Introduction to VHDL

Dilawar Singh

March 6, 2013

Abstract

This is not a tutorial on but a brief introduction to VHDL language. It has been assumed that you have been using at least one programming language such as C/C++.

To do the VHDL related assignments, you need not know full VHDL. Only what is minimal required is discussed here. You need to know entity-architecture pair, data-types, and use of components in VHDL. The best way to learn is, as the Mr. Dodo remarked in Alice in Wonderland, is to do it.

1 Introduction

Let's begin with a toy shown in following figure. One excercise is obvious : *Write its equivalent proto-RTL description.* Later, we can simply translate this proto-RTL description to VHDL (or any other HDL) by hand, or better, we can write a computer program to automate the conversion.

In this document, we'll describe this circuit directly in VHDL.

;

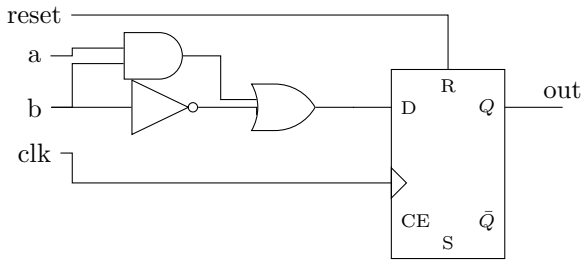

Figure 1: A small logic circuit. This represents one of the most commonly found hardware block : some combinational logic attached to the input of flip-flop. We have D-type flip-flop on the right. Unused pins are `S` (set), $\bar{Q}$, and `CE` (clock-enable).

While describing a hardware in VHDL, one needs to know the input and output (hereafter, **port**) of the system. We have `a`, `b`, `clk`, and `reset` as input to (input port of) this system and `out` as the output (port) of this system shown in figure 1. Immediately after this we ask the **type** of ports. We assume that all ports in this system have data-type **bit** [1].

Till now, we have the following information about the system :

Port	Direction	Type
a	in	bit
b	in	bit
clk	in	bit
reset	in	bit
out	out	bit

[1] See the types available in VHDL language. VHDL type system is stronger than Verilog. Why should it matter?

2 Entity-Architecture pair in VHDL

In VHDL, any digital system is described by **entity-architecture** pair. **Entity** describes what a system receives as input nd what it generates as output. **Architecture** describes how the system operates on input to generate it's output. If an analogy is to be drawn with C/C++ languages, it can be said that entity are like function declarations while architecture are like function definitions. Enough philosophies! Let's see some code.

```
-- NOTICE : VHDL does not differentiate between lower-case and upper-case words.

-- These are standard libraries one HAS TO include.
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

-- All of your current designs are stored in WORK library. In ghdl you can
-- change its path by --work option. We are pretending to use it as if a lot of
-- work has been done in WORK library.
USE WORK.ALL;

-- Here goes the declaration of our entity.
ENTITY demoEntity IS
  PORT(
  a, b : IN BIT;
  reset : IN BIT;
  clk : IN BIT;
  q   : OUT BIT
);
END ENTITY;
```

Listing 1: Entity declaration

And now we have to write the **architecture** of this entity. There are two style of writing architecture. One is called **structural** and other is **behavioural**. If we already know the components (as we do for this system) it is straightforward to write the structural description. When we do not know the components and their arrangements (in most of the case we do not know), we write behavioural description.

Remark 1 (Structral vs behaviour)**.** *Structural architecture describes* what is connected with what*. In our system we have three gates connected with each other and their output is reflected to* `out` *at each rising edge of clock (D-flip flop will ensure that.). Behavioural architecture describes* what happens when *which can be described by the following equation.*

$$out = \begin{cases} (a \wedge b) \vee \neg b & \text{on rising edge of clock and reset is unset} \\ \text{no change} & \text{otherwise} \end{cases} \tag{1}$$

Structural architecture Now let's see how to write the structural architecture.

```

-- Structural architecture
ARCHITECTURE structural OF demoEntity IS
  -- Declare components. These components have entity-architecture pairs
  -- declared some other file (or may be in this file also).
  -- We can also use already available components in some library like we
  -- use functions available in some library.
  COMPONENT and2
    port(in1, in2 : in bit;
         out1 : out bit
        );
  end COMPONENT;

  COMPONENT or2
    port(in1, in2 : in bit;
          out1 : out bit
         );
  end COMPONENT;

  COMPONENT not1
    port(in1 : in bit;
         out1 : out bit
        );
  end COMPONENT;

  COMPONENT dff
    port(d : IN BIT;
         clk : IN BIT;
```

```
29          reset : IN BIT;
            q : OUT BIT
31        );
    END COMPONENT;
33
    -- Local signal (like local variables in C).
35  signal andOut, notOut : bit;
    signal orOut : bit;
37
BEGIN
39  -- Now inside the architecture, describe the structure of model.
    -- Port map declares which port is connected with what.
41  -- There is one more style to map the ports.
    A1 : and2 port map(a, b, andOut);
43  N1 : not1 port map(b, notOut);
    O1 : or2 port map(andOut, notOut, orOut);
45  D1 : dff port map(orOut, clk, reset, q);

47 END structural;
```

Listing 2: We do not define the architecture of components used in this code. Each component again has **entity-architecture** pair. We can write those in different files and compile that file separately using ghdl. How to do it is available in ghdl manual.

Behavioural architecture This is a more natural way to describe the hardware. Most of the time we think in terms of behaviour of the circuit.

```
-- behavioural architecture
2 ARCHITECTURE behav of demoEntity IS
BEGIN
4 PROCESS(clk, reset)
  BEGIN
6   if(reset = '1') then
      q <= '0';
8   elsif(clk'event and clk='1') then
      q <= ( a and b) or (not b);
10  end if;
  END PROCESS;
12 END behav;
```

Listing 3: Behavioural architecture

Remark 2. *Multiple architecture*

We can have any number of architectures for one entity. If we do so, it is not clear which which architecture to use with the entity. Standard practice in VHDL is to write ***configuration****.*

Behavioural to hardware Structural architecture is essentially a netlist of the circuit : which component is connected to which component. Therefore much is known about the circuit topology. But what about behavioural architecture? It is not always easy to convert a behavioural description to hardware despite the rising maturity of synthesis tools [2] Fortunately for us, we are not concerned about synthesis in this course. Anyway lets have a look at the hardware is generated by Xilinx tool.

[2]Conversion of a description to a netlist of the hardware is called synthesis.

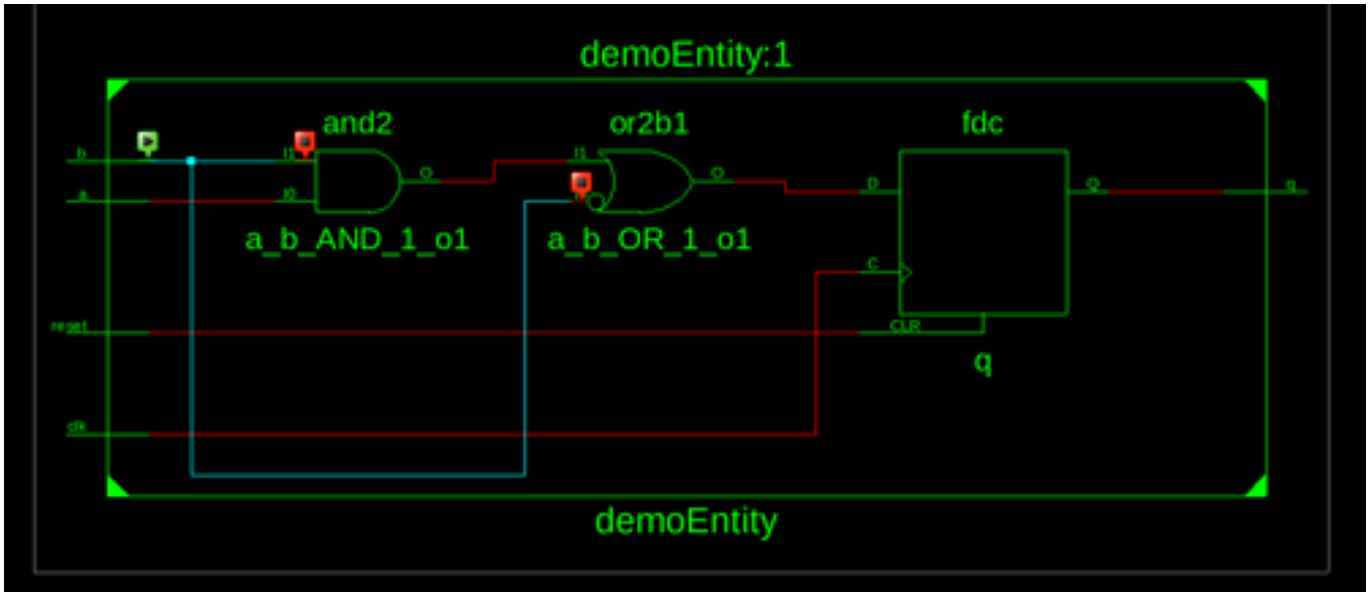

Figure 2: Hardware synthesised by Xilinx tool. Well, it looks like what we started with. In real life we usually do not know details of the hardware connections. Its bahaviour is specified and simulation is required to say something about its correctness. When we write code for simulation, we can full language. All constructs of a language are not supported by synthesise tool and one has to rewrite that particular section of code. A more recent language **bluespec** (available in HPC lab, IIT Bombay) guarantees that if you can compile it, you can synthesise it.