The Scheme Machine:
A Case Study in Progress
In Design Derivation at System Levels

Steven D. Johnson
DDD Project, Hardware Methods Laboratory
Computer Science Department
Indiana University

`sjohnson@cs.indiana.edu`
`http://www.cs.indiana.edu/hmg/hmg.html`

Thanks: NSF/MIP92 08745, NASA/NGT-50861

Outline

- The Scheme Machine Prototype
- Context, Motivation, and Goals of the Research
- Components of the Scheme Machine
- Issues in Verification
- Incremental Construction of Hardware
- Summary and Status

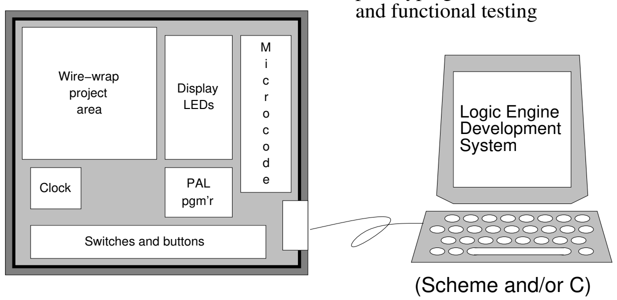
The Scheme Machine Prototype

Photo:
Logic Engine
Scheme Machine
Prototype

Emulator in Actel FPGAs, PLDs and standard DRAM simms.

The Logic Engine

An environment for sytem prototyping, VLSI emulation, and functional testing



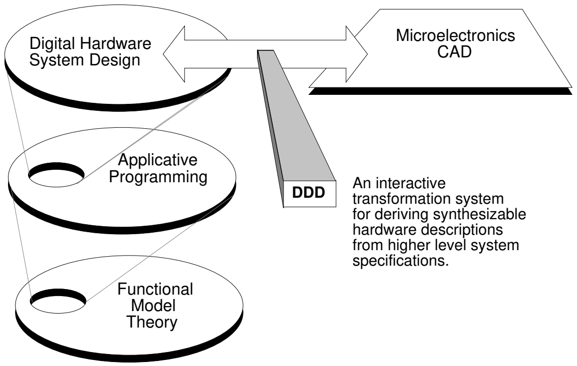
Scheme Machine
Memory:
$2 \times 4 \times 1,000,000$ bytes
Design Size:
20,000 gates (?)
Speed:
2 MHz+ (or $1.25 \times$ DRAM)

Photo:
Scheme
Machine
Detail

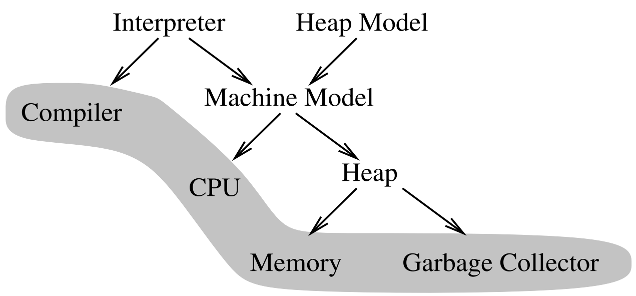
Context of the Research

1982 M. Wand, *Semantics-directed machine architecture* (POPL 92)
1983 S.D. Johnson, *Synthesis of Digital Designs from Recursion Equations*, Ch. 5 (PhD dissertation)
1984 W. Clinger, *The Scheme 311 Compiler* (L&FP 84)
1987 S.D. Johnson, et. al. *A Tactical Framework for Digital Design* (WG10.2 FM/VLSI).
1988 R. Wehrmeister, *Derivation of an SECD Machine* (IU/CS TR #290)
1990 D. Boyer and K. Rath derive boolean system for a Scheme machine.
1991 IEEE Std 1178-1990 *IEEE Standard for the Scheme Programming Language*.
1992 M. Wand, D. Friedman, C. Haynes, *Essentials of Programming Languages*, Ch. 12.
1993 B. Burger, *The Scheme Machine* (IU/CS TR #413).
1995 M. Wand, J. Guttman, J. Ramsdell, et.al. *VLISP: A Verified Implementation of Scheme* (*J. Lisp and Symbolic Computation*).

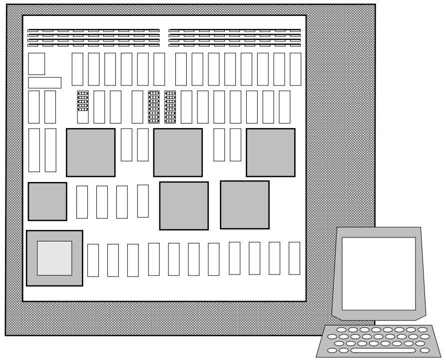
Digital Design Derivation Project



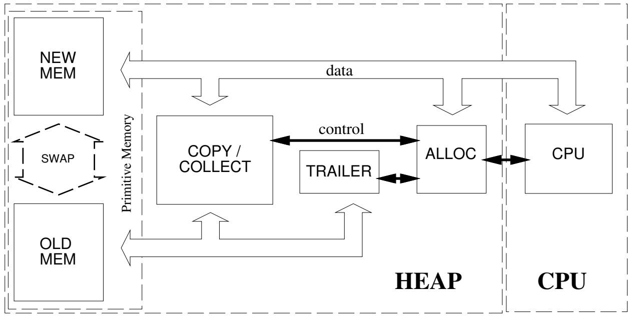
The Scheme Machine Project



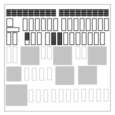
Scheme Machine Prototyping Environment



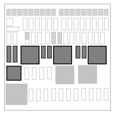
Scheme Machine Architecture



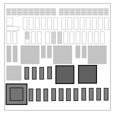
Components of the Scheme Machine



Memory
- ○ Dual-ported semispaces, multiplexed bus
- ○ typical case: 2 opns. per 1.25 cycles
- ○ refresh inhibits clock
- ○ manual design, unverified (one bug so far)

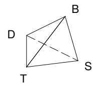


Heap Interface
- ○ collector, allocator, invalidator
- ○ derived from XFSM.
- ○ synchronization with CPU not verified
- ○ algorithm not verified (VLISP?)



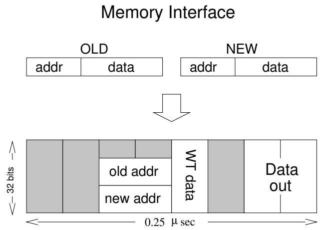
Processor
- ○ derived from X-FSM only.
- ○ SPEC close to VLISP machine?

Issues in Verification



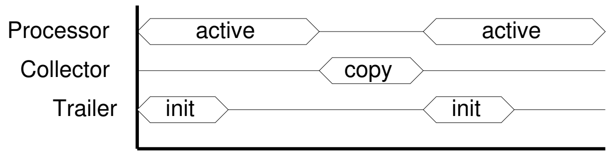
- Higher level behavior specification
- Memory model
- Process synchronization
- Data abstraction hierarchy
- Multiple views

Memory Model



rd(N, a) || wt(O, b,c)
wt(N, a, b) || rd(O, c)
rd(N, a) || mark–next(O)
wt(N, a, b) || mark–next(O)

Process Synchronization



- In Scheme Machine, heap integrity is guaranteed by hardware, with overhead absorbed (to be measured) by the parallel sweep.
- In VLISP, heap integrity is guaranteed by the compiler, but is interruption considered?

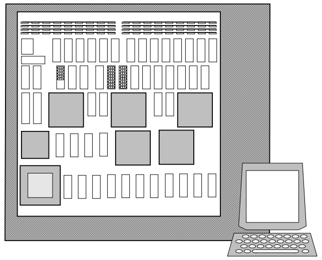
Data Abstraction

- semantic model: ρ: $Var \to Val$
- compiler factorization: ρ_c: $Var \to StaticOffset$
  ρ_r: $StaticOffset \to Store \to Val$
- Operational model: $\mathrm{Rep}[\rho_r] = List(Vector)$
- Implementation model: $List$: $HeapReference$
  $Vector$: $(HeapReference, Offset)$
- Representation: $HeapReference$: $(Tag, Address)$
  $Offset$: $Address$
  Tag: $\{\mathsf{list}, \mathsf{int}, \ldots\}$
  $Address$: $\mathcal{Z}_{2^{24}}$
- HW Realization: $Address, Offset$: $BitVector(24)$
  Tag: $BitVector(8)$

Multiple Views

- The CPU sees the heap as an assorted collection of cells, including numbers, list cells of various flavors, vectors, strings, *etc.*
- The allocator and collector see the heap as an array of words subject to certain invariants.
- The invalidator sees the heap as a sequence of bits.

Incremental Construction of Hardware



Summary and Status

- Design derivation extends and adapts programming methodology to hardware targets
- Language implementation, Scheme in this case, is a standard example,
  - exposes hard (higher order) modeling issues
  - a form of closure/completeness (Scheme Machine derivation & emulation could be run on the Scheme Machine).
- the Scheme Machine system specification exposes new verification problems in process coordination, algorithmic correctness, ….
- the design environment explores the gap between modeling and implementation.
- components of the Scheme Machine, or their variations are viable products.

Incremental Construction of Hardware

- The software specification of the processor runs directly against both the software specification of the heap and (with functional test points exposed) the hardware prototype.
- The software specification of the processor, allocator, and collector, run directly against both the modeling environment's heap image and the hardware realization.
- We are seeking to develop an environment where modeling, simulation, emulation, and realization are closely integrated with verification.
- We are seeking to integrate multiple reasoning tools to apply to the verification problem.