

# VHDL 101

Patrick Mintram

November 27, 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 is 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.
- **PMODs** are a type of plug in INFORMATION HERE

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>How to use this guide</b>	<b>2</b>
2.1	Terminology . . . . .	2
<b>3</b>	<b>Why Should I Care?</b>	<b>5</b>
3.1	Toolchain . . . . .	5
<b>4</b>	<b>Important things to remember</b>	<b>6</b>
<b>5</b>	<b>Finally lets get to doing some VHDL</b>	<b>6</b>
5.1	Reference Project . . . . .	6
5.1.1	File Types . . . . .	6
5.2	How to see output from our VHDL . . . . .	6
5.2.1	Simulation . . . . .	6
5.2.2	On the hardware itself . . . . .	7
<b>6</b>	<b>Further Activities</b>	<b>7</b>

## List of Figures

Manufacturer	Toolchain	Device
Intel	Quartus	SOME BOARDS
Xilinx	Vivado	SOME BOARDS
	ghdl	
ICE	IceBreaker	Lattice

Table 1: An overview of devices you might come across

### 3 Why Should I Care?

FPGAs enable Low Latency processing <sup>1</sup>, 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 files<sup>2</sup>. 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 <sup>2</sup>, and High Speed IO such as devices produced by SpeedGoat <sup>3</sup>.

#### 3.1 Toolchain

One important takeaway from this is that toolchains are important. Each device vendor will have their own proprietary toolchain. This means that you can approach FPGA development in one of two ways:

- Choose a toolchain you're familiar with, then a device from the manufacturer PUT IN LOGO
- Choose a device which suits your requirements, then potentially suffer with an unfamiliar toolchain PUT IN LOGO

Fortunately there are few realistic choices when it comes to this decision; use the Quartus toolchain with Intel <sup>4</sup> devices, or using the Vivado toolchain<sup>5</sup> with Xilinx devices. I am most familiar with Vivado, so this workshop is based around that. The reason for this choice is that the *Zynq* range of devices from Xilinx are a System on Chip (SOC) which allows me to use either the ARM core or some Programmable Logic in any projects I'm undertaking. Similar devices may exist from the Intel range, but at the time of buying my dev board they didn't. For an overview of some devices you might come across see Table 1.

There is also a third option when it comes to toolchains; if you don't care about synthesis a well known simulation tool is **ghdl**. This allows for your VHDL code to be written, analysed, elaborated, and testbenches run very quickly and without any synthesis. One of the obvious

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

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

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

<sup>4</sup>Altera

<sup>5</sup>Older Xilinx devices can use the ISE suit from Xilinx

limitations with this is that it doesn't allow you to put the hardware onto a board. There are plenty of docs available online to reference when it comes to using this and this projects `build.sh` in the `scripts` directory might help as a starting point.

I have seen on twitter lots of talk of the Lattice; this seems to be an open source FPGA and toolchain.

## 4 Important things to remember

There is one main things to remember: **It's not software it's hardware**. Everything you do should be done with the hardware you're creating in the back of your mind. You should make sure you are familiar with the design guidance for your device of choice. This is because different devices are made of different things - the Xilinx guidance for example states **THIS ABOUT THE ZYNQ DEVICES**.

## 5 Finally lets get to doing some VHDL

### 5.1 Reference Project

The reference project provided enables you to flash some LEDs at different speeds - this demonstrates the ability to create designs with different things happening at different times and at different rates. It also contains some stubs for you to put in your own functionality; turning off and on an LED from a switch.

#### 5.1.1 File Types

A VHDL project will typically contain the following files:

1. Design Source Files `*.vhd`: These contain the actual source code for the hardware you're describing. These should contain only synthesisable code.
2. Design Test Benches `*_tb.vhd`: These contain test benches for your design entities, usually suffixed with `_tb` so you know it's a test bench, and will likely contain non synthesisable code.
3. Constraints Files `*.{ucf|xcd}`: These files describe any constraints of the platform or design, such as pins used, timing requirements etc.
4. Scripts `*.{sh|tcl}`: These are provided to help with setting up and managing the project. `tcl` files seem to be what most tools prefer, and tools will often provide a number of utility functions to help with this.

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 VHDL<sup>6</sup>
- Awesome VHDL<sup>7</sup>
- Digital Fundamentals<sup>8</sup>

---

<sup>6</sup><https://www.amazon.co.uk/Effective-Coding-VHDL-Principles-Practice/dp/0262034220>

<sup>7</sup><https://github.com/VHDL/awesome-vhdl>

<sup>8</sup><https://www.amazon.co.uk/Digital-Fundamentals-Thomas-L-Floyd/dp/0132737965>