

VHDL 101

Patrick Mintram

November 17, 2019

1 Introduction

Things this session is intended to wet your appetite to the world of FPGAs, so that if you choose to you can start having a play in your own time. This session is:

1. A brief overview of VHDL
2. A chance to get hands on with some Hardware
3. A chance to make a 'hello world' in Hardware


Things this session is not:

1. An introduction to Digital Design
2. A comprehensive deep dive into VHDL
3. Likely to finish on time

By the end of this session you should have configured an FPGA to react to a switch input, and to see that output on an LED.

2 How to use this guide

It is not expected that session participants follow this document step by step. It expected that participants reference it to work at own pace, or to be able to recreate the work in their own time. Participants are encouraged to make notes are they require on this document, once printed.

The hand  icon will appear when something is purposely had some handwaving applied to it to keep it simple.

Anything in a box in an instruction to complete, like 'click this' or 'type this'

2.1 Terminology

This document contains a number of terms which may be new to people so a brief overview of these follows:

- **HDL** stands for Hardware Description Language, of which VHDL is one. Others include Verilog, System Verilog and System C. There are of course many more, but these 4 are the ones that I have come across most. Note that these are specifically used to describe digital circuits, if you want to describe analog circuits there are flavours such as Verilog-AMS(Analog and Mixed-Signal) or VHDL-AMS(Analog/Mixed-Signal Extension), I have never come across these in the wild though.
- **Entity** is a keyword in VHDL. Each item you describe in VHDL is in an *entity* and is split into two parts; the entity *declaration* and the entity *architecture*.
- **Module** is not a keyword in VHDL, but at a high level the term may be banded around as a way of describing a distinct design unit, e.g. an entity is a module of an overall design.
- **Architecture** is another keyword. The architecture of an entity is it's implementation details. An entity may have more than one architecture, where the one instantiated can be selected via a configuration.

- **Dataflow** is a type of architecture implementation in which the source code describes the flow of data in the module, and leaves the structure of the design up to the toolchain.
- **Behavioural** is a type of architecture implementation in which the source code describes the behaviour of the module. This is most familiar to us as software engineers as it is a high level description of the module using abstractions such as `if else` etc. This may not be the most efficient way of doing the job, but it's relatively quick and easier to understand, even if it's not synthesisable. *Writing the flowchart*
- **RTL** is a type of architecture implementation in which the source code is fully synthesisable may go into detail about the gates in the module. *Writing the circuit diagram*
- **Synthesis** is the process of turning your description into blocks of hardware. 🌀 There are a number of steps to turning VHDL source code into a format actually usable by an FPGA, and often people refer to the process as compilation or synthesis.

Contents

1	Introduction	2
2	How to use this guide	2
2.1	Terminology	2
3	Why Should I Care?	5
3.1	Toolchain	5
4	Important things to remember	5
5	Finally lets get to doing some VHDL	5
5.1	Reference Project	5
5.2	How to see output from our VHDL	5
5.2.1	Simulation	5
5.2.2	On the hardware itself	5
6	Further Activities	5

List of Figures

3 Why Should I Care?

FPGAs enable Low Latency processing ¹, so performing a transform on data coming in and getting the result output can be much faster than in a traditional CPU based approach. They also provide far more IO configurability than the traditional approach; the IO logic, and the pin it's connected to are totally configurable in code and constraints files². Say a requirement changes from an 8 bit UART bus to a proprietary 11 bit UART bus - this would require a whole new microcontroller in a traditional approach however with an FPGA this might only require a change to a `generic` and a recompile.

For the reasons stated above, typical uses include signal processing such as filtering ², and High Speed IO such as devices produced by SpeedGoat ³.

3.1 Toolchain

Vivado vs Quartus. Others - yosys, ghdl etc

4 Important things to remember

It's not software it's hardware

5 Finally lets get to doing some VHDL

5.1 Reference Project

This will be an overview of the reference project

5.2 How to see output from our VHDL

5.2.1 Simulation

How to interpret the waveforms

5.2.2 On the hardware itself

How to load onto the board

6 Further Activities

If this has been fun, and you want to learn more here are some useful resources I have had successes with:

- Effective Coding with VHDL⁴

¹<https://blog.esciencecenter.nl/why-use-an-fpga-instead-of-a-cpu-or-gpu-b234cd4f309c>

²<https://digital-library.theiet.org/content/journals/10.1049/iet-cdt.2016.0067>

³<https://www.speedgoat.com/products/simulink-programmable-fpgas-fpga-i-o-modules-io334>

⁴<https://www.amazon.co.uk/Effective-Coding-VHDL-Principles-Practice/dp/0262034220>

6 FURTHER ACTIVITIES

- Awesome VHDL ⁵
- Digital Fundamentals ⁶

⁵<https://github.com/VHDL/awesome-vhdl>

⁶<https://www.amazon.co.uk/Digital-Fundamentals-Thomas-L-Floyd/dp/0132737965>