FORMAL VERIFICATION SUPPORT FOR ELLA

IED 4/1/1357

Verification Requirements Analysis*

Alan Williams

University of Manchester†
Oxford Road, Manchester, UK.

September 1990

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

Abstract

This report examines current industrial hardware design processes, which are based on the ELLA hardware description language. It establishes a set of Designer Requirements for future design tools and support, studying the role that ELLA plays. The nature of hardware verification, involving the use of both informal and formal methods, is examined. A set of Verification Requirements is formed, followed by a discussion of methods needed to fulfil them.

1 Introduction

What is required of a design environment which caters for *hardware verification*? How would a design system carry out the tasks required? What would be the best way for a user to interact with the system? This report examines these issues, identifies those areas which should be considered in more detail, and forms a set of Verification Requirements. It will also provide background information and suggest possible solutions. The work is based on the results from three meetings with industrial design engineers, where ELLA is currently used in their design processes. The first meeting was held at Plessey Research, Caswell, with two subsequent meetings at Racal Research, Reading, and STL, Harlow. The aim of the meetings was to study the hardware design processes currently used, to determine the role that ELLA plays, and to establish a set of Designer Requirements for future design support. Many of these requirements would be fulfilled by a design system which satisfies the aforementioned Verification Requirements. Further details of the three meetings can be found in [1] and [2].

It should be noted that all of the sites were involved in designing devices with digital signal-processing (DSP) applications, and that this is likely to influence approaches to design, and requirements of the design systems. Other external factors which may influence the process of a design include commercial constraints such as delivery dates and cost.

Section 2 contains a description of a general industrial design process, together with variations, observed during the course of the three meetings. An analysis of this design process follows in §3, where present difficulties are noted, together with future requirements for design tool support. The problems and requirements of design verification are considered. Section 4 surveys the different verification methods which are available and which will fulfil the verification requirements noted in §3. Finally, in §5 a summary of future work is given. Appendices contain a summary of industrial designer requirements, a summary of verification requirements, and a glossary of terms as used in this report.

The nature of hardware verification will be considered first.

1.1 What is Hardware Verification?

Hardware verification will be used here to describe the process of determining whether a physical computing device (such as a VLSI chip) exhibits the behaviour required by the device User. This implies that hardware verification will be involved at every stage in the design process. It is to be distinguished from what shall be called *formal verification*. This is the process of determining that an abstract mathematical model representing, in this case, a computing device possesses a set of properties, and whether two models are equivalent in certain well-defined ways. Clearly, formal verification will form a major part of hardware verification.

Methods for hardware verification will range from animation — supplying an executable representation of the design with a finite set of inputs and analysing the output produced — to using tools for performing theorem-proving or equivalence checking applied to formal models of the design. A design system should therefore provide an integrated simulation and formal verification toolset. In current design systems, designers are limited to using animation only for predicting the behaviour of a design.

[3] and [4] provide useful commentaries on the features which should be provided by a verification system. The following list describes the useful properties of specification languages and verification methods:

1. executable specifications needed; this allows the designer or user to exercise the specification, and to gain confidence in it
2. integration of verification and specification techniques into existing VLSI CAD tools
3. traceback facility during theorem proving, so that the designer can determine where either the design violated the theorem, or where the theorem prover failed
4. simplicity and expressive power of specification language, leading to natural hardware descriptions (which will be easier for the non-expert to follow)
5. concise proof traces
6. hierarchical specification structures, allowing (de)composition
7. a formal specification will
   (a) aid problem understanding, gives comprehensive design document
   (b) provide the basis for design animation
   (c) be the input to theorem prover

2 Industrial Design Process

In this section, a general industrial design process will be established, based on results from visits to Plessey Research, Caswell, Racal Research, Reading, and STL, Harlow. The design processes are based on ELLA, whose role will be examined, together with comments on the IED/ELLA Project proposal. The Plessey design process serves as a good basis for the general process, with additional information and variations supplied by the other two sites.

ELLA can be used throughout most of the design process, from expressing the initial user specification, down to design representation at the gate-level. It is found to be particularly useful for design partitioning, even for multi-chip systems. At gate-level the design is converted into a format suitable for traditional CAD tools, such as standard cell layout. For example, at Plessey, a design is constructed in parallel using the semi-custom sea-of-gates Plessey Design System (PDS). At other sites, behavioural models are constructed in ELLA for standard cells in traditional chosen CAD tool libraries (for example, VTI, AT&T, Silvar-Lisco, LSI Logic), to facilitate translation of the ELLA design.

The basic design process can be summarised as follows:

1. Specification of 'numerical model' using ELLA
2. Split design into blocks
3. Produce/verify high-level ELLA model
4. Produce/verify intermediate-level ELLA model
5. Produce/verify hardware-level model
6. Layout

Translation between different design representations is performed using ad hoc translation programs, or by using synthesis tools such as LOCAM. The problem thus arises that if changes are made to the new design format, then either the initial ELLA representation must be changed also so that it is equivalent, or it is discarded. A design hierarchy is therefore not often maintained, because updates to previously produced designs must be made manually, and are thus time-consuming and error-prone. The current design representation is the only version retained.

Equivalence between different representations is checked for by comparing simulation results (at Plessey, the PDS representation is first translated into gate-level ELLA before comparison). The degree of difficulty in carrying out such a test depends on the similarity in format of the input and output of each design representation. Problems are caused by retiming and rescheduling of data, or by changes in datatypes.

The initial ELLA description, together with a carefully selected set of test vectors, can form the basis of a contract between them and the user; this is the case at Plessey. If the user has access to an ELLA system then the ELLA description can be embedded into a system-level description. Otherwise the ELLA-text forms a concise textual description.

The ELLA simulator is used for high-level examination and exploration of the design, because it is time-independent and thus allows the designer to concentrate on architectural issues. When real-time analysis is required then a specialised simulator or timing analyser is used. For example, at Plessey the event-driven PDS simulator is used for more detailed timing simulation. The time-independence of the ELLA simulator is thus seen to be at once an advantage, when considering architecture for example, and a disadvantage, clearly when circuit delay estimates are required. However, even when time-independent simulation is needed, the ELLA simulator was criticised for being too slow, as noted above.

A collection of file management systems (for example, the VMS Code Management System) and ad hoc programs are used to control source code dependencies during a design. A house style is also adopted so that separate parts of a design can be joined together more easily (for example, using common formats for ELLA-text files). These mechanisms all assist in maintaining code-compatibility in a multiple designer system throughout the design process.

2.1 Variations

At Racal and STL, the design processes are found to be slightly different from the Plessey process, mainly because ELLA is not used quite so extensively. High-level programming languages (for example C, Pascal) are used at different stages, primarily because simulations can be performed orders of magnitude faster using them instead of the ELLA simulator. In addition, programming languages are used at a high level because in general, they are more widely available to users than is the ELLA system. Users can therefore make use of the design descriptions in their own system-wide simulations.

Figure 1 gives an overview of the use of different design tools during the design processes of these industrial sites. High level programming representations $P_0 \ldots P_k$ are used for design entry or high level simulation (of course, at Plessey these are not used). The ELLA-text representations $E_0 \ldots E_m$ are used during design exploration, experiment and translation. CAD tool representations $C_0 \ldots C_n$ are used during low-level simulation, timing analysis and of course layout.

Figure 2 shows a typical test harness for checking the equivalence of two designs.

2.2 Comments on ELLA

The industrial designers noted the following useful features of ELLA:

- good for initial design exploration, (time-independent) simulation and specification, especially using sequential ELLA.
- hierarchical
- hardware-oriented
- integrated
- vendor independent

and the following criticisms were made:

- slow simulator, which is not real-time.
- function-sets/function-types difficult to understand; therefore bidirectionals are difficult to handle.[1]

[1] There will be a new and simplified approach to the use of function-sets and function-types in future versions of ELLA

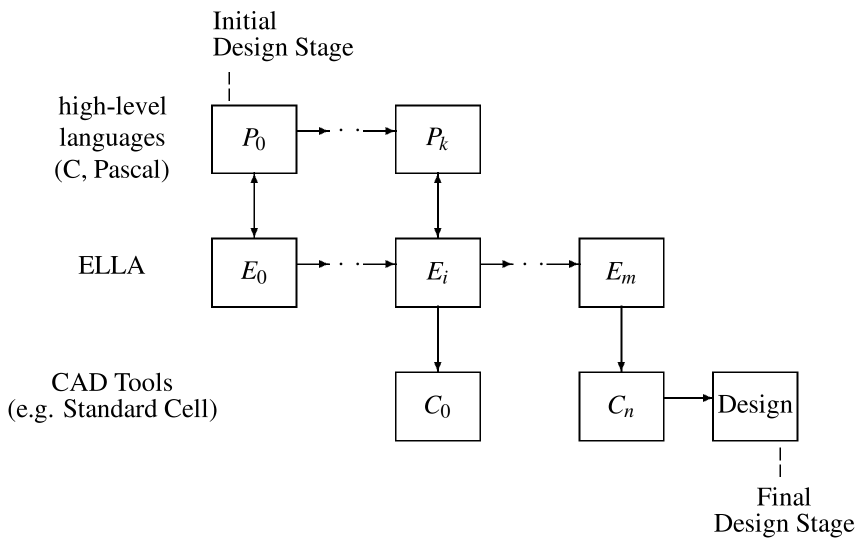

Figure 1: General Design Process

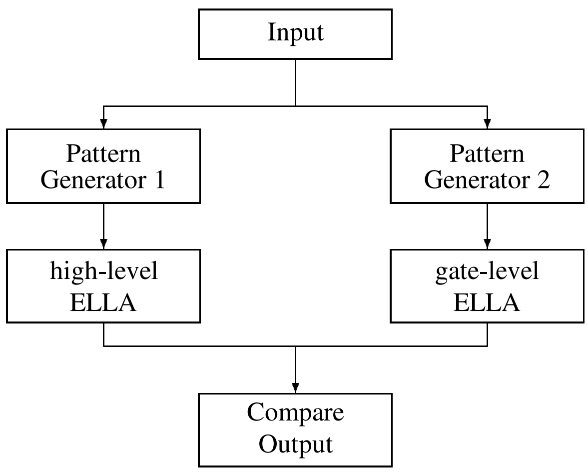

Figure 2: Design Verification by Parallel Simulation

2.3 Comments on IED/ELLA Project Proposals

The following notes give an indication of how various aspects of the proposed features of the ELLA Project Design Support Environment were received at the industrial sites.

1. *Equivalence Checking*:
   - **ALL WANTED SUPPORT HERE**. All checked for computational equivalence between designs in different representations (either ELLA-to-ELLA or ELLA-to-other, different levels/datatypes etc) by comparison of simulation results.
   - All said they could not maintain consistency in design hierarchy because of lack of support and consequent cost or 'waste of time'. Such support may be useful.
2. *Timing Issues*: Most said scheduling (eg. keeping track equivalent data when timing/clocks has been changed) is a big problem, currently coped with manually. Tool support would be useful.
3. (data) *Abstraction*: All were apparently happy with the data-types available to them in ELLA (but translation between representations is a separate issue)
4. *Notes/sketch Facility* (free use of text,graphics, with hyperlink capabilities): Currently, customer specifications are kept as (textual and pictorial) documents, either provided initially by the customer, or developed during design exploration.
5. *Schematic Capture*: Most were not fond of schematic capture (but most seemed to include schematics/timing diagrams in textual specification documents). The following comments and criticisms were made:
   - schematics are only useful for general understanding or documentation
   - 'low-level' schematic is too detailed; it is easier and quicker to input a design by text editing.
   - Schematic capture assumes once-only entry; it is very difficult to change a design.
   - self-consistency checks cause a problem (system may not allow inconsistent, incorrect or incomplete schematics to be entered or sketched)
   - schematic capture means too much detail at high-level; constrains design to being correct, complete or consistent
   - good for small designs
6. *Timing Diagrams*: not considered to be of benefit (although they do frequently appear in textual specification documents).
7. *Simulation* :
   - Some want hierarchical (logic) simulator
   - ELLA simulator (bit-level) is slow compared with logic simulators
   - ELLA simulator (higher-level) is slow compared with using high-level programming language description
   - All said they do not use ELLA simulator for detailed timing assessment (a timing analyser is used instead)
8. Design Management
   - Most considered that their design management system, including code management, was lacking or too much of a burden; would be keen for a new system.
   - Most like the library and *context* facilities of ELLA
9. *Use of other Specification Languages/tools*: ALL used ELLA at different 'levels' in design; augmented during initial stages by standard high-level language (C, Pascal) programs, and translated at later stages into a form suitable for a silicon vendor (using its standard cells, constructing bottom-level design in vendor language (and then comparing with ELLA))
10. *Interface with specialist simulation tools*: required, because different simulation tools are needed at different stages of the design. The ELLA simulator does not fulfil all design requirements, for example.

3 Analysis of Design Process

From the preceding description of current industrial design processes, it is observed that problems in design arise because there is a shortage of tools and techniques to support design translation, equivalence checking, abstraction and timing. This leads to the widespread use of animation and simulation in order to check for the correctness of a design as it progresses from the initial user requirements through to silicon. There is also a need for a design process *structure* to support animation, translation and verification tools, and to assist in design management. In this section, the main problem areas in design highlighted above will be examined. The management and structure of a design process is studied initially, followed by the important issue of translation and equivalence checking. In the remainder of this section the related problems of animation, timing, abstraction, testability and tool interfaces are examined.

3.1 Design Management and Structure

In the present design processes, a suitable top-level architecture is first developed so that the design can be partitioned into blocks with well-defined interfaces. One designer then works on each block, using a copy of the design in a separate library. The project design library is only updated when a design review has checked the interfacing of the blocks and resolved any compatibility problems. The ELLA library and context facilities are useful for maintaining design partitions.

A lot of effort is currently spent on recording design dependencies and modifications. It is essential that a complete design is constructed from the intended versions of its components, and that the design project library is not altered until the required version of a component is fully tested and approved. The correctness of a design therefore depends on the integrity of its design management system. It is necessary to impose a suitable structure onto a design history, so that the relationship between different design parts and versions, together with any design decisions taken in making a transition, are recorded.

3.1.1 An Approach to Design Structure

Three main stages may be identified in the translation of a design from initial user requirements to silicon:

Stage A: User Requirements to User Specification (Requirement Capture)

Stage B: Transformations applied to abstract design representation; for example:

(i) Specification S_1 to Specification S_2
(ii) Specification S to Draft Design D
(iii) Draft Design D_1 to Draft Design D_2

Stage C: Implementable Design D to Physical Device

Each stage requires different techniques to assist in design translation and to help ensure that the translation preserves correctness. For example, ELLA has been introduced into the current industrial design processes to help to reduce the gap between user requirements and concrete design representation. It allows the user and designer to explore and experiment with the design, so that high-level architectural and behavioural issues can be studied.

In [5] it is proposed that the design process is divided into three phases: Exploration, Abstraction and Implementation (see Figure 3). During exploration, a specification trajectory is built up in consultation with the user until a particular specification S_{u-c} is user-consistent and fulfils all of the user requirements. The specification is then further developed to resolve any inconsistencies, errors or omissions contained in the original user requirements. The abstraction phase is then entered, where parts of the design (such as implementation dependent factors) are modified to remove unnecessary details , so that the design is reduced to the simplest form possible. The final Implementation phase carries specification and design development through to implementation.

	$S_{initial} = S_0$	$D_1^0 \rightarrow \ldots \rightarrow D_i^0$
	$\downarrow$	
	S_1	$D_1^1 \rightarrow \ldots \rightarrow D_j^1$
Exploration	$\downarrow$ $\vdots$ $\downarrow$	
	S_{u-c}	$D_1^{u-c} \rightarrow \ldots \rightarrow D_k^{u-c}$
	$\downarrow$ $\vdots$	
Abstraction	$\downarrow$ S_{exp} $\downarrow$ $\vdots$	$D_1^{exp} \rightarrow \ldots \rightarrow D_m^{exp}$
Implementation	$\downarrow$ S_{final}	$D_1^{final} \rightarrow \ldots \rightarrow D_n^{final}$

Figure 3: Example of Design Structure Record

A simplified design process in [5] is represented by an array of draft specifications $S_1 \ldots S_m$ and designs $D_1^i \ldots D_n^i$ where S_{j+1} is a development of S_j and D_{j+1}^i is an development of D_j^i (see Figure 3). Each specification may give rise to a sequence of draft designs, one of which will progress through to implementation. Note that Stage A from above covers the specifications $S_{initial} \ldots S_{u-c}$, and Stages B and C cover the remaining draft specifications and designs.

Each specification S should be a 4-tuple

$$S = [L, I, F, E]$$

where $L =$ Hardware Specification Language HSL, determining the level of abstraction, $I =$ Informal specification, $F =$ Formal specification, and $E =$ set of Experiments. A draft algorithm Design D is a 5-tuple

$$D = [L, A, S, V, E]$$

where $L =$ Hardware Description Language HDL, $A =$ Algorithm written in the HDL, $S =$ Specification, both for functional correctness and performance, $V =$ Verifications, and $E =$ set of Experiments.

It will be necessary to record an explanation for each progression from one specification or draft design to the next. There may exist an equivalence proof, or application of a compiler to convert between representations. However it is also possible that the connection is informal or arbitrary. Alternatively, the connection may not yet have been completed: there could be an associated proof obligation still pending. This would still serve as a suitable explanation, and it would be recorded in the design history that such a step had been made. Note that the elements from Figure 3 may not necessarily be constructed in the order shown. For example, work on implementing parts of a design is often carried out in parallel with high level specification, to reduce the length of the design cycle.

Such a design framework may provide a mechanism for structuring a large design project. It should form the basis of both a design management and database system. Each specification or design entity would also require additional attributes, such as textual notes or design obligations (see [6]).

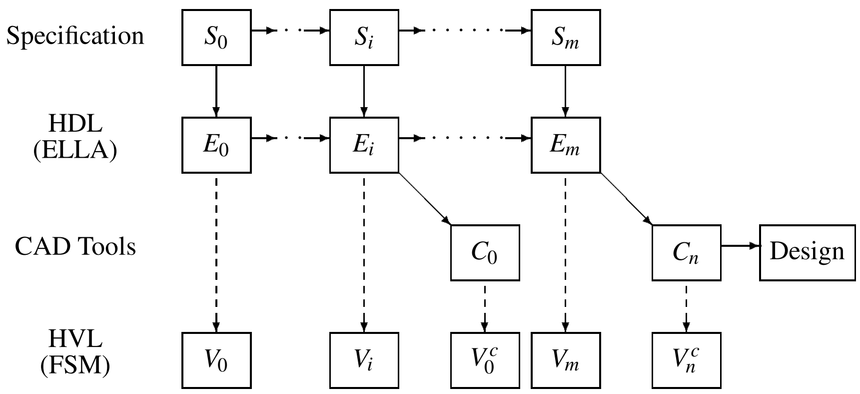

Figure 4: Design Process Structure with Verification

3.2 Design Translation and Equivalence

Having proposed a design framework, describing the different stages in design, it is now possible to consider translating a design between stages and checking its correctness. As stated above, translation and equivalence checking can be divided into three regions — Stage A: User Requirements Capture; Stage B: Abstract Design Translation; Stage C: Physical Implementation.

Current design processes rely on parallel simulation of different design representations in order to check for equivalence, and traditional current CAD tools focus assistance on Stage C. It is only during Stage B that formal verification methods can be applied. The requirements for formal verification during the design process will now be considered.

It is likely that during a design, several different design representations will be used, depending on the design level and operation being carried out. It is therefore necessary to provide design translation tools, and to ensure that translation or compilation between representations preserves equivalence.

An appropriate well-defined language must be used (the Hardware Verification Language HVL) as a basis for formal verification. Any other languages used during design, such as ELLA, will be compiled into the HVL for verification. For example, a finite state machine language may serve as the HVL (although it may be necessary to have more than one HVL), into which all other design representations (such as those used by simulators, standard cell libraries or other CAD tools) are translated before verifying.

Figure 4 shows a similar example design process structure to that of Figure 1, where ELLA is used as an HDL. In this scenario, however, each ELLA-tex $E_0 \ldots E_m$ or CAD tool design representation $C_0 \ldots C_n$ is now translated into a language HVL amenable to verification so that properties of, or equivalences between, design representations can be checked formally. In addition, formal specifications $S_0 \ldots S_m$ have been introduced using a suitable language HSL, obviating the need for the high-level language representations that appeared in Figure 1. Notice also that the design process structure shown in Figure 4 differs from that of Figure 3 in §3.1.1 due to the addition of the HVL-representations; of course, these help to establish the relationship between the draft specifications and design representations.

Once a design has been compiled into the appropriate HVL then the following properties may be formally verified:

- Computational equivalence between two design representations. This would have to take into account retiming and changes in datatype.

- Equivalence between *parts* of two design representations.
- Equivalence between specification and design representation.
- Checking that a design representation has certain properties; for example invariants, eventualities, pre/post-conditions.

If a verification fails, it should be possible to obtain diagnostic information to assist in tracing the error, such as a counter-example in a proof.

A formal verification system will also be able to assist during design partitioning and composition; it should be able to check that the specification for a block interface is satisfied by the block design, and to use this specification during verification of the composite design.

It may be beneficial to maintain a distinction between data, control and timing aspects in the design to assist in formal verification.

3.3 Animation

Animation can be used both for checking that a design performs the correct function for a given input, and for comparing the behaviour of two different representations of the same design. The first facility will be extensively used in Stage A, when developing the initial formal user specification, given an informal description of user requirements. It is to be expected that a user with not much experience with formal methods will have difficulty in accurately interpreting a formal model of their requirements, so by animating the initial formal user specification they be able to check that their requirements are captured correctly. In addition to an animation tool, there will need to be a set of support tools to process the design input and output so that helpful information is presented to the designer.

In [7] it is suggested that a simulation comparison can be carried out for two programs P and Q by creating a third program R, which runs the two programs connected in parallel and then checks for the equivalence of their outputs. The composite program will therefore produce a boolean value which depends on whether the comparison is good or bad. Further diagnostic tools would have to be added to such a program; it is vital to know which pairs of outputs from the simulator differ (and to determine the inputs which caused this). A traceback facility indicating the parts of each program responsible for producing the incompatible parts of the output would be useful.

3.4 Timing Issues

At a high-level of design, timing is not of great importance. A synchronous operation is thus assumed. For example, all combinatorial logic may be given zero delay, whilst registers are given unit delay. The ELLA time-independent simulator is useful here. The designer can then concentrate on architectural issues.

During design translation, significant problems can arise due to rescheduling or retiming of data in a design. It may be difficult to compare like data from a parallel simulation of two different design representations. Currently, problems caused during simulation by retiming or rescheduling are processed manually. and resulting simulation data must currently be processed manually. There is clearly a need for the introduction of suitable support tools, to support test pattern generation, retiming, datatype conversion (which affects timing) and like-data comparison.

For low-level design, then evaluation of real-time circuit delays becomes important, so that the performance and correct behaviour of the circuit is maintained. Timing analysis is employed for critical path search, and detailed circuit-level simulations are conducted for standard cells. The accuracy of such estimates can only be ascertained empirically. Attention must be given to the relationship between simulation results produced by a low-level real-time model and the behaviour of the corresponding low-level discrete design model.

3.5 Abstraction

At Racal and STL, only two levels of data abstraction (INT and BOOL) are used in ELLA-texts, in addition to other enumerated types and associated types. At Plessey, floating points are also used (included in the latest release of ELLA). This limited use of data abstraction may be due to the nature of the DSP applications of the designs being tackled, or to the present inability to perform datatype transformations, because of the lack of adequate methodology and tool support. These datatypes may not be sufficient for other areas of application.

3.6 Testability

There is no way of guaranteeing that a fabricated device will exhibit the behaviour required by the user, even if the lowest-level design model is correct. This may be because the low-level model does not capture device behaviour correctly, or because there are fabrication flaws in the design. So however accurate the low-level model there will always be a need for designs to have good testability, to assist during Stage C of verification.

It is important that testability issues are considered from the start of a design , so that test circuitry, such as debug or pseudo-random signal generation, can be built in. Verification and testing of silicon must also be considered; inclusion of scan-paths will assist the designer in evaluating internal circuit states. Generation of suitable sets of test vectors to obtain adequate coverage is an independent research problem in itself. Vectors are currently generated by hand or semi-automatically — for control circuitry a functional test is needed, whilst for ALU/datapaths an exhaustive test is performed. Circuit block isolation is a problem.

3.7 Interface to Verification Tools

A designer is likely to be more familiar with verification tools such as animators or simulators, than with formal verification tools. Care must be taken to produce new tools, especially those for formal verification, that feel ‘natural’ to the designer, and fit in with familiar operation practices.

The quality of interface between verification tools and their environment will affect the degree to which these goals can be achieved, and hence the quality of the whole system.

3.7.1 User Interface

Animation tools should accept an input data file to be applied to an executable design representation, and then permit the designer to manipulate the output so that it is in the most useful format. They will also allow monitoring of the activity of any node in a design.

In order to operate a formal verification tool, a designer will need to know the necessary syntax and meaning of commands, and how to interpret the results. Having received suitable training a designer will be aware of the capabilities of the verification tool, and just as important, of its limitations.

A draft specification or design may be in any of an approved set of formats, with the verification tool informing the designer the options for verification available at a particular stage. If the tool fails to complete then an error message indicating the required remedial action should be produced. If a specification or design fails a test then the tool should produce a traceback indicating the area of the problem, for example by presenting a counter-example.

Verification tools will be able to handle any design representation changes required in order to conduct verification. For example, if two ELLA descriptions of a design are to be tested for equivalence then the verification tool would first compile the ELLA descriptions into the appropriate HVL, perform the equivalence-check, and then relate any diagnostic information back to the original ELLA descriptions. Designer involvement should be kept to a minimum, and ideally it should be zero, whilst the verification is in progress. The output format of the tool should be controllable.

The role of a verification tool may also involve providing messages and warnings to the designer, indicating the state of proof obligations and dependencies, where the correctness proof of one design block will depend on the proof of another block.

3.7.2 Design System Interface

The proposed IED/ELLLA Project Design System will consist of a design support environment for an HDL such as ELLA, linked with a hardware verification system. This environment will then need to interface with existing CAD tools and design database systems, so that external design representations can be imported into the environment for processing, and internal representations can be exported for processing by external CAD systems or for storage in design databases.

The obvious problem with building such an interface is that a wide range of different design representation formats is used by the external tools. There have been encouraging signs of attempts to develop standard design exchange languages, which would be then supported by CAD tool vendors. However, it is not yet clear whether such efforts will be successful. Consequently, if the Project Design System opts to support a design exchange format which is not then accepted by a sufficient number of industrial designers, then it will not enjoy wide usage.

However, the concept of using an interface which provides a standard design exchange format is likely to form the basis for a viable solution to this problem. Tools within the environment will then be able to fully support the single design format, whilst a set of translators would be needed to convert between the standard format and any format used in an external CAD tool.

3.8 ELLA and Verification

ELLA has been put forward as a candidate for use as a Hardware Description Language in the IED/ELLA Project design environment. It is useful to examine the properties of ELLA which make it suitable or otherwise for such a role. There are likely also to be special requirements on the environment because of the nature of ELLA:

- simple, natural specification/description language, very well-suited to hardware specification, because it was designed with the aim that all constructions would be hardware implementable.
- formal syntax and semantics can be constructed, suitable for entry into formal verification tools.
- executable (it is equipped with a simulator/animator), although this is not yet formally based.
- can be used with different datatypes
- user-imposed delays allow time-independent study of design (this is also a disadvantage when delays are to be included)
- hierarchical compositional structure of designs
- equipped with design management database
- familiar format for engineers; it is already being used in industrial design systems.

3.9 Limitations of Verification

Hardware verification can only be performed informally, for it is not possible to give a precise meaning to the User Requirements, and so it is impossible to reason about them formally (see [8] for a discussion). At the implementation stage, it is also not possible to precisely model the physical behaviour of the VLSI device. Even if it were, then the unavoidable introduction of fabrication flaws would prevent exact prediction of chip behaviour. According to [4], formal verification can also only ever hope to reduce the chances of certain errors being introduced during the translation of the initial abstract model of user requirements to the final abstract model of the behaviour of the physical device.

4 Methods for Verification

It has been established in the previous section that hardware verification affects all aspects of the design process. Formal verification methods are suitable for application during Stage B of design translation, when design representation only involves abstract models. For verification during design stages A and C it is necessary to use other methods. In the following sections a complete cross-section of hardware verification methods is examined.

4.1 Formal Verification Methods Available

Temporal Logic-based Verification Methods (sample)

Advantages:

- Algorithmic: machine-processable
- Time
- Traceback possible

Problems:

- State explosion
- not good for combinatorial computations

SMG: (Project at Manchester [9])

EMC: offers traceback/counter-example facility [10]

Work by Kurshan: state reductions by homomorphisms [11]

TEMPURA: supports programming language constructions [12]

CESAR [13]

Methods based on Equational Reasoning (for example, OBJ [14])

Advantages:

- equational logic with rewriting (simple and natural for h/w) and
- nice proof tactics: rewrite, induction,...
- can express computable functions
- Links well with abstract datatypes
- First order
- semi-automatic proof construction
- (OBJ Project at RHBNC)

Problems:

- difficulty of constructing proof route and size of proof trace
- semi-automatic proof construction

Methods based on General Higher Order Predicate Calculus (for example, HOL [15])

- Problems:
  - higher order logic (according to [4])
  - long proof traces
- Advantages:
  - higher order logic
  - working system
  - embedded language facility
  - (HOL Project at Cambridge)

Other methods to consider possibly are CCS/CSP, Circal, μFP, VDM.

4.2 Other Methods Necessary for Verification

During User Requirements Capture (design Stage A) every effort must be made to facilitate user understanding of the initial user specification. It is therefore necessary to use a simple specification language, which gives a ‘natural’ description of hardware, from the User viewpoint. The language will be executable so that the initial design description can be animated. It will support abstraction so that unnecessary details, such as real-time issues, can be excluded from the specification. A multimedia system, which allows textual, pictorial and schematic elements to be associated with particular parts of a design, would be useful; user requirements may well be presented initially in such a form. User-controllable presentations of animation results may be of assistance, whilst machine-processable formats would enable that results from design animation to be embedded into the controlling environment.

At the physical implementation Stage C, there is currently the most support available from traditional CAD tools. However it is not possible to relate the frequently informal and empirically-based models used here, with the low-level discrete models used during design translation in Stage B. Development is required of low-level discrete design language which can be embedded in Stage B, whilst producing animation results in agreement with physical device behaviour.

5 Summary

In this report, the nature of hardware verification is explored, with the aim of establishing the verification requirements of a new design system developed by the IED/ELLA Project. The structure of a general industrial design process has been described, examining the problems currently involved. From this study, a set of verification requirements is obtained, which affect every aspect of the design process, from user requirements capture through to device fabrication. A set of suitable verification methods is suggested.

5.1 Future Work

A set of design case-studies should be assembled so that new verification tools developed in the IED/ELLA Project can be exercised. The following is an initial set of possibilities:

- Counter (integer and boolean)
- Traffic Light Controller
- Alternating Bit Protocol Communications
- Multiplier (pipelined, parallel), ALU
- Lift/disk Controller
- Railway Signalling
- Simple Processor (such as Viper)

Each design will be specified using a suitable language, and then developed using ELLA. The examples should be of varying complexity and address different aspects in design.

A Review and comparison of hardware description languages is required (for example, consider ELLA, VHDL and ABEL). Consideration should be given to the role ELLA can play in a design system.

Further reviews should be conducted of traditional CAD tools, and currently available formal verification methods.

6 Acknowledgements

The author would like to thank those people who helped to arrange and who participated in the industrial meetings: John Fox and Gareth Dixon who arranged and took part in the primary meeting at Plessey; Mike Hill (RSRE) who provided initial contacts for the follow-up meetings at Racal and STL; Eric Harper (Racal) and Andy McWilliam (STL) who arranged the Racal and STL meetings; Mark Jenkins, Kevin Donnelly, Dougal (all Racal) and Graham Fellows (STL) who took part in the Racal and STL meetings.

The author would also like to thank Brian Monahan (University of Manchester) for his useful comments on this report, and Doug Edwards (University of Manchester) who was participant in the visit to Plessey and in preparation of that visit report.

A Summary of Designer Requirements

The following list is a summary of Designer Requirements [i]-[xiii] identified at the industrial sites. There is a reference to the Verification Requirements ([A]-[C],[1]-[16]) which coincide with each Designer Requirement; these are described below in §B. If these Verification Requirements are met by a future design system then these corresponding Designer Requirements will also be met.

[i] support tools for reliable design translation (including datatype, level and timing translations) (met by Verification Requirements [A], [B], [C]; [4], [8])

[ii] test pattern comparison tool (to cope with rescheduling problems) (met by [4(b)])

[iii] high-level schematic capture, but with a sketch-pad feel for flexible use (for example, to cope with: varying degrees of detail; incomplete and inconsistent designs; (which would be worked on at a later date)) (met by [5])

[iv] hierarchical timing analyser to bridge gap between ELLA simulator (with no real-time consideration) and present timing analysers (which flatten design) applied to standard cells (met by [12])

[v] design management system/support (for example, code management, design route standardisation, design partition) (met by [7(b)])

[vi] methodology to construct abstract model which facilitates mapping to low-level physical implementation, especially with respect to timing. (met by [A], [B], [C], particularly [3])

[vii] behavioural/functional specification for block interfaces (met by [7(d)])

[viii] Interface with specialist simulation/CAD tools.

[ix] block diagram *output* from initial ELLA-text, so that dataflow can be observed (a mixture of text and picture would be good) (met by [5])

[x] more degrees of abstraction; do not need actual physical timing, just consider architectural aspects. (met by [7], [2])

[xi] behavioural synthesis tool

[xii] adaptable ELLA environment user-interface.

[xiii] dynamic storage in future versions of ELLA

B Summary of Verification Requirements

The requirements for the following three distinct design verification areas must be considered (only for [B] is it possible to use purely formal techniques).

[A] User Requirements vs. User Specification

[B] Transformations applied to Abstract Design Representation; for example:

- (i) Specification A vs. Specification B
- (ii) Specification vs. Draft Design
- (iii) Draft Design A vs. Draft Design B

[C] Implementation Design vs. Physical Device

Note that for each Verification Requirement [1]-[16], the corresponding Designer Requirements [i]-[xiii] which will be fulfilled have been noted:

[A] User Requirements vs. User Specification (see [3]); partly meets Designer Requirements [i], [vi]:

[1] *Formalised* Specification Language:

[2] Properties of Specification language: simplicity, 'natural' hardware description, (de)composition, (data/timing) abstraction (meets [x])

[3] Well-defined design process (especially during capture of user requirements) Iterative/consultative construction of User Specification in Exploration Phase of Design Process Abstraction Phase to remove unnecessary design details, such as implementation dependencies (partly meets [vi])

[4] Animation of User Specification (partly meets [i])

[5] Schematic capture/animator, timing diagram processor;
Documentation/Text/picture processing/annotation/hyper-linking with (formal) specification and rest of design environment (meets [iii], [viii])

[6] Designer Education

[B] User Specification vs. Implementation Design (partly meets [i], [vi]):

[4] Animation (and suitable animation processing tools)

[7] General model/language properties (partly meets [x]):

- (a) formal model/language, with many-sorted ADTs, multiple clocks
- (b) well-defined design process, with design/code management (meets [v])
- (c) Abstraction (data, control, timing)
- (d) Composition or decomposition (meets [vii])
- (e) machine-processable syntax/semantics

[8] Formal Verification (partly meets [i]):

- (a) equivalence checking
- (b) property checking (invariants, conditions, eventualities)
- (c) theorem proving
(This needs formal model, with abstraction, composition, retiming)

[9] realistic computational models (especially at low level)

[C] Implementation Design vs. Physical Device:

[10] Testing methodology (to test for fabrication faults *and* for functionality faults).

[11] experiments to demonstrate that lowest level design does not *violate* product design

[12] Realistic simulation and timing analysis (meets [iv]):

- (a) multi-level
- (b) compatible with physical device behaviour
- (c) compatible with low level discrete model of design

[14] experimental results

[15] links between simulators and discrete design representation

[16] Electronic/physics theoretical development of semi-conductor circuit modelling techniques.

The need for animation introduces further requirements on the design environment:
[4] Animation:

- (a) executable specifications
- (b) test pattern comparison tool (meets [ii]):
  - (i) test pattern generator
  - (ii) retiming/rescheduling tool
  - (iii) datatype conversion tool

(iv) like-data-comparison tool

(c) multi-level

[7(c)] Abstraction:

(i) Data abstraction: many-sorted ADTs, homomorphisms, compilers, ADT extensions; separate system Data from Control

(ii) Timing abstraction: multiple clocks, and maps between them (retiming); adoption of synchronous model

C Glossary

- *User*: the person, group or company who will use a fabricated device. They provide a set of requirements (behavioural, physical) which they wish the device to fulfil.
- *Designer*: the person or group of engineers who specify, design and implement a device, given the user requirements.
- *User Concepts* or *User Requirements*: information provided by the user to describe the behaviour of the device they want; typically this will be an ad hoc textual/pictorial description, algorithm description, interface requirements, including operating constraints (area, speed, power etc); mathematical formulae, timing diagrams, schematics.
- *User Specification*: the capture of user requirements in a formal specification language/model; needs consultation with user, animation, abstraction, iteration; there will only be an informal correspondence between user requirements and user specification
- *Hardware Verification*: determining whether a physical computing device exhibits the behaviour required by the device User.
- *Formal Verification*: determining, using formally-based techniques, that an abstract mathematical model of a computing device possesses a set of properties, and whether two abstract models are equivalent.
- *(traditional) CAD tools*: software to assist a designer in producing graphical descriptions of a VLSI circuit, and in checking that the circuit will operate correctly. For example: graphics input/editing packages, (layout) design-rule checkers, electrical rule checkers, circuit extractors, logic-level simulators, timing analysers, circuit-level simulators.
- *(Abstract) Data types (ADTs)*: can be defined in terms of a many-sorted algebra, which consists of a collection of non-empty sets of elements, with operations between them. An ADT is defined as the *isomorphism class* of a many-sorted algebra.
- *Schematic Capture*: entering circuit, manipulating it symbolically; i.e., draw circuits/wires, blocks/busses, annotate, and then process them using other design tools.
- *Simulation*: trying to imitate the behaviour of a device using model of the design
- *Animation*: applying inputs to a *model*, using it to perform *its* computation and observing the output of the model (so an animation may have nothing to do with the behaviour of any *real* computing device, whereas a simulation does try to reproduce the potential behaviour of the device)

References

[1] A Williams. Industrial Visits: Plessey Research Labs, Caswell, 10th July 1990. Technical Report, Department of Computer Science, University of Manchester, July 1990.

[2] A Williams. Industrial Visits: Racal Group Research, Reading, 9th August 1990; STL, Harlow, 10th August 1990. Technical Report, Department of Computer Science, University of Manchester, August 1990.

[3] V Stavridou, H Barringer, and D Edwards. Formal Specificition and Verification of Hardware: A Comparative Case Study. Technical Report UMCS-87-11-1, Department of Computer Science, University of Manchester, 1987.

[4] J A Goguen. OBJ as a Theorem Prover with Application to Hardware Verification. Technical report, SRI International, August 1988.

[5] N Harman. *Formal Specification for Digital Systems*. PhD thesis, School of Computer Studies, University of Leeds, October 1989.

[6] B Monahan. First-Cut Toolset Design Notes. Technical report, Department of Computer Science, University of Manchester, November 1989.

[7] N Halbwachs, D Pilaud, F Ounbdesselam, and A-C Glory. Specifying, Programming and Verifying Real-Time Systems using a Synchronous Declarative Language. In J Sifakis, editor, *Automatic Verification Methods for Finite State Systems*. LNCS 407, June 1989.

[8] A Cohn. The Notion of Proof in Hardware Verification. *Journal of Automated Reasoning*, 5, 1989.

[9] H Barringer, M D Fisher, and G D Gough. Fair SMG and Linear Time Model Checking. Technical Report UMCS-89-10-2, Department of Computer Science, University of Manchester, June 1989.

[10] E M Clarke, E A Emereson, and A P Sistla. Automatic Verification of Finte-State Concurrent Systems Using Temporal Logic Specifications. *ACM Trans on Programming Languages and Systems*, 8(2), April 1986.

[11] Z Har'El and R P Kurshan. Software for Analytical Development of Communications Protocols. *AT&T Technical Journal*, January/February 1990.

[12] B Moszkowski. A Temporal Logic for Multilevel Reasoning About Hardware. *IEEE Computer*, February 1985.

[13] J P Queille and J Sifakis. Specification and Verification of Concurrent Systems in CESAR. In M Dezani-Ciancaglini and U Montanari, editors, *International Symposium on Programming*. LNCS 137, April 1982.

[14] J A Goguen and T Winkler. Introducing OBJ3. Technical report, SRI International, August 1988.

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