Report on the Formal Specification and Partial Verification of the VIPER Microprocessor

Bishop Brock
Warren A. Hunt, Jr.[1]

Technical Report 46 January 15, 1990

Computational Logic, Inc.
1717 West Sixth Street, Suite 290
Austin, Texas 78703-4776

TEL: +1 512 322 9951
FAX: +1 512 322 0656

EMAIL: brock@cli.com, hunt@cli.com

[1] This work was supported in part at Computational Logic, Inc. by NASA purchase order L-39627C and by the Defense Advanced Research Projects Agency, ARPA Order 584155. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of Computational Logic, Inc., NASA, the Defense Advanced Research Projects Agency or the U.S. Government.

Foreword

By John Kershaw, VIPER Project Leader

Introduction

The VIPER microprocessor chip is one of the results of a research program on high-integrity computing being carried out at the Royal Signals and Radar Establishment at Malvern, England. RSRE (which is a research station belonging to the British Ministry of Defence) has been developing formal methods of specifying and analysing software for some 15 years, but only in 1983 did we begin to look at the equally challenging problem of computer hardware.

Correctness of computer hardware has only become an issue because of the pressure from computer users (civil and military) to put programmable electronics into ever more critical systems. The medical, automotive, and avionic communities now all use general purpose microprocessors in systems which could place the lives of their customers at risk, and the threshold is constantly being pushed higher: computer control offers so many advantages in cost, performance, and flexibility that its temptations are rarely resistible.

Conventional microprocessor chips, however, are neither well enough specified nor accurately enough implemented for life-critical (or security-critical) applications. Every month sees another press report of problems in widely-used devices. VIPER, like the FM8501 and 8502, was born of the need for a microprocessor with a precise, formal, specification, the highest possible assurance that the physical device conforms to it, and the special characteristics needed for high-integrity applications.

Computational Logic Incorporated were commissioned by NASA to review the work on VIPER, under a long-standing framework for collaboration in aerospace research between NASA and the Royal Aerospace Establishment in Britain. Information on VIPER was supplied by ourselves and by Marconi Electronic Devices Ltd.

The VIPER Design and Verification Process

VIPER was developed in response to a number of UK military requirements. We made an early decision (in 1984) to fabricate chips as a demonstration of what could be done, and therefore limited ourselves to technologies which offered a reasonably quick route to silicon. Gordon's LCF-LSM system (which has now matured into HOL, see the references at the end of the report) was the obvious choice, since a full account of it had been published with a tutorial example of hardware verification - a simple 8-bit machine now called "Tamarack."

HOL is syntactically quite similar to the hardware description language ELLA, so we decided to live with an informal (though technically easy) step in the design process by changing languages in the middle. A strongly-typed language like HOL is a big help when writing a specification (it detects a large proportion of errors) and a theorem prover for it is available. ELLA is an established HDL with a wealth of tools, some of which can refine a block-level description into a gate-level design with almost no human intervention. We changed at the point of least effort; the process could easily be mechanised though for a device as simple as VIPER this did not seem worth while.

From the language change downwards, the VIPER design was first verified by a process of "intelligent exhaustive simulation" whose coverage depends on the set of test vectors applied. These vectors (which contain "don't cares" in profusion) are generated by the ELLA simulator from a hand-written program. Though we believe this method to be sound (it is in fact pessimistic: it rejects some correct circuits but should never accept a wrong one) it depends too much on human assiduity for comfort, so we have replaced it. The new technique, developed by Clive Pygott at RSRE and based on the work of R. E. Bryant, is called NODEN; it was used in August 1989 to repeat the low-level verification of the latest version of VIPER. This device (VIPER 1A) is designed to operate in pairs and has built-in comparators to check for address and data bus errors. The comparators have too many inputs for the NODEN analyser to check them, but the rest of the design was confirmed to be correct.

The upper levels of the verification were done by Avra Cohn at the University of Cambridge, England, using the HOL Theorem Prover. The proof is not complete in a number of areas concerning the meaning of computer arithmetic; these points have been documented by Cohn herself and are commented on further in this report. We are confident from careful (but informal) argument, simulation, and testing of actual chips that the implementation is correct, but as Fetzer and others have pointed out it is not justified in strictly mathematical terms to claim a "proof" of any physical device.

Hardware, of any kind, is fundamentally different from computer software in that a truly formal proof (a "demonstration" in Fetzer's terms) of correctness is not possible. Physical devices wear out and break down, and no amount of formal logic can guarantee immunity: ultimately physics makes its own rules. With VIPER we have put almost as much effort into guarding against hard-

ware breakdown (using parity bits, master/slave operation, built-in test, and self-testable logic) as into assuring correctness of the design. The two aspects are of course complementary, for most redundant systems are built round the assumption that the channels will fail independently. If there is a common design fault, they may all fail together.

The Future

This report and Avra Cohn's work confirm our impression that more remains to be done, both to build up confidence in the existing VIPER design and to develop new techniques of design and verification which avoid the limitations of present methods. CLI themselves are major contributors to this field; in the UK we are sponsoring work on methods of refining a functional specification by correctness preserving transformations, so that the eventual gate-level design is "correct by construction." The first test vehicle for this work will probably be a much faster (but upwards compatible) development of VIPER.

At the silicon level, we are insuring against faults in the CAD software and manufacturing process (which are below the end-point of our verification work) by sponsoring a second, independent, gate-level design of the chip using a different technology.

In the longer term we feel that a combination of methods will usually be needed to achieve the highest assurance. A design may be pronounced "correct by construction" by a faulty software tool; to guard against this a separate proof of implication could be carried out using different tools. At the level of electrons and transistors formal logic is not very helpful, and the best safeguard is to repeat the design process in as different a way as possible. Interchange and co-operation between the various research teams is vitally important, and we hope to build on the knowledge we have gained from CLI and NASA to strengthen our own work and to provide secure foundations for VIPER and its descendants.

RSRE Malvern, UK John Kershaw
Email: KERSHAW@HERMES.MOD.UK

Contents

Foreword **i**

1 Introduction **1**

2 The VIPER Microprocessor **4**

3 Formal Specification **6**
3.1 The Top Level 7
3.1.1 Overview 7
3.1.2 Arithmetic Specifications 8
3.2 The Major-State Machine 9
3.3 The Block-Level Specification 10
3.4 The Implementation-Level Specification 11
3.5 Specification Summary 12

4 Proofs and Proof Attempts **13**
4.1 Block $\Longleftrightarrow$ Implementation 13
4.1.1 Intelligent Exhaustion Technique 14
4.1.2 Application to VIPER 15
4.1.3 Interface to Higher Levels 16
4.1.4 Summary 16
4.2 Top $\Longleftrightarrow$ Major-State (LCF_LSM) 17
4.3 Top $\Longleftrightarrow$ Major-State (HOL) 17
4.4 Top $\Longleftrightarrow$ Block 18
4.5 Proof Summary 20

5 Conclusion **21**

Chapter 1

Introduction

VIPER (Verifiable Integrated Processor for Enhanced Reliability) is a 32-bit microprocessor architecture designed by the Royal Signals and Radar Establishment (RSRE) in Malvern, England [Ker84]. Recent technical and marketing literature includes the following statements:

> ...formal mathematical methods have been used both to specify the overall behaviour of the processor and to prove that the gate-level realisations conform to this top-level specification. [CP87]
>
> [Formal methods] were used in the development of VIPER, the first commercially available microprocessor with a formal specification and a proof that the chip conforms to it. [Dyk88]

The purpose of this report is to examine the claim that the gate-level design of the VIPER microprocessor is mathematically verified. The sources for our study of VIPER included a number of technical documents from RSRE and Cambridge University. Additionally, in April, 1989, we personally interviewed the VIPER design team and their Cambridge University verification consultants. Although a great deal of effort has been expended on the formal specification and verification of VIPER, there is not sufficient evidence to substantiate the claim that the VIPER gate-level specification (the implementation netlist) has been proven to implement its top-level specification (the instruction interpreter). This is not a unique point of view; a recent paper by one of the Cambridge University consultants arrives at a similar conclusion [Coh89b].

Chapter 2 contains an informal description of the VIPER microprocessor. The analysis of VIPER begins in Chapter 3 with an outline of the abstract layers used to specify VIPER. Chapter 4 covers each of the proofs, and attempted proofs of correspondence between the levels in the specification. A schematic block diagram of the contents of Chapters 3 and 4 appears as Figure 1.1. The conclusion contains a discussion of some of the broader issues in managing formal hardware verification projects, using the VIPER project as an example.

Throughout the report we contrast the specification and verification approach used for VIPER with that used during the specification and verification of the FM8502 microprocessor [War87].

Figure 1.1: A schematic representation of the specification and verification of VIPER.

Chapter 2

The VIPER Microprocessor

The VIPER project was launched after the British Ministry of Defence became increasingly concerned that computer hardware and software errors had caused and would continue to cause loss of life. It was believed that the formal specification and verification of a microprocessor would yield an embedded-systems platform whose operational characteristics were completely known. Some other goals of the VIPER project, besides those stated in the VIPER design documents [Ker84], are listed below.

Design Stability. Different revisions of supposedly identical processors have been found to behave differently even though these processors bear the same part number. These differences are often the result of iterating a design as the processor implementation technology matures. To avoid this problem, VIPER implementations were to be verified before they were produced, thus insuring all VIPER implementations would have identical functionality.

Safety Critical Architecture. The VIPER architecture is straightforward and simple. A simple architecture is easier to specify, verify, and use correctly. To enhance the VIPER's use in safety-critical computing, the VIPER designers added features which force the processor to halt under various conditions. The VIPER enters a halt state whenever an unimplemented instruction is encountered, when the memory does not respond within a fixed amount of time, an unexpected arithmetic overflow is detected, etc.

Reliable Systems. Formal verification is not insurance against physical failures. The VIPER1A, a successor to the VIPER1, includes additional circuitry which allows two processors to be operated as a self-checking pair [Pyg87,HP87]. If the computations of the two processors ever diverge, then both processors will halt.

The above list points out the great success MOD has achieved with the VIPER project. They have implemented a safety-conscious architecture, and developed dual-processor systems capability which should provide enhanced reliability.

An informal, architectural-level specification for VIPER appears in [Ker84]. VIPER is a 32-bit machine with an accumulator, two index registers, a 20-bit program counter, and a 1-bit flag register. The processor supports word addressing of separate, 2^{20} word I/O and memory spaces. Each 32-bit instruction consists of a 12-bit opcode and a 20-bit literal value or memory address. As mentioned above, VIPER is designed to stop whenever an error is encountered during processing. The fact that the processor has halted due to an error is detectable by means of a dedicated output. The VIPER architecture does not provide interrupts or other kinds of exceptions except as noted above.

Chapter 3

Formal Specification

The formal specification of VIPER is divided into four abstract levels, summarized in Table 3.1. Starting from the top-level specification (most abstract), each level becomes more and more concrete until a gate-level description (least abstract) is reached. Partitioning the specification this way was inspired by Gordon's specification and verification of a simple 12-bit processor modeled in LCF_LSM [Gor81,Gor83]. Table 3.1 only lists the specifications published by RSRE. For some of the proofs the LCF_LSM specifications were recast into the derivative language HOL [Gor87] by consultants at Cambridge University. In the following Sections we focus on the original versions and treat the HOL translations along with the proofs in Chapter 4. The specification style used in the LCF_LSM specifications is not significantly altered by translation to HOL. It is also instructive to examine some of the deficiencies in the original LCF_LSM specifications, not all of which are solved in their HOL counterparts.

The VIPER Specification		
Level	**Language(s)**	**Reference**
Top	LCF_LSM	[Cul85]
Major State	LCF_LSM	[Cul86]
Block	LCF_LSM, Drawing; ELLA	[Pyg86]
Implementation	ELLA; HILO, FDL	N/A

Table 3.1: Levels in the VIPER Specification, in decreasing order of abstraction.

3.1 The Top Level

3.1.1 Overview

At the top level, VIPER was specified as a function in LCF_LSM named `NEXT`. Below we provide a schematic view of the operation of the function.

```
NEXT (ram,p,a,x,y,b,stop) -> (ram,p,a,x,y,b,stop)
```

`NEXT` takes the current programmer (visible) state (of seven components) and computes a new (seven component) programmer state based on the instruction referenced by the program counter. The programmer state consists of the registers `a`, `x`, and `y`, the program counter `p`, the multi-purpose flag `b`, the "processor-stopped" flag `stop`, and the data and I/O memory space `ram`. `NEXT` is an interpreter for a single instruction, which an assembly language programmer could use to predict the changes to the programmer state on the execution of one instruction. This is similar to the specification function for the FM8502, except that the FM8502 specification is an instruction interpreter for a processor executing a sequence of instructions.

Due to the very abstract nature of the top-level specification for VIPER, a number of the safety-critical features of the architecture do not appear. For example, the `NEXT` specification includes the `stop` flag, but the top-level specification function does not address all of the ways that real VIPER processors can be forced to halt, e.g., through a memory timeout. Instead, `NEXT` implicitly assumes a configuration in which the RESET, SINGLE-STEP, and ERROR inputs to the physical processor are never asserted, and all memory accesses complete normally. These assumptions are made explicit in the attempts to prove that the block-level implementation (Section 3.3) correctly implements the top-level specification. Since the top-level specification makes no account of any inputs to the system, the specification does not really model I/O, even though the I/O memory space is included in `NEXT`. In summary, when considering what has been proved about VIPER with respect to the high-level specification, one should realize that this specification does not cover every behavior, including several important safety-critical behaviors.

These deficiencies were known to RSRE, but addressing them would have required a radically different specification approach. Modeling memory timeout, for example, would have necessitated some notion of time in the high-level specification, as well as a nondeterministic input to model memory acknowledgment. At the time RSRE scientists began the work they did not feel confident in extending the specification methodology past the simple state-transition technique which was employed [Cul].

We do not mean to suggest that these are trivial problems; the formalization of peripheral behavior is an active area of research in hardware verification. For example, the FM8502 specification employs the notion of an *oracle,* a parameter

```
ADD32(r,m)
 ==
LET sum      = WORD34((VAL33(SIGNEXT r)) + (VAL33(SIGNEXT m))) IN
LET opposite = (EL 31 (BITS32 r)) XOR (EL 31 (BITS32 m)) IN
(TRIM34TO32 sum,                                          {sum}
 (EL 32 (BITS34 sum)) XOR opposite,                       {carry}
 ((EL 32 (BITS34 sum)) XOR ((EL 31 (BITS34 sum)))))       {overflow}
```

Figure 3.1: Top-level specification of addition with carry and overflow for the VIPER microprocessor.

to the low-level specification function which models the nondeterministic occurrence of RESET events and memory acknowledgments. The oracle abstraction made it possible to formally state and prove that the FM8502 implementation conforms to the top-level specification in the face of arbitrary delays from the memory. Another proof connected the programmer's view of resetting the machine with the hardware-level reset.

3.1.2 Arithmetic Specifications

The arithmetic behavior of VIPER is described in terms of operations on natural number abstractions of Boolean words. Interpreting these specifications is complicated by the fact that two equivalent representations are used for Boolean words in LCF_LSM and HOL: the built-in types `word`n, and lists of Boolean values. The top-level specification for addition in the ALU is presented in Figure 3.1. Hardware addition is defined as sign-extending 32-bit words to 33 bits (which involves an intermediate Boolean list), converting 33-bit words to numbers, adding the numbers, converting the sum to a 34-bit word, and then truncating this word to 32 bits. Computation of the carry and overflow are done with list forms of some of the intermediate results.

The `sum` computed by `ADD32` is a fairly straightforward definition of hardware addition, although the sign-extension of the addends is never explained.[1] More significant is the fact that nowhere in the formal work on VIPER is it ever demonstrated that `ADD32` is an abstraction for either signed or unsigned addition, nor is there ever any formal description of the significance of the `carry` and `overflow` outputs of `ADD32`. Whereas the informal specification states that "...overflow on either addition or subtraction causes the VIPER processor to stop ..." [Ker84], a statement about signed arithmetic, the formal specification never mentions signed numbers.[2] Thus the top-level specification leaves too

[1] An unconvincing justification appears in the specification [Cul85]. At the block level (Section 3.3), addition is defined by truncating the 33-bit result of a 32-bit addition.

[2] Signed integers are not a built-in type in LCF_LSM. They could have been modeled, for

much unspecified about the arithmetic operation of the processor. This does not mean that the top-level specification is wrong, simply that it is currently unable to support formal arithmetic analysis at any level higher than uninterpreted operations on Boolean words.

An example will clarify the above points. Imagine that a programmer had written a VIPER program whose abstract specification was to compute the sum of two signed integers. The programmer would like to *prove* the following:

> **If** `a`, `b`, and `c` are 32-bit words in the VIPER system, and **if** an instruction is executed such that `NEXT` places the `sum` output of `ADD32(a,b)` into `c`, and **if** the `overflow` output of `ADD32(a,b)` is not set, and **if** a, b, and c are the respective integer abstractions of `a`, `b` and `c` as 32-bit, 2's complement words, **then** $c = a + b$.

The programmer would further want to know that overflow is set only if $a + b$ is an integer which can not represented as a 32-bit, 2's complement word. The top-level specification of VIPER does not give the programmer the necessary tools to carry out this proof: no abstraction function from Boolean words to integers is provided, the result returned by `ADD32` has no abstract interpretation as either a signed or unsigned integer, and there is no formal definition of "representability" by which to judge the correctness of the overflow bit.

In contrast, the specification of the FM8502 provides a complete foundation for higher-level proofs. This was accomplished by proving a number of theorems relating the hardware operations of the FM8502 to abstract functions defining natural number and integer arithmetic. The complete specification of the FM8502 enabled the development of a formally verified system which includes a verified assembler and verified compiler [Moo88,You88].

3.2 The Major-State Machine

The next lower level of abstraction in the VIPER specification is the *major-state machine.* The major-state level abstracts VIPER as a cyclic graph whose nodes represent different phases of instruction execution, e.g., *instruction-fetch*, *perform-ALU-operation*, or *read-memory*. Each node in the major-state graph is modeled by an LCF_LSM function that specifies how the programmer state and internal state variables change as VIPER passes through the phases of instruction processing. The only concept of time at this level is the implicit ordering of the state transitions.

As specified by RSRE, the major state machine is *not* a single function in LCF_LSM, but rather a collection of functions with only an informal connection. In other words, RSRE's major-state specification cannot be used as a simulator for VIPER in the same way that the top-level specification can. Recasting the

example, as a sign/magnitude pair.

specification into HOL for purposes of formal proof (Section 4.3) necessitated the introduction of a new formal framework which is not present in the original specification, i.e., a framework which connects the individual state functions into a single function. Both the LCF_LSM and HOL versions continue to ignore external reset,[3] stop on external error, and memory timeout.

The major-state level was an attempt to bridge the gap between the high-level and low-level specifications, in order to simplify the formal correspondence proofs. Although two proofs (described in Chapter 4) involve this specification level, neither factors into the final correctness argument for VIPER hardware. We also point out in Section 4.3 that the original LCF_LSM specification was incorrect, in the sense that it was not equivalent to the top-level specification. There is no equivalent of the major-state level in the FM8502 specification.

3.3 The Block-Level Specification

A *block* is an abstract description of a major subsystem of the processor, e.g., the register file, instruction decoder, the ALU, etc. Each block has three equivalent specifications: RSRE's published LCF_LSM version, the HOL translation of the LCF_LSM used for formal proofs (Section 4.4), and an unpublished counterpart in the ELLA [MPT84] simulation language used for Intelligent Exhaustion simulation (Section 4.1). The block-level specification utilizes a register-transfer representation for sequential hardware. All of the registers are assumed to be activated by a common clock, and the block-level specification functions specify the behavior of the combinational logic. The latches never appear explicitly in the LCF_LSM or HOL versions. Instead, registers are modeled as parameters of the block-level functions. Whether an input argument or computed value is to be implemented as a latch is only informally specified by means of text and drawings. Latches *are* explicitly represented in the ELLA descriptions. In contrast, the FM8502 specification utilized a stylized hardware description methodology which makes apparent which parameters represent sequential state.

As with the major-state model, RSRE's LCF_LSM specification is partial; there is no formal description of the complete VIPER processor at the block level. This is a serious flaw, since an indispensable component of the original specification of VIPER is a schematic drawing that indicates the intended interconnection of the blocks. Connecting the block-level models to create a usable formal specification in HOL was a major hurdle in the high-level proof attempt (Section 4.4). The translation from LCF_LSM to HOL also uncovered a number of syntactic and typographical errors in the LCF_LSM specifications [Coh89a].

The block-level specification is also flawed by the need for the co-specification in ELLA. RSRE felt that it would have been prohibitively complicated to attempt gate-level verification against the block model using LCF_LSM [CP87],

[3] The specification does include a RESET state, but since external events are not modeled there is no way to enter that state.

and originally specified the block-level machine in ELLA. We comment further on the problems of the dual specification in Section 4.1.3.

3.4 The Implementation-Level Specification

The implementation-level specifications were produced by Marconi Electronic Devices, Ltd. and Plessey Company plc. The VIPER implementations were created from the block-level descriptions of VIPER. Two descriptions of each implementation actually exist: the circuit netlist in a proprietary CAD language, and translations of the netlists into ELLA which were used during Intelligent Exhaustion simulation (Section 4.1.1). The gate-level specifications are considered proprietary information, and are not publicly available.

The transfer from a formal specification language into an informal one, such as a hardware design language, is a weak link in the formal hardware verification process. The RSRE specification of VIPER is further weakened by the necessity of two informal translations: the translation of the gate-level models from proprietary languages to ELLA, and the translation of the ELLA descriptions of the block model to LCF_LSM and HOL. It is not possible to prove the correctness of either translation due to the lack of a formal theory relating the different languages.

In contrast, the low-level specification of the FM8502 can be viewed as a gate-level specification, in the sense that the specification can be formally expanded down to simple functions abstracting gates and registers. Although it has not been done, it would be possible to translate this formal expansion into a commercial CAD language in a single informal step. No special-purpose intermediate language was needed for the FM8502 verification because of the power of the Boyer-Moore theorem prover in dealing with induction, and the built-in Boolean decision procedure which simplified low-level proofs.

3.5 Specification Summary

The formal specification of VIPER is partitioned into four levels of abstraction. At the highest level, RSRE described VIPER with decreasingly abstract sets of functions in LCF_LSM. At the lowest level of abstraction are the gate-level models in proprietary CAD languages. The block-level and gate-level specifications are also given in the ELLA simulation language.

We noted several deficiencies in RSRE's specifications:

- There is no notion of external events in the top-level specification. Many of the safety-critical design features, such as externally forced error resets and memory timeouts, cannot be considered at this level.
- The top-level specification is incomplete with regard to the arithmetic operations which VIPER is said to provide. It impossible to use the top-level specification to prove abstract properties of programs running on VIPER computers.
- There is no complete formal description of the block-level machine (although Cohn later created one). The RSRE block-level specification only describes individual blocks; blocks are related to each other by informal text and drawings. The style used in the block-level specification was apparently geared toward low-level verification, without consideration for how it would be used in proofs at higher levels.
- There is no formal connection between the LCF_LSM and ELLA block-level specifications.

Chapter 4

Proofs and Proof Attempts

The VIPER verification effort includes proofs by two diverse groups over a three year period. The first proofs were carried out by RSRE without mechanical assistance. Later, Avra Cohn of Cambridge University was engaged to perform mechanical proofs using the HOL theorem prover. RSRE planned to verify VIPER in several steps, which when composed would constitute a formal verification of the processor. Three proof steps were to link the four abstract specification layers: Top $\Longleftrightarrow$ Major State, Major State $\Longleftrightarrow$ Block, and Block $\Longleftrightarrow$ Implementation. A complete HOL proof linking Top $\Longleftrightarrow$ Major-State was produced by Cohn. No HOL proof was attempted for the Major State $\Longleftrightarrow$ Block correspondence; instead, an HOL proof of Top $\Longleftrightarrow$ Block was attempted but never completed. There is no formal proof relating the block-level specification to the implementation, only an incomplete argument based on Intelligent Exhaustion simulation.

A briefly annotated, chronological list of the proofs and proof attempts can be found in Table 4.1. We discuss the nature and status of these proofs and proof attempts in the sections which follow. We follow the chronology, beginning at the implementation level and ending at the top level.

4.1 Block $\Longleftrightarrow$ Implementation

The first analyses of the VIPER specification were designed to show that the gate-level implementations proposed by the manufacturers correctly implemented the block-level specifications. Since the block-level specifications only deal with the combinational behavior of the blocks, this correspondence could have been demonstrated by exhaustive simulation. RSRE introduced a method called Intelligent Exhaustion (henceforth IE) which attempts to deliver the certainty of exhaustive simulation without explicit simulation of every possible input pattern. The remainder of this Section consists of an introduction to the IE tech-

VIPER Proof Efforts		
Levels	**Ref.**	**Notes**
Block $\Longleftrightarrow$ Implementation	[Pyg85]	ELLA specifications were analyzed by Intelligent Exhaustion simulation. The reference is to the method; the analyses are not documented.
Top $\Longleftrightarrow$ Major State	[CP85,Cul86]	A hand proof, later invalidated. The first reference is to the method, the second documents the proof attempt.
Top $\Longleftrightarrow$ Major State	[Coh87]	A machine-checked proof, using HOL versions of the (corrected) specifications.
Top $\Longleftrightarrow$ Block	[Coh89a]	A formal analysis in HOL; not a finished proof.

Table 4.1: Analyses of the VIPER specification, in chronological order.

nique, followed by a discussion of the application of IE to the VIPER verification and the problems with connecting the low-level proofs to the higher levels. Although the low-level specifications were extensively analyzed, this analysis does not constitute a formal proof of the Block $\Longleftrightarrow$ Implementation correspondence.

4.1.1 Intelligent Exhaustion Technique

Intelligent Exhaustion (IE) is a verification methodology which could be implemented in a number of high-level digital simulation systems. IE verification involves the simulation of one or more special purpose circuits encoded in a behavioral simulation language. In brief, gate-level and behavioral models of the circuit are simultaneously simulated, and the outputs of the two models are compared by a device which is also encoded in the simulation language.

If the behavioral and gate-level models agree on all possible inputs, then the two models are identical, although in general this would require the simulation of an exponential number of tests. IE exploits the fact that the values of functions are often determined by a proper subset of their input values. For example the output of a hardware AND gate will be low if one of the inputs is low; the value of the other input is irrelevant in that case. Irrelevant signal values can be modeled with *unknown* or *indeterminate* states, which are well-known

abstractions for digital logic simulation [BF76]. Since simulating a circuit with an indeterminate input is equivalent to simulating both high and low values for that input, intelligent analysis by the engineer can reduce the number of input patterns required to completely test the functionality of a combinational circuit. As long as it can be shown that every possible input has been considered, then IE is equivalent to exhaustive simulation. This condition is easy to check; for example a simple program can verify that a set of n-bit vectors which include indeterminate states actually covers all 2^n possibilities.

In practice, however, IE analyses are more complicated than suggested by the preceding paragraph. For example, if a device has several loosely correlated outputs, the most efficient verification may require a separate IE simulation for each output. Therefore it may also need to be shown that the IE simulations cover every output. This is not as straightforward as checking input coverage, since complete output coverage can only be determined by a careful examination of the behavioral source code. Another complication is that the behavior of a combinational circuit may only be specified for a subset of all possible inputs. For these cases it only needs to be shown that the inputs cover all interesting behavior. This requires a careful statement of exactly what the interesting cases are, and a corresponding proof that all of these cases have been considered.

Arithmetic circuits, like adders, also cause problems. For example, the IE verification of the carry output of an n-bit adder requires the simulation of $O(2^{n+2})$ patterns. While IE may provide economical verification of some types of circuits, it seems that the application of the method to arithmetic circuits will always be limited to relatively small devices; the combinatorial explosion is inescapable. RSRE also discovered cases where IE is too pessimistic, and would lead one to believe that correct circuits are incorrect [Pyg88].

4.1.2 Application to VIPER

RSRE implemented IE in the ELLA simulator, in part because the ELLA system had interfaces to the proprietary CAD systems used in the actual fabrication of VIPER. The block-level specification of VIPER was originally developed in ELLA, whereas the manufacturers provided gate-level realizations of the block-level designs in the proprietary CAD languages HILO and FDL. These gate-level designs were automatically translated to ELLA for IE simulation. The IE simulations uncovered errors in the initial designs "that would have been virtually impossible to find by simulation" [Pyg85].

The low-level verification of VIPER by Intelligent Exhaustion remains incomplete, however, for two important reasons. Most importantly, RSRE never proved that the input patterns used for IE simulation provided complete coverage of every possible case [Pyg]. The complete IE analysis of VIPER required the development and simulation of more than 6000 patterns [Pyg85]. There is no proof that these 6000 test patterns completely exercise the block-level specifications.

There is also no formalized argument that the implementations of the VIPER ALU meet the block-level specification. Recall that IE does not mitigate the combinatorial nature of verification of arithmetic circuits. In particular, IE simulation could not be performed on the entire 32-bit ALU. Instead, the ALU was partitioned into 8, 4-bit slices and associated "glue" logic for carry-lookahead. IE was only used to show that the 4-bit slices were correct; the correctness of the complete ALU is only supported by traditional engineering arguments. Since the later proof which begins at the block level (see Section 4.4) assumes a complete, 32-bit ALU, there remains an unverified gap between the block-level ALU specification, and the subcircuits that were analyzed with IE.

4.1.3 Interface to Higher Levels

Verification of the electronic block model of VIPER is not an end in itself, but has to be considered as a part of the overall verification effort. The block-level specification exists in two forms: the LCF_LSM which forms the basis for high-level verification, and the co-specification in ELLA. The LCF_LSM version was created by hand-translating ELLA to LCF_LSM. ELLA is *not* an LCF_LSM simulator; a number of subtle differences between the two representations are documented in [Pyg85]. For example the ELLA specification employs data types not available in LCF_LSM. There are also a number of ramifications of the presence of indeterminate states in ELLA with respect to interpreting LCF_LSM specifications.

The use of different languages in a verification effort increases the chance of errors, either in the translation process, or errors caused by differences in the semantics of the languages. In the case of VIPER, conjectures verified by IE simulation of the ELLA models are used as axioms in the high-level proofs based on the LCF_LSM versions of the specification, in spite of the fact that there is no formal connection between the two languages.

In the case of the formal systems built on the FM8502, a uniform logical theory (the Boyer-Moore logic) was used from the gate-level descriptions all the way up to the specification of a high-level programming language. All of the correctness proofs were checked with the Boyer-Moore theorem prover, which ensures that all of the proof obligations have been discharged.

4.1.4 Summary

We are not convinced that RSRE researchers have formally verified the gate-level implementation of VIPER. It was never verified that the IE test patterns provided complete coverage, and there is no proof that the complete ALU designs are correct. Even if the IE analysis of VIPER were to be completed there still remains the gap between the LCF_LSM and ELLA specifications, bridged only by informal arguments. The VIPER verification would be much more believable and satisfying if a uniform theory had been used at every level, and if

RSRE had completed all of the proof obligations of the methodology in use.

4.2 Top $\Longleftrightarrow$ Major-State (LCF_LSM)

Cullyer reported a proof, worked by hand, which showed that the major-state machine correctly implements the top-level VIPER specification. This proof attempt was carried out completely within the LCF_LSM framework, as Cullyer used the LCF_LSM specifications. Cohn later demonstrated that the major-state specification was wrong, thus invalidating this work (see Section 4.3). This failed attempt is important if for no other reason than to demonstrate the desirability of mechanically checked proofs.

4.3 Top $\Longleftrightarrow$ Major-State (HOL)

Desiring machine-checked, formal proofs of correctness, and feeling that it lacked the requisite experience in-house, RSRE contracted the Hardware Verification group at Cambridge University to produce high-level proofs of correctness for VIPER. The first proof, which showed the correspondence between the top-level specification and the major-state machine, was published in 1987 [Coh87]. Cohn reported that this proof required six months to complete, and involved over one million primitive inferences. Due to a change of plans (see Section 4.4) this proof is only of historical interest, and does not play any role in the formal correctness argument for VIPER.

Cohn began by translating the LCF_LSM specifications into the higher-order logic HOL [Gor87]. Since HOL was derived from LCF_LSM, the translation of the top-level and major-state specification from LCF_LSM into HOL was straightforward. The proof of equivalence also required Cohn to augment the original specifications in two respects. Recall that the major-state machine was modeled as a set of functions representing different phases of instruction execution, without any formal connection between the states. Cohn formalized the connection by combining all of the state transition functions into a single function. Cohn also formalized a notion of time, where each unit of time represented one transition in the major-state machine. Again, proofs at this level ignore the possibility of reset and memory timeout since these eventualities are not represented in the specifications.

Several blatant errors in the LCF_LSM major-state machine specification were uncovered during the proof. During the fetch cycle, for example, the check for illegal instructions was specified to be made against the *previous* contents of the instruction register, not against the instruction just fetched. Since the major-state machine was an abstraction created for proof, and not an integral part of the design, these errors are not manifested in the actual devices.

4.4 Top $\Longleftrightarrow$ Block

Cohn next considered the verification of the major-state machine with respect to the block-level specification. This approach was abandoned for technical reasons. Instead, Cohn attempted to prove the correspondence of the block-level specification with respect to the top-level specification directly. This effort did not result in a finished proof that the block-level machine correctly implements the VIPER top-level specification.

Cohn's first challenge was to convert the block-level specification into a form that was amenable to formal analysis. Recall from Section 3.3 that the block-level specification consists of a set of LCF_LSM functions, schematic drawings, and text. Using these sources Cohn created an HOL function which is believed to faithfully capture the intention of the VIPER designers for the block-level machine. This function is more complex than the VIPER major-state specification, as each major state is further divided into a number of minor states. The HOL block-level specification function was then expanded, using definitions and simplification lemmas, to produce what is essentially a symbolic execution of the block-level machine for each possible VIPER instruction schema. Cohn also proved that the expansion did cover every instruction schema.

To finish the proof, it would be necessary to prove that the results computed by the block-level specification match the top-level specification. Paraphrasing Cohn, this step was not taken because

1. Resources were limited, and the research results would not justify the effort.

2. No one had developed an HOL theory of bit-string operations, which is critical for completing the proof.

3. Relating the high-level results to the block-level results might require intricate knowledge of the design to understand exactly how the low-level design implements the specification.

Top:

```
WORDS32
(V
 (TL
  (TL
   (BITS34
    (WORDS34
     ((VAL33 (SIGNEXT (REGSELECT_ABBR(areg,xreg,yreg,preg,ram)))) +
      (VAL33 (SIGNEXT (MEM_ABBR(ram,preg))))))))))
```

Block:

```
WORDS32
(V
 (SEG
  (0,31)
  (BITS33
   (WORD33
    ((VAL32 (REGSELECT_ABBR(areg,xreg,yreg,preg,ram))) +
     (VAL32 (MEM_ABBR(ram,preg))))))))
```

Figure 4.1: Derived computations for an ADD operation for the VIPER top-level and block-level specifications.

Instead, Cohn delivered the symbolic expansion to the VIPER design team, who informally analyzed it and found no obvious errors. Quoting from [Coh89a] (the italics are Cohn's):

> For the non-ALU sequences, the results are not very complicated and they *appear* to be as intended. Some of the arithmetic-logic paths are also apparently correct. Others, in particular the additions, subtractions, and comparisons, are neither obviously correct nor incorrect, and require further study. So far, there do not seem to be any definitely incorrect results, but obviously, since the formal analysis ends at this point, there very well could be. For that reason, a great deal of care should be taken in describing the Viper block model as being 'verified'; it has to date only been *analyzed* ...and inspected ...

Cohn goes on to give examples of the types of obstacles remaining. For example, Figure 4.1 displays the result computed by the ALU during an addition, first from the top-level specification, then from the block-level specification. The informal argument is simple: both expressions compute the low-order 32 bits from addition of two 32-bit words. In the expression from the top level, the

result is obtained by truncating the 34-bit result of a sign-extended addition; at the block level, a 33-bit value is truncated. This is an example where it is fairly "obvious" that the two levels compute the same value. Cohn also gives other examples in which the correspondence between levels is far less clear. Unfortunately, the lemmas about bit-vector operations that were needed to prove the correspondence were never developed.

The status of this proof attempt is unchanged since the publication of [Coh89a]. There is no indication that this proof will ever be completed.

4.5 Proof Summary

Several attempts were made to prove correspondence between the various levels in the VIPER specification. These efforts were undertaken by RSRE and researchers at Cambridge University over a three-year period. So far there is no complete proof that the gate-level specifications implements the top-level specifications. RSRE used Intelligent Exhaustion simulation to analyze the gate-level implementations. The most satisfying formal work on VIPER is Cohn's proof that the major-state machine correctly implements the top-level specification. Unfortunately, this proof has no formal connection with any of the other proof attempts, except that the same HOL top-level specification was used for the Top $\Longleftrightarrow$ Block analysis. The final attempt to prove that the block-level model is equivalent to the top-level specification was prematurely terminated, but carried to the point that the results are at least plausibly correct. None of these efforts address resetting the machine, memory timeout, forced error, or single step modes.

A satisfactory completion of this work would require at least:

- The adoption of a more rigorous framework for gate-level verification than that provided by Intelligent Exhaustion.
- A formal proof that the gate-level ALU, which was partitioned in order to be analyzed by Intelligent Exhaustion, correctly implements the block level specification.
- The completion of Cohn's Block Level $\Longleftrightarrow$ Top Level proof.
- The specification of integers, along with integer operation appearing in the top-level specification.

Until these conditions are met, the claims that VIPER has been formally verified are unfounded.

Chapter 5

Conclusion

VIPER has been verified in the traditional hardware engineering sense, i.e., extensively simulated and informally checked. Before we would be satisfied that VIPER was verified in the formal sense, we would expect to see complete formal specifications at every hierarchical level, from the top-level instruction interpreter down to the gate-level design. Accompanying these specifications should be proofs which showed that the gate-level design correctly implements the top-level machine. These conditions could never have been met using RSRE's original specification and proof methodology. We pointed out several of these deficiencies, including the use of the informal simulation language ELLA for the gate-level specification, the lack of rigor in the Intelligent Exhaustion analyses, and the incomplete nature of RSRE's block-level specification. These points, and the fact that the attempt to prove the correspondence between the top-level and block-level machines in HOL is incomplete, lead us to the conclusion that VIPER has not been formally verified.

The VIPER work serves as a case study for several technology transfer issues, clearly demonstrating a need for improved formal systems. Although verification methods are an active area of research, application of these methods must eventually be placed in the hands of hardware designers, or specially trained engineers who are intimately familiar with the designs. It is significant that Hunt was an experienced hardware designer prior to the successful verification of the FM8501. Cohn was not, which only added to the problems caused by attempting an after-the-fact verification of an unfamiliar specification.

There is also a need to improve the Boolean decision procedures in mechanical reasoning systems, in order to avoid the necessity for special purpose methods such as Intelligent Exhaustion. For example, Bryant [Bry86] recently introduced a set of algorithms which provide extremely fast verification of Boolean circuits. It should be possible to soundly implement similar procedures in currently existing systems.

The VIPER project also pointed out the need for extensive libraries of lem-

mas about arithmetic and logical operations on the basic data types found in hardware specifications. This absence, and the presumed difficulty of creating these lemmas in HOL was one of the reasons that Cohn's last proof attempt was prematurely terminated. Something also clearly needs to be done about the large amount of detailed interaction required to complete a complex hardware proof using a mechanical assistant.

There are different degrees of rigor possible when applying formal methods to hardware design: hand-written specifications, hand proofs, mechanically recorded specifications, and mechanical proofs. The VIPER effort employed all of these techniques with varying degrees of success, but it is clear that the VIPER team was more thorough at specifying the abstract behavior of VIPER than traditional engineering techniques would allow. Without the use of formal techniques, the proofs of correctness could not have even been attempted. We are encouraged by the use of formal techniques in VIPER, as their use demonstrates what we believe to be a new paradigm in computer hardware specification and validation.

In conclusion, we admire the efforts of the groups at RSRE and Cambridge who took on a formidable verification task. We don't consider the shortcomings of the VIPER project as a pessimistic indication of the future of formal hardware verification. We are optimistic that the problems uncovered in the VIPER effort can be overcome, and that this hard-won experience will benefit future work.

Acknowledgments

We are pleased to acknowledge Avra Cohn, John Cullyer, Mike Gordon, Steven Johnson, John Kershaw, and Clive Pygott, who freely gave of their time and hospitality during the preparation of this report.

Bibliography

[BF76] Melvin A. Bruer and Authur D. Friedman. *Diagnosis and Reliable Design of Digital Systems.* Computer Science Press, Potomac, MD, 1976.

[Bry86] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[Coh87] Avra Cohn. *A Proof of Correctness of the VIPER Microprocessor: The First Level.* Technical Report 104, University of Cambridge, Computer Laboratory, January 1987.

[Coh89a] Avra Cohn. Correctness properties of the viper block model: the second level. In *Current Trends in Hardware Verification and Automatic Theorem Proving*, pages 1–91, Springer-Verlag, New York, 1989.

[Coh89b] Avra Cohn. The notion of proof in hardware verification. *Journal of Automated Reasoning*, 5(2):127–139, June 1989.

[CP85] W J Cullyer and C H Pygott. *Hardware Proofs Using LCF_LSM and ELLA.* Memorandum 3832, Royal Signals and Radar Establishment, Malvern, Worcestershire (United Kingdom), September 1985.

[CP87] W. J. Cullyer and C. H. Pygott. Application of Formal Methods to the VIPER Microprocessor. *IEE Proceedings*, 134, Pt. E(3):133–141, May 1987.

[Cul] W J Cullyer, Private Communication, April 1989.

[Cul85] W. J. Cullyer. *VIPER Microprocessor: Formal Specification.* Report 85013, Royal Signals and Radar Establishment, Malvern, Worcestershire (United Kingdom), October 1985.

[Cul86] W J Cullyer. *VIPER - Correspondence Between the Specification and the "Major State Machine".* Report 86004, Royal Signals and Radar Establishment, Malvern, Worcestershire (United Kingdom), January 1986.

[Dyk88] Digby A. Dyke. One-day seminars (advertisement). *Safety Net*, 1(1):10, April/May/June 1988. Published by Viper Technologies Ltd., PO Box 79, Worcester WR1 2PX, England.

[Fet88] James H. Fetzer. Program verification: the very idea. *Communications of the ACM*, 31(9):1048–1063, September 1988.

[Gor81] M. Gordon. *LCF_LSM.* Technical Report 41, University of Cambridge, Computer Laboratory, 1981.

[Gor83] Mike Gordon. *Proving a Computer Correct.* Technical Report 42, University of Cambridge, Computer Laboratory, 1983.

[Gor87] M. Gordon. *HOL: A Proof Generating System for Higher-Order Logic.* Technical Report 103, University of Cambridge, Computer Laboratory, 1987.

[HP87] M P Halbert and G P Pink. *The Design of Fault Detecting and Fault-Tolerant Computer Systems Based on the Viper Microprocessor.* Technical Report C2610, Cambridge Consultants Limited, June 1987. Science Park, Milton Road, Cambridge CB4 4DW, England.

[Ker84] J Kershaw. *VIPER: A Microprocessor for Safety Critical Applications.* Memorandum 3754, Royal Signals and Radar Establishment, Malvern, Worcestershire (United Kingdom), December 1984.

[Moo88] J Strother Moore. *A Mechanically Verified Language Implementation.* Technical Report 30, Computational Logic, Inc., September 1988.

[MPT84] J D Morrison, N E Peeling, and T L Thorp. Ella: hardware description or specification? In *Proceedings IEEE International Conference, CAD-84*, Santa Clara, November 1984.

[Pyg] Clive Pygott, Private Communication, April 1989.

[Pyg85] C H Pygott. *Formal Proof of Correspondence Between the Specification of a Hardware Module and its Gate Level Implementation.* Report 85002, Royal Signals and Radar Establishment, Malvern, Worcestershire (United Kingdom), November 1985.

[Pyg86] C H Pygott. *VIPER: The Electronic Block Model.* Report 86008, Royal Signals and Radar Establishment, Malvern, Worcestershire (United Kingdom), July 1986.

[Pyg87] C H Pygott. *Specification of the VIPER1A Micro-processor: Issue 2.* Divisional Memo (CC2) 412-87, Royal Signals and Radar Establishment, Malvern, Worcestershire (United Kingdom), November 1987.

[Pyg88] C H Pygott. NODEN_HDL: An Engineering Approach to Hardware Verification. In G. J. Milne, editor, *The Fusion of Hardware Design and Verification*, Elsevier Science Publishers, 1988.

[War87] Warren A. Hunt, Jr. *The Mechanical Verification of a Microprocessor Design.* Technical Report CLI-6, Computational Logic, Inc., 1717 West Sixth Street, Suite 290, Austin, TX 78703, 1987.

[You88] William D. Young. *A Verified Code Generator for a Subset of Gypsy.* Technical Report 33, Computational Logic, Inc., 1988. Ph.D. Thesis, University of Texas at Austin.