1

Introduction

This chapter looks at the way in which VHDL is used in digital systems design, the historical reasons why VHDL was created and the international project to maintain and upgrade the language.

1.1 The VHDL Design Cycle

From its conception, VHDL was intended to support all levels of the hardware design cycle. This is clear from the preface of the Language Reference Manual (LRM) (IEEE-1076, 2008) which defines the language, from which the following quote has been taken:

> VHDL is a formal notation intended for use in all phases of the creation of electronic systems. Because it is both machine readable and human readable, it supports the development, verification, synthesis, and testing of hardware designs; the communication of hardware design data; and the maintenance, modification, and procurement of hardware.

The key phrase is 'all phases'. This means that VHDL is intended to cover every level of the design cycle from system specification to netlist. As a result, the language is rather large and cumbersome. However, this does not necessarily make it difficult to learn. It is best to think of VHDL as a hybrid language, containing features appropriate to one or more of the stages of the design cycle, so that each stage is in effect covered by a separate language that also happens to be a subset of the whole. Each subset is relatively easy to learn, provided there is guidance as to what is in, and what is not in, that subset.

In the idealised design process, there are three subsets in use – since there are three stages that use VHDL. These are: system modelling (specification phase), register-transfer level (RTL) modelling (design phase) and netlist (implementation phase).

In addition to these VHDL-based phases, there will be an initial requirements phase that is conventionally in plain (human) language. Thus, there are three stages of transformation of a design: from requirements to specification, from specification to design and from design to implementation. The first two phases are carried out by human designers, the last phase is now largely performed by synthesis.

Figure 1.1 illustrates this idealised design cycle.

VHDL for Logic Synthesis, Third Edition. Andrew Rushton.

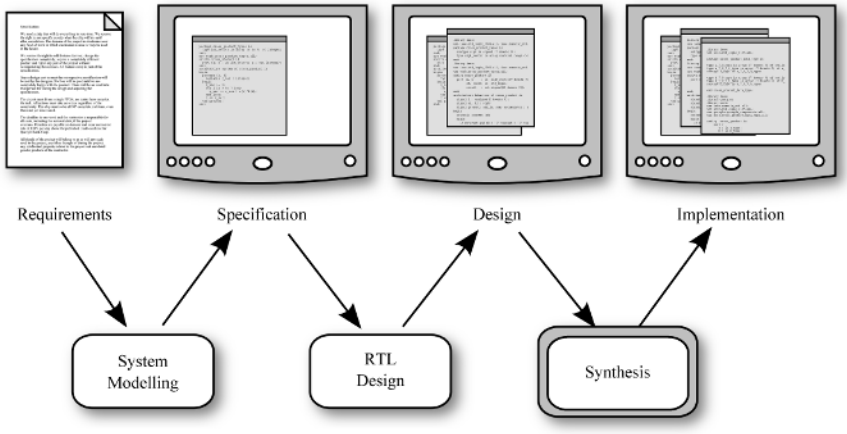

Figure 1.1 The VHDL-based hardware design cycle.

Typically, the system model will be a VHDL model that represents the algorithm to be performed without any hardware implementation in mind. The purpose is to create a simulation model that can be used as a formal specification of the design and that can be run in a simulator to check its functionality. This specification can also be used to confirm with a customer that the requirements have been fully understood.

The system model is then transformed into a register-transfer level (RTL) design in preparation for synthesis. The transformation is aimed at a particular hardware implementation but at this stage, at a coarse-grain level. In particular, the timing is specified at the clock cycle level at this stage of the design process. Also, the particular hardware resources to be used in the implementation are specified at the block level.

The final stage of the design cycle is to synthesise the RTL design to produce a netlist, which should meet the area constraints and timing requirements of the implementation. Of course, in practice, this may not be the case, so modifications will be required which will impact on the earlier stages of the design process. However, this process is the basic, idealised, design process using VHDL and logic synthesis.

1.2 The Origins of VHDL

VHDL originated from the American Department of Defense, who recognised that they had a problem in their hardware procurement programmes. The problem was that they were receiving designs in proprietary hardware description languages, which meant that, not only was it impossible to transfer design data to other companies for second sourcing, but also there was no guarantee that these languages would survive for the life expectancy of the hardware they described.

The solution was to have a single, standard hardware description language, with a guaranteed future. Specification of such a language went ahead as part of the Very-High Speed Integrated Circuits programme (VHSIC) in the early 1980s. For this reason, the language was later named the VHSIC Hardware Description Language (VHDL).

If the language had remained merely a requirement for military procurement, it would quite possibly have remained an obscure language of interest only to DoD contractors. However, the importance of the language development, and especially the importance of standardisation of the language, was recognised by the larger electronic engineering community and so the formative language was passed into the public domain by placing it in the hands of the IEEE in 1986. The IEEE proceeded to consolidate the language into a standard that was ratified as IEEE standard number 1076 in 1987. This standard is encapsulated in the VHDL Language Reference Manual (LRM).

1.3 The Standardisation Process

Part of the standardisation process was to define a standard way of upgrading the language periodically. Thus, there is a built-in requirement for the language to be re-standardised every five years. However, in practice updates have been irregular and driven by a desire to improve the language according to demand rather than this arbitrary 5-year cycle. Because the language has changed over the years, it is sometimes important to differentiate between versions. This is done in this book by referring to the year in which the standard was ratified by the IEEE. For example, the original standard, IEEE standard number 1076, ratified in 1987, is usually referred to as VHDL-1987. Subsequent revisions of the standard will be referred to in a similar way according to their year of ratification.

Here is a summary of the different versions and the features that affect the use of the language for synthesis:

VHDL-1987	The original standard.
VHDL-1993	Added extended identifiers, xnor and shift operators, direct instantiation of components, improved I/O for writing test benches. Most of the synthesis subset of VHDL is based on VHDL-1993.
VHDL-2000 (minor revision)	Nothing of relevance to synthesis.
VHDL-2002 (minor revision)	Nothing of relevance to synthesis.
VHDL-2008	Added fixed-point and floating-point packages. Added generic types and packages, enabling the use of generics to define reusable packages and subprograms. Enhanced versions of conditionals. Reading of out ports. Improved I/O for writing test benches. Unification of VHDL standards.

As you can see, there are only three versions of VHDL relevant to synthesis: VHDL-1987, VHDL-1993 and VHDL-2008. VHDL-1993 was the last revision to add features useful for synthesis. So VHDL-2008 is the first significant change in 15 years. A lot has been added in VHDL-2008 (Ashenden and Lewis, 2008) and most of it has some relevance to synthesis.

However, synthesis tool vendors are historically slow to adopt new language features. This is for good reasons – the focus of synthesis is the quality of the synthesised circuit and effectiveness of the synthesis optimisations, not the list of language features supported. This means that it is expected that several years will pass before the more significant changes in VHDL-2008 are implemented by synthesis tools and many never will be. In effect, synthesis users are still using VHDL-1993 and will continue to do so for the foreseeable future.

As a consequence, this book is based mainly on VHDL-1993. However, the more recent extensions are discussed where relevant, particularly with regard to the new fixed-point and floating-point packages added in VHDL-2008 but that have been made available as VHDL-1993 compatibility packages so that they can be used immediately on synthesisers that do not yet support the rest of VHDL-2008.

1.4 Unification of VHDL Standards

One of the largest changes in the VHDL-2008 standard is the unification of the many standards that define parts of the language and its environment.

The management of the standardisation process is down to the VHDL Analysis and Standardisation Group (VASG), part of the IEEE standardisation structure. In addition to the main standardisation process of the language itself, there are a number of working-groups working on standardisation of the ways in which VHDL is used. In the past, these working-groups have published standards of their own. For example, there was a group working on using VHDL for analogue modelling (VHDL-AMS – VHDL Analogue and Mixed-Signal – standard 1076.1), a group working on standard synthesisable numeric packages (VHDL Synthesis Package – standard 1076.3 (1997)), a group working on accelerating gate-level simulation (VITAL – the VHDL Initiative Towards ASIC Libraries – standard 1076.4), and a group working on the standard interpretation of VHDL for logic synthesis (VHDL Synthesis Interoperability – standard 1076.6). In addition, the 9-value logic type `std_logic` that is almost universally used for synthesis was developed as a completely different IEEE standard (VHDL Multivalue Logic Packages – standard 1164).

This separation of the standardisation of the various application domains of VHDL was effective in the early days of language development, because it allowed the subgroups to get on with their work independently of the main VHDL standardisation process and furthermore meant that they could publish their standards when ready, rather than waiting for the next formal release of the VHDL standard. However, this separation has become a problem as the working-groups' work has become mature, stable and in common use. For example, a release of a new standard for VHDL could leave the subgroups' standards lagging behind, compatible with the previous version and lacking the new language features.

So, in VHDL-2008, those working group standards that are specific to synthesis have been *partly* merged into the VHDL standard itself. Standard 1076 now includes the standard logic types (1164), the standard numeric types (1076.3) and some parts of the standard synthesis interpretation (1076.6). This doesn't make any difference to the user, but it does formalise these parts of the language as an integral part of VHDL and ensures that they stay in step with language developments in the future.

As you can probably imagine, this makes the Language Reference Manual (IEEE-1076, 2008) quite massive.

1.5 Portability

Synthesisable RTL designs can have a long life span due to their technology independence. The same design can be targeted at different technologies, revised and targeted at a newer technology and so on for many years after the original design was written. It is a wise designer

who plans for the long-term support of their designs. It is therefore good practice to write using a safe, common style of VHDL that can be expected to be supported for years to come, rather than use 'clever' tool-specific tricks that might not continue to be supported.

Also, it is not unusual for a company to change their preferred tools, or for a designer to be obliged to use a different synthesis tool because a different technology is being targeted. So it is good practice to write using a portable subset of synthesisable VHDL that will work across many different tools.

The problem with this principle is that synthesis relies on an interpretation of VHDL according to a set of templates, and historically each synthesis vendor has developed their own set of templates. This means that in practice, each synthesis tool supports a slightly different subset of VHDL. However, there has always been a lot of overlap between these subsets and this book attempts to identify the common denominator.

To make life more complicated, the IEEE Design Automation Standards Committee have specified a synthesis standard for VHDL (IEEE- 1076.6, 2004) that seems to be a superset rather than a subset of the VHDL supported by commercial tools. Therefore, adhering to the standard does not mean that a design will be synthesisable with any specific synthesis tool. It also seems unlikely that any single tool will implement every detail of this standard.

It is recommended that a subset is used that is common to all synthesis tools. As a consequence, this book focuses on the common subset and avoids the more obscure tool-specific features of VHDL, even if those obscure features are in the synthesis standard.