Hardware Verification at Computational Logic, Inc.

Bishop C. Brock
Warren A. Hunt, Jr.

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

+1 512 322 9951
Brock@CLI.COM, Hunt@CLI.COM

3 August 1990

Talk Topics

- Hardware Verification: What Is It?
- Formal Methods: What Good Are They?
- Verification Methodology
- Present Accomplishments
- Expected Near Term Results
- Present Trends
- Future Directions
- Collaborations and Technology Transfer
- Technology Enablers
- Conclusions

Hardware Verification: What Is It?

The mathematical formalization of the specification of any (all) aspects of hardware design.

We specifically are interested in the design of hardware for digital computing.

Goals:

- Completely replace programmer's manuals, timing diagrams, interface specifications, power requirements, etc. with clear precise formulas.
- Provide a perfectly clear foundation upon which systems can be built.

Formal Methods: What Good Are They?

Formal methods in the U.S. have a bad credit rating.

Over the years, good mechanized software verification systems have been constructed.

Good software verification tools are being extended to include hardware verification, thus providing good systems verification tools.

Hardware verification seems more tractable than software verification:

- few, repeatedly-used, low-level constructs;
- specification domain is less abstract (fairly concrete); and
- formal methods can be used incrementally.

Last point is critical, note Bryant's work.

Our Verification Methodology

We employ the Boyer-Moore logic to:

- write design specifications;
- write behavioral specifications; and
- record relations.

The Boyer-Moore theorem prover

- insures that definitions are well formed;
- checks that proofs are correct; and
- manages our evolving database of facts.

Present Accomplishments

Our application of formal methods to hardware specification and verification include:

- Core RISC specification;
- FM8502 microprocessor verification;
- verification of circuits using standard TTL components;
- a formalization of a simple HDL; and
- verified synthesis of combinational circuits.

Let us consider several in more detail.

Core RISC

Bill Bevier has formally specified a set of instructions that characterize a Core RISC-complient processor. This formalization includes:

- byte, half-word, and long-word memory accesses;
- Boolean, natural number, and integer ALU operations;
- a minimum register set; and
- an exception mechanism.

The emphasis here has been on mathmatically modeling the instruction set.

Our study of RISC architectures indicates that we need to be able to model multi-phase clocking schemes before we attempt to design a build a verified Core RISC processor. This effort is ongoing.

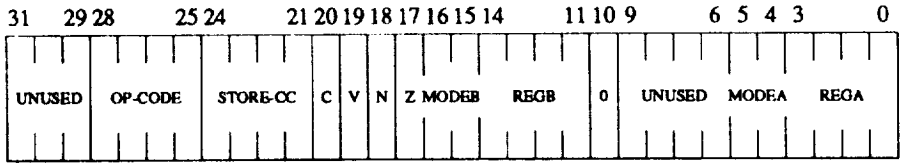
The FM8502 Fabrication

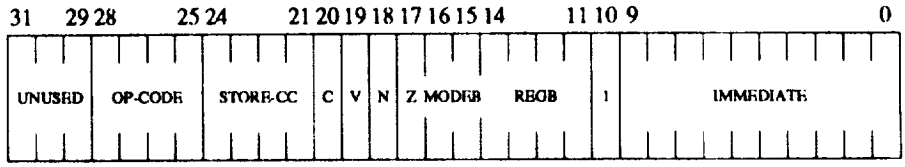
Currently, our primary effort involves the fabrication of the FM8502 microprocessor.

This fabrication effort is a test-of-concept; that is, can we manufacture formally modeled circuits and get them working?

The FM8502 microprocessor is a 32-bit general purpose microprocessor with:

- 32-bit addressing;
- 16 general-purpose registers;
- two-address architecture;
- 5 addressing modes;
- a 16-function ALU
- extensive flag support; and
- little else.





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	OPERATION	DESCRIPTION	STORE-CC	CONDITION
0000	b <- a	Move	0000	Carry clear
0001	b <- a + 1	Increment	0001	Carry set
0010	b <- a + b + c	Add with carry	0010	Overflow clear
0011	b <- b + a	Add	0011	Overflow set
0100	b <- 0 - a	Negation	0100	Not negative
0101	b <- a - 1	Decrement	0101	Negative
0110	b <- b - a - c	Subtract with borrow	0110	Not zero
0111	b <- b - a	Subtract	0111	Zero
1000	b <- a >> 1	Rotate right through carry	1000	Higher
1001	b <- a >> 1	Arithmetic shift right	1001	Lower or same
1010	b <- a >> 1	Logical shift right	1010	Greater or equal
1011	b <- b XOR a	XOR	1011	Less
1100	b <- b OR a	OR	1100	Greater
1101	b <- b AND a	AND	1101	Less or equal
1110	b <- NOT a	NOT	1110	True
1111	b <- a	Move	1111	False

The FM8502 Implementation Specification

To be able to manufacture the FM8502 with some precision, we have been working on the formalization of an HDL.

We will prove the correctness of our HDL description of the FM8502, and then translate our HDL description into a commercial HDL.

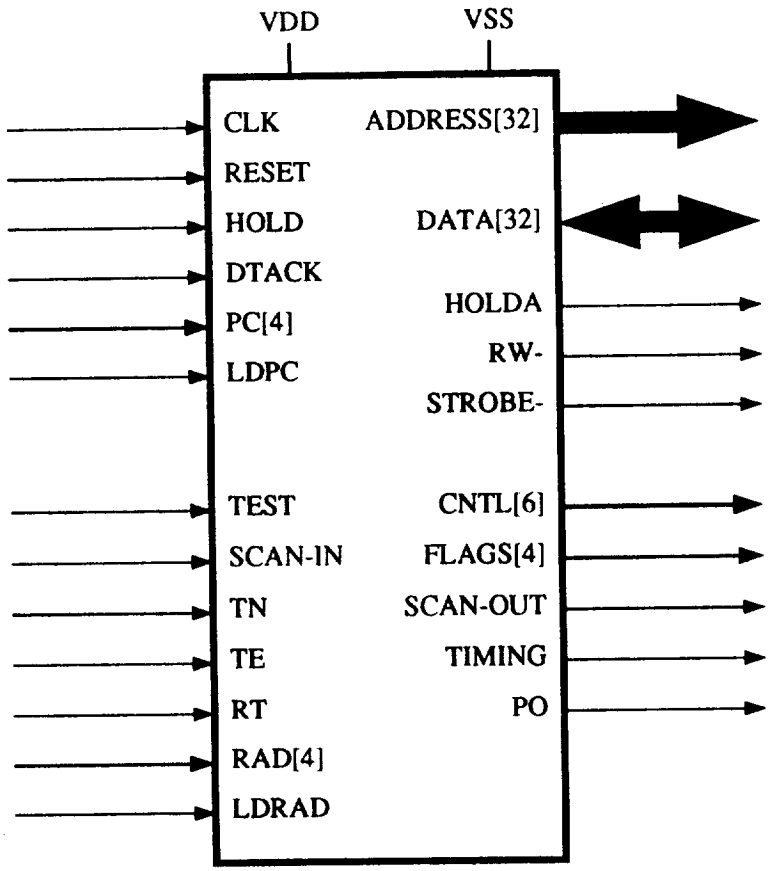
Our HDL provides our lowest-level model for the FM8502 implementation:

- every internal gate and register is described;
- every I/O pad is defined; and
- we expect to validate our test vectors directly on our HDL description.

Our HDL specification also includes all of the internal test logic.

The FM8502 Pinout

Below is a pictorial diagram of the FM8502 pinout. Quite a number of pins are allocated to testing purposes.



A Formal HDL

Our HDL is structured like commercial HDL's:

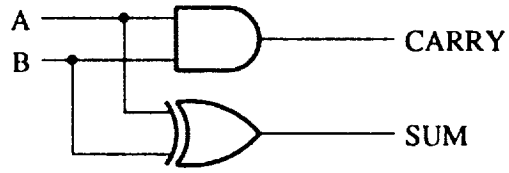
- netlist based;
- heirarchicaly structured;
- occurence-oriented; and
- allows multiple views of circuits.

We have a formal specification of our HDL:

- a predicate recognizes well-formed circuits; and
- several interpreters define the semantics.

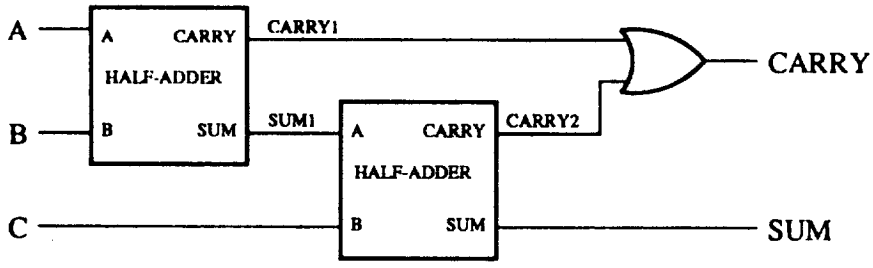
HDL Examples of Circuits

```
'(HALF-ADDER (A B)
             (SUM CARRY)
             (( G0(SUM)    B-XOR(A B))
              ( G1(CARRY)  B-AND(A B))))
```



The following full-adder specification refers twice to the half-adder specification above.

```
'(FULL-ADDER (A B C)
             (SUM CARRY)
             (( T0(SUM1 CARRY1) HALF-ADDER(A B))
              ( T1(SUM CARRY2)  HALF-ADDER(SUM1 C))
              ( T2(CARRY)       B-OR(CARRY1 CARRY2))))
```



Verified Synthesis

We perform synthesis by

- writing circuit generator programs;
- verifying the circuit generator programs; and
- then running the generators to produce provably correct circuits.

In other words, after a circuit has been generated we need not inspect it for the Boolean correctness.

An ALU Generator

We have an arbitrary size, 16-function ALU generator which is:

- programmable -- ALUs with different internal structure can be produced;
- "intelligent" -- internal buffers are only added when needed; and
- has been verified to generate correct *n*-bit, gate-level ALU descriptions.

Simple translators can convert the ALU descriptions into conventional CAD languages (e.g., VHDL).

To replay the proof only takes about 20 (Sun 3) minutes.

ALU Generator Output Summary

Summarized below are some characteristics of the ALUs generated by our verified ALU generator.

ALU Characteristics			
Size	Gate Count	Fanout	Delay
1 bit	126	8	12
2 bits	149	8	14
4 bits	196	8	17
8 bits	297	8	22
16 bits	491	8	26
32 bits	880	8	30
64 bits	1665	8	35
128 bits	3227	8	39

Payoff: It only takes 0.6 seconds to generate a correct 32-bit ALU, 1.3 seconds for a 64-bit ALU, and 3.1 seconds for a 128-bit ALU.

Expected Near Term Results

Several projects underway which will conclude this year are:

- an ability to verify sequential circuits generators; and
- the fabrication of the FM8502 microprocessor.

We are using both combinational and sequential logic synthesis techniques in the fabrication of the FM8502.

We will be able to generate a correct *n*-bit microprocessor (so long as the word size is large enough to contain FM8502 instructions.)

We will generate a gate-array specification directly.

We are generating our test-vectors directly from our formal circuit specifications.

Present Trends

There is increasing interest in:

- boolean comparison -- which should lead the way to more general purpose techniques;
- register-transfer specifications with circuit verification;
- formalization of self-timed circuits;
- formalization of timing behavior; and
- transformational systems.

These trends are all indicative of increased use of formal techniques for hardware specification and verification.

And these techniques are being applied incrementally.

Future Directions

In the future we hope to:

- formalize a subset of VHDL (using our Ada formalization experience);
- perform tool verification (e.g., logic minimizer, tautology checkers);
- verify a Core RISC microprocessor with memory management; and
- continue our work on formalizing hardware interfacing and use specifications.

This last item is hardest and has the biggest payoff.

Industrial Collaborations

We have been working with DEC for two years.

Motorola may attempt the specification (and possibly the verification) of one of their microcontrollers.

Technology Transfer

We highly value interactions with industry; we all profit.

Our formal techniques may be used incrementally, i.e., "creeping formalization."

Industry first employs our techniques for (unambiguous) specification, later for verification.

Specification is a big problem for industry -- formal specification allows analysis without exhaustive testing.

Technology Enablers

Is the state-of-the-art separating further from the state-of-the-practice?

To enable the use of formal techniques in hardware design we need to:

- train more engineers with formal methods (not train mathematicians to be engineers);
- make existing tools and techniques more accessible to engineers; and
- make formal techniques the most economical method of hardware validation.

A big success or two would help us get industry's attention.

Conclusions

Formal methods can be used to provide accurate specifications.

Hardware verification provides increased assurance of circuit correctness.

Formal techniques provide a good growth path; they scale up well.

The credit rating of formal techniques is improving.

Goals:

- Completely replace programmer's manuals, timing diagrams, interface specifications, power requirements, etc. with clear precise formulas.
- Provide a perfectly clear foundation upon which systems can be built.