VHDL GUIDANCE FOR SAFE AND CERTIFIABLE FPGA DESIGN

P.M. Conmy*, C. Pygott†, I Bate**

*SSEI, Department of Computer Science, University of York, Heslington, U.K. philippa@cs.york.ac.uk
†Columbus Computing Ltd, Malvern, U.K. clive@columbus-softwareandsafety.com
** RTS Group, Department of Computer Science, University of York, Heslington, U.K.

Keywords: VHDL, Safety, FPGA, Coding Standards.

Abstract

Field Programmable Gate Arrays (FPGAs) are becoming increasingly popular for use within high integrity and safety critical systems. One commonly used coding language for their configuration is the VHSIC Hardware Description Language (VHDL). Whilst VHDL is used for hardware description, it is developed in a similar way to traditional software, and many safety critical software certification standards require the use of coding subsets and style guidance in order to ensure known language vulnerabilities are avoided. At present there is no recognized, public domain guidance for VHDL. This paper draws together many different sources to provide a starting discussion for a VHDL subset.

1 Introduction

Field Programmable Gate Arrays (FPGAs) are becoming increasingly popular for use within high integrity and safety critical systems. FPGAs contain hundreds of thousands of programmable logic cells, which can be configured for a wide variety of tasks, and offer many benefits over traditional micro-processors, such as efficient parallel processing and very predictable performance. FPGAs also have advantages over conventional hardware implementations: lower component count, lower real-estate requirements and simpler assembly. They are configured by using a Hardware Description Language (HDL), such as the VHSIC (Very High Speed Integrated Circuit) Hardware Description Language (VHDL) [10], to describe the required logic; this is converted into a configuration file which is loaded onto the device. The HDL goes through a two stage synthesis process, converting it first to a low level gate description, then assessing where these gates should be placed on the target device i.e. the layout.

VHDL is structured and developed in a very similar way to standard software languages. For safety critical software development, most recognised standards require the use of coding guidelines to constrain how the language is used. This consists of a series of rules which should be adhered to, assisting certification by avoiding known vulnerabilities and pitfalls [12]. Rules can also be followed to assist analysis, maintenance and the avoidance of poorly defined language features. Simpler syntactic rules can be enforced by tool support, but other more complex stylistic rules may require manual assessment.

Whilst many companies have in house VHDL development standards and subsets, and some articles can be found on common VHDL errors (e.g.[13]), there is currently no widely agreed or publicly available VHDL subset or style guidance for safety critical use, as there is for other languages e.g. Ada [3] or C [15]. The recently released update to IEC 61508 [9] has some general guidance on producing HDL, but this lacks detailed analysis advice or specific rules. Obviously as the use of FPGAs becomes more widespread in the safety domain this issue is becoming more of a concern.

This paper brings together sources of information about safe and correct use of VHDL programming in general, with specific issues associated with FPGAs, to discuss the main areas of concern. This paper doesn't provide a complete solution, but is intended as a starting point from which further guidance can be developed, ideally as a recognised subset. Note that we have assumed the VHDL will be being inspected as part of a safety critical design, i.e. the emphasis is on safety, not security or other types of dependability. This may impact on the structure of the VHDL as will be discussed.

We have used some of the relevant areas of concern cited in [12] as criteria for examining VHDL. In addition we have considered some of the more specialist features of VHDL which make it difficult to analyse, such as highly parallel computation. In each case we discuss how the feature of interest should and could be enforced.

This paper is laid out as follows. Section 2 provides background information and discusses related work, including the draft ISO standard on language vulnerabilities. Section 3 applies the guidance from the ISO draft to VHDL. Section 4 considers additional VHDL areas which are not covered by the ISO draft. Finally, conclusions are presented.

2 Coding Standards and VHDL

This section provides background information on VHDL, certification guidance and other related work. It also discusses practical concerns when producing a complete subset.

2.1 Introduction to VHDL

VHDL [10] was originally developed as a tool for documenting the behaviour of electronic circuits in a strict way. Its development was initiated by the US Department of Defense who were finding hardware was not adequately documented and it was difficult to replace or reproduce components. However, the language then started to be used as a design tool, with simulation and synthesis tools being developed. The first IEEE VHDL language standard was published in 1987, with the most recent revision published in 2009.

It is a strongly typed language which can elegantly represent parallel processing. It is a modular language, similar in visual style to Ada. It differs from traditional software programming languages due to the inclusion of time/clock processing and the facilities for describing low-level circuits. It can be simulated, rather than executed, and these simulations may give different results to actual implementations if VHDL is poorly constructed [14]. This means that simulation cannot, in general, be considered equivalent to software testing.

2.2 Certification Guidance

Two documents which offer advice and guidance on the use of HDL in the safety critical domain are DO-254 "Design Assurance Guidance for Airborne Electronic Hardware" [17] and IEC 61508 "Functional Safety of Electrical /Electronic/ programmable electronic safety-related systems" [9].

DO-254 is from the civil avionics domain, and has extensive guidance on high level hardware design activities such as planning and requirements capture. It also suggests the types of results needed from detailed hardware analysis, but doesn't specify techniques. In addition, there is useful information on environmental concerns, e.g. heat affecting hardware performance. It offers little detail on the detailed design or configuration of hardware using an HDL. It notes that the similarity in appearance between HDL and traditional software "*can mislead one to attempt to use software verification methods on the design representation of HDL*". However, it offers very little in specific guidance for writing and assessing HDL, other than general requirements for traceability, V&V activities, and assessment and qualifications of synthesis tools.

IEC 61508 is a generic standard, used in the process industry and more widely. It has recently been updated to include detailed guidance on HDL design and analysis for both FPGAs and Application Specific Integrated Circuits (ASICs). Part 2 now has a detailed design lifecycle for these, similar in structure to a software lifecycle. This is complemented by detailed background information in part 7, which has some specific suggestions for construction and style of HDL (including VHDL), albeit in the context of ASIC design rather than FPGAs. Suggestions specifically related to HDL level design include:

- Use of structured and modular design - this eases analysis and maintenance
- Restricted use of asynchronous constructs
- Design for testability
- Observation of coding guidelines including:
  - Restrictive use of ambiguous constructs
  - Transparent and easy to use code
  - Use of comments and annotations
  - Defensive code and range checking
  - Limits on module sizes and number of ports to increase readability
  - Avoidance of multi-dimensional arrays, goto type commands, combining signed and unsigned data types
  - Avoidance of redundant logic and feedback loops
  - Avoidance of latches, asynchronous reset
  - Use of `std_logic` and `std_logic_vector` data types for module ports
- Use of tools to check coding guidelines

Whilst this is a useful checklist it lacks details for VHDL, for example in terms of which constructs could be considered ambiguous and what common errors can occur in day to day use.

One UK defence standard, 00-54 “Requirements for Safety Related Electronic Hardware in Defence Equipment” [19], which was used for a number of years but has now been superseded (to allow the use of civil standards) also recommended the use of HDL subsets.

2.3 ISO Draft on Language Vulnerabilities

The development of ISO guidance on programming language vulnerabilities [12] was originally brought about due to concerns in the security domain. It has been broadened to include safety, mission and business critical usage. In each case a system may be susceptible to unanticipated behaviour or even attacks, due to use of poorly defined or poorly constructed code.

Two main areas are covered: general programming language vulnerabilities, and more specific application vulnerabilities which relate to software functionality. The majority of the latter are known security exploits, such as buffer overruns and are not discussed further here.

At the time of writing this paper, 53 separate language vulnerabilities are listed in the draft, each with suggested solutions. Many of these are relevant to VHDL including syntactic measures, such as use of consistent naming conventions, maintenance problems due to macros, and ensuring variables are initialized. Other vulnerabilities are not applicable, for example memory management problems are simply not relevant when programming logic gates for an FPGA. All these areas are explored in more detail in section 3.

It should be noted that the draft of [12] doesn’t currently address object orientation or concurrency in much detail. Obviously the latter is of great importance when using VHDL. Missing areas are addressed in section 4

2.4 Other VHDL Specific Guidance

In [11] Isaac uses experience from developing programmable electronics within NASA in order to recommend the following when using VHDL:

- State machines should have no un-terminated states otherwise there can be inadvertent jumps in the code following a Single Event Upset (SEU).
- State machines should represent states with numbers greater than unity. No single bit flip should cause the state machine to go into an unwanted state.
- The VHDL code should be written to ensure the device powers up to a known state, with values for every pin defined.

No specific work looking at VHDL subsets is available, but there are a number of webpages and reports which document common VHDL traps and errors in the non-critical domain. For example, [13] has a list of common VHDL errors, produced to help students learning the language. Whilst these are basic, they should still be noted within a subset designed to capture a full range of issues. Items include understanding that variables are updated in time for the next statement using them, whereas signals are not updated until the end of a process block, so only the last signal assignment will be acted upon. This could have undetermined side effects if a signal or variable is used for calculation. A further useful observation is that event triggers can be very fast and must be carefully handled during parallel processing to ensure they execute in the order expected.

In [5] an extensive list of suggestions is presented to improve the HDL design process and hence have hardware which behaves as desired and expected. Many of these suggestions, such as robust requirements and specification management and traceability, are standard in the safety critical domain. It also contains suggestions on design, for example to improve efficiency. Interestingly, some of the suggestions are at odds with IEC 61508. For example, it specifically suggests using asynchronous processing of input data, unlike IEC 61508. Another example, is the advice to minimize complex hierarchy and modularity of design. This is to improve the maintainability of the VHDL, and ensure complex port mappings are not required. However, IEC 61508 encourages limiting the module size which may conflict with this. Limiting module size help improve the design's testability, as would synchronous processing. Modularization may be required to separate functionality to avoid common mode failures in the final FPGA layout. From this it can be deduced that VHDL guidance must be carefully considered, weighing up efficiency and optimisation with safety and the ability to statically analyse the code. When safety is a priority, then it should take precedence over ideal design structures.

2.5 Ensuring Subset Completeness

A final VHDL subset must cover all pertinent issues, however these are very wide ranging. The ISO draft divides issues up into two levels, language and application level, which leaves a large number of issues under the one heading of language, but allows the freedom to explore multiple areas. IEC 61508 presents a list of issues with no particular ordering but with some emphasis on the need to support analysis. Between the two sets of guidance there are many orthogonal and overlapping areas of concern including data representation and manipulation, function, style, low-level detail, high level design, integrity level requirements etc. Hence, it is difficult to construct a map or template which would ensure all issues can be rigourously captured. The following three paragraphs discuss possible overlaps and categories using three areas of concern.

Code Construction: This refers to the mechanics of how the code is put together. Some construction rules are restrictive e.g. certain constructs are not allowed, or their use is limited. Some construction rules are pro-active and preventative, for example adding range checking to look for value faults and respond accordingly. Low level construction concerns include Isaac's suggestions for how states should be represented (section 2.4). Some construction rules may be influenced by the underlying hardware.

Presentation: This refers to stylistic improvements to code. At the higher level it refers to design methods for maintainability and readability, and at a lower level refers to the need for comments, consistent naming and layout. It overlaps, and potentially conflicts, with construction in terms of designing to ease analysis.

Language Use: This is at the lowest level of concern, considering analysis mainly on a line by line basis. It includes ambiguities, common syntax errors, keywords, and arithmetic. Some of these areas may overlap with code construction, for example restrictions on keywords alter the overall design approach.

Due to the lack of a comprehensive list of concerns, this paper follows the broad outlines of the ISO draft as the most complete and generic source of information, with extra concerns gleaned from other sources discussed in a separate section. However, the completeness achieved by this approach needs further assessment.

3 Using the ISO guidance

This section looks at how the suggestions in the ISO language vulnerabilities draft are applicable to VHDL, collecting related issues using their layout. In each case some discussion of enforcement or checks is made.

3.1 Problems with the language specification

Many of the early issues discussed in the draft relate to problems with a languages specification. This includes the problem of ambiguities raised in IEC 61508. For example, behaviour many not always be fully defined in terms of operator precedence, and different compilers may produce different results. The ISO draft also discusses deprecated language features, as these may not be fully supported.

Another area is obscure language features that are not well understood.

According to the online VHDL Frequently Asked Questions (FAQ) [16] there can be issues of ambiguity introduced when the same function names, with the same parameters, are used within separate subprograms. A similar issue may occur during type conversion and resolution e.g. if different enumerated arrays have the same names included. Obviously these problems can be easily avoided by ensuring unique names are always used. In addition VHDL allows full name qualification by the developer which could be enforced in a subset.

Sometimes the final FPGA behaviour may depend on the target hardware. Altera notes a VHDL problem with undefined read/write behaviour in a dual clock device in [1]. This is a specific example of general problems which might occur due to asynchronous processing. This type of ambiguity needs to be addressed at a high level of design and cannot easily be enforced via language subsetting.

3.2 Generic issues

There are some issues which are generic to most languages. For example, the use of consistent variable naming conventions can help maintenance and readability of all coding languages. This doesn't directly affect the dependability or functionality but could indirectly if the code is difficult to alter. Consistent naming cannot be enforced statically if names depend on function and content, however a clear policy and independent manual review during static analysis can be used to provide assurance.

Pre-processing directives and templates are found in many programming languages now. A change in one of these can have unexpected side effects wherever they are used within the code, or if unintended code is included in the final product. The `generic` construct in VHDL has recently been greatly widened in functionality so that it can be used for both pre-processing directives and template code [2]. There are two options for a subset. The first is to avoid the use of generics altogether which can be simply enforced by a search for relevant keywords. This approach was taken for the SPARK Ada subset [3]. The second is to allow limited use of generics, with explicit documentation of all expected behaviour. Then every place they are used must be tested and analysed. This is more complex, and more expensive, so the benefits must outweigh the cost.

Namespaces can be emulated in VHDL using libraries. The concern raised in the ISO draft is that identical names within the namespaces can be compiled in, leaving uncertainty as to which items are being used. As discussed in section 3.1, using unique and/or fully qualified names at all times would prevent this issue.

3.3 Issues very pertinent to VHDL

Some of the issues raised in the draft are of extra relevance to VHDL, when compared with traditional programming languages, due to the low level nature of the language. The problem of floating point numbers is raised. These cannot be exactly represented, and are particularly difficult to manipulate on an FPGA. Therefore, inaccuracies result during calculations. One potential solution is to avoid a VHDL/FPGA design solution altogether when manipulating floating point numbers. Alternatively, detailed analysis and modelling to determine whether the level of inaccuracy and approximation is acceptable would be needed.

Another issue of particular relevance is low-level bit representations and binary mathematics. The problem raised by the draft is not so much with the language but with programmers being unfamiliar with this type of data manipulation and hence introducing value errors. It is arguable that a VHDL designer should always be competent in this area, however analysis and testing should be used for additional assurance.

Variable initialisation is important to ensure that correct values are used during startup routines. VHDL allows default values to be set for signals, but this may not necessarily be the initial value depending on how it is driven [8]. An initial value can be set by using the "reset" signal to initialise all other signals within processes, or the designer might assume an undetermined value for the first iteration. As noted in section 2.4 signals will not be immediately updated following an assignment, so care must be taken that the signal is not used until it's ready. A further complication is that signals are not guaranteed to hold their value (due to possible changes in the state of the physical hardware implementing the circuit). Note that a simulation will assume a preserved value, and this is one area where a simulation may give misleading results.

Therefore signals should ideally be assigned, or re-assigned on every iteration of a process. Static analysis can be used to help ensure this guidance is followed, checking all signals named in declarations. An example is shown in Figure 1, where `signala` will always be assigned a value on a clock tick.

```
if rising_edge(clk) then
        if reset = '1' then
                signala <= '0';
        else
                if [some comparison] then
                        signala <='0';
                else
                        signala <='1';
                end if;
        end if;
end if;
```

Figure 1 VHDL example showing initialization and continuous value assignment

```
if signalx = '1' then
        signaly <= signalz;
end if;
```

Figure 2 Example of a latch

Following on from this, use of latches is restricted by IEC 61508. These are produced in code via `if` statements not

completely specified, i.e. all possible input conditions are not covered. And example is shown in Figure 2. The difficulty with latches is that they can cause race conditions with old and new data being compared. They are a particular problem on a FPGAs which often don't have physical latches built in, so will represent them by combinatorial logic, leading to path delays and difficulty in performing timing analysis.

One way to ensure latches are not inferred is to always have a default assignment block, prior to any comparison statements. Note that this is an alternative method for ensuring signals are always assigned a value. Synthesis tools typically produce warnings when they are inferring latches, so the developer should look for these.

Finally, self modifying code is raised as potential concern. Some FPGAs are dynamically reconfigurable during run-time, which is potentially helpful in keeping full functionality after a partial hardware failure on the board [6]. VHDL which manages this process could be interpreted to be self-modifying, as it will be difficult to predict its layout at any given time. At present, dynamic reconfiguration is unlikely to be acceptable within a safety critical design, due to issues such as uncertainties of the final layout, and loss of functionality during reconfiguration. Therefore, this should be avoided altogether.

3.4 Irrelevant features

There are a large number of memory related issues raised within the ISO draft. It should be noted that the use of pointers and dynamic memory allocation is typically either avoided or restricted for safety critical software development. Their use can lead to many problems such as accessing invalid memory addresses, or difficulty anticipating size of memory allocations. However, although VHDL has these features they would not be used when configuring a fixed memory device such as an FPGA. Therefore their use should be prevented in a subset.

Finally, some language features are simply missing from VHDL, for example exception handling, therefore no guidance is required.

4 Additional VHDL concerns

This section discusses some issues missing from the ISO draft which are also of relevance to VHDL.

4.1 Specific Code Level Issues

VHDL allows hints to be passed to the synthesis tool. For example, suggestions for final layout can be made to separate certain areas or to meet timing deadlines. Alternatively, some tools have keywords to prevent optimization of some circuits and signals. These can be extremely important for safety critical design, e.g. to prevent a common mode failure. As keywords may be vendor specific, maintenance of the code may be required when changing to a different vendor.

Guidance on appropriate use of keywords should be provided with particular awareness of why they might be needed. The developer should take consideration of any comments made on synthesis about optimization, for example. A code parsing tool can flag each use of keywords so that they can be statically analysed and tested.

4.2 Design Level Issues

The issue of concurrency (touched upon in 3.1, and discussed at some length in IEC 61508) is extremely pertinent for VHDL and FPGA design. Signals may be updated in parallel, and feedback loops can be created. Race conditions may occur which mean variables are not updated as expected. Concurrency is a huge area of concern, and there are tools available to help assess it. In particular the use of formal specifications, analyses and proofs applied at the design and netlist levels can be used. Due to space restrictions the various methods are not explored in detail here, but they include Esterel [7] and the Communicating Sequential Processes (CSP) language [18]. One issue of note is that the application of formal methods can be time consuming and expensive, therefore their use may only be required for very high integrity VHDL applications (as suggested in IEC 61508, part 2).

Section 3.1. touched on the issue of behaviour depending on the underlying hardware. Another hardware related issue is the effect that poorly designed VHDL can have on the final performance of the FPGA. For example, power consumption can be significantly impacted [4]. Timing can also be affected by inefficient or poorly thought out code. This may be a safety issue, for example if power requirements or timing deadlines are not met. Synthesis tools will accurately assess time performance so related issues can be identified during design, however power problems will be harder to predict.

4.3 Defensive code

One additional issue from IEC 61508 (see section 2.2) is that of defensive code. Defensive code is written to pick up faults or anomalies and respond in a pre-determined way. This means checking that values are within expected ranges, and picking up on incorrect control flow. This is another area made more complex by concurrent execution. For example, it is possible that a signal may have the wrong value when checked, but the correct value by the time it is required after one or more clock cycles.

4.4 Future issues

Object Orientation (OO) is not mentioned in any depth within the ISO draft, however VHDL does not currently have well developed OO features, such as classes, although these are being developed [2]. Once integrated into the language these should be considered.

5 Conclusions

The purpose of this paper was to provide a starting discussion on subset and style guidance for VHDL, particularly when configuring FPGAs for critical applications. The ultimate objective is to encourage the development of a recognised

language subset, comparable with SPARK Ada or MISRA C. A number of different sources have been examined, particularly the application of the ISO Draft guidance on generic language vulnerabilities. From this a number of significant issues were identified, including variable and signal initialisation and bit representations.

In addition we have considered additional VHDL specific areas not covered by the ISO draft, such as concurrency and close coupling to the underlying hardware.

Whilst this paper has attempted to put together a comprehensive list of concerns, by looking at multiple resources, the areas discussed are wide ranging and occasionally conflicting. Further work is needed to assess the completeness, correctness and coverage of the listed areas, ideally using expertise from safety critical VHDL practitioners, and lessons learned from the creation of other subsets such as SPARK.

Following on from this paper a more formal list of rules will be developed, along with detailed methods for their enforcement. Also some areas, such as analysis of concurrency and acceptable levels of asynchronous behaviour, require deeper research and development. The guidance will also needed to be updated as new language features arrive, such as OO.

Acknowledgements

Philippa Conmy is a research associate with the Software Systems Engineering Initiative (SSEI), which is funded by the U.K. Ministry of Defence. She would like to thank them for their support and funding. She has worked in the area of software safety for over 10 years, looking at FPGAs, Integrated Modular Systems and Operating Systems.

Clive Pygott is an independent computing safety consultant. Previously he worked for a UK Ministry of Defence research organisation for some 30 years. He was co-author of two standards on the use of digital electronics in safety-critical systems and the designer of a micro-processor for safety specific applications.

Iain Bate is a Lecturer in real-time systems. His research includes scheduling and timing analysis, design and analysis of safety critical systems, and engineering of complex systems of systems including sensornets.

References

[1] Altera, “Altera QuartusII Handbook Version 9.1 Volume 1: Design and Synthesis”, Downloaded from http://www.altera.com/literature/lit-index.html June 2010.

[2] P. J Ashenden and J. Lewis, “VHDL 2008: Just the New Stuff (Systems on Silicon), Morgan Kaufman, ISBN-13 978-0123742490, 2007.

[3] J. Barnes, “High Integrity Software: The SPARK Approach to Safety and Security”, Addison Wesley, ISBN-13: 978-0321136169, March 2003.

[4] D. Chen, J. Cong, Y. Fan, “Low-Power High-Level Synthesis for FPGA Architectures”, Proceedings of the 2003 International Symposium on Low power electronics and design, pp. 134-139, 2003.

[5] Design Abstraction Ltd., “Common HDL Design Errors”, Downloaded from http://pldworld.info/_hdl/2/_ref/-designabstraction.co.uk/Articles/CommonHDLErrors.PDF, Accessed June 2010.

[6] J. M. Emmert, C. E. Stroud, B. Skaggs, and M. Abramovici, "Dynamic Fault Tolerance in FPGAs via Partial Reconfiguration," IEEE Symposium on Field-Programmable Custom Computing Machines. pp. 165-174, 2000.

[7] J. Hammarberg, and S. Nadjm-Tehrani, “Development of Safety-Critical Reconfigurable Hardware with Esterel,” *Electronic Notes in Theoretical Computer Science*, **vol**. 80, pp. 219-234, 2003.

[8] R. E. Harr, A. G. Stanculescu, “Applications of VHDL to circuit design”, Springer, ISBN-13 978-0792391531, 1991.

[9] IEC, “Functional Safety of electrical/electronic/programmable electronic safety-related systems (IEC 61508)”, 2010.

[10] IEEE, “1076-2008 IEEE Standard VHDL Language Reference Manual”, IEEE, Jan 2009.

[11] T. A. Isaac, "Firmware in Safety Critical Subsystems," International System Safety Conference. pp. 469-478, 2004.

[12] ISO, "Information Technology — Programming Languages — Guidance to Avoiding Vulnerabilities in Programming Languages through Language Selection and Use – DRAFT, Available at - http://aitc.aitcnet.org/isai/, 2009.

[13] R.Manion, “Common Mistakes in VHDL”, http://www.cs.ucr.edu/cs122a/cs122a_05fal/ddr/vhdl.html, Accessed April 2010.

[14] D Mills, C.E. Cummings, “RTL Coding Styles That Yield Simulation and Synthesis Mismatches”, Proceedings of SNUG 99, 1999.

[15] Motor Industry Software Reliability Association, “Guidelines for the Use of the C Language in Critical Systems”, MIRA, ISBN 0 9524156 2 3 (paperback, October 2004.

[16] E. Naroska (ed.), “VHDL Frequently Asked Questions”, http://www.vhdl.org/comp.lang.vhdl/FAQ1.html Accessed June 2010

[17] RTCA/EUROCAE, DO-254 "Design Assurance Guidance for Airborne Electronic Hardware", RTCA, 2000.

[18] J Willis, Z. Li, T. Lin, “Use of embedded scheduling to compile VHDL for effective parallel simulation, Proceedings of the conference on European design automation, pp 400-405, 1995.

[19] U.K. Ministry of Defence, “Requirements for Safety Related Electronic Hardware in Defence Equipment, 00-54 Part 2: Guidance”, March 1999.