

EC262: Intro to VHDL

We have looked primarily at many different ways of representing digital logic circuits:

Long Boolean expressions can be very cumbersome to read. Looking at a long Boolean expression, the intent of the designer can be difficult to discern. Similarly, a long truth table often cannot casually convey what a circuit is designed to do, absent detailed analysis.

Similarly, circuit diagrams (for large circuits) can be difficult to draw, and, as with Boolean expressions, sometimes just looking at a circuit diagram provides few clues about the intended operation of the circuit.

Another means of representing a digital logic circuit is through the use of a *hardware description language*. A hardware description language allows you to describe the desired behavior of your circuit using text, and then simulate the operation of your circuit.

Although lines in a hardware description language look similar to lines of code written in a programming language (such as C++ or Java), there is a critical distinction: lines of code in a programming language are executed line by line sequentially without any reference to a time axis. Hardware description languages, on the other hand, attempt to model actual circuit operation, which may be very much time dependent. Specifically, lines of code in VHDL can execute concurrently or sequentially, depending on the scenario (as we will see). For this reason, we will normally say “Look at this VHDL *code*” instead of “Look at this VHDL *program*”.

By looking at simulation results based on our hardware description, we can spot design flaws before implementing hardware (or even drawing a schematic).

It is important to emphasize again that for all but the simplest designs, you always want to

The two major hardware description languages in use today are both IEEE standards:

- Verilog (developed by Gateway, now owned by Cadence Design)
- VHDL (developed by the Department of Defense)

Verilog has the reputation for being somewhat easier to learn. Thus, guess which one we are going to use?

VHDL stands for **V**ery high-speed integrated circuit **H**ardware **D**escription **L**anguage. The language has a formal syntax that must be followed.

VHDL code has three main sections:

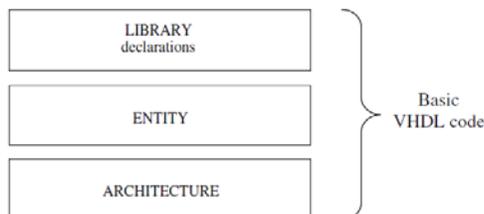


Figure 2.1
Fundamental sections of a basic VHDL code.

From Pedroni, *Circuit Design with VHDL*, MIT Press, 2004

A template for a vhdl file is shown below (not a working file).

```
1  -- A library clause declares a name as a library.  It
2  -- does not create the library; it simply forward declares it.
3  library <library_name>;
4
5  -- We will use IEEE library
6  library IEEE;
7
8  -- Import all the declarations in a package
9  use <library_name>.<package_name>.all;
10
11 -- Commonly imported packages:
12
13 -- BIT, STD_LOGIC and STD_LOGIC_VECTOR types, and relevant functions
14 use ieee.std_logic_1164.all;
15
16 -- SIGNED and UNSIGNED types, and relevant functions
17 use ieee.numeric_std.all;
18
19 ENTITY <entity_name> IS
20     PORT
21     (
22         -- Input ports
23         <name>      : IN  <type>;
24         <name>      : IN  <type> := <default_value>;
25
26         -- Inout ports
27         <name>      : INOUT <type>;
28
29         -- Output ports
30         <name>      : OUT  <type>;
31         <name>      : OUT  <type> := <default_value>
32     );
33 END <entity_name>;
34
35 ARCHITECTURE <arch_name> OF <entity_name> IS
36
37     -- Declarations (optional)
38
39 BEGIN
40     -- This section includes statements (code) that define the behavior of this entity.
41
42     -- Some commonly used statements:
43     -- Concurrent Signal Assignment (optional)
44     -- Process Statement (optional)
45     -- Conditional Signal Assignment (optional)
46     -- Selected Signal Assignment (optional)
47     -- Component Instantiation Statement (optional)
48     -- Generate Statement (optional)
49
50 END <arch_name>;
```

The library declarations: This section lists the libraries of code that others have already written that you would like to use in your own code. We will have this section read:

```
library IEEE;
use ieee.std_logic_1164.all;
```

This looks cryptic, but basically it makes available to you a set of code that you can freely use in your own design. For now, just consider this an opening stamp that you place on your code.

Note the semicolons in the two lines above. **They are not optional.**

As we write VHDL, we may want to include comments on our code—basically notes to ourselves that we do not want to be viewed as part of the intrinsic design. We can add comment to our code by using a **double-dash**. For example, the following library declaration will be treated exactly the same as the one above:

```
library IEEE;
use ieee.std_logic_1164.all;
```

Comments can be placed in the same line as code, such as:

```
library IEEE;                -- EC262 = Fun Times
use ieee.std_logic_1164.all;
```

The **ENTITY** section: This section specifies the name of our design, and all of its inputs and outputs. What the design actually does to the inputs to arrive at the output is NOT of interest in this section. Put another way, if you imagine your design as a black box that has inputs and outputs, the entity section is where you will describe the inputs and outputs.

The basic format for the entity section using just input bits and output bits is:

```
ENTITY entity_name IS PORT (
    input_1, input_2, ... , input_n      :      IN BIT;
    output_1, output_2, ... , output_m  :      OUT BIT
);
END entity_name ;
```

After the word “ENTITY” is the entity name that you pick. Can you pick any name?

Entity names (in fact, all names that you choose to represent items in your code) must begin with a letter, and then can consist of **letters, numbers, or the underscore character, with no spaces allowed.**

Example. Which of the following are valid names that I can choose in VHDL?

1. EC262
2. Go_Navy
3. Project_25
4. Practice_Parades_are_Fun
5. 2cofer_surprise_quiz
6. _cofer_surprise_quiz
7. cofer_surprise_quiz_
8. cofer__surprise_quiz
9. disconnect
10. ec262 (if I am already using number 1 above)

Here are the VHDL reserved words:

abs	entity	next	select
access	exit	nor	severity
after	file	not	signal
alias	for	null	shared
all	function	of	sla
and	generate	on	sll
architecture	generic	open	sra
array	group	or	srl
assert	guarded	others	subtype
attribute	if	out	then
begin	impure	package	to
block	in	port	transport
body	inertial	postponed	type
buffer	inout	procedure	unaffected
bus	is	process	units
case	label	pure	until
component	library	range	use
configuration	linkage	record	variable
constant	literal	register	wait
disconnect	loop	reject	when
downto	map	rem	while
else	mod	report	with
elsif	nand	return	xnor
end	new	rol	xor

Between the open and close parenthesis we list all of our inputs and outputs.

Look at the line of code:

```
input_1, input_2, ... , input_n    :           IN BIT;
```

Consider this line to be two sections, divided at the colon. Everything to the left of the colon is a list of names, and everything to the right of the colon describes what those names are to function as. So, if we were to have the line:

```
a, b : IN BIT ;
```

that would mean that my entity is to have |

Similarly, in the line of code:

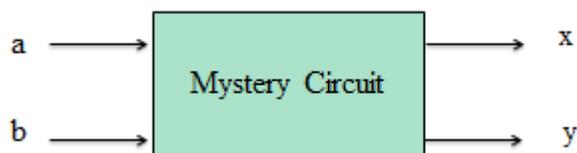
```
out_1, out_2    :           OUT BIT
```

we are saying that our circuit will have two outputs, and both will be bits.

Example. What is accomplished by this entity:

```
ENTITY fun_times IS PORT (  
    a , b      :           IN BIT;  
    x , y      :           OUT BIT  
);  
END fun_times ;
```

We have told VHDL that we are planning on designing a circuit that will have two input bits and two output bits. At this point, our circuit is a black box:



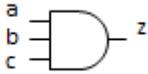
You might be thinking that VHDL is wonderful for designing mystery circuits, but how do I design real circuits. Ah, that is where the architecture section comes in!

The architecture section: This section tells how the entity that you named above actually carries out what you intended it to do. How the outputs are derived from the inputs is detailed in this section.

The basic format for the architecture section is:

```
ARCHITECTURE arch_name OF entity_name IS  
BEGIN  
  
    the statements that define the behavior ;  
  
END arch_name ;
```

Example. Write VHDL code to implement the circuit below:



```
library IEEE;  
use ieee.std_logic_1164.all;
```

Notice that strange symbol: a less than sign, followed by an equal sign, with no space in between: `<=`. That symbol is the assignment operator. So the statement

means “Compute the value of *a* and *b* and *c*, and put the answer in *z*.”

Questions:

- Since I am designing an *and gate*, why did I not label my entity as “*and*”.
- Suppose I wanted to design a three-input or gate. What changes would I make to the above program?
- How would you write VHDL code to implement this truth table:

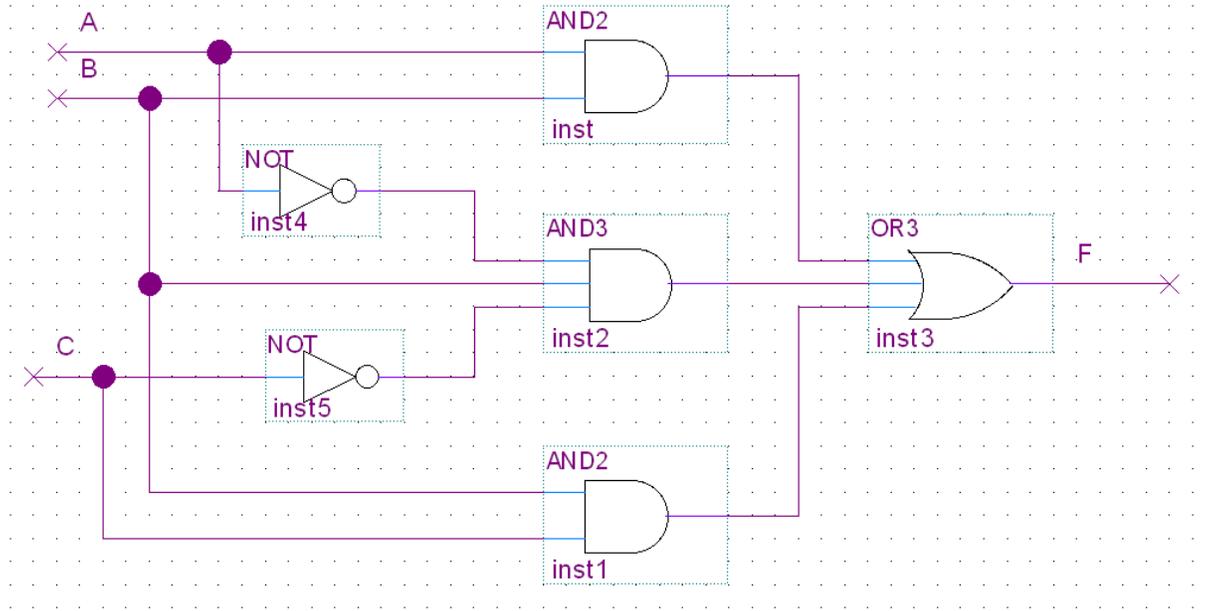
<i>a</i>	<i>b</i>	<i>out</i>
0	0	1
0	1	1
1	0	1
1	1	0

Example

To ground this in a more practical actual example, let's write VHDL code to implement the following logic function,

$$F(A,B,C) = AB + A'BC' + BC$$

The circuit diagram for this function is:



Library declaration:

```
1  --Library declaration
2
3  library IEEE;           -- Declare which VHDL library
4  use IEEE.std_logic_1164.all; -- and packages to use
5
```

Entity section:

```
7  -- Entity declaration
8
9  ENTITY lab1_vhdl IS
10 PORT(
11     -- Input ports
12     A, B, C : IN BIT ;    -- can use std_logic also
13
14     -- Output port
15     F:      OUT BIT      -- can use std_logic also
16 );
17 END lab1_vhdl ;
18
```

Note: syntax for ;

Note again that the entity section is just a description of the inputs and outputs—the pins of the circuit.

Architecture section:

```
20 --Architecture body
21
22 ARCHITECTURE dataflow OF lab1_vhdl IS
23 BEGIN
24
25     -- Implement function  $F = AB + A'B C' + BC$ 
26     F <= (A AND B) OR ( NOT A AND B AND NOT C) OR (B AND C);
27
28 END dataflow ;
29
```

Notice the use of the assignment operator. The statement

$$F <= (A \text{ AND } B) \text{ OR } (\text{ NOT } A \text{ AND } B \text{ AND NOT } C) \text{ OR } (B \text{ AND } C) ;$$

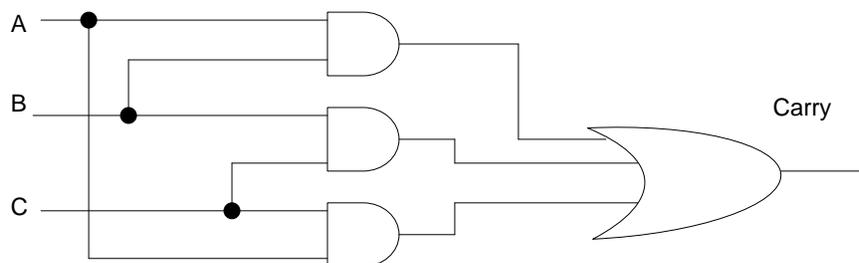
should be read:

“Compute the value of $(AB + A'BC' + BC)$ and then assign that result to F .”

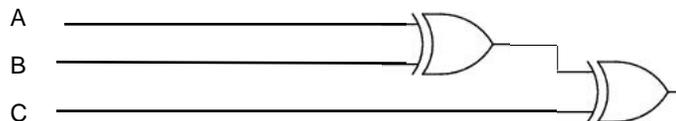
Example

To ground this in another example, suppose we wanted to implement a full adder using VHDL. Recall the full adder, which you designed about a couple of years or so ago, had two input bits and a carry-in bit, and produced a sum bit and a carry-out bit, using the logic below:

The Boolean expression for the carry-out bit is $AC + AB + BC$ and the logic circuit for the carry-out bit is:



The sum bit can be implemented by two XORs.



So, let's complete the three sections of our VHDL code: The library declarations, the entity section and the architecture section.

```
1  --Library declaration
2
3  library _____      -- Declare which VHDL library
4  use _____ .all;    -- and packages to use
5
6
7  -- Entity declaration
8
9  _____ full_adder IS
10 PORT(
11     -- Input ports
12     a, b, cin : _____ ;
13
14     -- Output ports
15     s, cout: _____
16 );
17 END full_adder ;
18
19
20 --Architecture body
21
22 ARCHITECTURE dataflow OF _____ IS
23 BEGIN
24
25     -- Implement the Boolean expression for sum bit
26     s <= _____
27
28
29     -- Implement the Boolean expression for carry out bit
30     cout <= _____
31
32
33 END _____ ;
34
```

Note again that the architecture section describes how the circuit is supposed to function.

Save your VHDL code in a file named *full_adder.vhd*.