DOULOS

Lesson 2

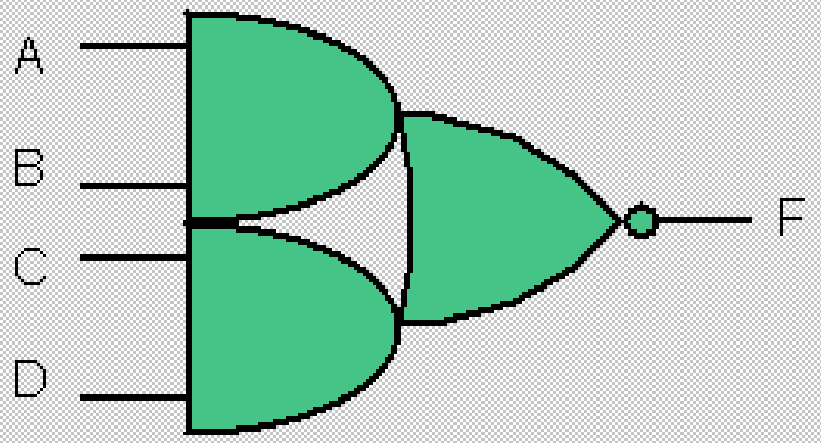
Design Entities

This lesson will introduce you to the design entity, which is the VHDL language construct for describing hierarchical hardware designs. Not everything will be explained here; some of the detail will be covered in subsequent lessons.

The Design Entity

We're going to start by learning about the design entity. A design entity is used to describe a block of hardware.

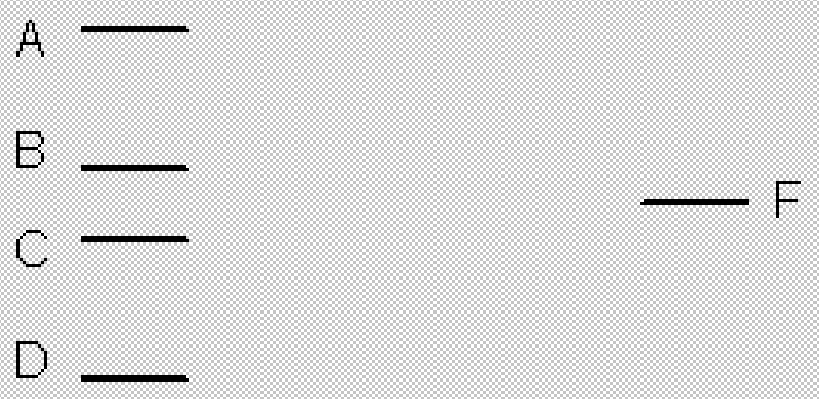
Let's begin with a simple example: an and-or-invert gate.



The Design Entity

A design entity is divided into two parts: the entity and the architecture. The entity describes the interface - the inputs and outputs.

The entity must be given a name, which may optionally be repeated at the end of the entity.

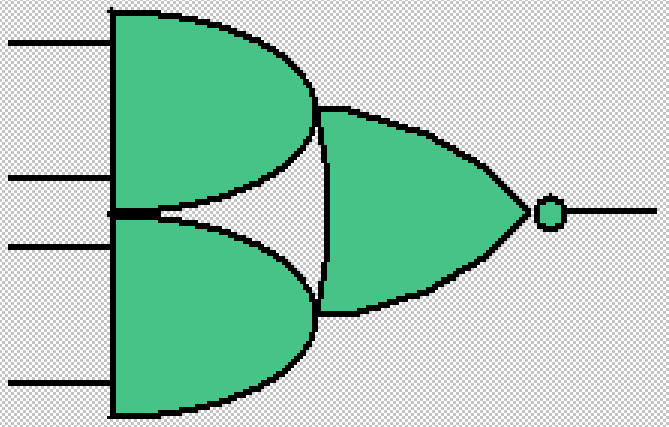



```
entity AOI is
   ...
end AOI;
```

The Design Entity

The architecture describes what is inside the block of hardware - its behaviour or structure. The architecture must also be given a name, which may optionally be repeated at the end of the architecture.

An entity can have more than one architecture.

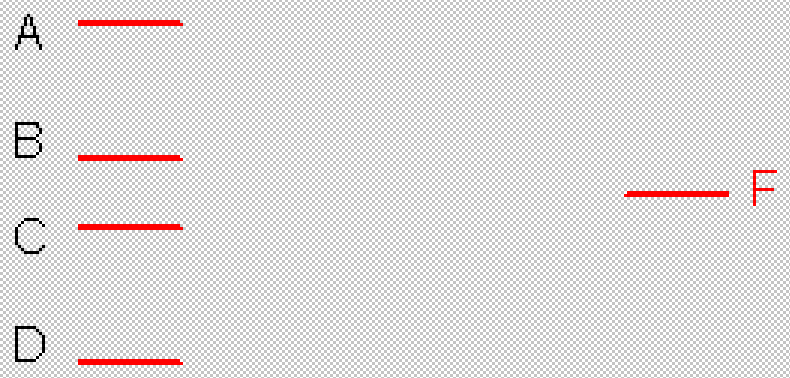


```
entity AOI is
   ...
end AOI;

architecture V1 of AOI is
   ...
begin
   ...
end V1;
```

The Design Entity

The pins around the block of hardware are described as ports in the entity.

Each port is named in the entity, and is classified as **in** or **out**. Ports with the same classification can be listed together, separated by commas, as shown.

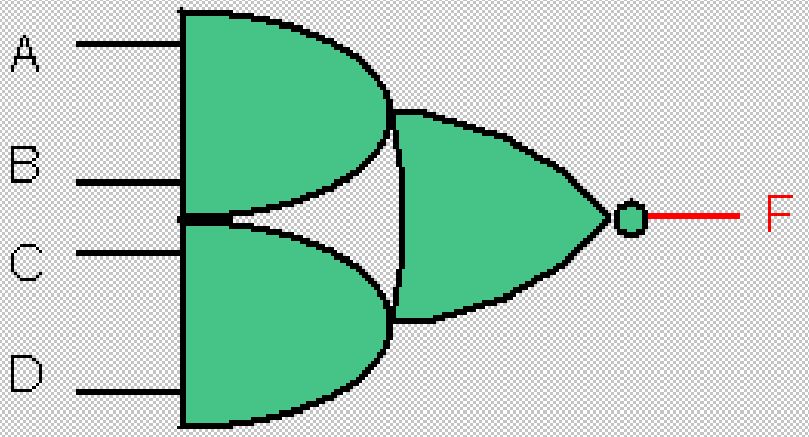



```
entity AOI is
   port (A, B, C, D: in ...
         F: out ...);
end AOI;
```

The Design Entity

In a simple example like this, the architecture can describe the boolean function of the hardware component using a signal assignment, indicated by the special symbol <=.

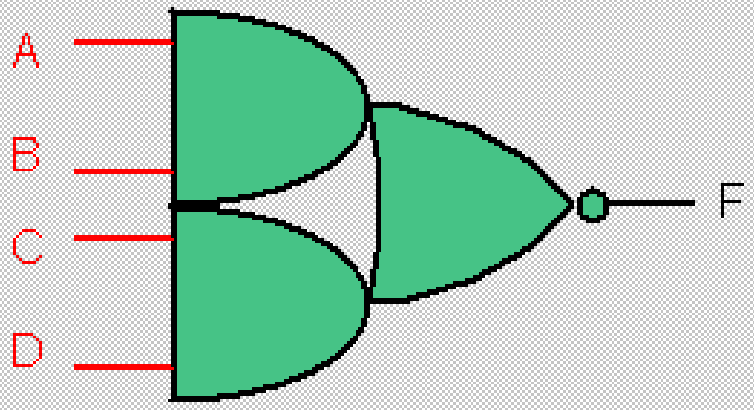
The name of the output port goes on the left hand side of the assignment.




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

architecture V1 of AOI is
begin
   F <= ...
end V1;
```

The Design Entity

On the right hand side of the signal assignment is an expression which calculates the value of the output from the value of the inputs.

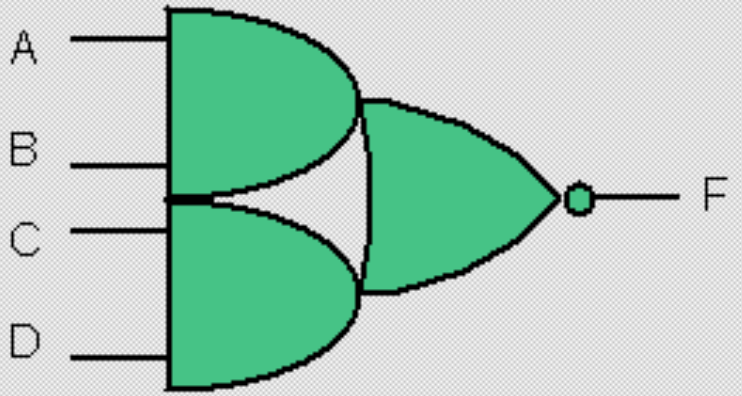



```
entity AOI is
   port (A, B, C, D: in ...
         F: out ...);
end AOI;

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

The Design Entity

Each port has a data type which describes the kind of information which can pass through the port. This may seem odd, since it is clear that each port represents just one physical pin, but this will not always be the case.

The type STD_LOGIC indicates that each port carries just one digital logic value, either '0' or '1'. The value on each port can change over time.

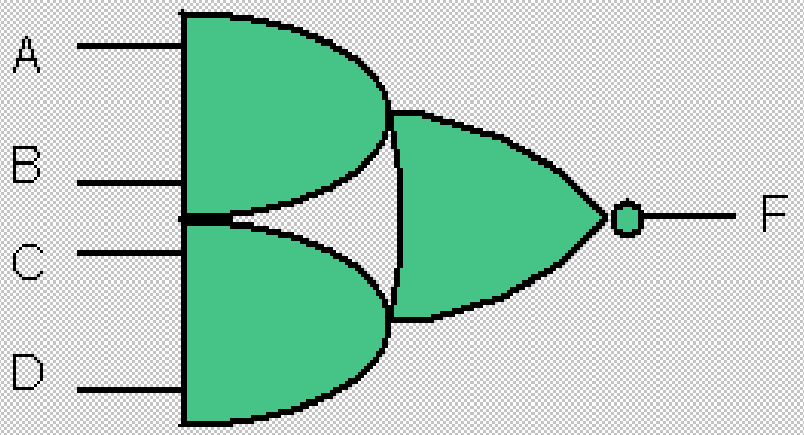



```
entity AOI is
  port (A, B, C, D: in STD_LOGIC;
        F: out STD_LOGIC);
end AOI;

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

The Design Entity

The type STD_LOGIC is defined in the package STD_LOGIC_1164 on library IEEE.

The two lines starting library and use appear at the top of many entities, and tell the VHDL compiler where to find the definition of STD_LOGIC.

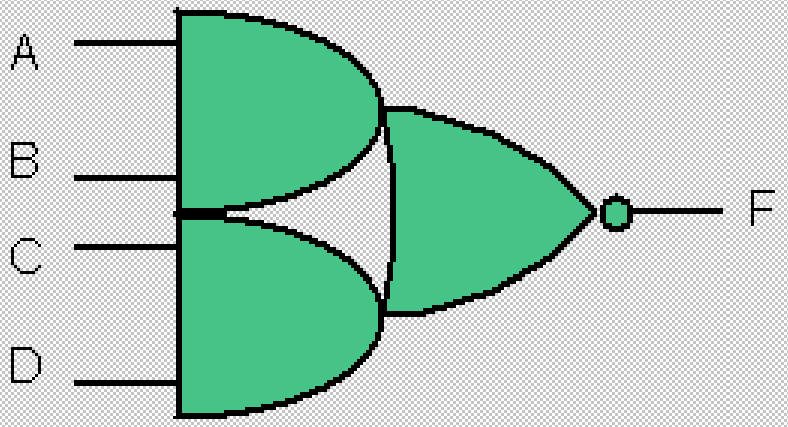



```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

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

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

The Design Entity

Data types are often defined in a package.

Confusingly, a VHDL package does NOT represent a physical package, but is just a language construct where common, shared definitions can be placed.

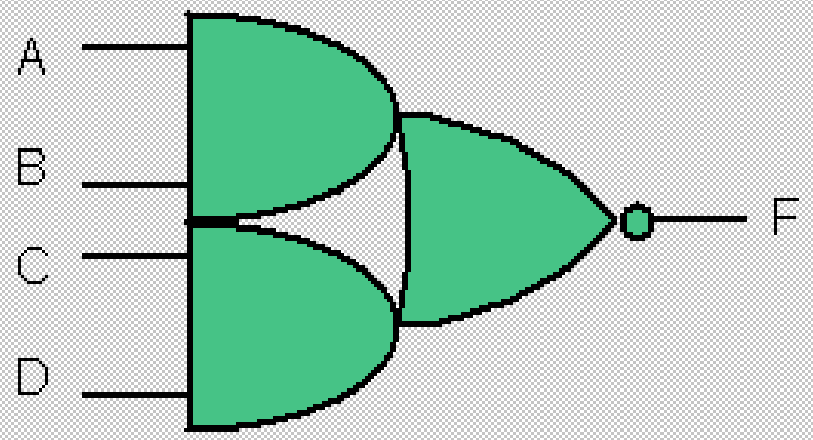



```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

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

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

The Design Entity

By the way, you may have noticed that some words are written in lower case, and others in upper case.

Well, VHDL is **not** case sensitive, so it does not actually matter! However, it is a good idea to have a convention, so in these examples built-in reserved keywords are in lower case, and user defined names in upper case.

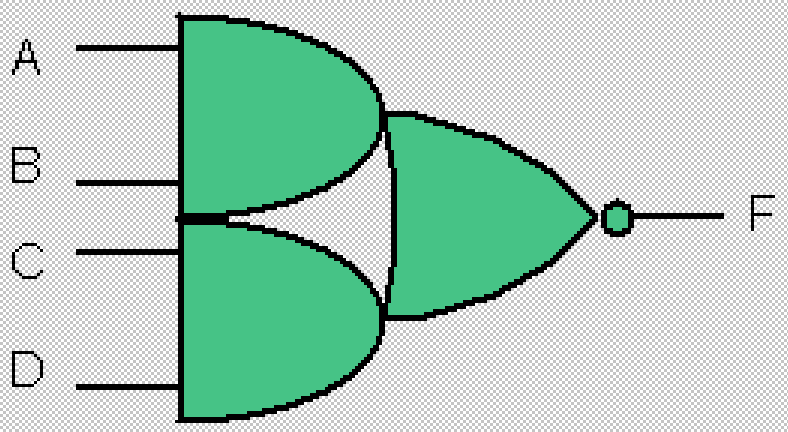



```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

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

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

The Design Entity

Here is the complete design entity. Click on the hotwords for a further explanation of each detail of the VHDL description.

When you've finished, move on to the exercises that follow. If you have difficulty with an exercise, you should review the earlier part of the lesson before proceeding.




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

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

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

Exercise 1

Using the mouse, sort the lines into the correct order by dragging them up and down.
Do not press the up arrow key!

Check Answer

Correct! Well done. Try the next exercise.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity AOI is
  port (A, B, C, D: in STD_LOGIC;
        F: out STD_LOGIC);
end AOI;
architecture V1 of AOI is
begin
  F <= not((A and B) or (C and D));
end V1;
```


The Answer

Exercise 2

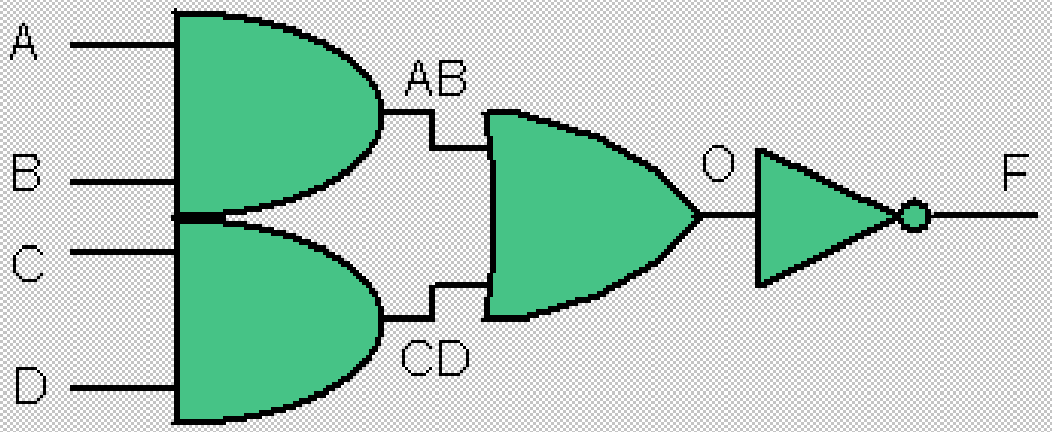
Drag and drop each VHDL term into the appropriate slot.

Correct.

architecture	= the internal function of a block of hardware
port	= a pin
std_logic	= a type representing one bit of information
design entity	= the complete definition of a block of hardware
entity	= the definition of the interface to a block of hardware
package	= a place to put common definitions

Signals

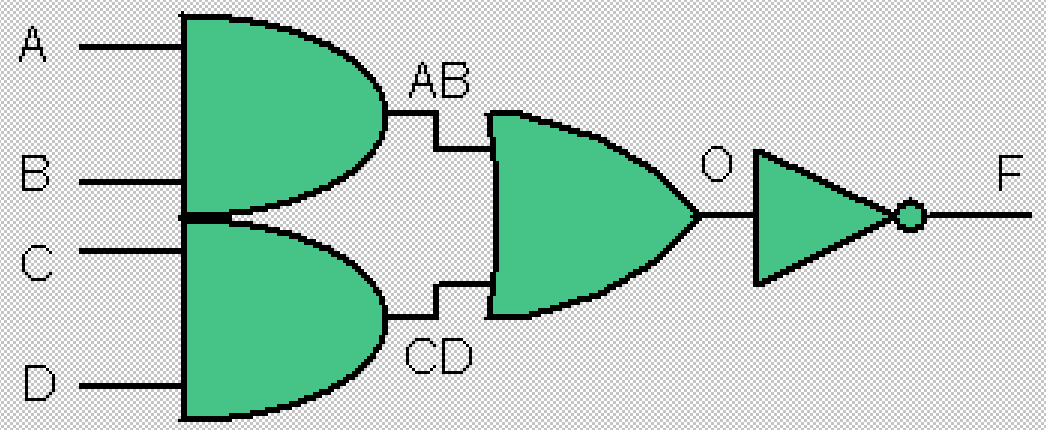
Let's now see how we can add internal connections to our design entity. We'll modify the AOI to include separate interconnected gates, instead of one boolean function.



Signals

We're now going to construct a second alternative architecture named V2 for our AOI entity.

The main reason for having architectures in VHDL is so that you can create more than one description for the same piece of hardware. This second variant of the AOI is named V2, although the name can be whatever you like.

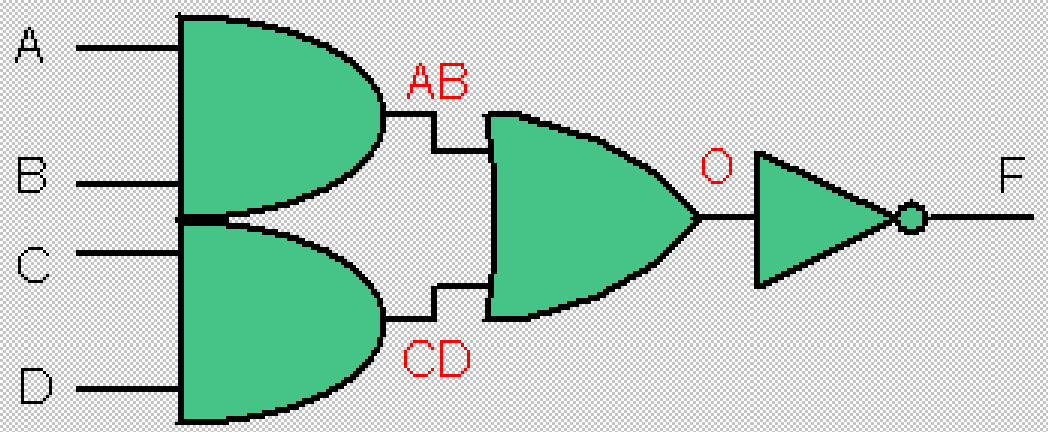



```
architecture V2 of AOI is
   ...
begin
   ...
end V2;
```

Signals

An internal connection is described in VHDL as a signal defined inside an architecture.

Each signal has a type, and signals with the same type can be listed together.

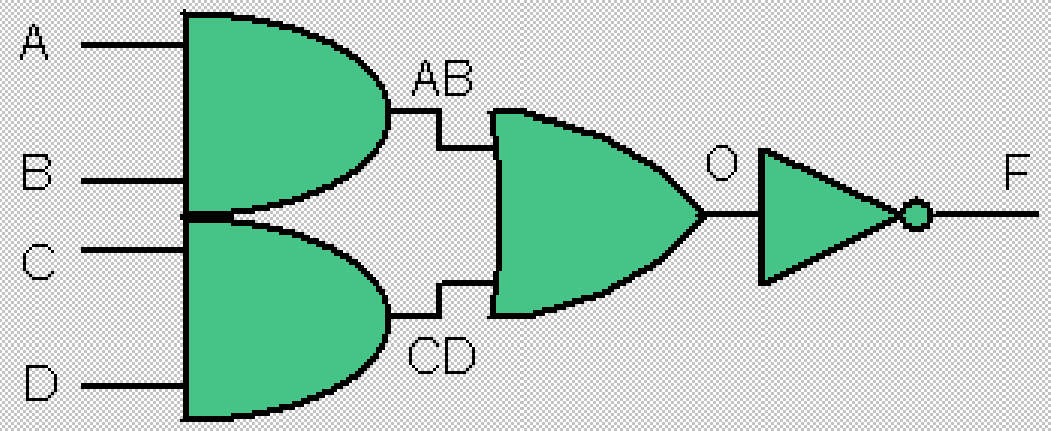



```
architecture V2 of AOI is
   signal AB, CD, O: STD_LOGIC;
begin
   ...
end V2;
```

Signals

The function of each gate is described by a concurrent signal assignment. The order in which the assignments are written has no effect on their behaviour.

Each signal assignment will be synthesized to the equivalent combinational logic gates, as shown in the diagram.




```
architecture V2 of AOI is
  signal AB, CD, O: STD_LOGIC;
begin
  AB <= A and B after 2 NS;
  CD <= C and D after 2 NS;
  O <= AB or CD after 2 NS;
  F <= not O after 1 NS;
end V2;
```

Concurrent Assignments

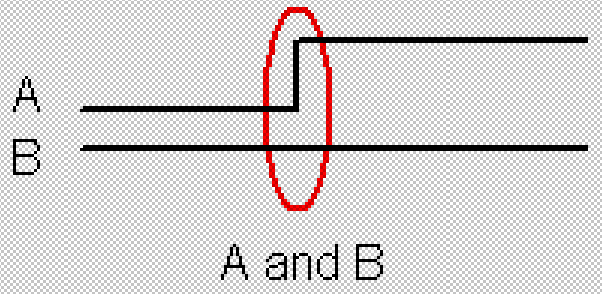
A concurrent signal assignment is triggered by an event on a signal. An event is a change in value.

An event on the input port A would trigger the assignment to AB.


```
port (A, B, C, D: in STD_LOGIC;
...
AB <= A and B after 2 NS;
```

Concurrent Assignments

The signal assignment first calculates the value of the expression on the right hand side.

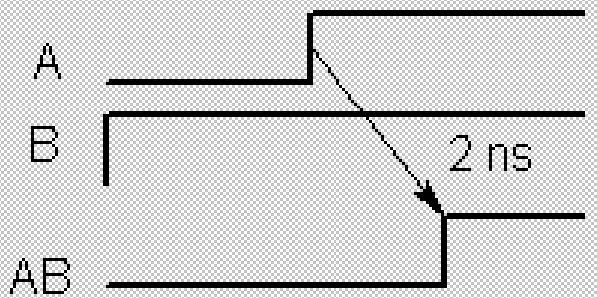



```
AB <= A and B after 2 NS;
```

Concurrent Assignments

The signal assignment then schedules a future event on the signal on the left hand side of the assignment to occur after the given delay.

The delay is very important for simulation, but not for synthesis. Synthesis tools ignore delays in signal assignments! Delay information for synthesis must be specified separately in the form of timing constraints.

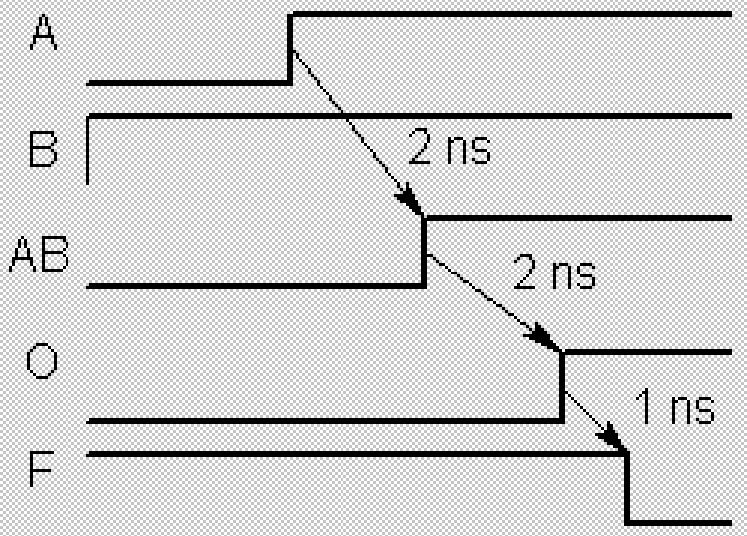



```
AB <= A and B after 2 NS;
```

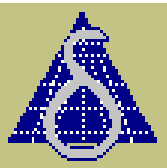
Concurrent Assignments

After 2 NS have elapsed, the event on AB occurs, which triggers the assignment to O. An event is scheduled to occur on O after a further delay of 2 NS. That event in turn triggers the assignment to F.

The total delay from A to F is 2 + 2 + 1 = 5 NS.




```
AB <= A and B after 2 NS;
CD <= C and D after 2 NS;
O <= AB or CD after 2 NS;
F <= not O after 1 NS;
```



Exercise 3

If an event occurs on A at time 0 NS, at what time does F change value. (Type your answer in the box, then hit the Enter key.)

```
architecture V1 of Logic is
  signal A, C: Std_logic;
begin
  A <= not D after 1 NS;
  F <= C xor D after 2 NS;
  C <= A xor B after 4 NS;
end V1;
```

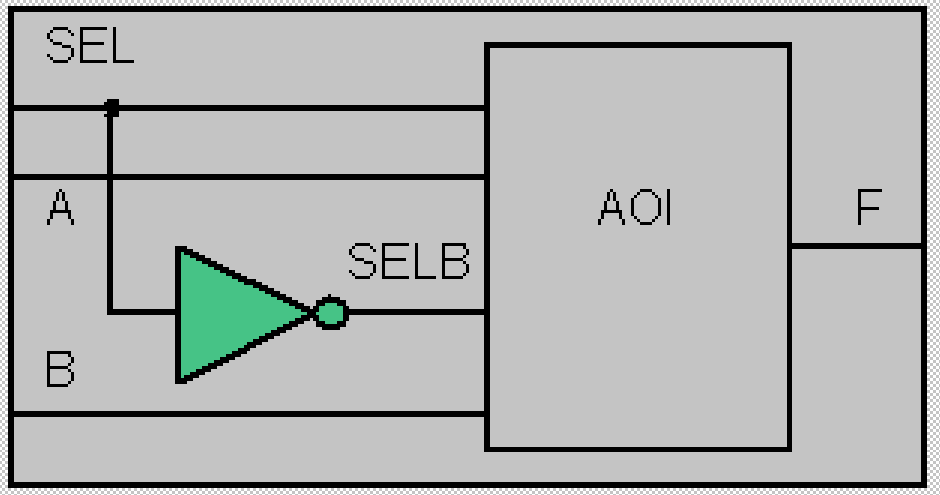
6 NS

Correct. The event on A triggers the assignment to C, which then triggers the assignment to F. The 1st assignment to A does not contribute.

Components

Let's now see how to build a hierarchy of design entities.

We will take the AOI gate and connect it to an inverter to build a multiplexer. The entity is shown here.



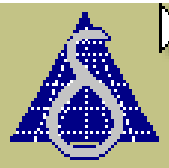

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity MUX2 is
   port (SEL, A, B: in STD_LOGIC;
         F: out STD_LOGIC);
end MUX2;
```

Components

Each lower level design entity must be declared as a component within the architecture. The component declaration repeats the name and the ports of the corresponding lower level entity.

A component is analogous to a chip socket - it creates an extra interface which allows more flexibility when building a hierarchical system out of its component parts.

```
architecture STRUCTURE of MUX2 is
   component INV
     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;
   ...
begin
   ...
end STRUCTURE;
```

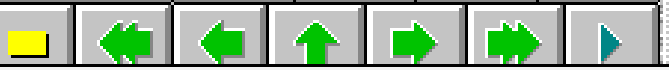



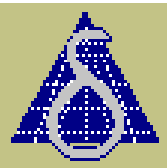
Components

Each component must now be **instantiated** within the architecture. The term instantiation may seem like obscure jargon, but the idea is very simple. An instantiation is VHDL jargon for a unique, concurrent copy of a design entity.

This architecture contains instantiations of the components INV and AOI.

```
architecture STRUCTURE of MUX2 is
  component INV
    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: INV port map(SEL, SELB);
  G2: AOI port map(SEL, A, SELB, B, F);
end STRUCTURE;
```



Components

In other words, having defined a piece of hardware as a design entity, an instantiation allows you to use that description as part of another higher level description.

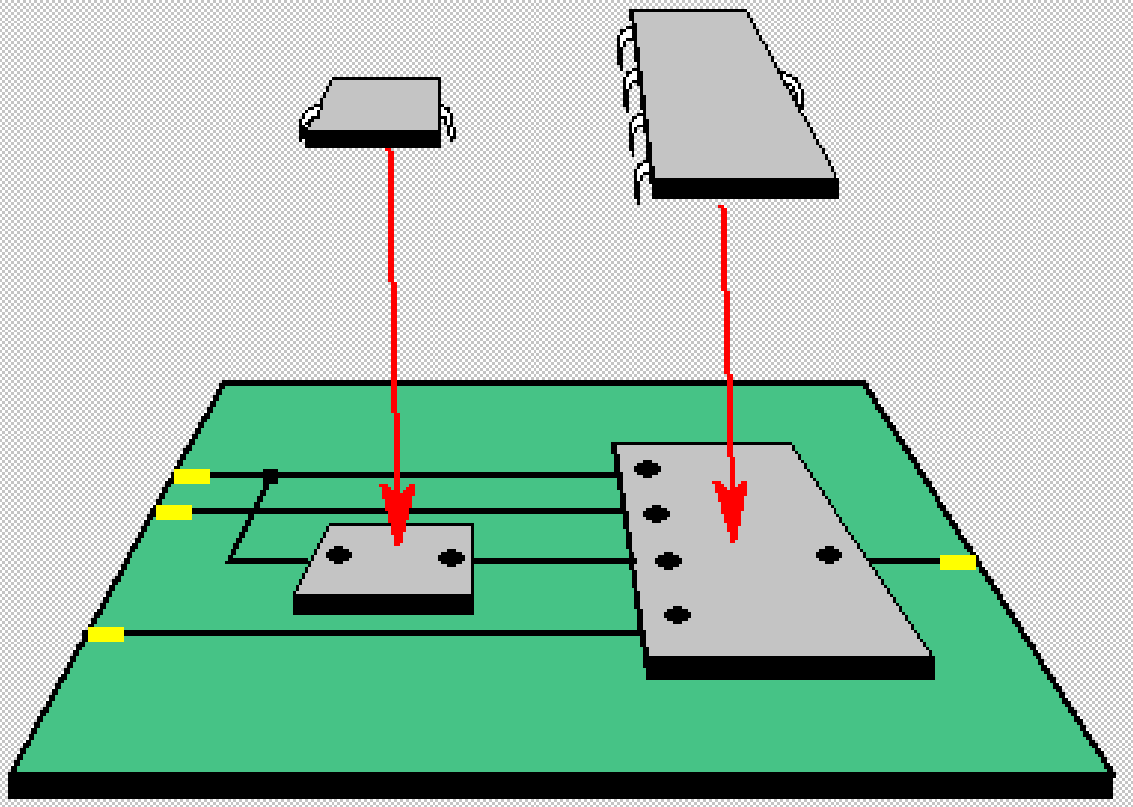
What is more, you can use it as many times as you wish by writing lots of instantiations, each with its own label. The same label can only be used once in a particular architecture.

```
architecture STRUCTURE of MUX2 is
  component INV
    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: INV port map(SEL, SELB);
  G2: AOI port map(SEL, A, SELB, B, F);
end STRUCTURE;
```


Components

A picture is worth a thousand words, so they say! Let's illustrate our MUX2 example with a diagram.

Click anywhere on the diagram with the mouse pointer and the appropriate VHDL term will appear in the Pop-up window! Note that component instances are equivalent to chips in sockets.

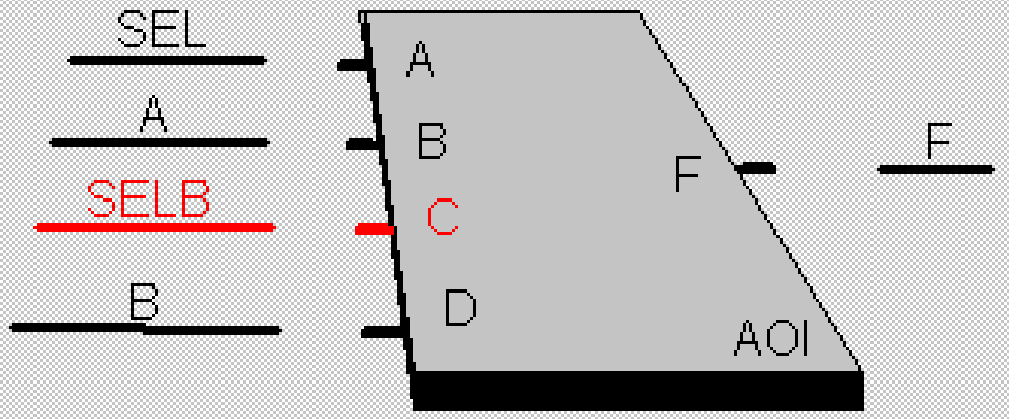


Components

Now for the connections.

The port map makes the connections to ports on a component instantiation according to the order of the ports in the component declaration. So SELB, the third name in the port map, is connected to port C, the third port on the component.

```
architecture STRUCTURE of MUX2 is
   ...
   component AOI
      port (A, B, C, D: in STD_LOGIC;
            F: out STD_LOGIC);
   end component;
   ...
begin
   ...
   G2: AOI port map(SEL, A, SELB, B, F);
end STRUCTURE;
```

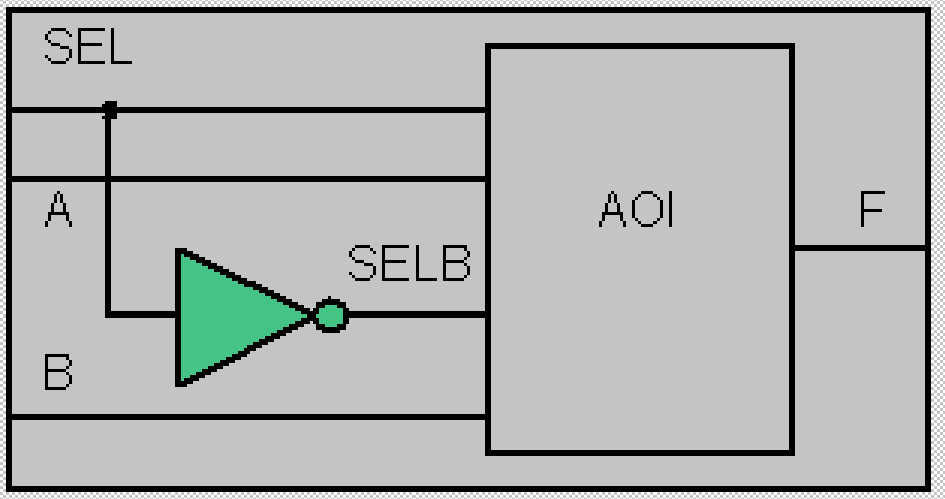


Components

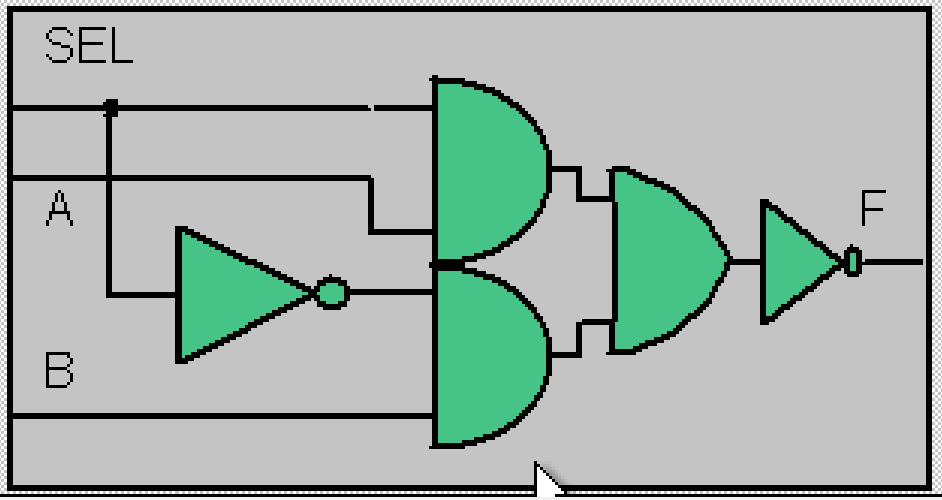
The use of components is very important for synthesis, since it allows the hierarchy of the synthesized design to be controlled.

You can choose to synthesize your design as several separate blocks, thus steering the architecture of the design, or to "flatten" the design to a single level and allow the optimizer to change the architecture.

Hierarchical synthesis



Flat synthesis



Components

In VHDL '93, there is an important change to the way hierarchy is described. You do not need components! It is possible to instantiate design entities directly, without using components as sockets! This makes the VHDL code much more straightforward and less verbose.

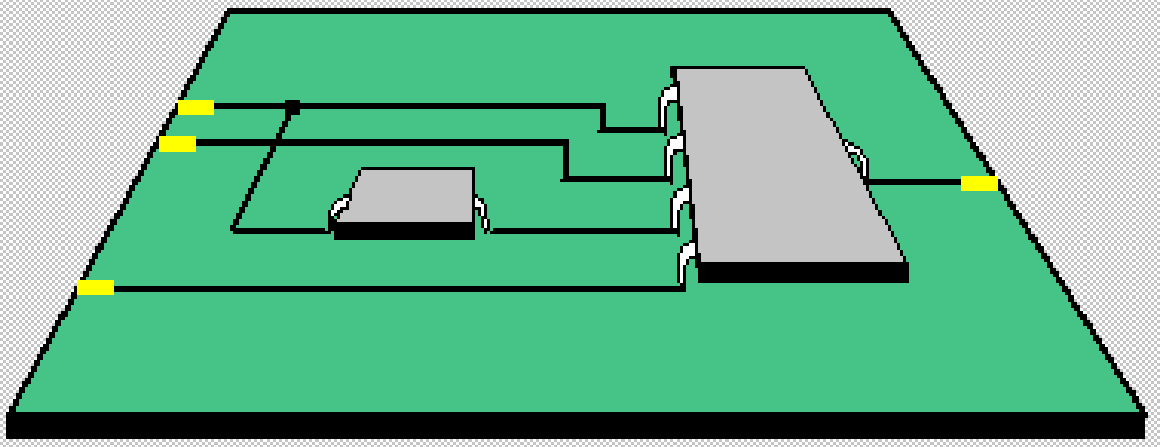
Once again, explore the diagram by pointing and clicking!

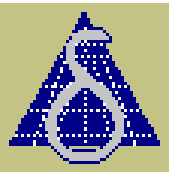
Entity

Architecture

```
G1: entity WORK.INV(ARCH)
   port map (SEL, SELB);

G2: entity WORK.AOI(V2)
   port map (SEL, A, SELB, B, F);
```

VHDL '93 only!





Exercise 4

Use the mouse to drag the lines up and down into the correct order. There is only one correct answer.

Check Answer

Correct! Well done.

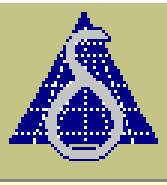
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity E is
    port (A: in STD_LOGIC;
          F: out STD_LOGIC);
end E;
architecture STRUCTURE of E is
  component C
    port (X: in  STD_LOGIC;
          Y: out STD_LOGIC);
  end component;
begin
  G1: C port map (A, F);
end STRUCTURE;
```

Exercise 5

Drag and drop each of the VHDL terms into the appropriate slot.

Correct!

signal	= a piece of wire
event	= a change in the value on a piece of wire
instantiation	= a specific chip socket
port map	= the electrical connections to a specific chip socket
assignment	= the thing that changes the value on a piece of wire
component	= the footprint of a chip socket



Exercise 6

Which port of the FullAdder is the signal Cout connected to in the instantiation labelled FA? (Type your answer in the box, then hit the Enter key.)

```
component FullAdder
   port (A, B, Cin: in Std_logic;
         Sum, Cout: out Std_logic);
end component;


FA: FullAdder
   port map (X, Y, Cout, A, B);
```

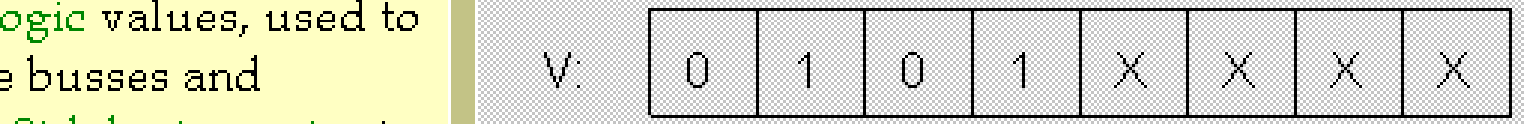
Std_logic_vector

Now let's explore the types Std_logic and Std_logic_vector. Std_logic represents a single digital logic value. A value of this type is written in single quotes, as shown. The value 'U' means uninitialized, and 'X' means unknown.

A Std_logic_vector is a row of Std_logic values, used to describe busses and vectors. Std_logic_vector is an array type.

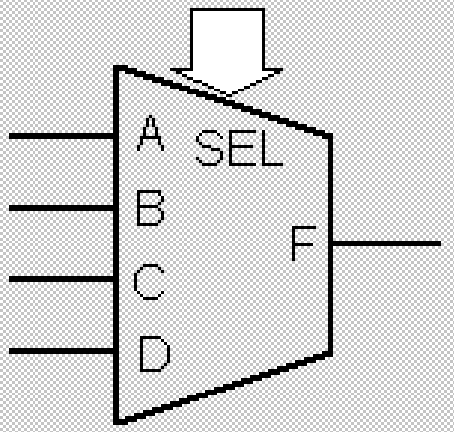
```
signal B: STD_LOGIC;
...
B <= '0';
...
B <= '1';
...
B <= 'X';
```

```
signal V: STD_LOGIC_VECTOR(7 downto 0);
```

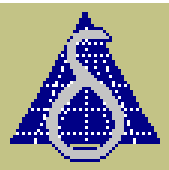


Std_logic_vector

The port SEL in this 4 to 1 multiplexer is a two bit array, described by using the type Std_logic_vector, and giving the width of the array.




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

entity MUX4 is
 port (
    SEL:in STD_LOGIC_VECTOR(1 downto 0);
    A, B, C, D: in STD_LOGIC;
    F: out STD_LOGIC);
end MUX4;
```



Std_logic_vector

The range of an array can be defined in either ascending or descending order, using the keywords **to** or **downto**. This implies the order in which the bits of the array are numbered, but actually says nothing about which bit is the most significant.

Descending

```
signal V: STD_LOGIC_VECTOR(7 downto 0);

signal W: STD_LOGIC_VECTOR(0 to 7);
```

Ascending

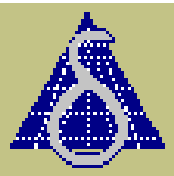
Std_logic_vector

A value of type Std_logic_vector is written as a text string enclosed in double quotes. The length of the string must match the number of elements defined in the type.

Array of 8 elements

```
signal V: STD_LOGIC_VECTOR(7 downto 0);

...

V <= "0101XXXX";
```

String of 8 characters



Std_logic_vector

Individual bits within an array can be accessed using an indexed name, which is a name followed by an index expression in parenthesis.

```
signal V: STD_LOGIC_VECTOR(7 downto 0);
...
V(7) <= V(6);
```

Copy bit 6 into bit 7

Std_logic_vector

Each element of a Std_logic_vector is actually a value of type Std_logic, so its value has to be written in single quotes.

```
signal V: STD_LOGIC_VECTOR(7 downto 0);

...

V(7) <= '1';
```

Std_logic_vector

It is also possible to access part of an array, rather than the entire array or a single element of the array. Accessing part of an array is called a slice name in VHDL jargon.

Array elements are copied from left to right, such that the leftmost bit on the right of the assignment is copied into the leftmost bit on the left of the assignment.

```
signal V: STD_LOGIC_VECTOR(7 downto 0);
signal W: STD_LOGIC_VECTOR(0 to 3);

...

W <= V(7 downto 4);
```

V(7) copied into W(0), V(6) into W(1) etc

```
V(3 downto 0) <= "01XZ";
```

leaving V(3) = '0', V(2) = '1' etc

Exercise 7

Drag and drop each of the VHDL expressions into the appropriate slot. There is only one overall solution in which every item is correct.

'0' "11" "0000"

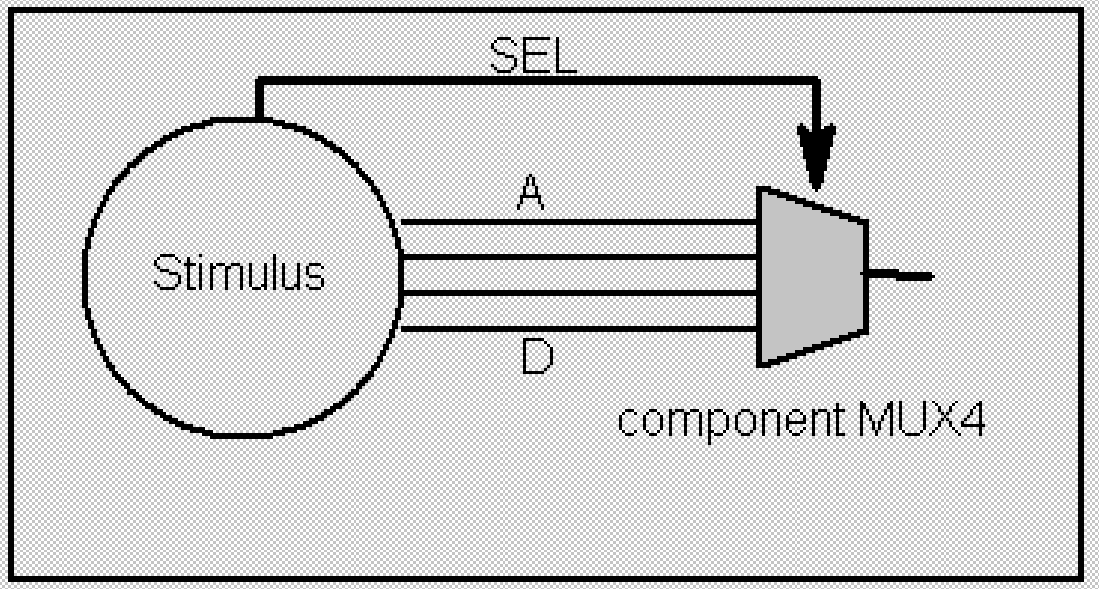
0 2 3

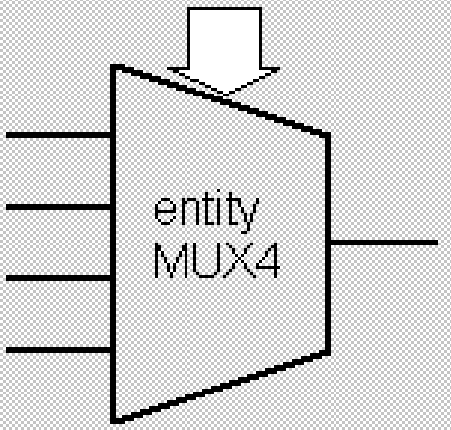
```
signal P: Std_logic_vector(3 downto 0);

P <=      ;

P(      downto      ) <=      ;

P(      ) <=      ;
```

Test Benches

A VHDL simulation requires not only the hardware design described in VHDL, but also test vectors described in VHDL.

A design entity which provides a simulation environment for the hardware design is commonly known as a test bench in the VHDL world.

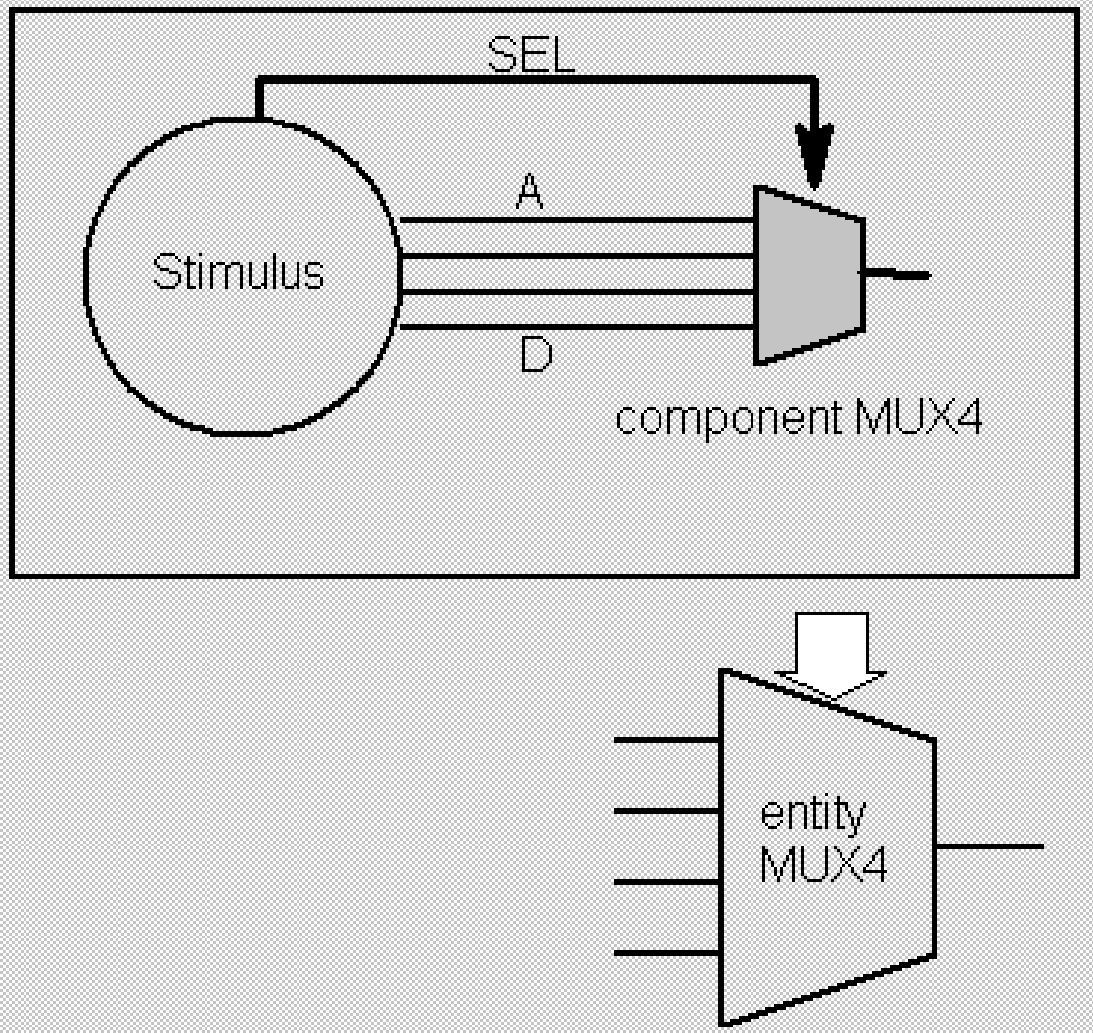




Test Benches

A VHDL test bench makes an instantiation of the design entity at the top of the hardware design hierarchy, and contains VHDL code to apply test vectors to the inputs of the design.

The test bench architecture may also include VHDL code to analyze the outputs of the design. Test benches are never synthesized, of course!



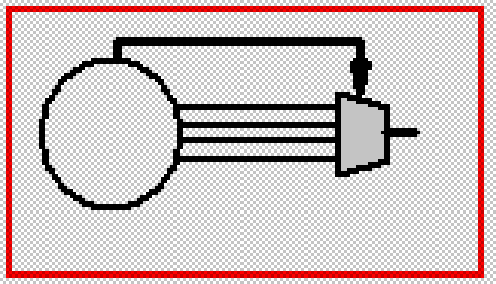
Test Benches

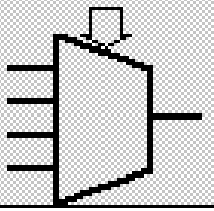
Let's look at a test bench for the design entity MUX4.

The entity for a test bench is usually empty. This is important, as many VHDL simulators do not permit ports on top level entities.

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

entity TEST_MUX4 is
end;
```

Empty!



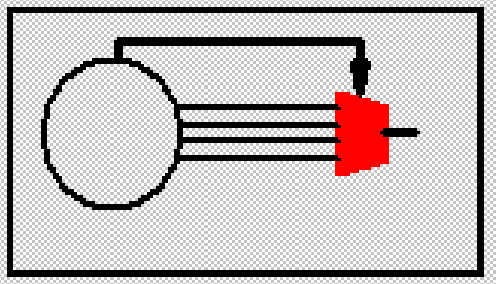


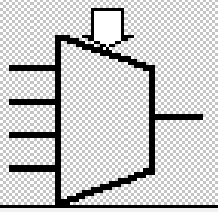
Test Benches

The test bench architecture includes a component declaration corresponding to the MUX4 design entity, because we want the MUX4 as a lower level hierarchical block in the test bench.

The component declaration repeats the ports from the entity.

```
architecture BENCH of TEST_MUX4 is

   component MUX4
      port (SEL: in
            STD_LOGIC_VECTOR(1 downto 0);
            A, B, C, D: in STD_LOGIC;
            F: out STD_LOGIC);
   end component;
   ...
begin
   ...
end BENCH;
```



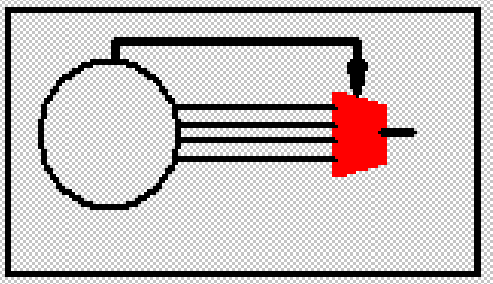


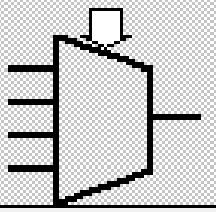
Test Benches

We also need an instantiation of the component. This is critical, because it is the instantiation that actually creates a copy of the MUX4, not the component declaration.

The instantiation includes a port map to connect up the ports of the component.

```
architecture BENCH of TEST_MUX4 is

  component MUX4
    port (SEL: in STD_LOGIC_VECTOR
                          (1 downto 0);
          A, B, C, D: in STD_LOGIC;
          F         : out STD_LOGIC);
  end component;
  ...
begin
  ...
  M: MUX4 port map(SEL, A, B, C, D, F);

end BENCH;
```

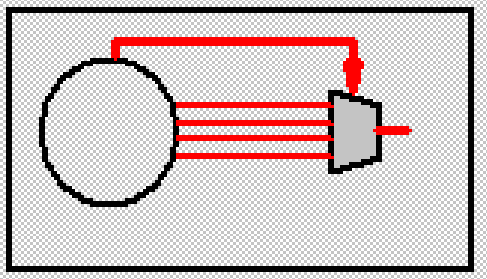


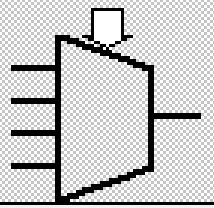


Test Benches

The signals that are connected to the ports of the MUX4 must also be declared in the architecture. These signals represent internal connections within the test bench.

```
architecture BENCH of TEST_MUX4 is
   ...
   signal SEL:
          STD_LOGIC_VECTOR(1 downto 0);
   signal A, B, C, D, F: STD_LOGIC;

begin
   ...
   M: MUX4 port map(SEL, A, B, C, D, F);

end BENCH;
```



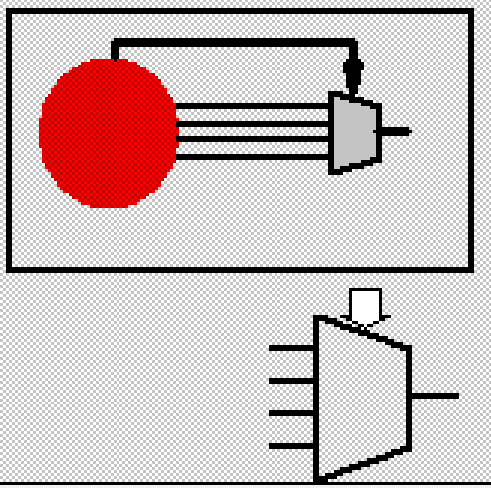


Test Benches

Finally, the test bench must generate the test vectors to be applied to the input ports of the MUX4. This is done with concurrent signal assignments to the signals declared in the architecture.

Each signal assignment creates several events on a signal at the times given in the assignment.

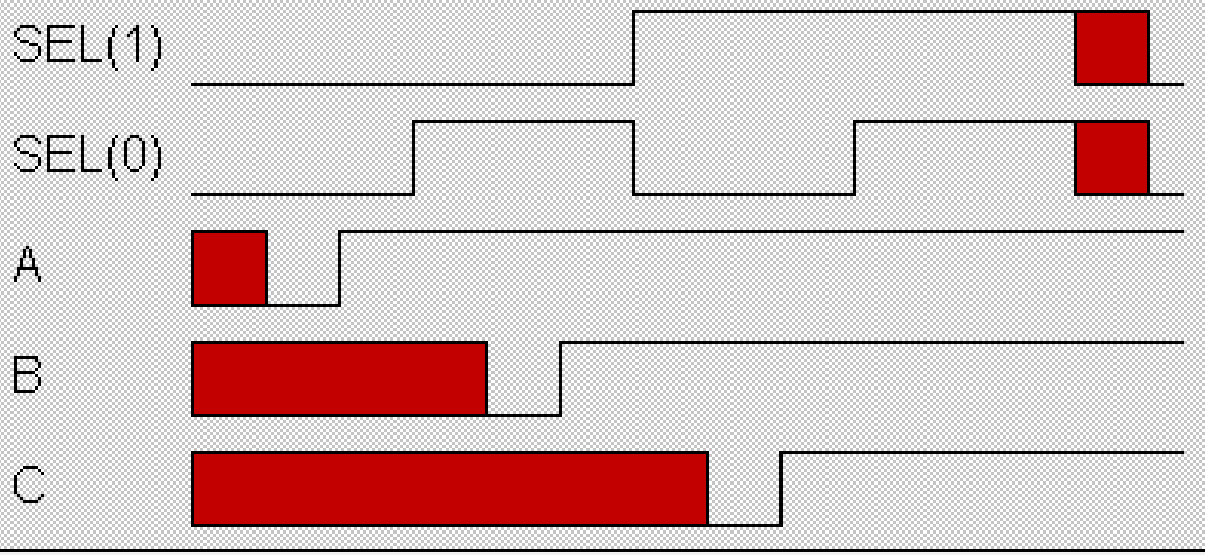
```
architecture BENCH of TEST_MUX4 is
  ...
begin
  SEL <= "00", "01" after 30 NS,
  "10" after 60 NS, "11" after 90 NS,
  "XX" after 120 NS, "00" after 130 NS;

  A <= 'X',
       '0' after 10 NS,
       '1' after 20 NS;
  B <= 'X',
       '0' after 40 NS,
       '1' after 50 NS;
  C <= 'X',
       '0' after 70 NS,
       '1' after 80 NS;
  D <= 'X',
      '0' after 100 NS,
       '1' after 110 NS;

  ...
end BENCH;
```



Test Benches

In order to clarify the meaning of these signal assignments, let's try to visualize what these test vectors look like graphically. This is shown in the diagram opposite.

```
SEL <= "00", "01" after 30 NS,
"10" after 60 NS, "11" after 90 NS,
"XX" after 120 NS, "00" after 130 NS;

A <= 'X', '0' after 10 NS,
          '1' after 20 NS;
B <= 'X', '0' after 40 NS,
          '1' after 50 NS;
C <= 'X', '0' after 70 NS,
          '1' after 80 NS;
```



Exercise 8

Drag and drop the fragments of VHDL code into the appropriate slots in the architecture to make a complete test bench.

```
library IEEE; use IEEE.STD_LOGIC_1164.all;
entity TestBench is
[                                ]
end;
architecture  V1 of TestBench is
[                                ]
[                                ]
begin
[                                ]
[                                ]
end;
```

A <= '0',
'1' after 10 NS;

component C
port(P:in Std_logic;Q:out Std_Logic);
end component;

L: C port map (A, F);

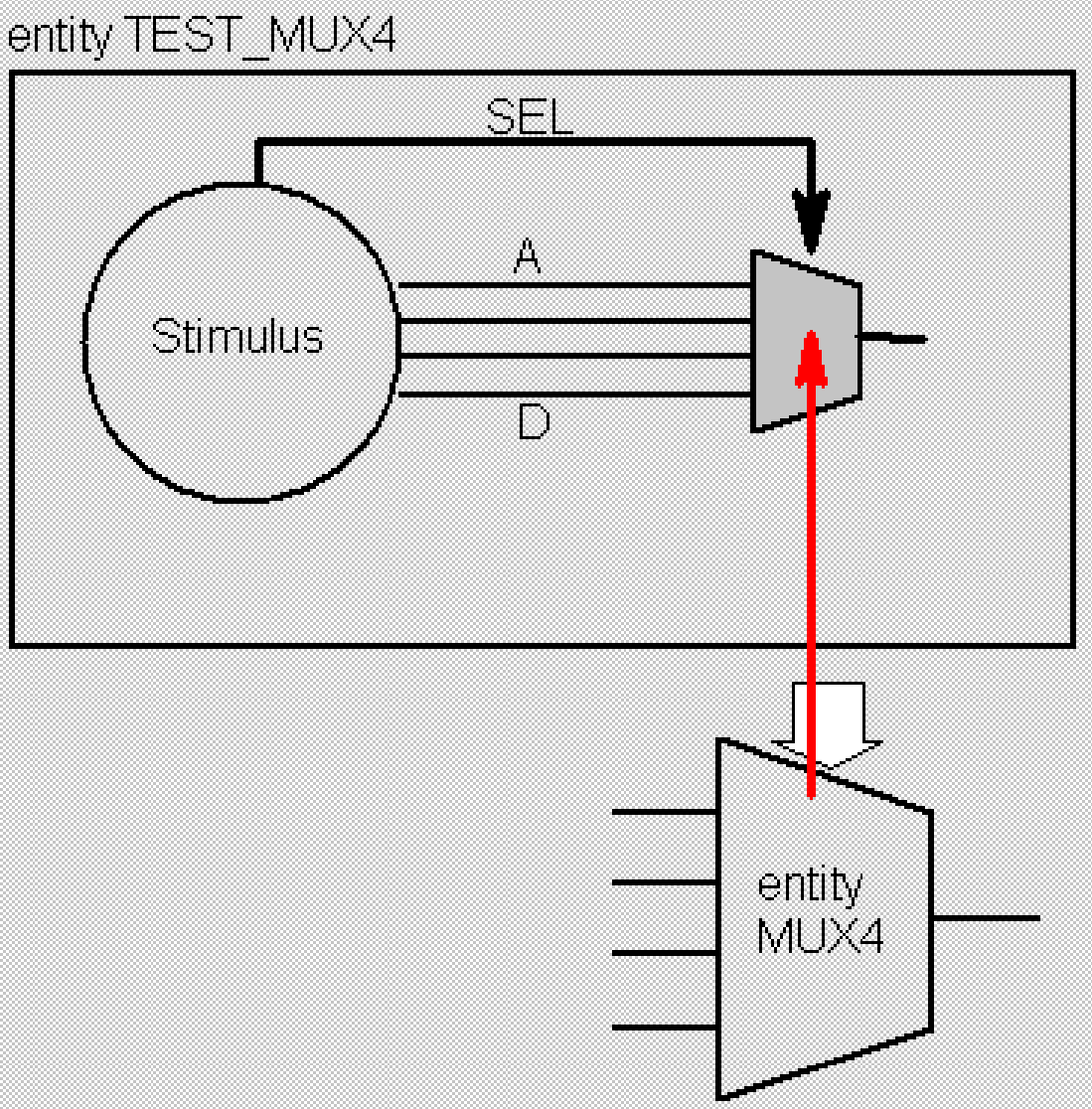
signal A: Std_logic;
signal F: Std_logic;

port (A: in Std_logic;
F: out Std_logic);

Configurations

So, a design written in VHDL consists of many design entities (representing hierarchical blocks), and also a few packages (containing common definitions).

In order to simulate or synthesize the hierarchy, we must select the entity and architecture to be used for each component instantiation throughout the hierarchy.

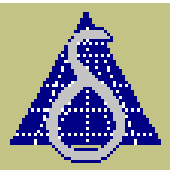


Configurations

VHDL contains a construct for selecting which entities and architectures are to be used. This is called the configuration.

A configuration is a separate piece of VHDL text, and must be given a name which optionally may be repeated at the end of the configuration.

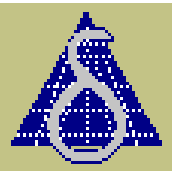
```
configuration CFG_MUX4 of TEST_MUX4 is
  for BENCH
  end for;
end CFG_MUX4;
```



Configurations

A configuration always belongs to a specific entity, in this case the top level test bench TEST_MUX4.

```
entity TEST_MUX4 is
end;

...

configuration CFG_MUX4 of TEST_MUX4 is
   for BENCH
   end for;
end CFG_MUX4;
```

Configurations

A configuration always configures a specific architecture, which is named immediately inside the configuration. BENCH is the architecture of entity TEST_MUX4.

Note that it is the test bench being configured, not the design under test. BENCH is the architecture of the test bench.

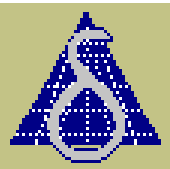
```
architecture BENCH of TEST_MUX4 is
  ...
end;


configuration CFG_MUX4 of TEST_MUX4 is
  for BENCH
  end for;
end CFG_MUX4;
```


Configurations

The VHDL standard defines some default rules for how component instantiations are to be configured in the absence of explicit instructions.

By default, an entity with the same name and ports as the component is used, together with the most recently compiled architecture.

```
component MUX4
   port (SEL: in
         STD_LOGIC_VECTOR(1 downto 0);
         A, B, C, D: in STD_LOGIC;
         F: out STD_LOGIC);
end component;


entity MUX4 is
   port (SEL: in
         STD_LOGIC_VECTOR(1 downto 0);
         A, B, C, D: in STD_LOGIC;
         F: out STD_LOGIC);
end MUX4;
```



Configurations

Despite these rules, many VHDL tools insist that the minimal configuration declaration shown here must exist for the top level design entity in the hierarchy!

```
configuration CFG_MUX4 of TEST_MUX4 is
   for BENCH
   end for;
end CFG_MUX4;
```

Configurations

Some VHDL tools go further, and insist that any entities to be used are made directly visible to the compiler in the configuration declaration.

The use clause shown makes the entity MUX4 visible, so it can be used for the MUX4 component in architecture BENCH.

This completes our tour of design entities.

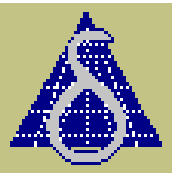
```
use WORK.all;
configuration CFG_MUX4 of TEST_MUX4 is
  for BENCH
  end for;
end CFG_MUX4;
```

Exercise 9

Choose the definition which most accurately describes the purpose of a configuration.

- To define the functionality of the design hierarchy
- To define the connections between components
- To tell the compiler where to find the architectures available for an entity
- To define which entities and architectures are used in the design hierarchy

Correct! The "minimal" configurations we've seen use the default rules to select entities and architectures.



Exercise 10

Assemble a minimal configuration for the given architecture out of the parts provided! Note that some of the parts are not used!

Incorrect! Try again.

```
entity Top is
end Top;

architecture Bench of Top is
   component Main
      ...
   end component;
   ...
begin
   ...
end Bench;
```

Bench

for

is

Main

Main

configuration Cfg of Top is

for Bench

of

end Top ;

end Cfg ;