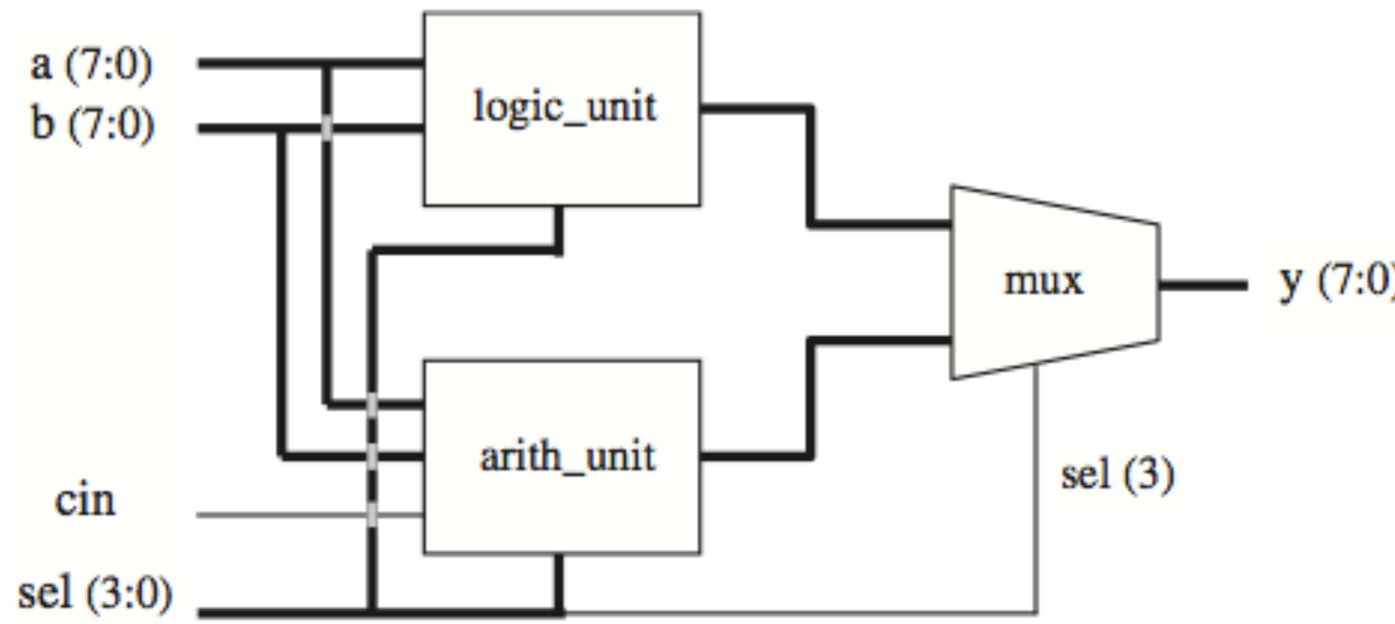


sel	Operation	Function	Unit
0000	y <= a	Transfer a	Arithmetic
0001	y <= a+1	Increment a	
0010	y <= a-1	Decrement a	
0011	y <= b	Transfer b	
0100	y <= b+1	Increment b	
0101	y <= b-1	Decrement b	
0110	y <= a+b	Add a and b	
0111	y <= a+b+cin	Add a and b with carry	
1000	y <= NOT a	Complement a	Logic
1001	y <= NOT b	Complement b	
1010	y <= a AND b	AND	
1011	y <= a OR b	OR	
1100	y <= a NAND b	NAND	
1101	y <= a NOR b	NOR	
1110	y <= a XOR b	XOR	
1111	y <= a XNOR b	XNOR	

This design will have 4 different files: One for each component description (logic_unit, arithm_unit and MUX) and a single file for the top level.

arith_unit

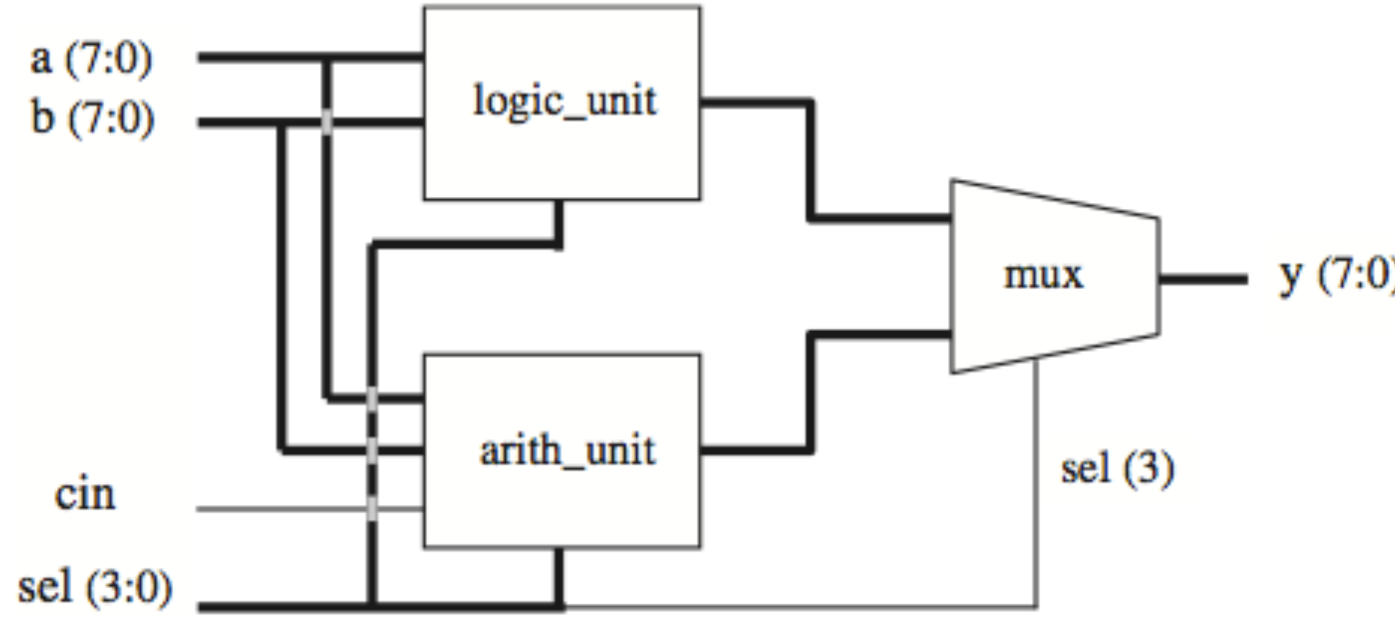
```
-------- COMPONENT arith_unit: -------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
-----------------------------------------
ENTITY arith_unit IS
   PORT ( a, b: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
          sel: IN STD_LOGIC_VECTOR (2 DOWNTO 0);
          cin: IN STD_LOGIC;
          x: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END arith_unit;
-----------------------------------------
ARCHITECTURE arith_unit OF arith_unit IS
   SIGNAL arith, logic: STD_LOGIC_VECTOR (7 DOWNTO 0);
BEGIN
   WITH sel SELECT
      x <=  a WHEN "000",
            a+1 WHEN "001",
            a-1 WHEN "010",
            b WHEN "011",
            b+1 WHEN "100",
            b-1 WHEN "101",
            a+b WHEN "110",
            a+b+cin WHEN OTHERS;
END arith_unit;
---------------------------------------------------
```



sel	Operation	Function	Unit
0000	y <= a	Transfer a	Arithmetic
0001	y <= a+1	Increment a	
0010	y <= a-1	Decrement a	
0011	y <= b	Transfer b	
0100	y <= b+1	Increment b	
0101	y <= b-1	Decrement b	
0110	y <= a+b	Add a and b	
0111	y <= a+b+cin	Add a and b with carry	
1000	y <= NOT a	Complement a	Logic
1001	y <= NOT b	Complement b	
1010	y <= a AND b	AND	
1011	y <= a OR b	OR	
1100	y <= a NAND b	NAND	
1101	y <= a NOR b	NOR	
1110	y <= a XOR b	XOR	
1111	y <= a XNOR b	XNOR	

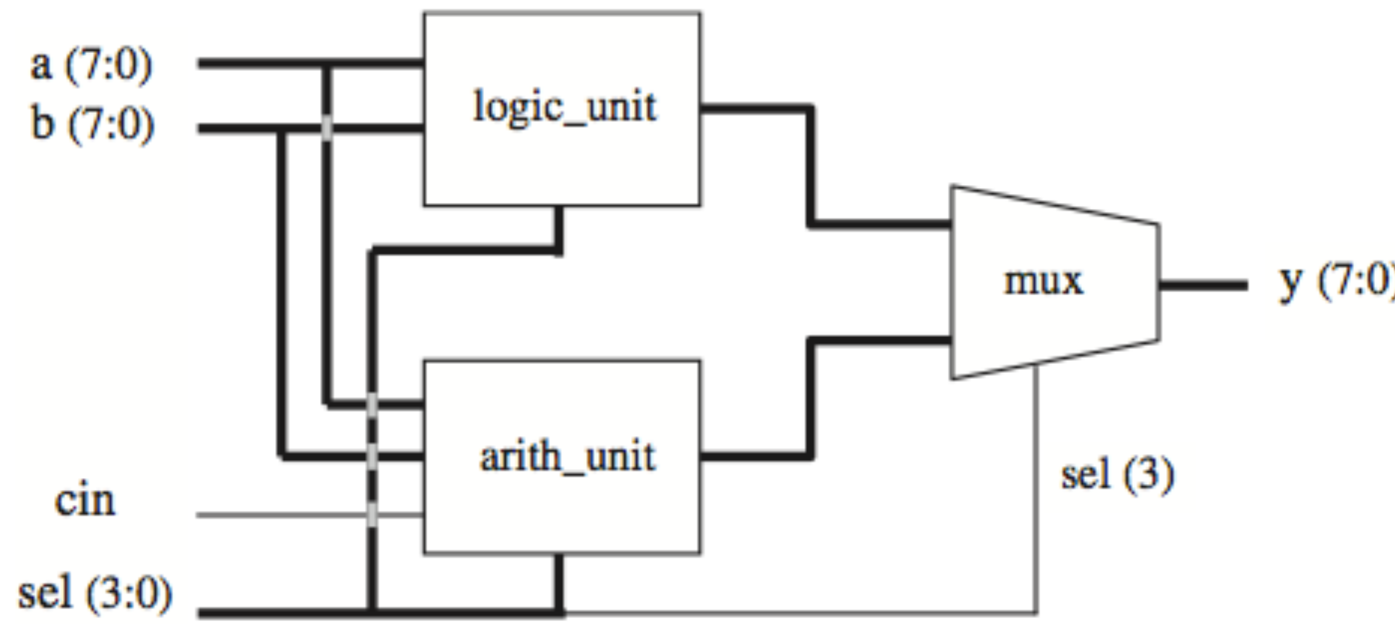
logic_unit

```
-------- COMPONENT logic_unit: -------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
----------------------------------------
ENTITY logic_unit IS
   PORT ( a, b: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
          sel: IN STD_LOGIC_VECTOR (2 DOWNTO 0);
          x: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END logic_unit;
----------------------------------------
ARCHITECTURE logic_unit OF logic_unit IS
BEGIN
   WITH sel SELECT
      x <=  NOT a WHEN "000",
            NOT b WHEN "001",
            a AND b WHEN "010",
            a OR b WHEN "011",
            a NAND b WHEN "100",
            a NOR b WHEN "101",
            a XOR b WHEN "110",
            NOT (a XOR b) WHEN OTHERS;
END logic_unit;
--------------------------------------------------
```



sel	Operation	Function	Unit
0000	y <= a	Transfer a	Arithmetic
0001	y <= a+1	Increment a	
0010	y <= a-1	Decrement a	
0011	y <= b	Transfer b	
0100	y <= b+1	Increment b	
0101	y <= b-1	Decrement b	
0110	y <= a+b	Add a and b	
0111	y <= a+b+cin	Add a and b with carry	
1000	y <= NOT a	Complement a	Logic
1001	y <= NOT b	Complement b	
1010	y <= a AND b	AND	
1011	y <= a OR b	OR	
1100	y <= a NAND b	NAND	
1101	y <= a NOR b	NOR	
1110	y <= a XOR b	XOR	
1111	y <= a XNOR b	XNOR	

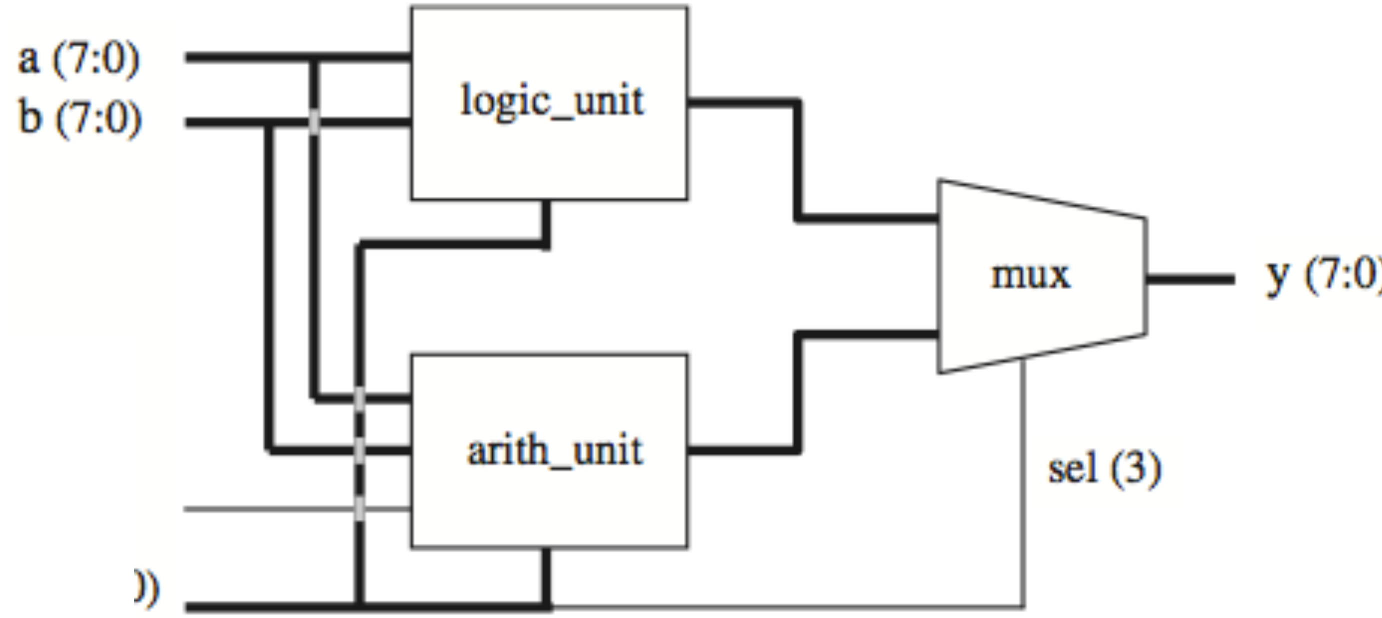
mux



sel	Operation	Function	Unit
0000	y <= a	Transfer a	
0001	y <= a+1	Increment a	
0010	y <= a-1	Decrement a	
0011	y <= b	Transfer b	Arithmetic
0100	y <= b+1	Increment b	
0101	y <= b-1	Decrement b	
0110	y <= a+b	Add a and b	
0111	y <= a+b+cin	Add a and b with carry	
1000	y <= NOT a	Complement a	
1001	y <= NOT b	Complement b	
1010	y <= a AND b	AND	
1011	y <= a OR b	OR	Logic
1100	y <= a NAND b	NAND	
1101	y <= a NOR b	NOR	
1110	y <= a XOR b	XOR	
1111	y <= a XNOR b	XNOR	

```
-------- COMPONENT mux: --------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-----------------------------------------
ENTITY mux IS
   PORT ( a, b: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
          sel: IN STD_LOGIC;
          x: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END mux;
-----------------------------------------
ARCHITECTURE mux OF mux IS
BEGIN
   WITH sel SELECT
      x <=   a WHEN '0',
             b WHEN OTHERS;
END mux;
---------------------------------------------------
```

```
-------- Project ALU (main code): ----------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-----------------------------------------
ENTITY alu IS
   PORT ( a, b: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
          cin: IN STD_LOGIC;
          sel: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
          y: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END alu;
-----------------------------------------
ARCHITECTURE alu OF alu IS
-----------------------
   COMPONENT arith_unit IS
   PORT ( a, b: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
          cin: IN STD_LOGIC;
          sel: IN STD_LOGIC_VECTOR(2 DOWNTO 0);
          x: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
   END COMPONENT;
   -----------------------
   COMPONENT logic_unit IS
   PORT ( a, b: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
          sel: IN STD_LOGIC_VECTOR(2 DOWNTO 0);
          x: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
   END COMPONENT;
   -----------------------
   COMPONENT mux IS
   PORT ( a, b: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
          sel: IN STD_LOGIC;
          x: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
   END COMPONENT;
   -----------------------
   SIGNAL x1, x2: STD_LOGIC_VECTOR(7 DOWNTO 0);
-----------------------
BEGIN
   U1: arith_unit PORT MAP (a, b, cin, sel(2 DOWNTO 0), x1);
   U2: logic_unit PORT MAP (a, b, sel(2 DOWNTO 0), x2);
   U3: mux PORT MAP (x1, x2, sel(3), y);
END alu;
---------------------------------------------------
```



	Operation	Function	Unit
0	y <= a	Transfer a	Arithmetic
1	y <= a+1	Increment a	
0	y <= a-1	Decrement a	
1	y <= b	Transfer b	
0	y <= b+1	Increment b	
1	y <= b-1	Decrement b	
0	y <= a+b	Add a and b	
1	y <= a+b+cin	Add a and b with carry	
0	y <= NOT a	Complement a	Logic
1	y <= NOT b	Complement b	
0	y <= a AND b	AND	
1	y <= a OR b	OR	
0	y <= a NAND b	NAND	
1	y <= a NOR b	NOR	
0	y <= a XOR b	XOR	
1	y <= a XNOR b	XNOR	

main code

PORT MAP tricks

- There are two ways to map the PORTS of a COMPONENT during its instantiation: positional mapping and nominal mapping.
- Consider the following component declaration:

```
COMPONENT inverter IS
   PORT (a: IN STD_LOGIC; b: OUT STD_LOGIC);
END COMPONENT;
...
```

- The following three instantiations are all identical.

```
    U1 : inverter PORT MAP (x,y);
U2 : inverter PORT MAP (y=>b,x=>a);
U3 : inverter PORT MAP (x=>a,y=>b);
```

- Only in U1 the mapping is positional; that is, PORTS x and y correspond to a and b, respectively.

PORT MAP tricks

```
COMPONENT inverter IS
   PORT (a: IN STD_LOGIC; b: OUT STD_LOGIC);
END COMPONENT;
...
```

```
U1 : inverter PORT MAP (x,y);
U2 : inverter PORT MAP (y=>b,x=>a);
U3 : inverter PORT MAP (x=>a,y=>b);
```

- Positional mapping (U1) is easier to write.
- Nominal mapping (U2 and U3) is less error prone.

Ports can also be left unconnected (using the keyword OPEN). For example:

```
U2: my_circuit PORT MAP (x=>a, y=>b, w=>OPEN, z=>d);
```

Next time

- Next class we will talk about:
    - Packages
    - Generic Components