Entity, Architecture, Ports

A VHDL models consist of an *Entity Declaration* and a *Architecture Body.*

The entity defines the <u>interface</u>, the architecture defines the <u>function</u>.

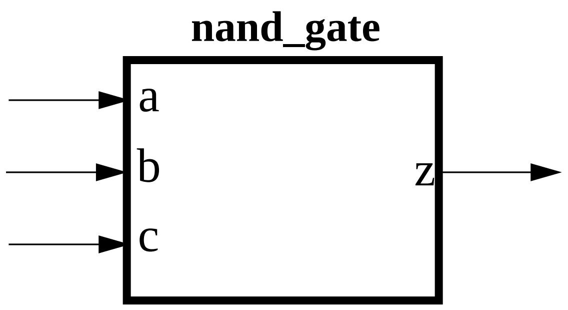
The entity declaration names the entity and defines the interface to its environment.

Entity Declaration Format:

```
ENTITY entity_name IS
   [GENERIC (generic_list);]
   [PORT (port_list);]
END ENTITY [entity_name];
```

There is a direct correspondence between a ENTITY and a block diagram symbol. For example:

```
ENTITY nand_gate IS
PORT(
       a : in   std_logic;
       b : in   std_logic;
       c : in   std_logic;
       z : out  std_logic);
END ENTITY nand_gate;
```



Port Statement

The entities *port* statement identifies the ports used by the entity to communicate with its environment

Port Statement Format:

```
PORT(
    name_list  : mode type;
    name_list  : mode type;
    name_list  : mode type;
    name_list  : mode type);
```

This is legal but poor form:

```
ENTITY nand_gate IS
  PORT(a,d,e,f : in  std_logic;
    b,j,q,l,y,v : in  std_logic;
    w,k  : in  std_logic;
    z : out: std_logic);
END nand_gate;
```

This is much less error prone:

Use one line per signal. This allows adequate comments.
Capitalize reserved names.

```
ENTITY nand_gate IS
  PORT(
    a : IN  STD_LOGIC;  --a input
    b : IN  STD_LOGIC;  --b input
    c : IN  STD_LOGIC;  --c input
    z : OUT STD_LOGIC); --nand output
END ENTITY nand_gate;
```

Port Mode:

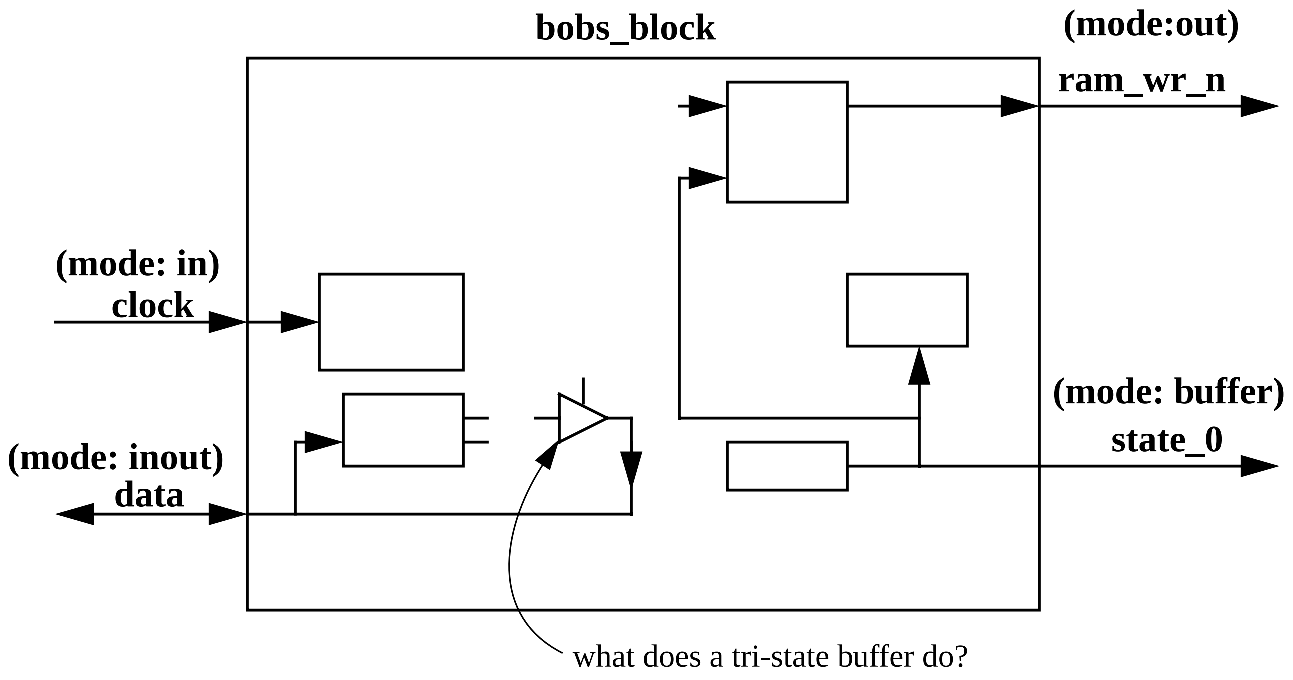
Identifies the direction of data flow through the port.

The PORT statement is optional. At the top (testbench) level, none is needed. (why?)

All ports must have an identified mode.

Allowable Modes:

• **IN**	**Flow is into the entity (input only)**
• **OUT**	**Flow is out of the entity (output only)**
• **INOUT**	**Flow may be either in or out (either in or out)**
• **BUFFER**	**An OUTPUT that can be read from**



Architecture Body

The architecture body describes the operation of the component.

Format:

```
ARCHITECTURE body_name OF entity_name IS
   --this is the ->declarative area<-
   --declare signals, variables, components,
   --subprograms
BEGIN
   --this is the ->statement area<-
   --in here go statements that describe
   --organization or functional operation of
   --the component
   --this is the “execution part” of the model
END [body_name]
```

The entity_name in the architecture statement must be the same as the entity declaration that describes the interface to the outside world.

```
ENTITY entity_name IS
```

```
ARCHITECTURE body_name OF entity_name IS
```

The “body_name” is a user-defined name that should uniquely describe the particular architecture model.

```
ARCHITECTURE beh OF nand_gate IS
```

```
ARCHITECTURE struct OF nand_gate IS
```

Note: multiple architectures are allowed.

Commenting Code

A double hyphen (--) indicates everything from that point on in that line is to be treated as a comment.

```
ARCHITECTURE example OF xor_gate IS
   --The following is a silly example of how
   --to write comments in VHDL.
BEGIN
   --comment from the beginning of a line
   a <= b XOR c; --or...comment from here on
   --
   --each line must have its own
   --comment marker unlike “C”
   --
END [body_name]
   --
   --
   --this is the end and there ain’t no more!
```

Comments can be put anywhere except in the middle of a line of code.

Entity and Architecture for a NAND gate Model

```
--
--the following is a behavioral description of
--a three input NAND gate.
--
ENTITY nand3 IS
PORT(
      a  : IN   std_logic;
      b  : IN   std_logic;
      c  : IN   std_logic;
      z  : OUT  std_logic);
END ENTITY nand3;

ARCHITECTURE beh OF nand3 IS
 BEGIN
    z <= ‘1’  WHEN a=’0’ AND b=‘0’ ELSE
         ‘1’  WHEN a=’0’ AND b=‘1’ ELSE
         ‘1’  WHEN a=’1’ AND b=‘0’ ELSE
         ‘0’  WHEN a=’1’ AND b=‘1’ ELSE
         ‘X’;
END ARCHITECTURE beh;
```

You can create VHDL source code in any directory.

VHDL source code file may be anything......but,
Use the name of the design entity with the extension “*.vhd*”

The above example would be in the file: nand3.vhd

Question: Why the ‘X’ in the above code?

Signal Assignment

The assignment operator (<=) is used to assign a waveform value to a *signal*.

Format:

```
target_object <= waveform;
```

Examples:

```
my_signal  <= ‘0’; --ties my_signal to “ground”
his_signal <= my_signal; --connects two wires
```

```
--vector signal assignment
```

```
data_bus <= “0010”; -- note double quote
bigger_bus  <= X”a5”;  -- hexadecimal numbers
```

Note: I am using framemaker to generate these slides. Frame causes the first right slanting “tic” to slant to the left. This is correct English language but incorrect VHDL.

Declaring Objects

Declaration Format:

```
OBJECT_CLASS  identifier:  TYPE [:= init_val];
```

Examples:

```
CONSTANT  delay   :  TIME:= 10ns;
CONSTANT  size    :  REAL:=5.25;
VARIABLE  sum     :  REAL;
VARIABLE  voltage :  INTEGER:=0;
SIGNAL    clock   :  BIT;
SIGNAL    spam    :  std_logic:=’X’;
```

Objects in the port statement are classified as signals by default.

Objects may be initialized at declaration time.
(Danger, danger, Will Robinson!)

If an object is not initialized, it assumes the left-most or minimum value for the type.

Why not initalize a flip flop as shown below?

```
PORT (d   : IN  STD_LOGIC;
      q   : OUT STD_LOGIC := ‘0’;
      clk : IN  STD_LOGIC)
```

Naming Objects

Valid characters:

- **alpha characters (a-z)**
- **numeric characters (0-9)**
- **underscore (_)**

Names must consist of any number of alpha, numeric, or underline characters.

Underscore must be proceeded and followed by alpha or numeric characters.

The underscore can be used to separate adjacent digits in bit strings:

```
CONSTANT big_0 : STD_LOGIC_VECTOR(15 DOWNTO 0) :=
B"0000_0000_0000_0000";
```

Names are not case sensitive. (be consistent!, use lowercase!)

Coding hints:

Use good names that are meaningful to others. If your code is good, somebody else will want to read it.

Name signals by their function. For example, if you have a multiplexor select line that selects addresses, give it a name like “`address_select`” instead of “`sel_32a`”.

Name blocks by their function. If a block generates control signals for a DRAM controller, call the block “`dram_ctl`” not something obscure like “`block_d`”.

A Simple Example to Recap

```
------------------------------------------
--and-or-invert gate model
--Jane Engineer
--3/13/04
--version 0.5
------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY aoi4 IS
PORT(
     a  : IN    std_logic;
     b  : IN    std_logic;
     c  : IN    std_logic;
     d  : IN    std_logic;
     z  : OUT   std_logic);
END ENTITY aoi4;

ARCHITECTURE data_flow OF aoi4 IS
  SIGNAL  temp1, temp2  :  std_logic;
  BEGIN
    temp1 <= a AND b;
    temp2 <= c AND d;
    z     <= temp1 NOR temp2;
END ARCHITECTURE data_flow;
```