An Overview of the Formal Specification and Verification of the FM9001 Microprocessor

Bishop C. Brock and Warren A. Hunt, Jr.
brock@cli.com, hunt@cli.com

Fall, 1994

The use of mathematical logic for modeling and reasoning about hardware designs promises assurance of circuit correctness beyond what is available from current state-of-practice techniques. The development and use of formal techniques in hardware design is spreading [5, 13, 16, 19, 20, 25, 28, 36, 45]. This approach to circuit validation is known generally as *hardware verification*. Circuits with the complexity of microprocessors [5, 30, 35, 46] have been given mathematical specifications, and their designs have been proved to implement their specifications. Yet, the transfer of hardware verification techniques to commercial engineering practice has been hampered by such factors as the use of non-standard notations, inaccessibility of the tools, and the significant mathematical sophistication required to use these approaches. In addition, formal techniques have been directed at only selected aspects of the design process. Important hardware characteristics such as testability and I/O behavior have been largely neglected by the formal hardware modeling and verification community.

We have attempted to address some of these issues by considering the formal specification, verification, and physical implementation of the FM9001 microprocessor. The FM9001 is a general-purpose 32-bit microprocessor whose gate-level netlist design implementation was developed using a theorem-proving environment in conjunction with a traditional CAD system. The behavioral specification for the FM9001, the definition of the hardware description language (HDL) used to represent the design of the FM9001, the simulator for the HDL, and the verification of the FM9001 were all carried out using the Boyer-Moore theorem-proving system Nqthm [9]. The FM9001 netlist was mechanically translated to LSI Logic's Netlist Description Language and implemented by LSI Logic, Inc., as a CMOS gate-array. Rigorous testing has not uncovered any situation where the manufactured device fails to meet its specification. The FM9001 also serves as the target for the verified assembler, Piton [42], which in turn serves as the target of the verified μ-Gypsy compiler [49]. This document presents the details of the FM9001 development, its specification, and its verification.

1 RESULTS

We believe that a significant result of the FM9001 microprocessor study is that we have shown it is possible to formally model a microprocessor at several levels of abstraction and then prove that these different levels are formally related. The FM9001 was formally modeled at four levels of abstraction as shown in Figure 1.

- A high-level behavioral (user-level) model that operates as an instruction interpreter for FM9001 programs;
- A Boolean (two-valued) model that "mirrors" the intermediate, register-transfer model;
- An intermediate, (four-valued) register-transfer model that links the low- and high-level specifications for verification purposes; and
- A complete gate-level (netlist) model presented in the `DUAL-EVAL` HDL.

The high-level and the register-transfer models are captured as executable Nqthm logic functions. The semantics for the gate-level implementation, which is described by an Nqthm constant, is given by our hardware description language simulator, `DUAL-EVAL`. Even though we often prove that parts of the register-transfer level are equivalent to the "Boolean level", we do not have a complete Boolean level; this is an artifact of our verification approach. The gate-level model includes all of the logic required to physically construct the FM9001, including the test logic. To have LSI Logic manufacture the FM9001, we translated only the `DUAL-EVAL` netlist into LSI Logic's Network Description Language (NDL) and provided the test vectors required by LSI Logic, Inc.

The implementation of the FM9001 is described as a `DUAL-EVAL` netlist containing a reference for every primitive gate, latch, register, I/O buffer, and wire that is required for the FM9001 implementation. Both the syntax and the semantics of this netlist have been mechanically checked: the netlist is well-formed and it has been proven to implement the FM9001 specification. We have a formal definition of acceptable netlists; that is, we have a predicate that identifies well-formed netlists.

The proof of correctness of the FM9001 gate-level design consists of three major lemmas. First, we have shown that the FM9001 can be forced to a known state, i.e., reset, by a suitable sequence of inputs. This proof is carried out using the gate-level models; the concept of resetting the machine does not appear at the behavioral level of the instruction interpreter. Second, we show that given a set of initial conditions, the gate-level model correctly implements the high-level specification. This proof involves both time and data abstractions to link the clock-cycle based semantics of the gate-level FM9001 model with the high-level instruction interpreter. Finally, we show that the state at the end of the reset sequence satisfies the initial conditions for the previous lemma. This sequence of results provides the formal basis for believing that behavior specified by the

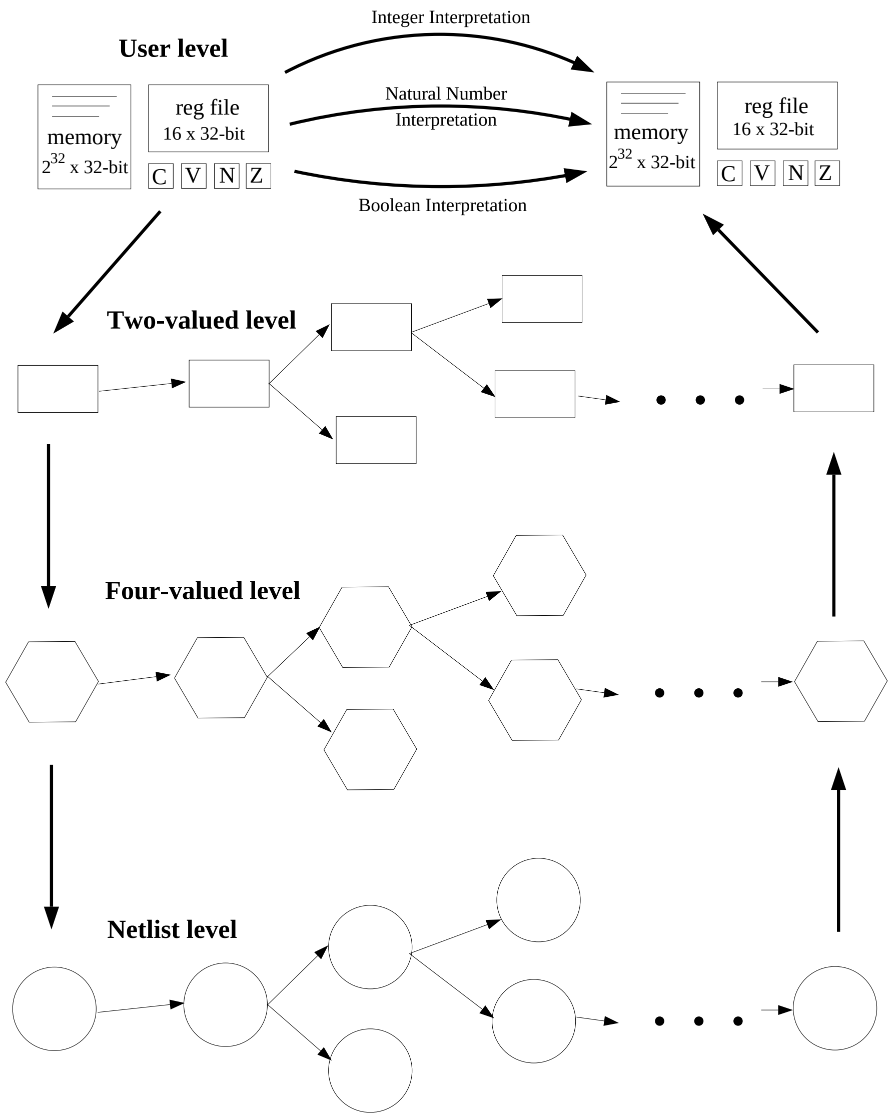


Figure 1: Specification Levels

instruction interpreter is implemented by the gate-level design. Of course, any physical flaws could invalidate everything. We did, during the design of the FM9001, make provisions for thoroughly testing the manufactured devices.

Physically, the FM9001 is a general-purpose CMOS 32-bit microprocessor implemented as a gate array. The FM9001 features a two-address programming architecture, five addressing modes, multi-processor bus capability, and extensive arithmetic flag support. Instructions require, assuming fast enough external memory, from five to 17 cycles to execute; typical instructions require five to nine clock cycles to execute. At 25 MHz the FM9001 only requires 70 mA of power exclusive of I/O drivers; the static design allows the clock rate to be reduced for lower power consumption. All internal flip-flops are connected by a scan chain for testing purposes, except for the register-file latches which have dedicated test signals. We have been testing the FM9001 chips for three years, and we have never found a case where the behavior of FM9001 chips deviates from its specifications.

The proof of correctness of the FM9001 microprocessor model was mechanically checked by the Nqthm theorem prover. Our FM9001 proof script contains 2957 entries that expand into 4851 theorem-prover events. The total time required for Nqthm to check the proof of the FM9001's implementation with respect to its specification is about 24 hours on a Sun Microsystems 3/60. The statement of the three correctness lemmas takes almost 10,000 lines to "prettyprint".

	Prettyprinted Lines
User-level semantics of FM9001	915
Statement of correctness lemmas	197
Semantics of HDL	3459
FM9001 implementation description	3479
Existential witness for the clock	1942

Total:	9992

Given the size of the formula that we needed to manipulate, we considered it necessary to have mechanical assistance. Our choice of the Nqthm system was based in our belief that it could successfully discharge the rather large proof obligations we faced during our effort.

To get a rough measure of the size of the FM9001 proof, J Moore modified the theorem prover to count its smallest step. By adopting a suitable collection of derived inference rules, a formal proof can be constructed with one line for each step. Among our one-step rules are instantiation, *modus ponens*, generalized equality substitution, tautology recognition, and cross-multiplication and addition of inequalities.

The sum of the proof steps in all the formal proofs done for FM9001 is 6,100,315. That is, more than six million lines of formal proof were "virtually constructed." It would be easy to increase this number by reducing the size of our proof steps. For example, the cost of substitution-of-equals-for-equals in HOL [27] is not constant (it is one in our count of proof steps) but is proportional to the depth at which the target occurrence is found. Similarly, we recognize very large `IF` expressions as tautologies but charge only one step. We have expressions containing more than 50 `IF` expressions through which there are more than 10^9 branches—but because the branches were pruned dynamically, the theorem prover did not have to explore them all.

The definition of the FM9001 architecture, its design, the definition of the `DUAL-EVAL` netlist simulator, the verification of the three lemmas mentioned above, generation of the test vectors, and the conversion of the FM9001 `DUAL-EVAL` netlist to LSI Logic's NDL, required about three man-years of effort. About an additional six months of effort was devoted to testing the manufactured FM9001. The time required to document this effort is not included in the estimates just given.

2 HISTORY OF THE PROJECT

The FM9001 effort began in earnest in November 1989 when two of us (Brock and Hunt) defined a simulator, which we called `DUAL-EVAL`, for a language that was general enough to represent the design of a microprocessor along with its memory system. Representing circuits as data was a turning point in our efforts at hardware specification. Previously Hunt [30, 31] had represented hardware with functions in the Boyer-Moore logic. We had also investigated representing circuits as predicates [10], in the tradition of the HOL hardware verification community [17]. Although both of these approaches were successfully used in significant hardware verification examples, neither approach permits direct, formal operations on the circuit descriptions.

Immediately prior to the definition of the `DUAL-EVAL` simulator and language recognizer we had defined several different hardware description languages which differed in representational power. For example, the language/simulator `HEVAL` [11] was restricted to combinational logic circuits only. We also defined `WAVE-EVAL`, which worked similarly to an event-driven simulator. `WAVE-EVAL` was more general than `DUAL-EVAL`, but we considered its semantics too difficult to use given the size of our task. Once we had defined `DUAL-EVAL`, we developed techniques to simplify the specification and verification of circuits described in the `DUAL-EVAL` language, at least until we had automated the process sufficiently to verify the FM9001 design. In actuality, the evolving FM9001 design was a "technology driver" for the development of the `DUAL-EVAL` language.

Another part of the HDL development was the evolution of its language recognizer. At first, the recognizer checked for proper syntax and absence of combinational circuit loops; however, over time the recognizer was extended to check for loading and fanout violations, timing properties, restrictions on names, and a host of other properties. We used feedback from the LSI Logic toolset to restrict the class of circuits we admitted by changing the circuit recognizer to disallow any problem encountered. Later, we produced a much nicer circuit recognizer that produced error messages; the original recognizer only returned true or false. Another facet of the design process was ensuring that circuits could be tested. We implemented a parallel stuck-at fault simulator that was used during our design process and the creation of test vectors. Often during the design of the FM9001 logic, we would first check the testability of the circuit before we went to the effort to verify its correctness. We also used the fault simulator to identify redundant logic.

The FM9001 netlist, along with the necessary test vectors and signal pad assignments, was passed off to LSI Logic, Inc., at the end of July of 1991. We received FM9001 microprocessor chips some six weeks later. After receiving the chip, we built a very simple test jig that allowed any single instruction to executed repeatedly so we could observe the pin-level timing of the chip. In 1992, we built a single-board computer using the FM9001 microprocessor; this board had two serial ports and a monitor program. This board was constructed in such a way that all of the FM9001 microprocessor signals could be attached to a logic analyzer. We executed programs by downloading them from a workstation, and then observed the behavior of the FM9001 on our logic analyzer. We ran numerous programs to check to see if the manufactured part worked as we expected it to [1].

We were still limited from checking the complete functionality of the FM9001 because, for instance, this single-board computer did not have a complete 2^{32} word memory. In 1993, we obtained a Tektronics chip tester. By early 1994, we had interfaced the FM9001 to the chip tester; we then proceeded to test the FM9001 in all operational modes. We are able to generate random, pin-level inputs and check that the operation of the FM9001 chip perfectly matched what was predicted by the `DUAL-EVAL` simulator.[1] This testing has been completely automated and has been ongoing for some time now. At this point, we have found no unexpected operational differences.

The FM9001 behavioral and design specifications, the hardware description language and its semantics, the large majority of the FM9001 correctness proof, and the preparation of the FM9001 for fabrication were performed entirely by Bishop Brock and Warren Hunt. Matt Kaufmann proved the ALU interpretation lemmas and assisted with proving the reset properties. Ann Siebert did much of the work on our recognizer for well-formed circuit descriptions that produces simple error messages, and Ken Albin built the FM9001 single-board computer.

[1] There are two dedicated test pins for the level-sensitive register file that cannot be randomly set; we knew this prior to our testing. Also, the clock and power inputs cannot be randomly changed — the power supply must be stable and the clock input must be a suitable square wave.

3 INSTRUCTION SET

We now present a short, informal description of the instruction set of the processor, and the pinout of the actual chip. All of our formal specifications represent varying degrees of abstraction about the behavior of the physical device. This preliminary introduction of the FM9001 is a foundation on which to build the formal specifications and proofs.

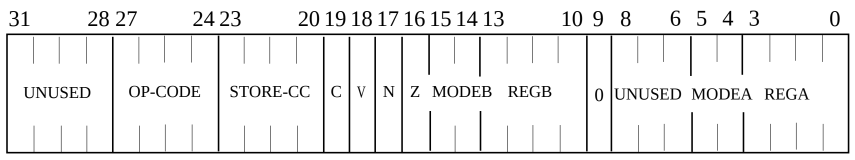
The FM9001 is a general-purpose 32-bit microprocessor featuring a two-address programming architecture with five addressing modes, 16 general-purpose 32-bit registers, 15 arithmetic and logical operations, extensive arithmetic flag support, and a memory interface that supports multiple processors. Figure 2 shows the two FM9001 instruction formats, a two-address instruction and an immediate datum instruction. The two formats only differ in the origin of operand A. In the immediate datum case the nine-bit datum in the instruction word is sign-extended to 32-bits. Otherwise, operand A is selected in the same manner as operand B. Except for the immediate datum mode, operand A and operand B are selected by any of the following modes: register direct, register indirect, register indirect with pre-increment, and register indirect with post-increment. Thus, operand A has five addressing modes and operand B has four, and every addressing mode for both operand A and B works with every instruction. Except for the immediate addressing mode for operand A there is no provision for data sizes other than 32-bit words.

One of the general-purpose registers is used as the program counter, and the choice of the PC is programmable externally. The number of the register used as the PC is read from dedicated pins during reset and hold operations on the FM9001. The PC number can be neither read nor written by any instruction. This feature provides a very primitive interrupt facility: The processor can be halted by a hold request, the PC number changed, and then processing resumed on a different instruction stream. Since branching instructions and certain types of immediate data instructions depend on knowledge of which register is being used as the PC, programs written for the FM9001 will depend on assumptions about the hardware and execution environment of the FM9001. Generally, we select register 15 as the program counter.

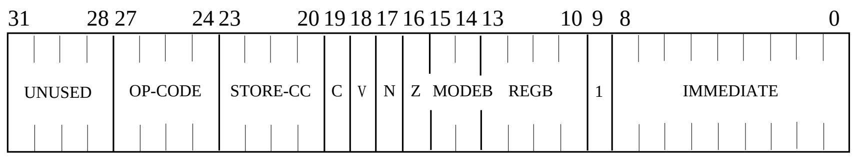
The 15 different arithmetic and logical instructions are listed in Figure 2, as are the conditions that determine if the computed result is to be stored. Each instruction also contains four bits that determine individually whether the arithmetic flags carry, overflow, zero, and negative, are updated.[2] During instruction execution the PC is incremented by 1 before the calculation of the effective address of the A operand, and all side effects to the A operand register (pre-decrement or post-increment) are completed before the computation of the effective address of the B operand.

[2]This feature was added to the FM8502 processor to simplify the proofs of correctness of the PITON interpreter [42]. We retained this feature in the FM9001.

TWO-ADDRESS MODE



IMMEDIATE DATUM MODE



MODE	**OPERAND**	**DESCRIPTION**
00	Rn	Register Direct
01	(Rn)	Register Indirect
10	-(Rn)	Register Indirect Pre-decrement
11	(Rn)+	Register Indirect Post-increment

OP-CODE	**MNEMONIC**	**OPERATION**	**STORE-CC**	**MNEMONIC**	**CONDITION**
0000	MOVE	b <- a	0000	CC	~C
0001	INC	b <- a + 1	0001	CS	C
0010	ADDC	b <- a + b + c	0010	VC	~V
0011	ADD	b <- b + a	0011	VS	V
0100	NEG	b <- 0 - a	0100	PL	~N
0101	DEC	b <- a - 1	0101	MI	N
0110	SUBB	b <- b - a - c	0110	NE	~Z
0111	SUB	b <- b - a	0111	EQ	Z
1000	ROR	b <- c,a >> 1	1000	HI	~C & ~Z
1001	ASR	b <- a >> 1	1001	LS	C | Z
1010	LSR	b <- a >> 1	1010	GE	(N & V) | (~N & ~V)
1011	XOR	b <- b XOR a	1011	LT	(N & ~V) | (~N & V)
1100	OR	b <- b OR a	1100	GT	(N & V & ~Z) | (~N & ~V & ~Z)
1101	AND	b <- b AND a	1101	LE	Z | (N & ~V) | (~N & V)
1110	NOT	b <- NOT a	1110	T	True
1111	M15	b <- a	1111	F	False

Figure 2: FM9001 Instruction Word Format

Some simple examples will help clarify some of the features of the architecture and instruction set. Since the results of any instruction may be conditionally stored, and the register used as the PC is always valid as a destination, there are no differences between control and data operations. For example a "Jump" instruction can be encoded as an unconditional MOVE (i.e., STORE-CC = 1110) of the target address to the PC. A conditional jump is obtained simply by encoding a condition other than "True" in the STORE-CC field of the instruction. For example "Jump if Carry clear" is encoded as a MOVE instruction where the target address is the PC, with STORE-CC = 0000. Non-storing instructions (i.e., STORE-CC = 1111) can be used to side effect the flags without destroying the destination. For example, a "Compare" can be effected by a non-storing SUB that sets the condition codes (WOP-CODE = 0111, CVNZ=1111, STORE-CC=1111).

The FM9001 specification was derived from the FM8502 specification [31]. There are several important differences.

- The FM8502 reserves register 15 for the PC. The PC is hardware selectable in the FM9001.
- The FM9001 has the ability to conditionally store every result. Only conditional moves were permitted in the FM8502.
- The FM9001 performs the post-increment operations as each operand is fetched. The FM8502 post-increment operations were done after both operands were fetched and the result was stored.
- The FM9001 has an immediate data mode for operand A; the FM8502 did not have this feature.

The ALU operations remain the same as do the fetch-execute cycle. We did this to ensure that the Piton assembler [44] for the FM8502 could be easily transported to the FM9001.

4 SIGNAL GROUPS

A pictorial diagram of the FM9001 signal groups appears as Figure 3. The implementation of the FM9001 microprocessor does not include a memory; some of the signals are used to interface to an external memory. Likewise, there are signals devoted to testing the FM9001. Not all signals are relevant to all the levels of formal specification. For example, the power and ground pins VSS and VDD never appear in *any* of our specifications; we always assume that the power is turned on! All of the other signals appear in our netlist-level specification, although the clock input CLK is given special treatment. The high-level specification of the FM9001 does not include any signals, it is a self-contained specification that includes the memory.

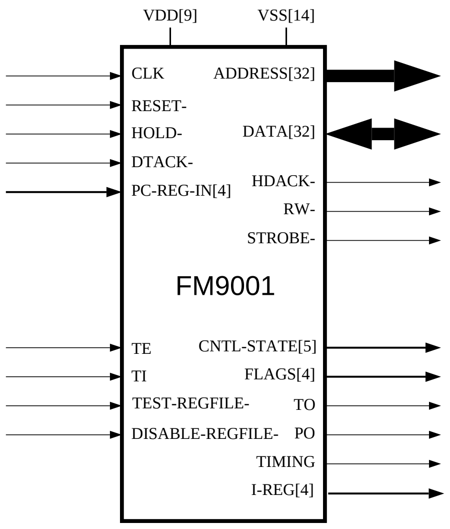


Figure 3: Signal Groups of the FM9001

The following is a brief description of each signal or signal group that is important for the normal operation of the FM9001. These descriptions should in no way be construed as complete or formal specifications. For signal groups the number of signals appears in square brackets, e.g., FLAGS[4].

ADDRESS [**32**] The 32-bit tri-state address bus. It can be set to float while HDACK– is asserted.

CLK The clock input.

DATA [**32**] The tri-state 32-bit bidirectional data bus. The FM9001 drives the bus only during memory-write operations; otherwise, the FM9001 sets the bus to float.

DTACK– Active low, synchronous memory data acknowledgment.

HDACK– Active low, synchronous hold acknowledgment. The processor acknowledges a hold request by asserting HDACK–, disabling the bus drivers on the ADDRESS and DATA pins, and cycles in a no-op state until HOLD– is deasserted. The processor loads the PC register number during each cycle that HDACK– is asserted; therefore, the PC register can be changed.

HOLD– Active low, synchronous hold request. When HOLD– is asserted, and if it remains asserted until acknowledged by HDACK–, the processor will halt at the end of the execution of the current instruction. Whenever the processor is halted (i.e., HDACK– is asserted), the DATA, ADDRESS, STROBE–, and RW– pins are set to float.

PC-REG-IN [4] The number of the register to be used as the program counter. These pins are read only during reset and hold operations.

RESET– Active low, synchronous reset. RESET– must be asserted for one full clock cycle. After one clock cycle, the FM9001 will enter a reset state, and remain there as long as RESET– is asserted. When RESET– is deasserted, the processor begins a short reset sequence which clears all registers and loads the PC register number. As long as HOLD– or RESET– are not asserted during the reset sequence, instruction execution will begin at memory address 0 after the reset sequence.

RW– Memory read/write control, low to write, set to float while HDACK– is asserted.

STROBE– Active low memory strobe, set to float while HDACK– is asserted.

The following pins are used for test purposes, and allow the processor to be controlled during testing, as well as provide glimpses of the internal state of the FM9001 internal state during normal operation.

CNTL-STATE [5] These pins provide an encoding of the current internal state of the processor.

DISABLE-REGFILE– Active low, register file test pin.

TEST-REGFILE– Active low, register file test pin.

FLAGS [4] The arithmetic flags carry, overflow, zero, and negative, are made available for thorough testing.

I-REG [4] The four high-order bits of the instruction register. Note that these bits are unused in the FM9001 instruction word.

PO A parametric test output required by LSI Logic, Inc.

TE Active low test enable. When TE is active, the processor is in test mode, and all of the internal latches are connected in a serial scan chain with input TI and output TO.

TI Serial scan input for scan test mode.

TIMING The signal corresponding to the longest combinational path in the device, used to test the speed of the FM9001. In particular, this signal is connected to the ZERO output of the ALU.

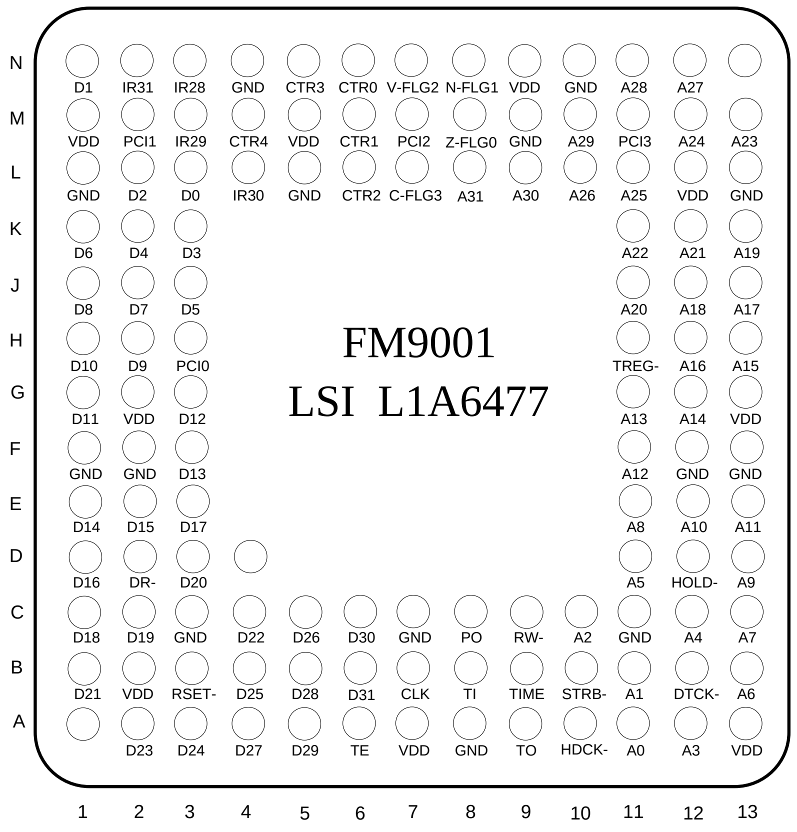
TO Serial scan output during test mode.

There are a large number of power and ground connections. Providing many such connections ensures the FM9001 of the necessary power and provides a large number of AC grounds, thus helping to quiet the noise associated with rapidly changing signal values.

VDD [**9**] These commonly connected signals should all be connected to +5 volts.

VSS [**14**] These commonly connected signals should all be connected to the power-supply ground. Signal values are measured relative this signal group.

Figure 4 is a pictorial drawing of pinout for the FM9001.


FM9001
LSI L1A6477
N
M
L
K
J
H
G
F
E
D
C
B
A
1 2 3 4 5 6 7 8 9 10 11 12 13
D1 IR31 IR28 GND CTR3 CTR0 V-FLG2 N-FLG1 VDD GND A28 A27
VDD PCI1 IR29 CTR4 VDD CTR1 PCI2 Z-FLG0 GND A29 PCI3 A24 A23
GND D2 D0 IR30 GND CTR2 C-FLG3 A31 A30 A26 A25 VDD GND
D6 D4 D3 A22 A21 A19
D8 D7 D5 A20 A18 A17
D10 D9 PCI0 TREG- A16 A15
D11 VDD D12 A13 A14 VDD
GND GND D13 A12 GND GND
D14 D15 D17 A8 A10 A11
D16 DR- D20 A5 HOLD- A9
D18 D19 GND D22 D26 D30 GND PO RW- A2 GND A4 A7
D21 VDD RSET- D25 D28 D31 CLK TI TIME STRB- A1 DTCK- A6
D23 D24 D27 D29 TE VDD GND TO HDCK- A0 A3 VDD

A:	Addr-out	HDCK:	HDack–
CTR:	Ctrl-state	IR:	I-reg
D:	Data-bus	PCI:	PC-reg-in
DR:	Disable-regfile–	RSET:	Reset–
DTCK:	Dtack–	STRB:	Strobe–
FLG:	Flags	TIME:	Timing
GND:	VSS	TREG:	Test-regfile–

Figure 4: Pinout of the FM9001

References

[1] Kenneth L. Albin and Warren A. Hunt, Jr.. The FM9001 Single-board Computer. Technical Report 90, Computational Logic, Inc., 1717 W. Sixth St., Suite 290, Austin, Texas 78703, 1993.

[2] William R. Bevier. *A Verified Operating System Kernel.* PhD Thesis, University of Texas at Austin, October 1987.

[3] William R. Bevier, Warren A. Hunt, Jr., J Strother Moore, and William D. Young. An Approach to Systems Verification. *Journal of Automated Reasoning*, 5(4):411–428, December 1989.

[4] W.R. Bevier. Kit and the Short Stack. *Journal of Automated Reasoning*, 5(4):519–530, December 1989.

[5] Graham Birtwistle, Brian Graham, Todd Simpson, Konrad Slind, Mark Williams, and Simon Williams. Verifying an SECD Chip in HOL. In *Proceedings of the IFIP TC10/WG10.2/WG10.5 Workshop on Applied Formal Methods for Correct VLSI Design.* Elsevier Science Publishers, Amsterdam, 1990.

[6] Dominique Borrione, Laurence Pierre, and Ashraf Salem. PREVAIL: A Proof Environment for VHDL Descriptions. In *Advanced Research Workshop on Correct Hardware Design Methodologies.* (assembled by Politecnico di Torino, Turin, Italy), June 1991.

[7] Richard Boulton, Mike Bordon, John Herbert, and John van Tassel. The HOL Verification of ELLA Designs. In *Final Program: 1991 International Workshop on Formal Methods in VLSI Design*, January 1991.

[8] Robert S. Boyer and J Strother Moore. *A Computational Logic.* ACM Monograph Series. Academic Press, Inc., 1979.

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

[10] Bishop Brock. Transforming Functional Hardware Models into Structural Models. Internal Note 150, Computational Logic, Inc., July 1989.

[11] Bishop C. Brock and Jr. Warren A. Hunt, Jr. The Formalization of a Simple HDL. In *Proceedings of the IFIP TC10/WG10.2/WG10.5 Workshop on Applied Formal Methods for Correct VLSI Design.* Elsevier Science Publishers, Amsterdam, 1990.

[12] 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.

[13] Geoffrey M. Brown and Miriam E. Leeser. From Programs to Transistors: Verifying Hardware Synthesis Tools. In *Workshop on Hardware Specification, Verification and Synthesis: Mathematical Aspects.*, Volume 408 of *Lecture Notes in Computer Science*, pages 128–150. Springer Verlag, 1989.

[14] M. Browne and E.M. Clarke. SML—A High Level Language for the Design and Verification of Finite State Machines. In D. Borrione, editor, *From HDL Descriptions to Guaranteed Correct Circuit Designs*, pages 269–292. North-Holland, Amsterdam, 1986.

[15] M. Browne, E.M. Clarke, D.L. Dill, and B. Mishra. Automatic Verification of Sequential Circuits Using Temporal Logic. *IEEE Transactions on Computers*, 35:1035–1044, December 1986.

[16] R.E. Bryant. Verification of Synchronous Circuits by Symbolic Logic Simulation. In *Hardware Specification, Verification and Synthesis: Mathematical Aspects.*, Volume 408 of *Lecture Notes in Computer Science*, pages 14–24. Springer Verlag, 1989.

[17] A. Camilleri, M. Gordon, and T. Melham. Hardware Verification Using Higher-order Logic. In D. Borrione, editor, *From HDL Descriptions to Guaranteed Correct Circuit Designs*, pages 43–67. Elsevier Science Publishers, Amsterdam, 1987.

[18] E.M. Clarke, D.E. Long, and K.L. McMillan. A Language for Compositional Specification and Verification of Finite State Hardware Controllers. In *Proc. of the Ninth Symposium on Computer Hardware Description Languages and Their Applications*, pages 281–295. IEEE, 1989.

[19] Avra Cohn. A Proof of Correctness of the VIPER Microprocessor: The First Level. In G. Birtwistle and P.A. Subrahmanyam, editors, *VLSI Specification, Verification and Synthesis*, pages 27–71. Kluwer Academic Publishers, Boston, MA, 1988.

[20] W.J. Cullyer. Implementing Safety Critical Systems: The VIPER Microprocessor. Divisional Memo (CC2) 411-87, Royal Signals and Radar Establishment, Malvern, Worcestershire (United Kingdom), August 1987.

[21] D.L. Dill and E.M. Clarke. Automatic Verification of Asynchronous Circuits Using Temporal Logic. *Proc. of the IEEE*, 133(5):276–282, September 1986.

[22] Matt Kaufmann et.al. Response to FM91 Survey of Formal Methods: Nqthm and Pc-Nqthm. Technical Report 75, Computational Logic, Inc., 1717 W. Sixth St., Suite 290, Austin, Texas 78703, 1992.

[23] Hans Eveking. Formal Verification of Synchronous Systems. In G. Milne and P.A. Subrahmanyam, editors, *Formal Aspects of VLSI Design*, pages 137–151. North-Holland, Amsterdam, 1986.

[24] Ivan V. Filippenko. VHDL Verification in the State Delta Verification System. In *1991 ACM SIGDA International Workshop on Formal Methods in VLSI Design*, January 1991.

[25] Simon Finn, Michael P. Fourman, Michael Francis, and Robert Harris. Formal System Design - Interactive Synthesis Based on Computer-assisted Formal Reasoning. In *Proceedings of the IFIP TC10/WG10.2/WG10.5 Workshop on Applied Formal Methods for Correct VLSI Design*. Elsevier Science Publishers, Amsterdam, 1990.

[26] Steven M. German and Karl J. Lieberherr. Zeus: A Language for Expressing Algorithms in Hardware. *IEEE Computer*, 18(2), February 1985.

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

[28] M.J.C. Gordon. Why Higher-order Logic is a Good Formalism for Specifying and Verifying Hardware. Technical Report 77, University of Cambridge, Computer Laboratory, September 1985.

[29] Jr. Guy L. Steele. *Common LISP: The Language*. Digital Press, 1984.

[30] Warren A. Hunt, Jr. *FM8501: A Verified Microprocessor*. LNCS 795, Springer-Verlag, 1994.

[31] Warren A. Hunt, Jr. Microprocessor Design Verification. *Journal of Automated Reasoning*, 5(4):429–460, December 1989.

[32] 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.

[33] Warren A. Hunt, Jr. and Bishop C. Brock. The Verification of a Bit-slice ALU. In *Workshop on Hardware Specification, Verification and Synthesis: Mathematical Aspects.*, Lecture Notes in Computer Science, pages 281–305. Springer Verlag, 1989.

[34] The Institute of Electrical and Electronics Engineers. *IEEE Standard VHDL Language Reference Manual*, 1988.

[35] Graham Birtwistle Jeffery Joyce and Mike Gordon. Proving a Computer Correct in Higher Order Logic. Technical report, University of Calgary, Department of Computer Science, August 1985.

[36] Steven D. Johnson. Manipulating Logical Organization with System Factorizations. In *Hardware Specification, Verification and Synthesis: Mathematical Aspects.*, Volume 408 of *Lecture Notes in Computer Science*, pages 259–280. Springer Verlag, 1989.

[37] Matt Kaufmann. An Integer Library for Nqthm. Internal Note 182, Computational Logic, Inc., March 1990.

[38] Matt Kaufmann. A Hardware Reset Lemma and Its Proof. Internal Note 230, Computational Logic, Inc., May 1991.

[39] Matt Kaufmann. Some Tools for Using the Brock/Hunt HDL. Internal Note 237, Computational Logic, Inc., July 1991.

[40] Matt Kaufmann. A Translator from Brock/Hunt HDL to VHDL. Internal Note 231, Computational Logic, Inc., May 1991.

[41] Beth H. Levy. An Overview of Hardware Verification Using the State Delta Verification System (SDVS). In *Final Program: 1991 International Workshop on Formal Methods in VLSI Design*, January 1991.

[42] J S. Moore. Piton: A Verified Assembly Level Language. Technical Report 22, Computational Logic, Inc., 1717 West Sixth Street, Suite 290 Austin, TX 78703, 1988.

[43] J S. Moore. Mechanically Verified Hardware Implementing an 8-bit Parallel IO Byzantine Agreement Processor. Technical Report Technical Report 69, Computational Logic, Inc., 1717 W. Sixth Street, Suite 290, Austin, TX 78703, August 1991.

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

[45] Mary Sheeran. μFP—an Algebriaic VLSI Design Language. Technical Report PRG-39, Oxford University Computing Laboratory, September 1984.

[46] Mandayam Srivas and Mark Bickford. Formal Verification of a Pipelined Microprocessor. *IEEE Software*, 7(5):52–64, September 1990.

[47] John Van Tassel and David Hemmendinger. Toward Formal Verification of VHDL Specifications. In *Applied Formal Methods for Correct VLSI Design, Volume 1*, pages 261–270, November 1989.

[48] Michael Yoeli, editor. *Formal Verification of Hardware Design.* IEEE Computer Society Press Tutorial, 1990.

[49] William D. Young. *A Verified Code Generator.* Computational Logic Technical Report Number 37, 1988.