FORMAL VERIFICATION SUPPORT FOR ELLA

IED 4/1/1357

Comparison of ELLA and VHDL*

Alan Williams

Department of Computer Science
University of Manchester†
Oxford Road, Manchester, UK.

January 1991

†The University of Manchester acknowledges support from the SERC via research grant GR/F 38174.

Abstract

This report compares the VHDL and ELLA hardware description languages. VHDL is the standard language for the US DoD and ELLA is likely to become its equivalent for the UK MoD. Aspects of each language which may affect its use during hardware verification are considered. An RS flip flop is taken as an example design, with descriptions constructed using both languages. Developments in research which use either language during verification are reviewed. It is found that the two languages have many similarities. VHDL has the **ASSERT** and **ATTRIBUTE** statements which are useful during simulation. ELLA is designed to be suitable for synthesis, having a core language which has been extended to produce high-level constructs. Both languages support different design styles so that clear descriptions for general hardware systems can be produced. These features are useful during design-checking via simulation comparison. However, neither language is yet equipped with a formal semantic definition, which would then enable formal verification techniques to be applied.

1 Introduction

This report compares two hardware description languages that have been developed during the last decade. VHDL (VHSIC Hardware Description Language) was started in 1982, viewed as a basic requirement for the US DoD VHSIC (Very High Speed Integrated Circuit) Program. Development of ELLA began slightly earlier in 1978 at the UK RSRE. The report will consider aspects of each language which may affect its suitability for use in future design environments involving hardware verification. Developments in research which use either language during verification will be noted. A knowledge of the two languages may assist the reader but is not a prerequisite for this report.

There have been many other HDLs developed before and during this period, but the two languages considered here are notable because they attempt to provide a standard description medium for general hardware design, for systems procured by the appropriate defence organisations. VHDL has been adopted as an IEEE standard (1987), and the DoD requires all new ASICs to be provided with a VHDL description. The UK MoD is currently considering ELLA to become its preferred ASIC hardware description language. There is therefore a strong possibility that either or both languages will be used extensively in industry in the future.

Against this background, some difficulty will be encountered during comparison. Much of the available literature for each language is understandably promotional (this report is not necessarily exempt from this), because the authors have a vested interest in the product, either because they have assisted in its development, or they are using it for design. Even if a designer has used both products, it is difficult to obtain unbiassed opinions about the relative merits of each.

In §2 a brief history and outline of each language will be given. Many features are found to be common to both languages. The two languages are illustrated in §3 using a small example design description of an RS flip flop. The example is too basic to exercise a wide range of language features, or to give a true test of the ease of expressibility. However, it shows the basic components of each language. It is interesting that the two descriptions do have many similarities. Simulation results are also provided, obtained using the Vantage simulator for VHDL, and the standard simulator for ELLA. Results appear to be identical, providing that certain assumptions concerning format and timing are made.

Section 4 reviews research work undertaken, that attempts to use the hardware description language as a basis for verification of hardware designs. Language features which are exploited for this, or which maybe problematic, are considered. A common technique is to compare simulation results obtained from different design representations. For VHDL, attempts are made in [1] to translate VHDL descriptions into HOL statements, so that theorem-proving tools can be applied. In [2] ELLA descriptions are converted into state machine representations for verification using temporal logic. However, all methods currently lack the foundation that would be provided by a formal semantics for each language.

Finally, §5 contains concluding remarks, and recommendations for future work.

2 History and Overview of ELLA and VHDL

Although the origins of the two languages are slightly different, they are similar in many respects in that they offer a strongly-typed functional style of description for hardware. They support design modularity and offer library facilities. A range of description methodologies is also supported. The languages provide a range of hardware constructs, and facilities for modelling time. VHDL has become the preferred medium for system description for the US DoD, and ELLA is likely to become the preferred medium for the UK MoD. However, there are important differences in the features of the two languages which may affect both their ease and power of expression, and their ability to support hardware verification.

2.1 ELLA

Work on the development of ELLA was started in 1978 at the RSRE, the aim being to produce a language for circuit design (as opposed to documentation). A core language was the result. This has since been expanded to include new features deemed useful by industrial ELLA-users, enabling higher-level abstract design descriptions to be produced.

The aim behind the design of ELLA is to permit maximum generality in design, and to provide a design transformation route from behaviour to structure, with all constructs being hardware-implementable [3]. This will facilitate automatic computer-aided synthesis and support different design methodologies. It therefore permits different description methods (structural, behavioural and procedural) to be mixed freely, subject to the overriding condition that all ELLA code must have a hardware representation. ELLA enables designs to be described at various levels, from system architecture down to gate level.

ELLA is equipped with an integrated design system, with language and compiler accessed via the ELLA Application Support Environment (EASE), a simulator, and the ELLANET programming interface [4]. A design can be partitioned and integrated within the EASE, using the library management facility; the design can be divided into suitable units called *Contexts*. Within EASE, an ELLA context will contain **FUNCTION** *headers*, with *bodies* (unless imported), **TYPE** and **IMPORT** declarations; it may also contain **INT**, **CONST** and **MAC**ro declarations. A single function interface may be supplied with several alternative function bodies, using *multiple* contexts.

The original core ELLA has been expanded to include higher level constructs, many of which can be translated to their functionally equivalent core elements. A very simple model of time is used (although it is hoped to include basic forms of *retiming*, currently being developed in a separate research project, in future versions of ELLA).

Leaf operations include **CASE**, **ARITH**, **RAM** and **DELAY** statements. Enumerated types are user-defined, with integers and element lists allowed. Function-types provide a mechanism for bidirectionality, and the latest version of ELLA accommodates floating point numbers.

Further details of ELLA can be found in [4].

2.2 VHDL

The $20million VHDL language Project was started in 1982 because of a need identified by the US DoD VHSIC Program for a standard medium in which to describe and document hardware designs. This was seen to be essential for the procurement and long term use of hardware systems by the DoD, so that design data could be transferred between different organisations involved in a product development, including system fabrication, maintenance and second-sourcing. One requirement of the resulting language which made a fresh start necessary, instead of being able to use an existing product, was that current hardware description languages were not seen to be general enough, their facilities influenced by their area of application. VHDL has been designed with the aim of being universal, so that it includes all the language features believed to be necessary so that any general hardware system can be expressed efficiently and with clarity (see [5]). VHDL must therefore be able to express all aspects of design, from system architecture level, through to layout and including technology-dependent factors.

VHDL has now been adopted as an IEEE Standard [6], and as the standard hardware description language preferred by the US DoD; all new designs considered by the DoD must now have description given in VHDL. They should have a high-level behavioural description, a low-level structural description, and include a suitable test harness. This DoD requirement has clearly influenced the design of VHDL. VHDL also has been influenced by the style of the Ada programming language [7].

Each VHDL description contains three main parts:

1. the **ENTITY** section, describing the interface for use by the calling environment, its inputs and outputs.
2. the **ARCHITECTURE** section, of which there may be several for any particular entity, which describes the body of an entity structurally, behaviourally or procedurally.
3. the **CONFIGURATION** section which defines the particular architecture to be used for an entity by its calling environment.

These correspond to the ELLA **FUNCTION** *header*, **FUNCTION** *body*, and **IMPORT** statements. Note that the main difference here between VHDL and ELLA is that in VHDL the structural, behavioural and procedural design descriptions must be separate, whereas in ELLA they can be freely mixed.

In addition to library facilities which are similar to those included with ELLA, VHDL is equipped with the package facility, used in Ada; collections of types and functions, such as Booleans, can be defined as a whole so that new datatypes can be created.

VHDL is designed so that hardware descriptions can be simulated efficiently; this is aided by including predefined types and functions. Other features which may facilitate simulation, especially during verification by simulation comparison, are **ATTRIBUTE** and **ASSERT**.

Predefined **ATTRIBUTE**s may access particular properties of a signal. For example, **S'STABLE(10ns)** evaluates to true if **S** has remained unchanged for at least 10nS. An attribute can also be user-defined, allowing the designer to include syntactic information in a design, which will have no meaning within VHDL itself but which may assist when a description is processed by external design tools.

Assertion statements are evaluated during design simulation. If they evaluate to false then some action may be taken, with varying severity: perhaps printing an error message describing what has gone wrong and where, or aborting the simulation. The assertion capabilities of VHDL will be extended with the VAL system (§4.1.1), and will be exploited in §4.1.2 to enable VHDL description to be translated into HOL statements, for use with the theorem-proving tools developed for HOL.

Having been publically released, a number of vendors have developed VHDL design environments, compilers and simulators, conforming to the language reference manual. For example, the Intermetrics toolset contains an analyser or compiler operating on the design library and a reverse analyser. There is also a design simplifier and full-specification simulator. Vantage have produced a complete VHDL design environment, with menu-driven graphics interface, incremental compiler, simulator, and schematic capture. Sperry on the other hand have produced an interface so that their own MIXSIM simulator can be used with VHDL. Some vendors are therefore developing VHDL tools by producing interfaces for their existing tools, possibly only supporting a subset of the complete language. Other vendors are producing dedicated VHDL support tools from scratch, which obviously takes longer, but has the advantage that the complete language is supported.

Further details of VHDL can be found in [8], [9] and [10].

3 Design Example: RS Flip Flop

An example of a circuit description for an RS flip flop is now given using both VHDL and ELLA. The example is small, having the advantage that each description is complete and can be simulated. It shows the basic format of each language, and the methods used to create a modular design description. However the full range of language features is not exercised in each language and major differences between them are not revealed here.

The complete VHDL description and simulation results are taken from [11], where the Vantage VHDL simulator is used, employing the VHDL **STANDARD** *package*. The ELLA description has been written to resemble the VHDL description as far as possible, so that equivalent parts of each text can be compared readily.

Figure 1 shows the schematic diagram from [11] used for the RS flip flop; only one output (**Q**) is defined. The node and component labels are utilised in the following circuit descriptions.

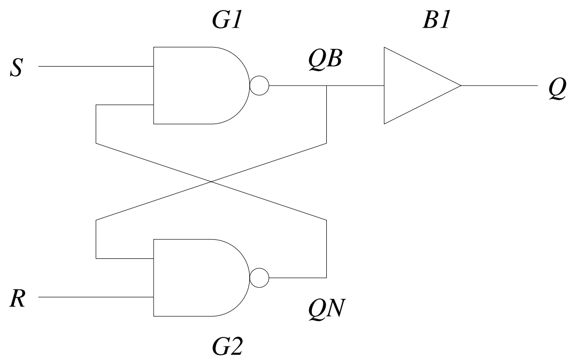

Figure 1: RS Flip Flop

The example illustrates the underlying functional and modular aspects of each language, outlined above in §2. Each function contains an interface description (**ENTITY** in VHDL, **FUNCTION** *header* in ELLA) giving the inputs and outputs with their datatypes, and then a body describing the structure or behaviour of the function (**ARCHITECTURE** in VHDL, **FUNCTION** *body* in ELLA). The third component is for design management, giving the location of each function body in the design library (**CONFIGURATION** in VHDL, **IMPORT** in ELLA). A function body must be declared an **EXPORT** from its context in ELLA before it can be used as an import to another context. A large design is thus subdivided into individual files and compiled separately. For the example, the RS flip flop function **RSFF** uses externally declared functions **NAND_GATE** and **BUF_GATE**, which incorporate delays. In VHDL the basic delay-free nand-gate and Boolean type are predefined; in ELLA they are user-defined. Function interconnections are made using **PORT MAP** in VHDL and **JOIN** in ELLA.

Figure 2 contains the ELLA-text to describe the RS flip flop and its components. It is compiled into a library with two separate contexts (*rs_gates* and *rsff*). The *rsff* context must import design elements from *rs_gates* before it can make use of them (of course, the elements must also be declared as **EXPORT**s in context *rs_gates*). It is not possible however to import type-declarations in ELLA; each context must contain a declaration of any type that is used within it. Note that any function compiled within a single context is available to any other function within that context without declaring it as an import.

The labelled statements in the ELLA-text have corresponding parts in the VHDL description. For example, the **FUNCTION** *header* for **BUF_GATE** labelled [A] in the ELLA-text corresponds to the *buf_gate* **ENTITY** in the VHDL description, which is also labelled [A]. The delay-less Nand gate and *bit* type declaration from ELLA are not included in the VHDL description.

Figure 3 contains the VHDL description of the RS flip flop. Note that *nand* and **BIT** are predefined. Different function bodies can be called up (for example, structural, behavioural or procedural, or even different versions of one of these) by including several **ARCHITECTURE** sections for each entity. The desired function body is selected in the **CONFIGURATION** statement; for example the behavioural architecture for *nand_gate* will be used in entity *rsff*. An equivalent construction is possible in ELLA by using multiple contexts and selecting the required function by modifying the relevant **IMPORTS** statements.

```
#****************************************#
#              Context rs_gates          #
#****************************************#

# Type Declarations:                     #
TYPE bit = NEW (t | f).

# 2-input delay-less Nand gate:          #
FN NAND = (bit: input1 input2) -> bit:
CASE (input1, input2)
OF (t, t): f,
   (f, bit)|(bit, f): t
ESAC.

# Delay buffer:                          #
FN BUF_GATE = (bit) -> bit:                   # [A] #
  DELAY(f,1).                                 # [B] #

# 2-input Nand gate with delay:          #
FN NAND_GATE = (bit: input1 input2) -> bit:   # [C] #
   BUF_GATE ( input1 NAND input2 ).           # [D] #

#****************************************#
#              Context rsff              #
#****************************************#

# Type Declarations:                     #
TYPE bit = NEW (t | f).

# Import Declarations:                          [E] #
FN NAND_GATE = ([2]bit) -> bit: IMPORT.
FN BUF_GATE = (bit) -> bit: IMPORT.

IMPORTS rs_gates: NAND_GATE BUF_GATE.         # [F] #

# RS Flip Flop Description:              #
FN RSFF = (bit: r s) -> bit:                  # [G] #
BEGIN
   MAKE BUF_GATE: q,                          # [E] #
        NAND_GATE: qb qn.
   JOIN (r, qb) -> qn,                        # [H] #
        (s, qn) -> qb,
        qb      -> q.
   OUTPUT q
END.

#****************************************#
```

Figure 2: ELLA Description of RS Flip Flop

```
-- **** nand gate model ****
ENTITY nand_gate IS                              -- [C]
       PORT (a,b: IN BIT; y: OUT BIT);           -- external ports
END nand_gate;

ARCHITECTURE behavioural OF nand_gate IS         -- [D]
BEGIN
       y <= a NAND b AFTER 1 ns;                 -- internal behaviour
END behavioural;

-- **** buffer model ****
ENTITY buf_gate IS                               -- [A]
       PORT (a: IN BIT; y: OUT BIT);             -- external ports
END nand_gate;

ARCHITECTURE behavioural OF buf_gate IS          -- [B]
BEGIN
       y <= a AFTER 1 ns;                        -- internal behaviour
END behavioural;

-- **** rs flip flop model ****
ENTITY rsff IS                                   -- [G]
       PORT (r,s: IN BIT; q: OUT BIT);           -- external ports
END rsff;

-- internal behaviour:
ARCHITECTURE structural OF rsff IS
     -- component types to use:                  -- [E]
     COMPONENT nand_gate
        PORT (a,b: IN BIT; y: OUT BIT); END COMPONENT;
     COMPONENT buf_gate
        PORT (a: IN BIT; y: OUT BIT); END COMPONENT;
     SIGNAL qn, qb: BIT;                         -- internal signals
BEGIN                                            -- [H]
     -- buffer named b1, connected to signals qb and q
     b1: buf_gate PORT MAP(qb,q);
     -- gate named g1 connected to signals s, qn and qb
     g1: nand_gate PORT MAP(s,qn,qb);
     -- gate named g2 connected to signals qb, r, and qn
     g2: nand_gate PORT MAP(qb,r,qn);
END structural;

-- design management/configuration:
CONFIGURATION parts OF rsff IS                   -- [F]
FOR structural
    -- use behavioural architecture for gates g1 and g2:
    FOR g1, g2: nand_gate
         USE ENTITY work.nand_gate(behavioural); END FOR;
    -- use behavioural architecture for buffer b1:
    FOR b1: buf_gate
         USE ENTITY work.buf_gate(behavioural); END FOR;
END FOR;
END parts;
```

Figure 3: VHDL Description of RS Flip Flop

Having compiled the circuit descriptions, they can then be simulated. In [11], a stimulus function and test harness are declared in VHDL itself, so that the simulation input is described using VHDL in exactly the same way as the circuit. This part of the design is one of the requirements of the DoD for every VHDL description submitted. The resulting structure has no inputs or outputs. The simulator is then used to monitor desired node values. This method of simulation has the advantage of type-checking the inputs and outputs during compilation instead of relying on the simulation phase. Results from the VHDL simulation are taken from [11] and are shown in Figure 4.

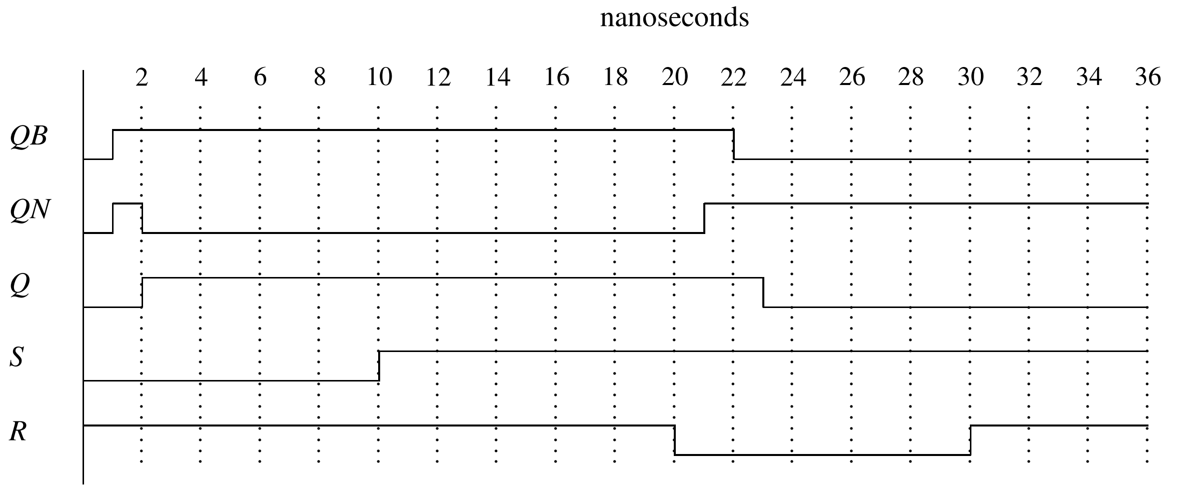

Figure 4: Simulation Results from VHDL

In ELLA, it is possible to construct a test harness and stimulus function in exactly the same manner. However, for this example, an external inputs file was provided for the ELLA simulator. Results are shown in Figure 5. Note that the ELLA simulator originally produced results in the form of strings of ‘t’ or ‘f’ values for each node. These have been manually converted into a timing diagram form, so that both the VHDL and ELLA results are presented in the same way for easy comparison. It is quite possible that this interpretation and therefore the translation are wrong.

The two sets of results are identical during the simulation period, with the test vectors provided and the interpretation given to the output from the ELLA simulation. It has therefore been shown that the VHDL and ELLA descriptions of the RS flip flop have similar behaviour, taking account of the limited tests that have been performed. It is assumed that since the results *look* identical then they are identical. Note that the ELLA description includes delays in terms of arbitrary *time-units*, the length of which is user-defined. However the VHDL description declares that the time-steps are in *nanoseconds*. If it is assumed here that the ELLA time-unit has also been defined as 1nS by the user, then the two simulation results are indeed identical. This of course may not always be the case, with the time-unit set to a different length at another point in the design; the simulation results would then not be the same. It is quite possible that a similar checking strategy would be used for design-checking by simulation comparison, with similar dangers, if the two circuit descriptions were written in the same language.

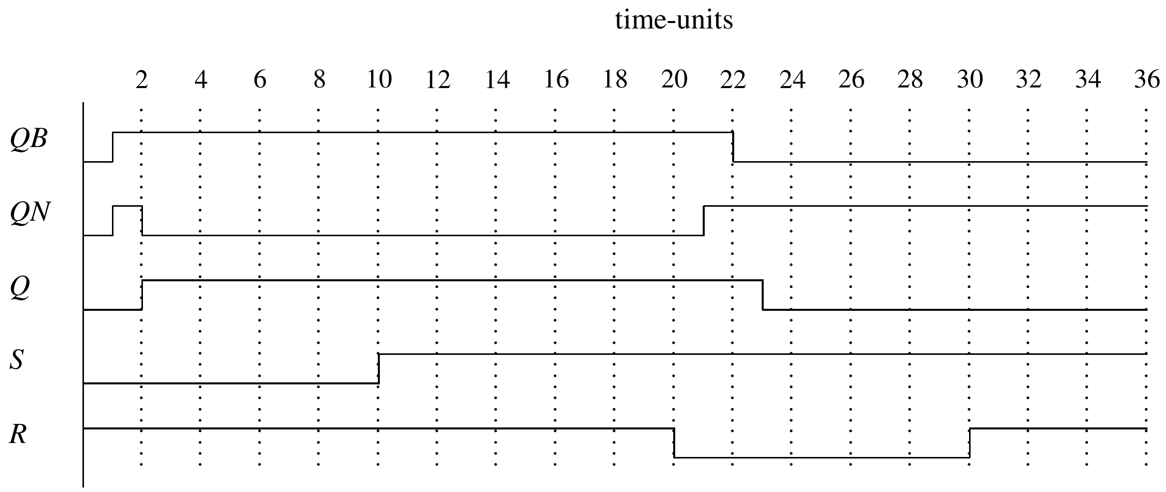

Figure 5: Simulation Results from ELLA

4 Verification and Hardware Description Languages

In [12], it was stated that hardware verification covers a range of methods, including informal design checking or testing, usually by some form of simulation comparison, and formal verification techniques using appropriate mathematical tools. Both of these methods have a role to play during design; informal techniques can be used during initial user-specification capture or when a particular stage in design development is incomplete – the designer may know that the design is carrying errors, but simply wishes to gain confidence and familiarity with the design behaviour so far. Formal techniques may be used at a final stage, because they are likely to be time-consuming and therefore costly. If good trace-back facilities are provided, then they may be applicable earlier in the design cycle, highlighting any remaining problem areas.

A hardware description language is unlikely to provide directly any form of verification itself, but by the design facilities that it offers it may assist or hinder any verification effort. For example, if it is difficult to produce easily understandable high-level design descriptions, then it will be more difficult to verify informally the behaviour of the design against the (informal) user-requirements. Again, if the language contains loosely defined elements then there will clearly be problems during the construction of mathematical models or formal semantics for the language.

This section reviews features of VHDL and ELLA which may enhance or diminish their amenability to hardware verification support. Current developments in research, for applying verification techniques to the languages will be described.

Neither language has yet been provided with a formal semantics, as far as it is known, although several research groups are in the process of developing formal semantic definitions. At present then, any claim for formal verification involving either language is unlikely to be developed on completely firm foundations. A some stage, an informal interpretation of the meaning of a design representation will be encountered. The scope for using formal verification techniques is therefore limited.

The ease with which formal semantics may be added to each language should be considered. VHDL has the advantage of a standard language reference manual, but is big and allows user-defined features, such as attributes, which have no meaning within the language. ELLA has been developed around a **CORE** language, with translations from more complex high-level constructs into the core statements. This may well facilitate development of a formal semantics, if the translation mechanism can be exploited. Problems may be caused in both languages by the precise meaning of delay and timing statements during simulation. For example, a gate in VHDL may be declared to have zero delay. However, during simulation a non-zero *delta-time* delay will be inserted into the design model. This will lead to difficulties in deciding whether two events take place *precisely* at the same time or not.

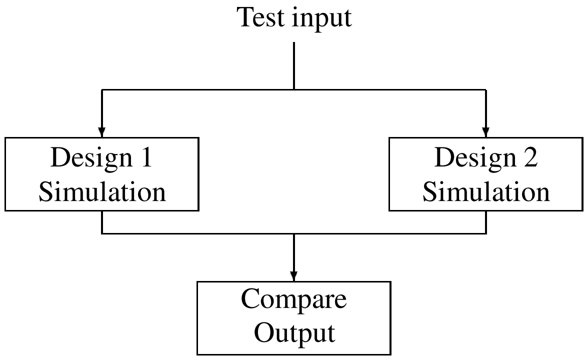

Figure 6: Verification using Simulation Comparison

Figure 6 shows a typical simulation comparison arrangement, for verifying or checking that the behaviour of two different implementations of a design is the same. Simulation results may also be compared with those produced by high level programming languages on an ad hoc basis. Details and limitations of methods used by industrial sites using ELLA appear in [12]. For example, the reliability of results from simulation comparison depends on the selection of test vectors used and the accuracy of simulation.

Both VHDL and ELLA have features which will help simulation comparison. However, although the VHDL simulator algorithm is described in the language reference manual [6], there is no guarantee that different implementations of the simulator will produce the same results for a particular design.

4.1 VHDL and Verification

VHDL has been developed for efficient design simulation, and is equipped with a number of features which will assist during design checking by simulation. The **ATTRIBUTE** statement allows a designer to access particular characteristics of design elements, such as the activity of signals. The **ASSERT** statement enables the designer to set constraints and then to monitor these during simulation; if they are violated then a user-defined report is output. The two verification systems described in the following sections are largely based on these particular VHDL features.

4.1.1 Verification of VHDL Designs using VAL

In the VAL (VHDL Annotation Language) system [13], a designer can add new language constructs to VHDL descriptions in the form of comments. These express the desired behaviour of design sections, which is to be compared with different implementations. For example, Figure 7 shows a VAL construct describing the desired behaviour of an AND-GATE in the design entity (taken from [13]). This annotation declares that the output *c* of the AND-gate entity must be the logical **AND** of its two inputs. Each **ARCHITECTURE** for AND-GATE should conform to this behaviour. It would not be possible to produce such an assertion, using the standard VHDL **ASSERT** statement, because in VHDL, outputs of an entity cannot be monitored from within the entity itself.

```
ENTITY AND-GATE is
   PORT (a, b: IN BIT;
         c: OUT BIT)
--| behaviour
--|    assert ((a AND b) = c)
--|      severity FAILURE
--|      report ``Error in AND-GATE'';
--| end behaviour
end AND.
```

Figure 7: VAL Annotation of VHDL ENTITY

Other examples of new constructs available in VAL include the **FINALLY ASSERT** statement, which delays evaluation of the assertion until the last *delta* timestep within a single simulation step. VAL describes timing behaviour using *relative time*; for example, **S[-5]** gives the value of **S** five time steps ago.

Having been inserted into a design description, VAL annotation statements are then translated into equivalent standard VHDL using a translation program. The result is a self-checking VHDL description, which can be processed using the standard VHDL analysers and simulators. Without altering the basic language, VAL has thus extended the expressive capability of the VHDL assertion mechanism, which will facilitate design-checking using simulation comparison.

4.1.2 VHDL and HOL

In [1] an initial attempt is made to use the theorem-proving tools of HOL (Higher Order Logic) [14] to verify VHDL designs. A two-stage approach is adopted so that the resulting constructs will also be of use during design simulation, and for checking that different simulator implementations are operating correctly. This strategy makes extensive use of the VHDL **ASSERT** and **ATTRIBUTE** constructs. The final output is a set of HOL specifications of the design, which can then be used in the HOL theorem-proving environment.

First, signal assignment statements in VHDL (such as '`y <= a NAND b AFTER 1 ns`' from architecture [B] in Figure 3) are translated into equivalent **ASSERT** statements, which should always evaluate to TRUE during a simulation. If this is not the case, then either the simulator is not operating as intended, or there is a fault with the interpretation of the meaning of the signal assignments, and therefore with the translation program. Now the **ASSERT** statements can be translated into HOL for processing by a theorem-prover. Either the HOL description of the circuit can be checked for satisfaction of a separately constructed HOL specification of the design, or two automatically generated HOL descriptions of different implementations of the same circuit can be compared. It is then necessary to develop the appropriate proof strategies for use within HOL.

A translation system could easily be designed that produced HOL statements directly from the VHDL description, omitting output of VHDL **ASSERT** statements. This is not done because the system in its present form can be used as an assistant for simulation, and checking for the correct operation of different VHDL simulator implementations.

The translation program runs into difficulties in interpreting precisely the VHDL simulator timing constructs, namely the use of *delta-time*. There is consequently a need for a broader range of **ASSERT** constructs, such as those provided by the VAL system (§4.1.1). The most important new constructs are **FINALLY ASSERT** and the ability to monitor both the inputs and outputs to an entity.

In [1] it is stated that the **ASSERT** generator program can accept most of the sequential constructs in VHDL. However, it is not clear yet whether the process can be extended to accept *all* valid VHDL descriptions. The following questions are raised by this approach:

1. is it possible to generate an equivalent **ASSERT** for every valid VHDL statement?

2. can all **ASSERT** generated by (1) be translated into HOL statements?

3. can efficient proof strategies be developed for HOL so that satisfaction and equivalence of HOL statements possibly all generated in (2) can be ascertained?

Most of the problems encountered in [1] arise of course from the lack of a formal semantic definition for VHDL; for example it is stated in [1] that "in the absence of a completely formal specification of the language, any translations from VHDL to HOL must be provisional", and that "the complexity of the VHDL timing model for signals needs formal specification". It is hoped however, that this translation technique may be used "to demonstrate the semantics of VHDL directly in VHDL itself".

4.2 ELLA and Design Verification

ELLA has been developed for circuit design, with the aim of supporting automatic synthesis from high-level behavioural descriptions to low-level structural descriptions. Consequently, it has several features which will assist during simulation comparison. It has a simple time model, which may be beneficial when developing formal semantics, but which may limit its expressiveness. Translation of complex constructs into a core language may also be of assistance when developing semantics. During pre-simulation processing, ELLA warns the designer of any zero-delay loops within a description.

Several attempts are being made to use ELLA within a formal verification framework. In [2] ELLA-text is translated into a state machine representation so that it can be verified using the temporal logic model-checking tool MCB[15]. The translation process is semi-automatic, but is not based on a formal semantics.

A method is described in [16] for verifying ELLA designs using HOL. A subset of ELLA is *semantically embedded* into higher order logic. Finally, as part of this IED project, an SOS-style semantics is being developed for ELLA [17]. As with VHDL, one newly developed feature in ELLA which may cause problems when developing semantics, is the ability to include functions written in an 'alien' language.

5 Conclusions

The VHDL and ELLA hardware description languages have been compared in this report, concentrating on features which make them suitable for supporting hardware verification. Although there are important differences between the languages, they are similar in many respects and include some desirable features for describing general hardware systems.

Most notably, VHDL provides the **ASSERT** and **ATTRIBUTE** statements, which are found to be very useful during simulation. Several systems for providing verification support for VHDL are largely based on them. However, it may not be most suitable when translating a design between a structural, behavioural or procedural form, because these different expression styles must be kept separate in VHDL. In ELLA they can be mixed and so a translation route exists between high-level and low-level design descriptions. ELLA also translates complex constructions into its core language.

However neither language is currently suitable for use in formal verification because they lack a formal semantics. This is particularly problematic when considering timing aspects, during simulation.

It may be worthwhile to consider whether constructs similar to the assertion/attribute statements in VHDL could be introduced into ELLA[1]. However, much more important is the development of a formal semantics.

Since both VHDL and ELLA are likely to be extensively used during procurements by their respective defence organisations, then an ELLA-VHDL translation system would be useful. The development of such a system would be easier with the introduction of formal semantics for both languages.

The author would like to thank Mike Hill and John Morison (RSRE) for their useful comments on this report, and Brian Monahan (Manchester) for his assistance.

[1]Note that ELLA currently supports a general text attribute system, and static assertions, which are evaluated during design assembly.

References

[1] J Van Tassel and David Hemmendinger. Toward Formal Verification of VHDL Specifications. In *Applied Formal Methods For Correct VLSI Design*. IMEC-IFIP Workshop, 1989.

[2] A Beck. Using Temporal Logic to Verify Properties of Circuit Designs Described in ELLA. Master's thesis, University of Oxford, 1987.

[3] C O Newton and M G Hill. Comparison of Implementations of Real Arithmetic in ELLA and VHDL. In *The European Design Automation Conference, Glasgow*. IEEE, 1990.

[4] Praxis Systems PLC. *ELLA Manuals, Issue 3.0*, 1987.

[5] J H Aylor, R Waxman, and C Scarratt. VHDL - Feature Description and Analysis. *IEEE Design and Test of Computers*, 3(2), April 1986.

[6] *IEEE Standard Language Reference Manual 1076*, 1987.

[7] R D A Costa, M Alexandre, G Imken, and B Real. The Role of VHDL in the MCC CAD System. In *25th Design Automation Conference*. ACM/IEEE, 1988.

[8] R Lipsett, E Marschner, and M Shahdad. VHDL - The Language. *IEEE Design and Test of Computers*, 3(2), April 1986.

[9] M Shahdad. An Overview of VHDL Language and Technology. In *23rd Design Automation Conference*. ACM/IEEE, 1986.

[10] J D Nash and L F Saunders. VHDL Critique. *IEEE Design and Test of Computers*, 3(2), April 1986.

[11] D R Coelho. *The VHDL Handbook*. Kluwar, 1990.

[12] A Williams. Verification Requirements Analysis. Technical Report Man/2.1a, Department of Computer Science, University of Manchester, September 1990.

[13] L M Augustin, B A Gennart, Y Huh, D C Luckham, and A G Stanculescu. Verification of VHDL Designs Using VAL. In *25th Design Automation Conference*. ACM/IEEE, 1988.

[14] M Gordon. HOL: A Machine Oriented Formulation of Higher Order Logic. Technical Report 68, University of Cambridge Computer Laboratory, July 1985.

[15] M C Browne, E M Clarke, D L Dill, and B Mishra. Automatic Verification of Sequential Circuits Using Temporal Logic. *IEEE Trans. Computing*, C-35(12), 1986.

[16] R Boulton, M Gordon, J Herbert, and J Van Tassel. The HOL Verification of ELLA Designs. Technical report, University of Cambridge Computer Laboratory, 1990.

[17] H Barringer, M Hill, and B Monahan. Semantic Framework. Technical Report Man/1.1a, Department of Computer Science, University of Manchester, September 1990.