Some notes on VHDL

Dilawar Singh [*]

March 6, 2013

Abstract

It has been assumed that you have been using at least C/C++.

To do VHDL related assignments, you need not know full VHDL. Only what minimal is required discussed here. You need to know entity-architecture pair, data-types, and use of components in VHDL. Mimicking Dodo the bird of *Alice in Wonderland* one can say that the best way to learn a language is to write few programs in it.

1 Introduction

Let's begin with a toy shown in following figure. One exercise is immediate : *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. [1]

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

;

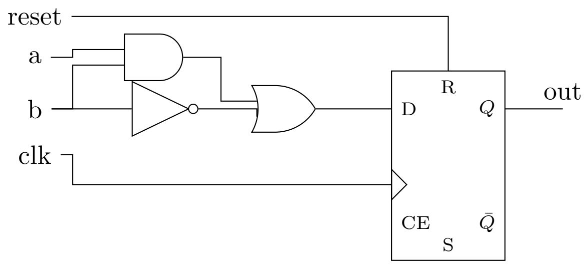

Figure 1: A small logic circuit. This represents one of the most commonly found hardware block : some combinational logic attached to the input of flip-flop. We have D-type flip-flop on the right. Unused pins are `S` (set), $\bar{Q}$, and `CE` (clock-enable).

Abstraction available in VHDL VHDL provides a block-level abstraction of system. One needs to know the input and output (hereafter, **port**) of the system. We have `a`, `b`, `clk`, and `reset` as input (port), and `out` as the output (port) of system shown in figure 1. Immediately after setting the ports, we ask what sort of data these ports carries : the **type** of ports. We assume that all ports in this system have data-type **bit**. It is usually not the case. We can have vectors of bit, int, double and other custom made data-types. VHDL strong type systems enables a designer to catch some bugs earlier in development process. [2].

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

[*]His name is pronounced 'the law were'. He works in HPC lab less commonly known as L.W.C.A. (Lab Wihtout a Cool Acronym). He can be contacted at `dilawar@ee.iitb.ac.in`

[1]Dr. Sameer Sahashrabudhdhe work (guide Prof. Madav Desai) AHIR tool-chain does something similar. It converts a C-program to VHDL.

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

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

2 Entity and Architecture pair

In VHDL, any digital block is described by **entity-architecture** pair. **Entity** describes what a system receives as input and what it generates as output. **Architecture** describes how the system operates on input to generate its outputs. If an analogy is allowed to be drawn with C/C++ languages, it can be said that entity are like function declarations while architecture are like function definitions. 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

Now we should write the **architecture** for this entity. There are two styles of writing architecture : **structural** and **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 (usually the case!), 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 it.). 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. We are using `COMPONENT` keyword in following listing. Components are previously defined entities. This is similar to the idea of instantiation of functions.

We have not defined the entity-architecture pairs for components used in this listing. We assume that they are readily available (`WORK` library).

```

-- 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;
        reset : IN BIT;
        q : OUT BIT
      );
  END COMPONENT;

  -- Local signal (like local variables in C).
  signal andOut, notOut : bit;
  signal orOut : bit;

BEGIN
  -- Now inside the architecture, describe the structure of model.
  -- Port map declares which port is connected with what.
  -- There is one more style to map the ports.
  A1 : and2 port map(a, b, andOut);
  N1 : not1 port map(b, notOut);
  O1 : or2 port map(andOut, notOut, orOut);
  D1 : dff port map(orOut, clk, reset, q);

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
ARCHITECTURE behav of demoEntity IS
BEGIN
PROCESS(clk, reset)
  BEGIN
    if(reset = '1') then
      q <= '0';
    elsif(clk'event and clk='1') then
      q <= ( a and b) or (not b);
    end if;
  END PROCESS;
END behav;
```

Listing 3: Behavioural architecture

Remark 2. *Handling multiple architectures of a single entity*

*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**. We don't discuss them here. If only one architecture is written, ghdl will figure it out by itself.*

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 [3] 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.

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

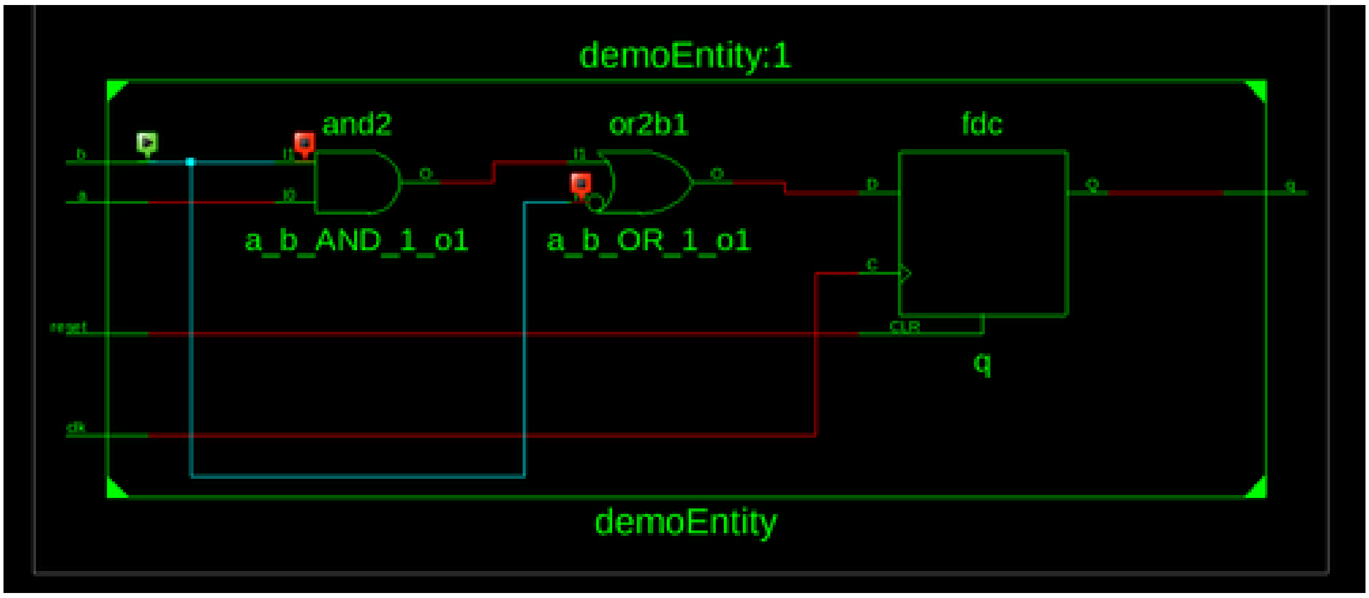

Figure 2: Hardware synthesised by Xilinx tool. Well, it looks like what we started with. We synthesised the behavioural architecture. When we write code for simulation, we can use full language. But all constructs of a language are not supported by synthesise tools and one has to rewrite that particular non-supported section of code.