1 Introduction

TITLE Building a better VHDL testing environment

To understand what VHDL is used for, a small introduction on digital electronics is best read first. Digital electronics differs from analog electronics in that it uses a discrete set of voltage levels to transmit signals. The most common number of items in the set is 2, namely one and zero. The advantage of using a discrete number of levels rather than a continuous signal is that noise generated by the environment, thermal noise and other interfering factors will have a far less significant influence on the signal. To process these discrete signals, electronics are made up of transistors that form *logic gates*, these logic gates make it so that only a certain combination of ones and zeroes at the inputs result in certain ones or zeroes at the outputs.

The most common gates are *nand* (not and, two ones make a zero, all other combinations result in one) and *nor* (not or, any one at the input results in a zero at the output). A certain combination of these logic gates are used to build higher-level blocks such as flip-flops, which are used to make registers and so on up to the entire chip design. An HDL can be used to describe any one of these levels, right down to the logic gate level, however this last one might not be a good idea considering most synthesis tools can produce superior logic gate-level layouts[1]. The level that uses certain blocks of logic gates to describe more complex behaviour is called the *Register Transfer Level* or RTL. Some blocks are standard implementations that have been widely used and nearly fully optimized, such as memories, flipflops and clocks. An RTL flipflop implementation is shown here:

```
library IEEE;
use IEEE.std_logic_1164.all;
entity DFF is
     Port ( D   : in  std_logic;
            CLK : in  std_logic;
            Q   : out std_logic;
end DFF;
architecture Behavioral of DFF is
     begin
          process (CLK)
          begin
               if rising_edge(CLK) then
                    Q <= D;
               end if;
          end process;
end Behavioral;
```

The IEEE library provides a number of extensions on the original VHDL code that allow a more realistic simulation and description of hardware behaviour. An entity defines the inputs and outputs of a certain building block, in this

case the D-FlipFlop or DFF. The D stands for Delay, and it simple puts on its output Q that which was on the input one clock cycle earlier. The architecture, in this case Behavioural, takes the description of an entity and assigns a real implementation to it. All processes are parallelly-executed, this does not mean that all are triggered at the same time, nor do they take as long to finish, but it means that any process can be executed alongside any other process. In this case there is only one (nameless) process that describes the entire behaviour of the flipflop. It waits for the rising edge of the clock, which is a transition from zero to one, and then it schedules the value of D to be put on Q until the next rising edge appears.

This is a basic example of an entity, an architecture and a process. This flipflop could be used in certain numbers to build a *register*, a collection of ones and zeros (henceforth named *bits*) that is used to (temporarily) store these values. The register could then be used alongside combinational logic to build an even bigger entity. The idea here is that small building blocks can be combined to produce vast and complex circuits that are nearly impossible to describe in one go. Adding all these layers together also creates a lot of room for error, and having a multi-level design makes it somewhat difficult to pinpoint the exact level and location of any errors. Therefor it is paramount that all code on all levels is tested thoroughly, this is done by use of *testbenches*.[2]

Testbenches are made up of code that takes a certain building block, the *Unit Under Test* (UUT) or *Device Under Test* (DUT). The testbench then performs a certain sequence of inputs and monitors the outputs. If the device performs normally, the received output sequence should match a certain *golden reference*, the exepected output sequence. In these testbenches it is also interesting to see how well a device performs if its inputs behave outside the normal mode of operation. When all of these tests have finished and the ouput performs as expected, the device is ready to be put into production or further down the developmental process.

It is easy to see that if a device is not tested properly and faults propagate it can be very expensive to correct, especially at the stage of production, where a single photomask, used to "print" part of the layout, can easily cost $100,000[3]. Therefore a large portion of time is spent testing the code and writing more tests for this purpose. Both writing and testing the code can

References

[1] http://www.asic-world.com/vhdl/intro1.html

[2] Writing Testbenches: Functional Verification of HDL Models

[3] Mask Cost and Profitability in Photomask Manufacturing: An Empirical Analysis