

Introduction to IP Cores

Part 1: Digital Design -- Using IP Cores to Simplify Design

In the world of digital design, one uses Hardware Description Languages to describe complex logic functions. These are encapsulated into design suites such as Xilinx's ISE and similar tools. However, if a digital engineer were to code an adder, or create a cosine lookup table each time he or she worked on a project, the engineer would be wasting his or her time by reinventing the wheel. Alternatively, if the design engineer had to continually re-code commonly used complex digital circuits in large projects; they would also end up wasting money.

Because of this, a digital design engineer may just use an IP (Intellectual Property) core. An IP core is a block of HDL code that other engineers have already written to perform a specific function. It is a specific piece of code designed to do a specific job. One can use these cores in a complex design where an engineer wants to save the time he or she might have otherwise used to design the given function from scratch.

As with any engineering tool, IP cores have their advantages and disadvantages. Though an IP core may simplify a given design, the engineer has to design the whole project around the requirements of the IP core. And while an IP core may reduce design time, the engineer frequently has to pay for the right to use the core, since it is someone else's intellectual property.

Part 2: VHDL – Structural description

Structural description is the lowest level of abstraction possible to describe a circuit or system using VHDL. The description will basically consist on the specification of the connection between the different components of the subsystem. Consider the circuit shown in figure 1.

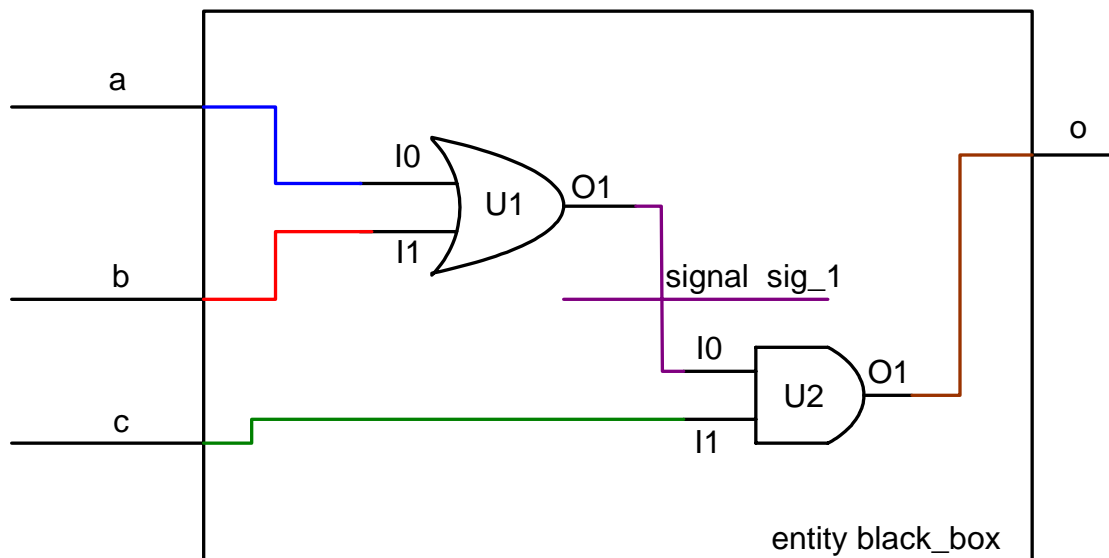


Figure 1. Circuit to be described using structural level of abstraction

The system **black_box** is made out of two components: **U1** and **U2**. The first step is to declare these components as part of the system. This is done as shown in the piece of code below.

```
architecture STRUCTURAL of black_box is           (line 1)

component U1                                       (line 2)
  port (                                           (line 3)
    I0, I1: IN std_logic;                          (line 4)
    O1: OUT std_logic;                             (line 5)
  );                                              (line 6)
end component;                                    (line 7)

component U2                                       (line 8)
  port (                                           (line 9)
    I0, I1: IN std_logic;                          (line 10)
    O1: OUT std_logic;                             (line 11)
  );                                              (line 12)
end component;                                    (line 13)

begin                                             (line 14)
```

Note that the components are declared between the architecture declaration line (line 1 in the code above) and the begin (line 14). The declaration of the component consists simply on the description of its inputs and outputs.

After the component declaration, follows the component instantiation which is the specification of the connections between the components. Component instantiation for the circuit shown in figure 1 would be as described by the code below:

```
begin                                             (line 1)
your_instance_name : U1                          (line 2)
  port map (                                       (line 3)
    I0 => A,                                       (line 4)
    I1 => B,                                       (line 5)
    O1 => sig_1                                    (line 6)
  );                                              (line 7)

your_instance_name2 : U2                         (line 8)
  port map (                                       (line 9)
    I0 => sig_1,                                   (line 10)
    I1 => C,                                       (line 11)
    O1 => o                                        (line 12)
  );                                              (line 13)
end structural;                                   (line 14)
```

Line 1 is the begin line for the structural description. Note that in the name of the inputs and outputs of the components are declared in the left column while the name of the connections is declared in the right column. For instance line 4 says that the input line I0 of the component U1 is connected to the input of the circuit A. Similarly, line 10 says that the input line I1 of the component U2 is connected to the input C of the circuit.

Now, from the schematic of the circuit shown in figure 1 we know that we want to connect the output O1 of component U1 with the input I0 of component U2. This connection cannot be made directly because one point of the connection is an input (I1) and the other point is an output (O1). We cannot mix apples and potatoes (inputs and outputs) together. That is why we need to declare a **signal** which will act as a bridge between the input and output that we want to connect

together. That is why, line 6 of the code above shows a connection between O1 of component U1 with the signal sig_1, and line 11 shows a connection between O1 of the component U2 with the same signal. The connection has now been made.

The signal we used must be declared in the same section of the code as the component declaration as follows:

```
signal sig_1 : std_logic;
```

Finally, note that the code above shows a tag “*your_instance_name*” before the name of the component U1 or U2. That tag will carry a name for the component instantiation. For instance, if our system has more than one component of the same kind (lets say two components U1), that tag will allow us to differentiate between the instantiation of the two similar components.

Structural description is very useful when piece of code representing black boxes, is given to you to put then together to build a system.

Part 3: Two's Complement

Two's complement is a positional number system that allows you to represent both positive and negative numbers. The radix (r) of the system is 1; meaning, there is a distance of 1 between any two consecutive digits and the number representation. If the most significant bit of the number is 1, then the number is negative.

Finding the Value of a Two's Complement Number

Figure 2 is the equation to find the value of a two's complement number. In the equation, b is a bit in the number and n is the number of bits.

$$Value = -b_{n-1} \times 2^{n-1} + \sum_{i=0}^{n-2} b_i \times 2^i$$

Figure 2: Value of a Two's Complement Number

Example 1: Find the decimal value of the number: **11001**

Number of bits: 5

	1	1	0	0	1
Bit	4	3	2	1	0

Figure 3: Bit Positions of 11001

Solution:

$$Value = -b_{5-1} \times 2^{5-1} + \sum_{i=0}^{5-2} b_i \times 2^i$$

$$Value = -b_4 \times 2^4 + b_3 \times 2^3 + b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0$$

$$\text{Value} = -1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

$$\text{Value} = -7$$

Figure 4: Solution to Example 1

Creating a Two's Complement Representation of a Negative Number

1. Represent the magnitude of the number in binary.
2. Complement each bit of the number.
3. Increment the binary number by one.

Example 2:

Find the Two's Complement of **-7**.

1. Binary representation of 7: 00111
2. Complement each bit of the number: 11000
3. Increment the binary number: **11001**

Maximum and Minimum Representable Values

The maximum and minimum values that can be represented depend on the number of bits.

For a number with 5 bits, the maximum value is 15. The minimum value is -16.

$$\text{Maximum} = 2^{n-1} - 1 \qquad \text{Minimum} = -2^{n-1}$$

Figure 5: Max and Min Values