Testing the FM9001 Microprocessor

Kenneth L. Albin, Bishop C. Brock,
Warren A. Hunt, Jr., Lawrence M. Smith

Technical Report 90 January 6, 1995

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

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

EMAIL: hunt@cli.com

This work was supported in part at Computational Logic, Inc. and by the Defense Advanced Research Projects Agency, ARPA Orders 6082 and 9151. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of Computational Logic, Inc.

Abstract

The FM9001 is a general-purpose 32-bit microprocessor that was fabricated for Computational Logic, Inc., by LSI Logic, Inc., as an ASIC. Prior to fabrication, the FM9001 netlist was formally and mechanically proved to implement its user-level specification by Brock and Hunt using the Nqthm theorem prover. In this report, we document our post-fabrication testing of the physical device. The testing included both executing FM9001 machine code and also low-level testing with a Tektronix LV500 chip tester. To date, all tests have confirmed that the FM9001 behaves as formally specified.

1 Introduction

The FM9001 is a general purpose CMOS, 32-bit microprocessor that was fabricated for us by LSI Logic in 1991. Prior to fabrication, the netlist design of the FM9001 that we later supplied to LSI Logic was formally proven, using the mechanical theorem prover Nqthm [7, 5], to implement its user-level, i.e., machine-code level, specification. (See the report "The FM9001 Microprocessor Proof" [7] for the details of this formal proof effort.) In the present report, we describe a series of post-fabrication physical tests that have increased our confidence that the manufactured device does indeed meet its netlist specification.

One purpose of testing the fabricated devices, of which we received fifty, was to insure that these integrated circuits were free of physical defects. Testing is necessary because even given a logically perfect netlist, one may encounter problems with the physical manufacturing process, e.g., dirt, broken wires, and cracked packages.

A second reason for testing was to validate our assumptions that we made about the LSI Logic primitives when we formalized the behavior of the primitive elements used in making the FM9001. Our lowest level formal model was one that defined such LSI Logic primitives as an AND gate and a one-bit latch. Even though LSI Logic provides a databook [11] of gate-level primitives, their actual gate array devices are composed of columns of N-type and P-type transistors. It was important to us to discover whether the translation process from our gate-level model to LSI Logic's Network Description Language (NDL) and then on to transistors all worked as we expected.

Finally, we wanted to validate our effort in producing a verified hardware and software computing platform. The CLI "short stack" [15] includes the FM9001 microprocessor as a base, upon which the Piton assembler [13, 14], the Micro-Gypsy compiler [22, 21, 23], and the Micro-Nqthm compiler [8] have been proven to operate correctly. We find that our having actually completed this verification exercise to be compelling evidence [12] that we do not have to settle for hardware and software validated only with the conventional test-oriented approaches.

Upon first receiving the fabricated FM9001 integrated circuits we evaluated

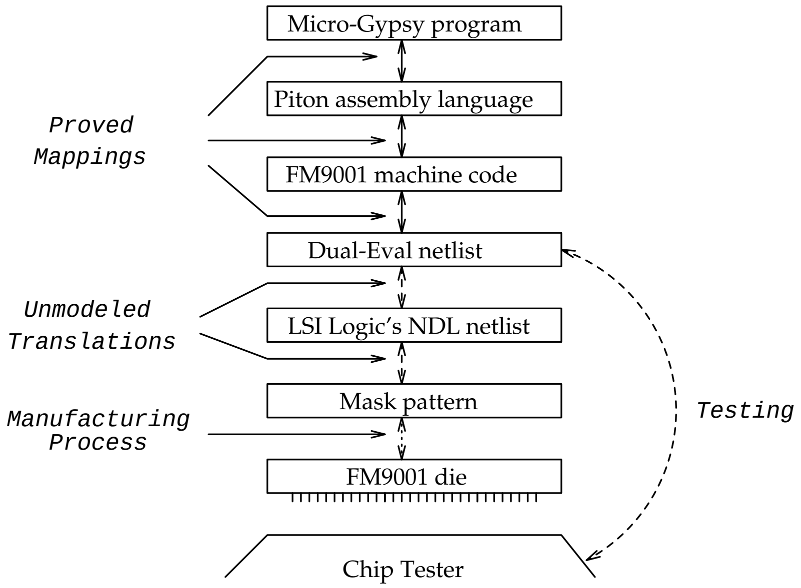

Figure 1: Proved and Tested Portions of the CLI Stack

the timing of the various control signals by stimulating the FM9001 with a fixed instruction. Using an oscilloscope, we observed the low-level timing of FM9001 control signals. See Section 3. Next we built a single-board computer which contained an FM9001, ROM, RAM, a USART, and an interface to a logic analyzer, through which we were able to monitor all FM9001 signals. Using this single-board computer we have run a number of pieces of software. See Section 4. To obtain pin-level control and testing of the FM9001, we also connected the FM9001 to a Tektronix LV500 [17], which is an industrial "stand-alone verification tester for ASICs." See Section 5. Some additional testing of the FM9001 has been performed at Indiana University at Bloomington. See Section 6.

Testing is an important component of our CLI stack. It provides assurance that our formalization of the netlist of the FM9001 (in the `DUAL-EVAL` formal hardware description language[6]) is a good clock-cycle by clock-cycle model of the operation of the FM9001. Testing is always required because the manufacturing process can introduce physical flaws. The testing we have performed here is primarily aimed at checking the translation process from `DUAL-EVAL` netlists into LSI Logic's design environment. These various layers are shown in Figure 1.

The physical act of testing is dependent upon how we physically access the device under test, that is, how we supply signals to and read results from the device. Therefore, as we discuss the testing below, we first identify the physical test jig used because the nature of the jig determines the kind of tests we can run and the kinds of measurements we can make.

2 Interacting with LSI Logic

The very first physical testing of fabricated FM9001s was actually performed by LSI Logic as part of its "contract" with the customer. That contract stipulates that the customer must provide both a netlist and also some means of testing the result of fabricating the netlist. The test information we provided included both inputs, called "test vectors" and also some simulation control language statements, which were used to feed the test vectors both to LSI Logic's simulator and to its test equipment. The results of the physical testing of the fabricated device must match the results predicated by running those same tests on LSI Logic's software simulator. LSI Logic only delivers parts for which the supplied physical tests yield results identical to those predicted by LSI Logic's simulator.

The test vectors provided to LSI Logic were of several varieties: functional, stuck-at-fault, RAM-specific, and parametric. Initially, we concentrated on producing test vectors that would reveal manufacturing defects. On the first occasion that we attempted to transfer the entire FM9001 design to LSI Logic for manufacture, we had no "functional" test vectors. LSI Logic was appalled because the typical ASIC design flow was to specify a netlist along with a series of test vectors and their expected results. The post-fabrication test vectors that we had created were usually the last thing produced by designers. However, in our case, the post-fabrication stuck-at tests were the first set of test vectors we produced, because we did not feel there was a need for functional test vectors. That is, our verification approach suggested to us that our netlist design was logically correct, but LSI Logic insisted that we provide functional test vectors.

We actually produced the test vectors using our `DUAL-EVAL` simulator and a parallel-fault simulator we developed. We produced test vectors that satisfied our fault simulator before we translated both the netlist and the test vectors into a LSI Logic compatible format. Our `DUAL-EVAL` hardware description language [9, 6] was designed in such a way that it was extremely simple to translate designs into the NDL. In fact, NDL was based upon TEGAS, a simulation control language that Bishop Brock helped develop. To validate our translation process, we executed test vectors on our `DUAL-EVAL` simulator and on LSI Logic's simulator, and compared the results on thousands of test vectors for tens of thousands of clock cycles.

Finally, before we proceed to describe the physical testing that we performed, it is perhaps worth noting that it is also straightforward to translate `DUAL-EVAL` netlists into VHDL[10], and we have in fact also, post-fabrication, simulated a VHDL version of the FM9001 netlist under the Vantage VHDL simulator, using LSI-Logic-supplied VHDL models of LSI Logic's primitives.

3 Signal Timing Validation

The first physical test jig we built allowed us only to program the databus with resistors. See Figure 2 for a picture of this jig. That is, what the processor read for every data access was set by 32 resistors connected either to +5 V (logic true) or to ground (logic false). This allowed a single instruction to be repetitively executed so we could carefully observe the timing of signals being generated by the processor using an oscilloscope. In addition, we connected a series of LEDs to the address bus so we could observe the changing of the address bus. For simple programs, the address bus could be observed to be counting up in binary, as the program counter was incremented after each instruction. In addition we had multiple clock drivers on this board to permit the clock rate to be changed between ten, twenty, and twenty-five megahertz. This board also contained "buttons" so we could reset and hold the processor.

Through the use of different loads on the output signals to the processor we observed the rise and fall times of the various control and data signals emanating from the processor. This was done by changing the amount of resistance and capacitance on various processor signals and observing the result on an oscilloscope. This kind of testing is performed principally to discover the analog behavior of the pad drivers. Further, measurements were made to "characterize" the relationships between the control signals that the processor generates and the externally supplied clock. In effect, we were trying to insure that the processor exhibited the very low-level analog behavior predicted by the LSI Logic simulator. As far as we could tell, the processor satisfied the expected behavior.

After validating the basic analog behavior of the signal pins of the FM9001, we constructed a single-board computer. This single board computer is a wire-wrap board with pre-loaded pins upon which we put the following components:

- 1 FM9001 microprocessor
- 4 32x8 byte-wide static RAMs
- 4 32Kx8 byte-wide EPROMs
- 1 National Semiconductor 16550 USART
- 1 RS232 signal level interface
- 4 PALs
- 3 clocks
  - an 8 megahertz clock for the USART
  - a 10 megahertz clock for the FM9001
  - a 25 megahertz clock for determining memory delay

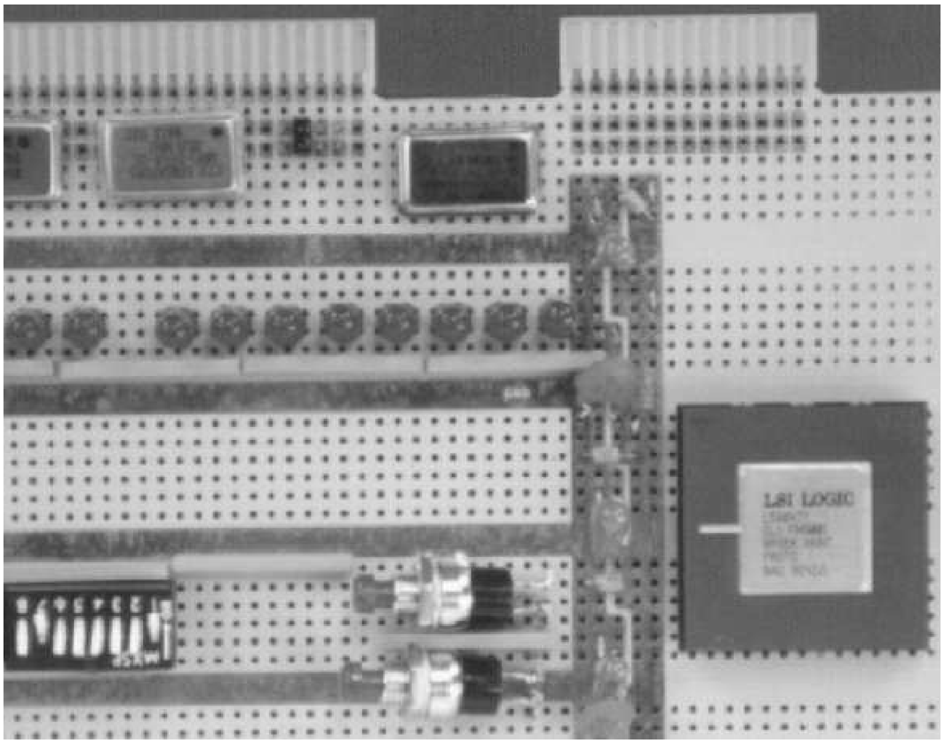

Figure 2: The Analog Signal Testing Jig

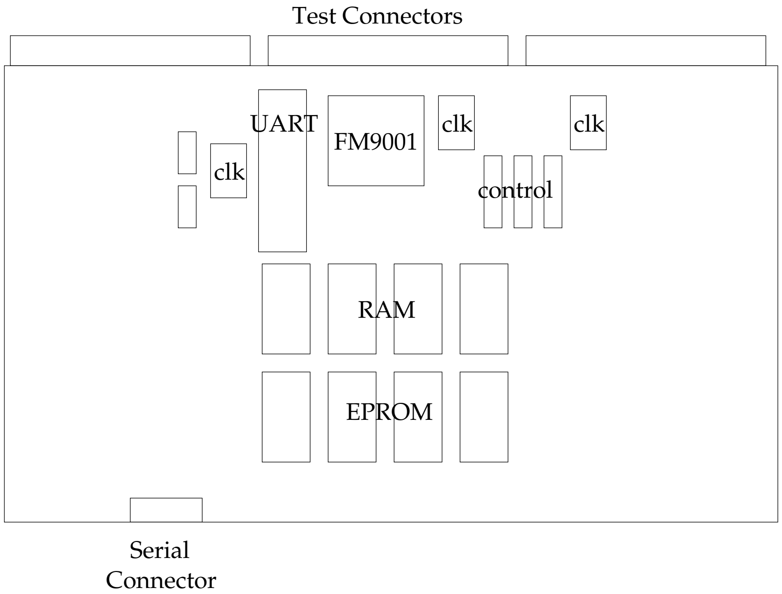

Figure 3: Pictorial Diagram of Single-Board Computer

The board is connected to a +5 volt source to supply power to the logic elements and also a ±12 volt source to supply power to the RS232 interface. Some of these major pieces are shown in the pictorial diagram Figure 3.

In addition this board contains two large connectors whose pins are connected to all the FM9001 data and control signals. We use these connectors to interface the single-board computer to a Tektronix DAS 9100 Digital Analysis System, a typical logic analyzer.

The EPROM (ROM) contains an FM9001 monitor program that permits other FM9001 machine code programs to be loaded over the serial interface into the RAM from a controlling workstation, which can been seen in Figure 4. A picture of a part of the single-board computer can be found in Figure 5. After a program has been loaded into RAM, the ROM monitor program can then be instructed to "jump" to the loaded code as a subroutine. Upon completion of a subroutine execution, control is returned to the ROM monitor program, and then the contents of the register file and the memory may be determined with suitable commands to the ROM-based program.

At any time, the signal values on the FM9001 microprocessor may be read every clock cycle by the attached logic analyzer. This testing is nonintrusive. That is, the logic analyzer merely "listens" to all the FM9001 signals. We checked that various programs caused the correct pin-level behavior on the FM9001 by running the programs using the monitor control program and simultaneously recording the

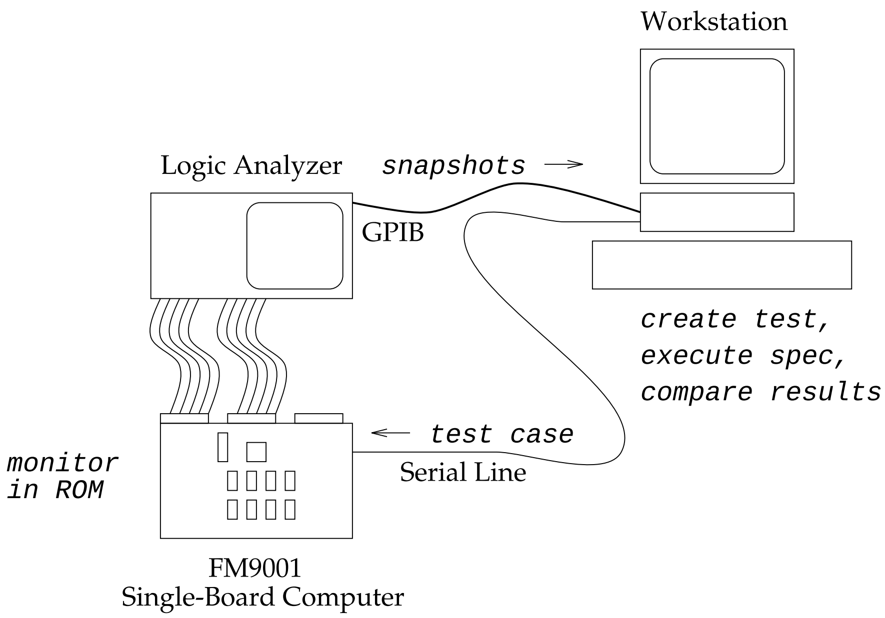

Figure 4: Single Board Testing Environment

signal values for all FM9001 signals. After accumulating this data with the logic analyzer, the identical program was loaded into our `DUAL-EVAL` based simulator and the same pin-level signal values were collected. We then compared the predicted pin-level behavior with the observed pin-level behavior.

4 The FM9001 Single-Board Computer

Below is an abbreviated example of the information recorded by the logic analyzer during the execution of a single FM9001 instruction, a move instruction:

```
4000002c 00000000 w-d fetch0
4000002d ffffffff r-- fetch1
4000002d 10e0fc02 rs- fetch2
4000002d 10e0fc02 rs- fetch3
4000002d 10e0fc02 rs- fetch3
4000002d 10e0fc02 rs- fetch3
4000002d 10e0fc02 rsd fetch3
4000002d 10e0fc02 rsd fetch3
4000002d fffffdfa r-d decode
4000002d ffffffff r--   rega
4000002d ffffffff r-- write0
4000002e 00000000 w-- write1
4000002e 00000000 ws- write2
4000002e 00000000 ws- write3
4000002e 00000000 ws- write3
4000002e 00000000 ws- write3
4000002e 00000000 wsd write3
4000002e 00000000 wsd write3
```

The columns, in order, indicate the contents of the address bus, data bus, read/write/strobe signals, and the decoded internal major control state (microinstruction). Each of these samples were taken during consecutive clock cycles. These values are actually read as binary values of individual signal lines. The display seen above has been "formatted" into hexadecimal numbers, symbols and state names. This formatting is part of the programming done to the DAS 9100 so as to identify logic-analyzer inputs with FM9001 signal lines.

A number of programs were loaded into the RAM and executed. This testing was not particularly complete because the DAS 9100 had small buffers permitting us only to record 256 clock cycles of signal information per test. After this information was gathered, it was then copied to the workstation so that we could compare it with the values predicted by the `DUAL-EVAL` simulation of the FM9001 netlist. Altogether, information for approximately several thousand clock cycles was gathered

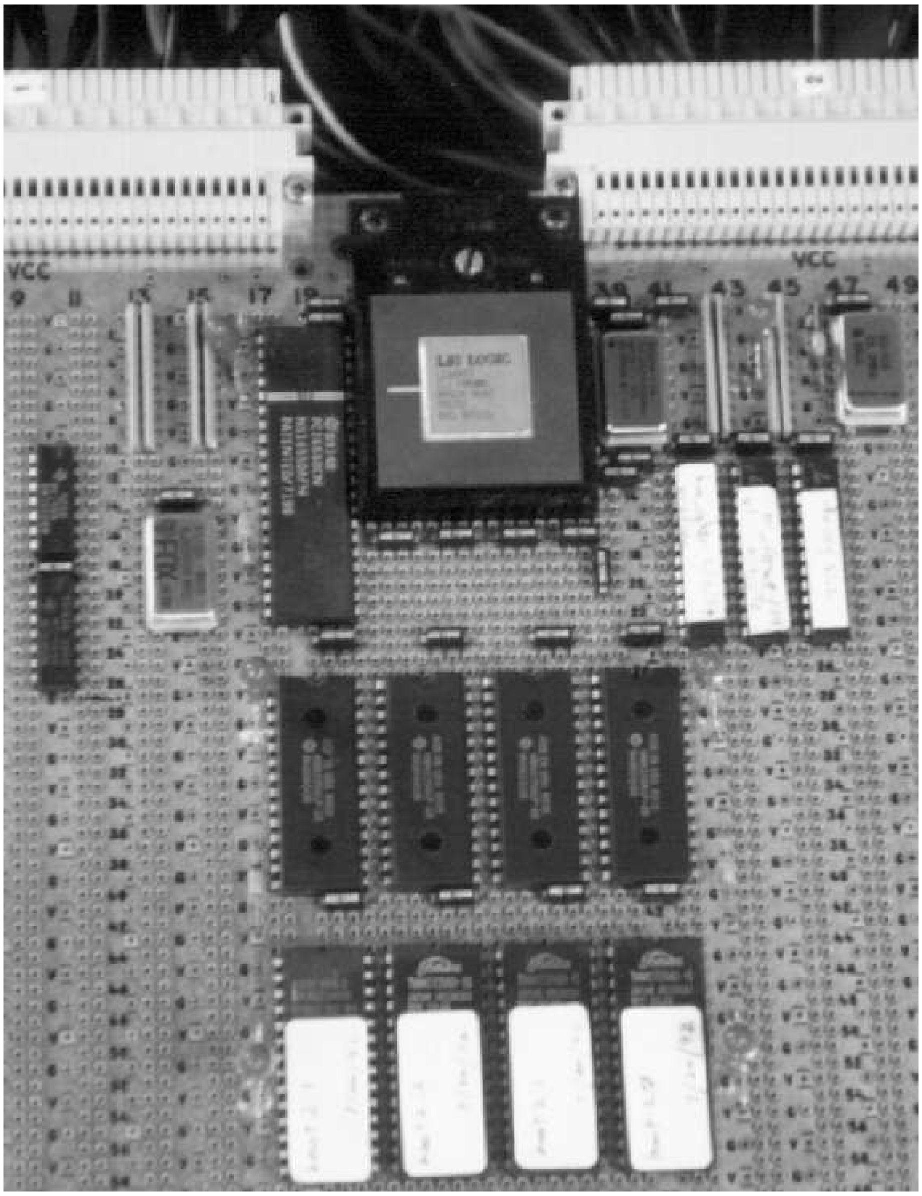

Figure 5: The Single-Board Computer

and compared in this fashion. Further details concerning the construction of the single-board computer may be found in Ken Albin's report [2], which includes the schematic wiring diagram, the monitor program, and the workstation-to-monitor interface program. Subsequently, for work on real-time software verification, the board has been adapted to include a simple switch and a light, both of which are memory mapped.

The single-board computer has been used to execute the following example programs:

- various utility programs and the ROM-based monitor program;
- a Piton program that adds multi-word integers [13];
- a Piton program that plays the game Nim, proved to win if possible [19];
- a factorial program;
- a program [18] that computes the Takeuchi function, which is commonly used to benchmark Lisp systems; and
- a real-time program that senses a switch and turns on a light when the switch is depressed [20].

In all these cases, the FM9001 microprocessor worked as expected. Perhaps the most satisfying of these software tests is the Nim game-playing program. It was most rewarding to see this rather subtle Piton program, which has been formally proved, with Nqthm, to win if possible, actually win, when possible, while executing on a physical FM9001—and winning within a proven real-time performance envelope.

There are about ten "layers" of formal, mathematical abstraction between the user interface to the Nim game and the interpretation of the FM9001 netlist by the `DUAL-EVAL` semantics. It is a source of real joy, and relief, to see the "real" world behave as predicted by all this formal mathematics. This is the only example we know of a proven stack of hardware and software components.

5 Testing with the Tektronix LV500

The most thorough testing of the FM9001 microprocessor was performed by interfacing an FM9001 microprocessor to a Tektronix LV500 chip tester. This device is similar to the logic analyzer used above, but can also provide pin-level stimulation to the device under test. That is, all inputs are driven solely by the chip tester, and all outputs are monitored on a clock-by-clock cycle basis. The LV500 permits programming of the clock, sampling of signals several times per clock cycle, and complete control over data input lines, including full flexibility in setting the hold, reset, test, and memory signals.

Figure 6: The LV500 Test Jig with the FM9001

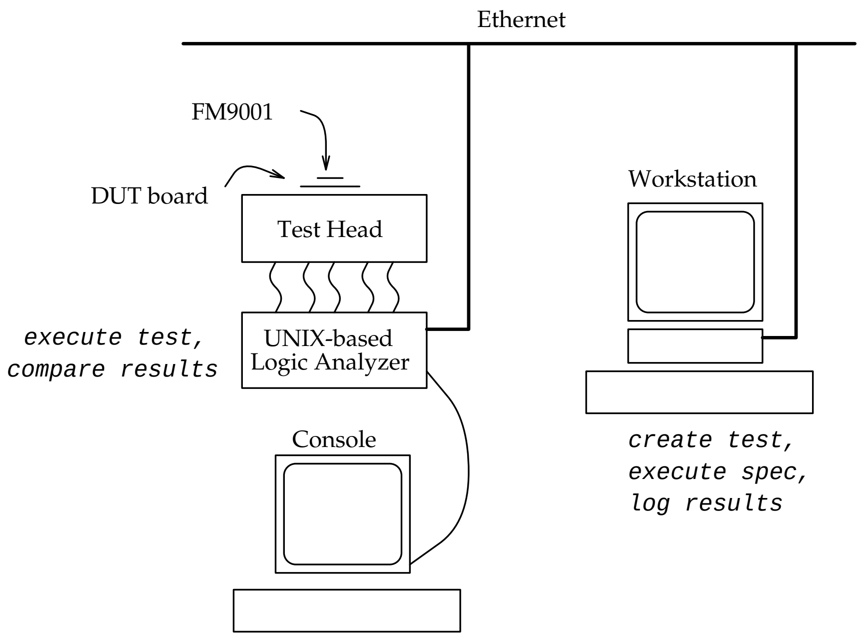

Figure 7: The LV500 Testing Setup

By interfacing the FM9001 to the chip tester, it was possible for us to randomly generate inputs for every FM9001 signal and record the processor responding to this stimulation. That is, we could simulate any kind of memory, with any contents desired. This was not possible with the single-board computer, because it contains only a modest amount of RAM and ROM. Further, the FM9001 can be tested in ways that would never be used in any typical application. For instance, it is possible to randomly enable and disable the FM9001 test inputs, causing the processor to change between normal operational modes and the test modes on a clock-cycle by clock-cycle basis. To take advantage of this kind of capability, we generated random sequences of inputs for the LV500 to provide to the FM9001. We took these same randomly generated sequences and provided them to the `DUAL-EVAL` simulator. We then could compare every signal output from a physical FM9001 and with the output predicted by our lowest-level formal model.

With the LV500, we undertook a substantial series of random tests of the FM9001. We now describe the major tests that were performed:

- A set of hand-crafted test vectors were created to insure that the FM9001 was correctly interfaced to the LV500.

- A small set of 130 input test vectors was created to force each input to both logic false and logic true and to cause each output to produce both logic false and logic true. Additionally the processor was forced to enter each major control state.

- A test suite of 12,000 random input test vectors was generated. The reset and test inputs were kept disabled, but all other inputs were allowed to vary randomly. The processor was given these input vectors after it was driven into its reset state. These same 12,000 vectors were given to `DUAL-EVAL`, which predicted the identical results that were observed by the LV500. Note that the "pseudo interrupt" facility of the FM9001 was exercised by these tests, which varied the input signals that determine which register is taken as holding the program counter.

- Another test suite of over 1,000,000 random input test vectors was generated. In this case, we allowed all inputs to vary randomly except the `TEST-REGFILE-` input. The processor was given these input vectors after it was driven into its reset state. These same million-plus vectors were given to `DUAL-EVAL`, which predicted the identical results that were observed by the LV500. This is our most general testing procedure.

The testing of the FM9001 was not completely straightforward because the FM9001 design contains a level-sensitive register file. Avoiding a level-sensitive implementation of our register file by using standard one-bit scanned latches would have simplified testing but would have tripled the size of the register file implementation and necessitated a shift to a substantially larger die. The `DUAL-EVAL` model used to specify the netlist-level implementation does not permit the formalization of level-sensitive devices. We implemented the register file by surrounding it with registers that operate within `DUAL-EVAL`'s modeling capability, and we believe that this entire register file implementation works exactly like a collection of 32x16 one-bit latches. Because the level-sensitive register file was a single LSI Logic primitive, we were required to provide direct testing inputs: `TEST-REGFILE-` and `DISABLE-REGFILE-`.

In attempting to derive our most general testing procedure, we realized that not all inputs could be varied on every clock cycle. In particular, we discovered that we could not vary the `TEST-REGFILE-` input, which combinationally controls the write-enable signal to the level-sensitive 16x32 register file. This was obvious in retrospect. Our `DUAL-EVAL` model of the four gates that control the register-file, write-enable signal ignores the `TEST-REGFILE-` input. Thus our `DUAL-EVAL` model does not accurately predict the operation when the `TEST-REGFILE-` signal is varied.

The `DISABLE-REGFILE-` input also controls the write-enable line to the level-sensitive 16x32 register file, but this is only partially reflected in the `DUAL-EVAL` model. The actual implementation of the write-enable control logic is "gated-clock" combinational logic. Therefore, to ensure that the `DUAL-EVAL` model correctly tracks the operation of the physical register file, we must be careful when, between successive clock pulses, we allow this value to change; otherwise, we can cause spurious writes into the register file. To get results from the FM9001 that were identical to those predicted by the `DUAL-EVAL` model, we had to insure that the `DISABLE-REGFILE-` signal for each clock pulse was stable during the previous cycle's clock-high time.[1] To make the effect of the `DISABLE-REGFILE-` signal mirror the `DUAL-EVAL` model, we moved its stimulation period "backward" in time to satisfy this timing requirement.

This timing issue reveals the need for the study of formal hardware models that include low-level timing information. The `DUAL-EVAL` model is for Mealy machines. Because we chose to use a level-sensitive register file, a very common component, we exposed the limitations of such simple hardware models. Certainly, simulation environments, such as VHDL, do provide low-level timing simulation capability, but they do not provide any facility other than simulation to insure the correctness of hardware designs. Given a formal model for a VHDL-like language one could verify such low-level timing properties.

Extensive details of the LV500 testing work may be found in Lawrence M. Smith's report "FM9001 Model Validation on the LV500 Logic Verifier" [16].

6 Independent Testing at Indiana University

Bhaskar Bose of Indiana University, working under the supervision of Steven Johnson, studied the FM9001 design with an eye towards attempting to automatically synthesize an FM9001 design from its behavioral specification using the DDD synthesis tool [3, 4]. As a part of the preparation for this synthesis project, William Hunt (Warren's brother) of the VLSI laboratory of the Computer Science Department at Indiana University interfaced a fabricated FM9001 to the Indiana University Logic Engine. This provided a way to test the FM9001 in a manner that was somewhat more general than we could with our single-board computer but not so thoroughly as is possible with the LV500 chip tester. In fact, the first machine code program testing was performed at Indiana, before our single-board system was built.

7 Conclusion

We have rather extensively tested fabricated FM9001 microprocessors in a variety of settings. Thus far, this formally verified microprocessor has behaved as predicted.

[1] Also, the `DISABLE-REGFILE-` signal must be stable for at least ten nanoseconds after the falling edge of the clock to account for internal gate delays.

References

[1] Kenneth L. Albin and Lawrence M. Smith. FM9001 Model Validation. Internal Report 298, Computational Logic, 1994.

[2] Kenneth L. Albin. The FM9001 Single-Board Computer. Internal Note 303, Computational Logic, 1994.

[3] Bhaskar Bose. DDD—A Transformation System for Digital Design Derivation. Technical Report 331, Computer Science Department, Indiana University, May, 1991.

[4] Bhaskar Bose. DDD—FM9001: Derivation of a Verified Microprocessor. Ph. D. Dissertation, Indiana University, 1994.

[5] R.S. Boyer and J S. Moore. *A Computational Logic Handbook*. Academic Press, Boston, 1988.

[6] Bishop C. Brock, Warren A. Hunt, Jr., and William D. Young. Introduction to a Formally Defined Hardware Description Language. *Theorem Provers in Circuit Design*, V. Stavridou, T. Melham, and R. Boute, eds., North-Holland, pp. 3–35, 1992.

[7] Bishop C. Brock, Warren A. Hunt, Jr., and Matt Kaufmann. The FM9001 Microprocessor Proof. Technical Report 86, Computational Logic, December, 1994.

[8] Arthur D. Flatau. *A Verified Implementation of an Applicative language with Dynamic Storage Allocation*. Ph. D. Dissertation, The University of Texas at Austin, December, 1992. Also available (minus certain appendices) from Computational Logic as CLI Technical Report 83.

[9] Warren A. Hunt, Jr. and Bishop Brock. A Formal HDL and Its Use in the FM9001 Verification. In C.A.R. Hoare and M.J.C. Gordon, editors, *Mechanized Reasoning and Hardware Design*, pages 35–48. Prentice-Hall International Series in Computer Science, Englewood Cliffs, N.J., 1992.

[10] IEEE. Standard VHDL Language Reference Manual, ANSI/IEEE Std 1076-1993. IEEE, 345 East 47th St., NY, NY, June 6, 1994.

[11] LSI LOGIC. 1.5-Micron Array-Based Products Databook. LSI Logic Corporation, Milpitas, CA. 1990.

[12] John McCarthy. It's Past Time for Practical Computer Checked Proofs of Program Correctness. *Computational Logic*, Symposium Proceedings, Brussels, Esprit Basic Research Series, DG XIII, Commission of the European Communities, Springer-Verlag, November 1990.

[13] J Strother Moore. PITON: A Verified Assembly Level Language. Technical Report 22, Computational Logic, 1988. To appear as a book in the Kluwer's series on automated reasoning under the title *Piton: A Mechanically Verified Assembly-Level Language.*

[14] J Strother Moore. A Mechanically Verified Language Implementation. *Journal of Automated Reasoning*, 5(4):493–518, December 1989. Also published as CLI Technical Report 30.

[15] J Strother Moore, et al. Special Issue on System Verification. *Journal of Automated Reasoning*, Vol. 5, No. 4, pp. 409-530, 1989.

[16] Lawrence M. Smith. FM9001 Model Validation on the LV500 Logic Verifier. Internal Report 299, Computational Logic, 1994.

[17] Tektronix, Inc. LV500/SE Operator's Manual. Tektronix, Inc. Walker Road Industrial Park, Beaverton, OR 97076. 1992.

[18] Matthew Wilding. Using the Fabricated FM9001. Internal Note 260, Computational Logic, August, 1993.

[19] Matthew Wilding. A Mechanically Verified Application for a Mechanically Verified Environment. *Fifth Conference on Computer-Aided Verification*, Lecture Notes in Computer Science, LNCS 697, pp. 268–279, Springer-Verlag, 1993. Also available as Technical Report 78, Computational Logic, 1994.

[20] Matthew Wilding. A Real-time Programmer's Model of the FM9001. Internal Note 302, Computational Logic, December, 1994.

[21] William D. Young. A Mechanically Verified Code Generator. *Journal of Automated Reasoning*, Vol. 5, Number 4, (December, 1989), pp. 493–518. Also available from Computational Logic as Technical Report 37.

[22] William D. Young. A Verified Code Generator for a Subset of Gypsy. Ph. D. Dissertation, University of Texas at Austin, 1988.

[23] William D. Young. Verified Compilation in Micro-Gypsy. *Proceedings of the Software Testing, Analysis and Verification Symposium*, Key West Florida, December, 1989, pp. 20–26.