Thesis – Joren Guillaume

## Introduction q

### Context en inleiding

In de wereld van hardware ontwikkeling wordt er veel gebruikt gemaakt van *Hardware Development Languages* (HDLs) om zowel prototypes te ontwikkelen op *Field Programmable Gate Arrays* (FPGAs) als de beschrijving voor het finale ontwerp dat gemaakt zal worden. Een van zulke talen is de *Very High Speed Integrated Circuits HDL, VHSIC HDL* of kortweg VHDL. Dit is de taal die gebruikt wordt in deze thesis.

Een typische ontwikkelcyclus bestaat uit verschillende stappen, waarvan het software ontwerp en vooral het testen hiervan het meest relevant zijn voor deze thesis. Omdat het onmiddellijk schrijven van een foutloos ontwerp praktisch onmogelijk is moet alles getest worden. Deze testen moeten niet alleen controleren of het ontwerp doet wat het moet doen onder normale omstandigheden, maar ook verkennen wat het doet onder abnormale omstandigheden en randgevallen.

Aangezien testen een groot deel van de tijd innemen die nuttiger zou kunnen worden besteed zou er dus een manier moeten bestaan om snel en efficient te testen. Er zijn een aantal factoren die daarbij een rol spelen. Vooraleerst wordt er veel code geschreven specifiek voor een enkele *Unit* of *Device Under Test* (UUT, DUT). Indien er een standaard bibliotheek bestond met een groot scala aan verscheidene, veel gebruikte testen zou dit dus een grote tijdsbesparing zijn.

Verder is er geen standaard aanpak om te testen. Er zijn wel verscheidene manieren maar deze verschillen van bedrijf tot bedrijf of soms zelfs van ingenieur tot ingenieur. Indien er een gestandardiseerde aanpak was zou dit bijdragen tot het vlotte overdragen van code en het algemeen begrip onder ingenieurs verbeteren.

Als laatste moet er zeker nog vermeld worden dat er 2 types ingenieurs zijn: ontwerpers en testers. De ontwerpers schrijven de code, doen enkele rudimentaire testen en schuiven het ontwerp door naar de testers die meer rigoreuze testen uitvoeren. Indien de ontwerpers een makkelijk bruikbare bibliotheek met gestandardiseerde testen zouden zij zelf een groter deel van de *bugs* (fouten in de code) kunnen opsporen zonder dat zij daar meer tijd aan verliezen. Deze bibliotheek kan al dan niet onderdeel zijn van een gestandardiseerde aanpak die ,indien gebruikt, nog vlotter code doet schrijven.

We hebben dus gesproken over een gestandardiseerde bibliotheek met testfuncties, een gestandardiseerde manier van programmeren en testen en we zullen verder nog spreken over een gestandardiseerde manier van verwerken van deze testen.

### Uitgebreidere beschrijving van doel

Een korte omschrijving is dus:

*Het ontwikkelen en uitleggen van een gestandardiseerde bibliotheek, aanpak en verwerker om het vlot ontwikkelen van code in VHDL te promoten.*

Iets langer gezegd betekent dit dus dat er vooraleerst een manier moet zijn om de testcode te schrijven. Eenmaal dat er een vaste aanpak is kan deze op een vaste manier verwerkt worden. Bij het schrijven op deze vaste manier kan er dan gebruik gemaakt worden van een bibliotheek met functies die zowel veel gebruikt worden alsook makkelijk kunnen worden begrepen en gebruikt.

De manier van aanpak moet makkelijk te verstaan zijn, makkelijk aan te leren zijn en moet het gehele gebied van testen omvatten. Dit zodat ontwikkelaars niet hoeven te wisselen tussen aanpakken voor kleine stukjes van hun code.

De manier van verwerken moet globaal beschikbaar zijn zodat ontwikkelaars op alle platformen dezelfde verwerker kunnen gebruiken. Ze moet snel en efficient zijn maar toch eenvoudig te verstaan en moet een vlot en leesbaar overzicht geven van de fouten en successen van de code.

De bibliotheek moet voldoende uitgebreid zijn maar niet onoverzichtelijk. De functies en procedures moeten de vele gevallen dekken die vaak voorkomen in *testbenches* (code die specifiek dient om een ontwerp te testen). Zij moet ook handige uitbreidingen aanbieden om zowel snel als rigoreus te testen waarbij de code steeds vlot leesbaar blijft.

## Huidige situatie en doelstellingen

Bij het schrijven van deze inleiding zijn er 3 dingen bereikt en reeds bepaalde keuzes gemaakt.

### Huidige situatie

Voor het verwerken van de code is er gekozen voor een script in Python. Dit script wordt opgeroepen vanop de *command line* met voorlopig als enig argument de *filename* van de te verwerken code. Het script zoekt de testen in de code en deelt deze in aparte files op, zodat alle testen afzonderlijk van elkaar kunnen worden uitgevoerd. Dit is belangrijk om alle randgevallen te dekken (*test independence)*.

Er wordt verwacht dat de code zelf geschreven is op een bepaalde wijze. Voor het huidige script is dit simpelweg dat alle testen *asserts* zijn. In de *architecture* (implementering) van de testbench moeten/mogen enkel asserts staan.

De bibliotheek is klein en niet volledig getest. Zij wordt gebruikt in een klein testontwerp dat verwerkt wordt door het script.

Al de bovenstaande items zijn *proof of concept*, zij dienen enkel tot bewijs dat code modulair kan gemaakt worden en verwerkt. Het is heel goed mogelijk dat geen enkele van deze code in het finaal ontwerp terecht komt.

### Doelstellingen

Echt specifieke doelstellingen kunnen nog niet worden gezet bij het schrijven van deze inleiding, maar algemene doelen kunnen wel al gezet worden.

De volledige aanpak zou *Operating System* (OS) onfanhankelijk moeten blijven. De programmeertaal Python is beschikbaar op zowel Windows als Linux, en dit geldt ook voor de gebruikte compiler en simulator Modelsim (een Apple systeem was niet ter beschikking om te testen).

De verwerker zou een aantal opties kunnen hebben. Momenteel is er de optie tot het opruimen van de gebruikte en *gecompileerde* bestanden, zodat enkel het logboek blijft staan. Dit zou kunnen worden uitgebreid tot bijvoorbeeld:

* Instellen van de taalstandaard (93, 98, 2008 ...)
* Selectieve uitvoering van testen
* Tonen van simulatieprogramma (momenteel is dit “silenced”)
* Optioneel inbrengen van simulatieparameters

Bij de manier van coderen zou een implementatie van *Test Driven Development* moeten aanwezig zijn. Deze programmeurstechniek laat toe testen individueel en onafhankelijk van elkaar te definieren en zou zowel vlotter als beter moeten doen programmeren.

De bibliotheek moet uiteraard veel uitgebreider worden. Verwacht wordt dat deze zo goed als alle noden kan vervullen van een normale testbench. De functies in de bibliotheek moeten makkelijk verstaanbaar zijn en makkelijk te gebruiken zijn. Uiteraard zijn zij bedoeld om te gebruiken met het verwerkingsscript, maar zij zouden ook onafhankelijk moeten kunnen gebruikt worden.