Use of the E Language

Warren A. Hunt, Jr. and Sol O. Swords

March, 2009

Computer Sciences Department
University of Texas
1 University Way, M/S C0500
Austin, TX 78712-0233

E-mail:
{hunt,sswords}@cs.utexas.edu
TEL: +1 512 471 {9748,9744}
FAX: +1 512 471 8885

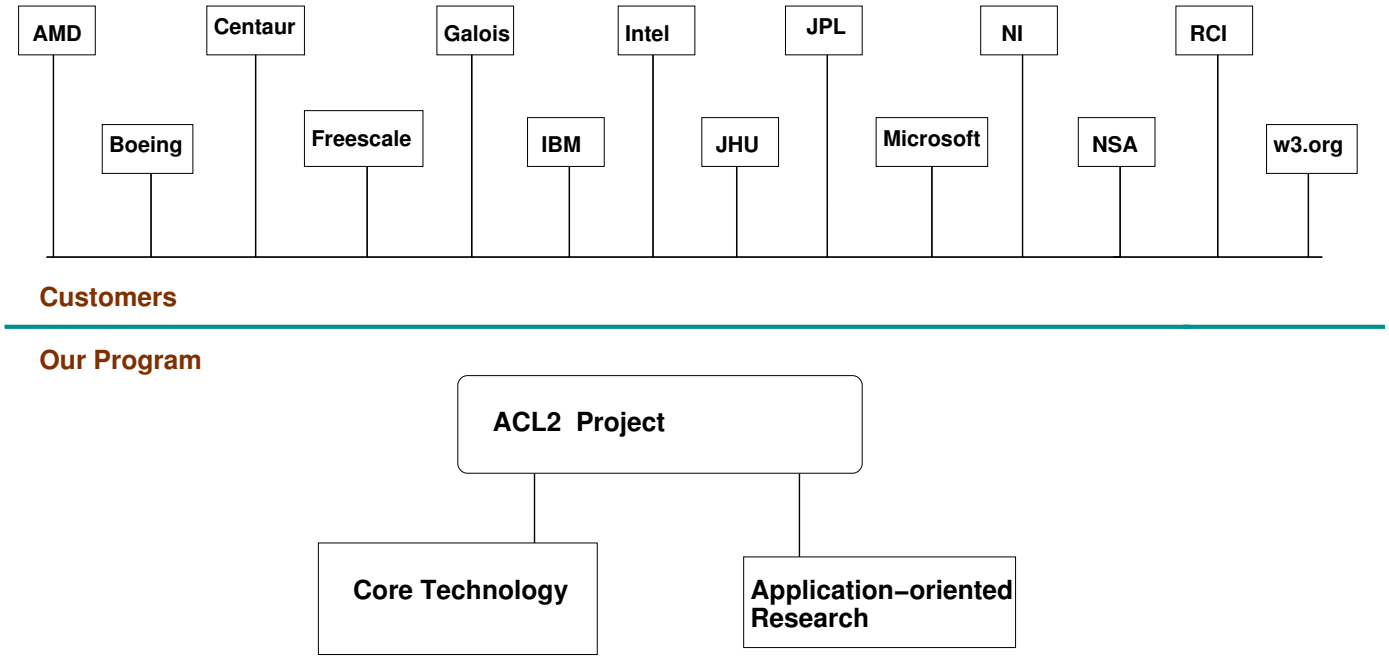
Centaur Technology, Inc.
7600-C N. Capital of Texas Hwy
Suite 300
Austin, Texas 78731

E-mail:
{hunt,sswords}@centtech.com
TEL: +1 512 418 {5797,5751}
FAX: +1 512 794 0717

Outline

Ecosystem

We have significant collaboration with the industry.



Our own research includes:

- **Development** of core technologies
- **Application** of these technologies on different verification domains
- **Focus** on specification and verification of Centaur X86 design

Use of the **E** Language

We have developed a formalized HDL in support of industrial design.

- Deeply embedded **E** language in ACL2 logic.
- Language descriptions are represented as Lisp constants.
- ACL2 theorem-proving system used to verify **E** descriptions.

The **E** langauge is formal.

- Syntax of **E** language is recognized by ACL2 predicate.
- Semantics given by interpreter.
  - Multiple evaluators defined: BDD, four-valued BDD, AIG, four-valued AIG, dependency, and delay.
  - Symoblic simulation for all modes (except delay).

The **E** Language is in everyday industrial use at Centaur Technology, Inc.

E-Language Features

The **E** language is:

- hierarchical, and
- occurrence-oriented.

We use the **E** language much like a database; it includes:

- HDL descriptions
- Hierarchical state representation
- Signal sense and direction
- Clock discipline
- Properties
- Annotations

E-language recognizer requires two passes:

- first pass ensures well-formedness of language names and data structures,
- second pass checks inter-module connections and dependencies.

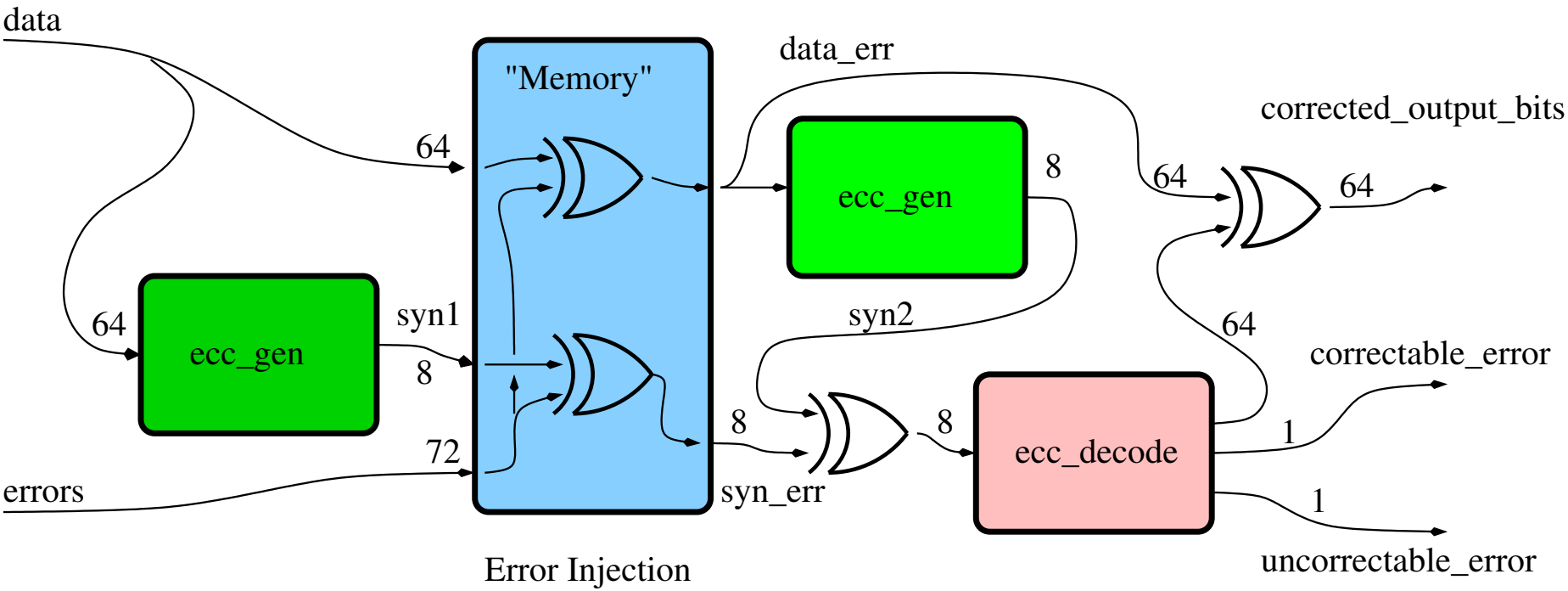
E-Language Example

```
(defm *simple-ff-latch*
  `(:i (clk a)
    :o (o)
    :s (l- l+)
    :c (clk)
    :cd ((t) (nil))
   :occs
    ((:u l+ :o (n n~) :op ,*latch+*  :i (clk a))
     (:u o1 :o (clk~) :op ,*not1*    :i (clk))
     (:u l- :o (o o~) :op ,*latch+*  :i (clk~ n)))))
```

Simple two-latch, flip-flop

- Interface: **:i**, **:o**, and **:s** fields.
- Clock: **:c** and **:cd** fields.
- Occurrences are in a list, but treated as a set
- Multi-phase and gated clocking supported

ECC Example



Model to analyze the ECC circuitry.

- Syndrome unit produces error-correcting code
- ECC unit decodes syndrome to produce 1-hot, correction position

Verilog for ECC Model

```
module ecc_model (data,                    // Input Data
                  errors,                  // Error Injection
                  corrected_output_bits,   // Output Data
                  correctable_error,       // Corrected?
                  uncorrectable_error);    // Can't be corrected

   ecc_gen gen1 (syn1, data);      // Generate syndrome bits for "memory"

   assign          data_err = data ^ errors[63:0];   // Fault injection
   assign          syn_err  = syn1 ^ errors[71:64];  // Fault injection

   ecc_gen gen2 (syn2, data_err);  // Syndrome bits for "memory" output

   assign          syn_backwards_xor = syn_err ^ syn2;  // Compute syndrome

   ecc_decode make_outs (bit_to_correct,      // One-Hot output correction
                         correctable_error,   // Correctable error?
                         uncorrectable_error, // UnCorrectable error?
                         syn_backwards_xor);  // Syndrome input

   assign          corrected_output_bits = bit_to_correct ^ data_err;
endmodule
```

E-Language for ECC Model

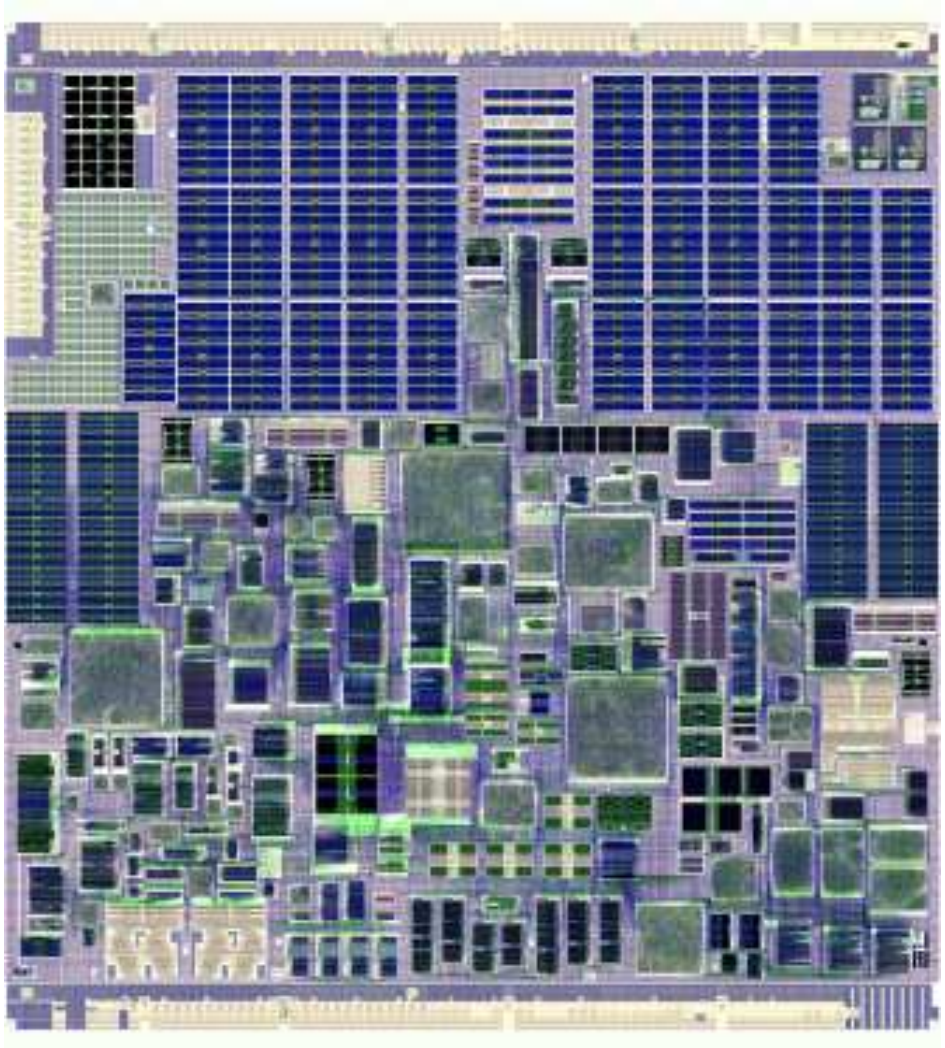
```
(:n  |*ecc_model*|

 :i (|data[0]| |data[1]| |data[2]| |data[3]| |data[4]|
     |data[5]| |data[6]| |data[7]| |data[8]| |data[9]| ...)

 :o (|corrected_output_bits[0]| |corrected_output_bits[1]|
     |corrected_output_bits[2]| |corrected_output_bits[3]|
     |corrected_output_bits[4]| |corrected_output_bits[5]|
     |corrected_output_bits[6]| |corrected_output_bits[7]|
     |corrected_output_bits[8]| |corrected_output_bits[9]| ...)

 :occ ((:full-i #@53# :full-o #@54# :u |_gen_3|
                :op #.*vl_64_bit_buf* :o #@55# . #@56#)
       (:full-i #@57# :full-o #@58# :u |_gen_4|
                :op #.*vl_8_bit_buf* :o #@59# . #@60#)
       (:full-i #@61# :full-o #@62# :u |_gen_5|
                :op #.*vl_64_bit_pointwise_xor*
                :o #@63# . #@64#)
       (:full-i #@19# :full-o #@20#
                :u |gen1|
                :op #.|*ecc_gen*|
                :o #@21#
                :i #@22#)  ...   ))
```

ACL2 Specification for ECC Model

We can verify the 94,299,755,704,803,227,860,992 cases in a few seconds.

```
(defn our-one-bit-error-predicate (bad-bit)
  ;; Check output correctness if one error injected.
  (declare (xargs :guard (natp bad-bit)))

  (let* ((data   (qv-list 0 1 64))
         (errors (q-not-nth bad-bit (make-list 72 :initial-element nil)))
         (inputs (ap data errors)))

    (equal (mv-let (s o)
             (emod 'two |*ecc_model*| inputs nil)
             (declare (ignore s))

             (list :corrected-bits        (take 64 o)
                   :correctable_error     (nth 64 o)
                   :uncorrectable_error   (nth 65 o)     ))

           (list   :corrected-bits        data
                   :correctable_error     (< bad-bit 64)
                   :uncorrectable_error   NIL            ))))
```

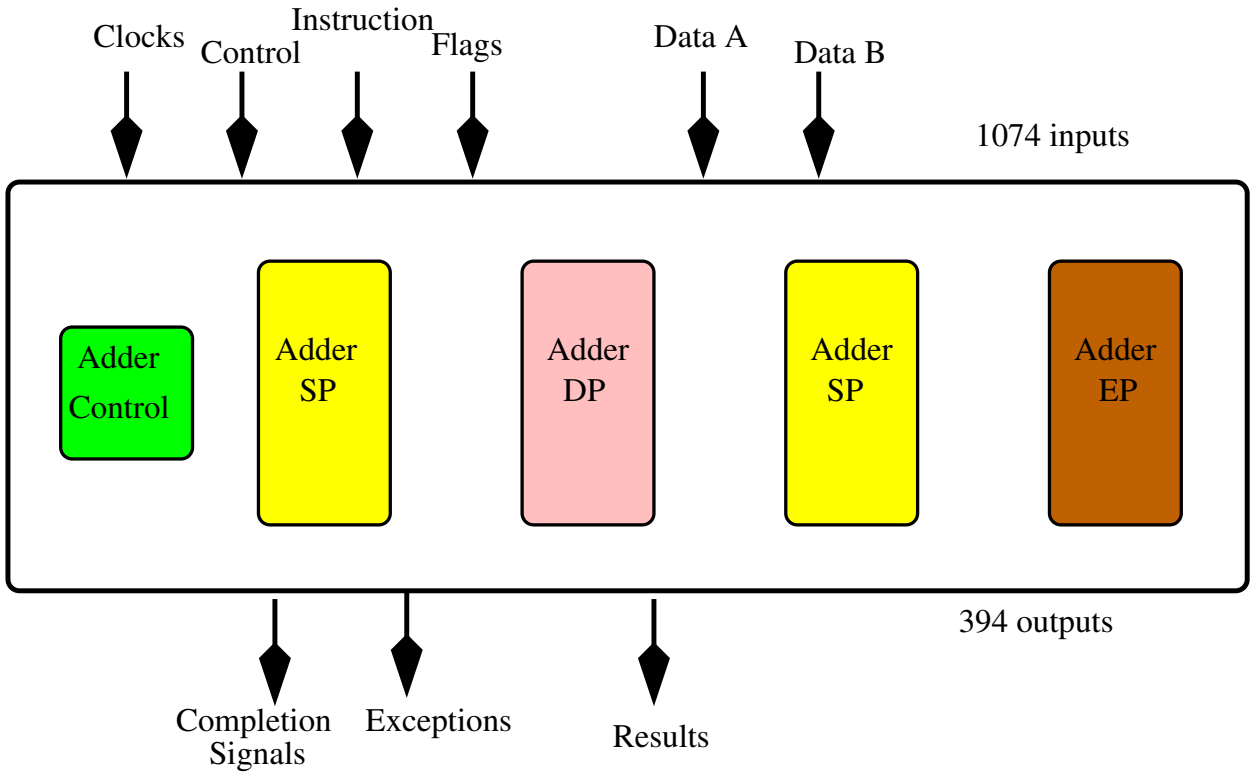
Centaur CN(x) X86-64 Microprocessor



Contemporary Example

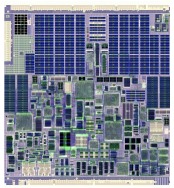
- Full X86-64 design including VMX
- 65-nanometer design of 94.5M transistors
- AES, DES, SHA, and random-number generator hardware
- Runs 40 operating systems and four VMs
- Used by IBM, HP, Samsung, and soon Dell

Centaur CN(x) Media Unit – Adders



- 33,700 line Verilog description of 680 modules
- Modules represent 432,322 transistors
- Unit has 374 outputs and 1074 inputs (26 clocks)
- Two-cycle-latency (data in to data out)

The Centaur Media-Unit, Verification Tool Flow



Verilog Files

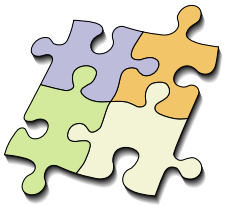
fadd EMOD module

EMOD Symbolic Simulator

fadd AIG function

Specialize

Per-instruction AIGs



Case-splitting, Parametrization

AIG2BDD

Instruction Spec

Symbolic Spec

Spec Output BDDs

=?

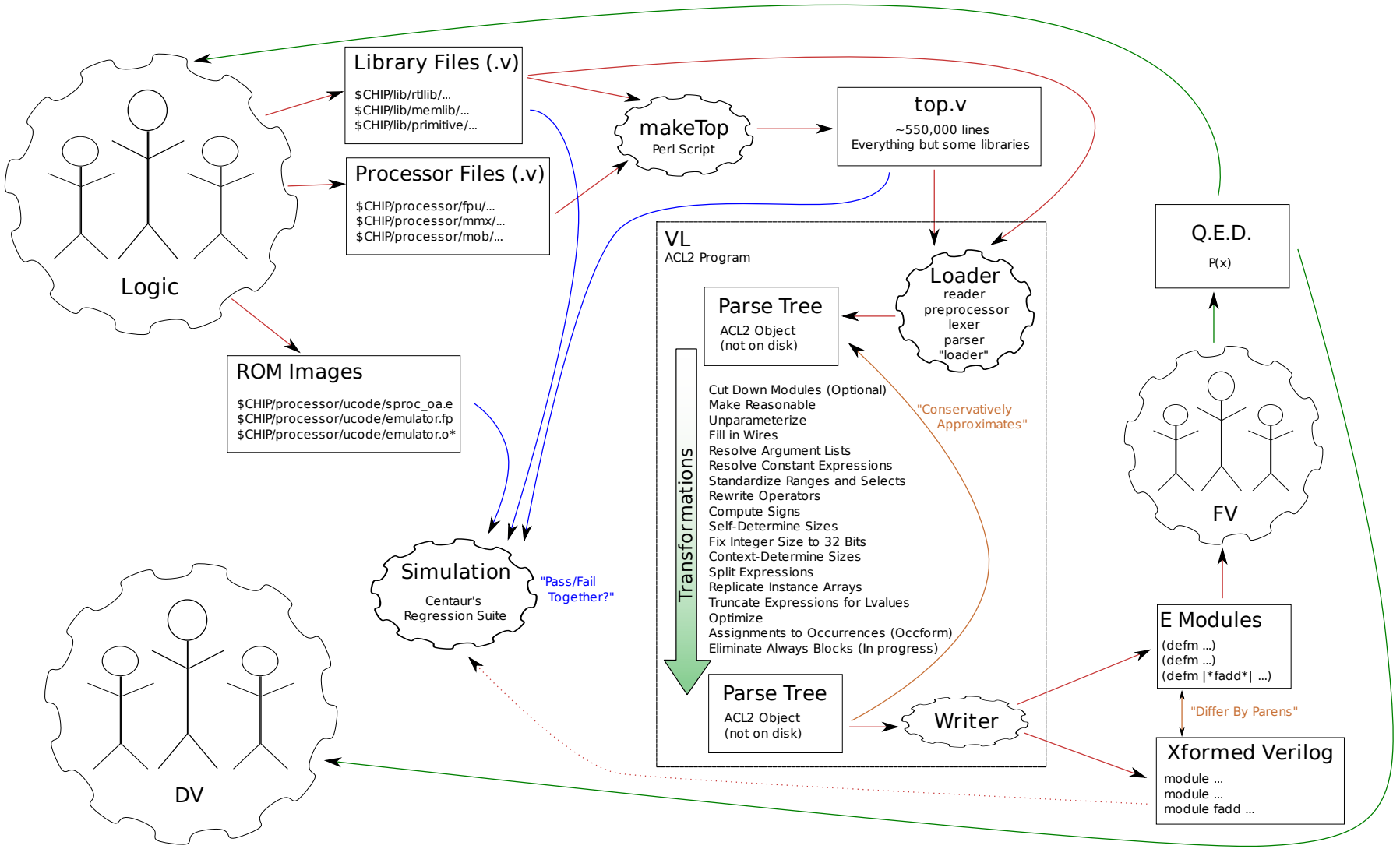
Hardware Output BDDs

c=a+b

The Centaur Media-Unit, Verification Approach

- Verilog-to-E translation performed by ACL2-based program.
- Four-valued, AIG-based symbolic simulation of entire design for eight half-cycles.
- AIGs specialized for the instruction under investigation
- AIGs are converted to BDDs
  - Too big to verify directly, so case splitting employed
  - For each case, BDD approximated until exact.
  - For each case, compared to symbolic simulation of specification
- Cases are shown to be exhaustive

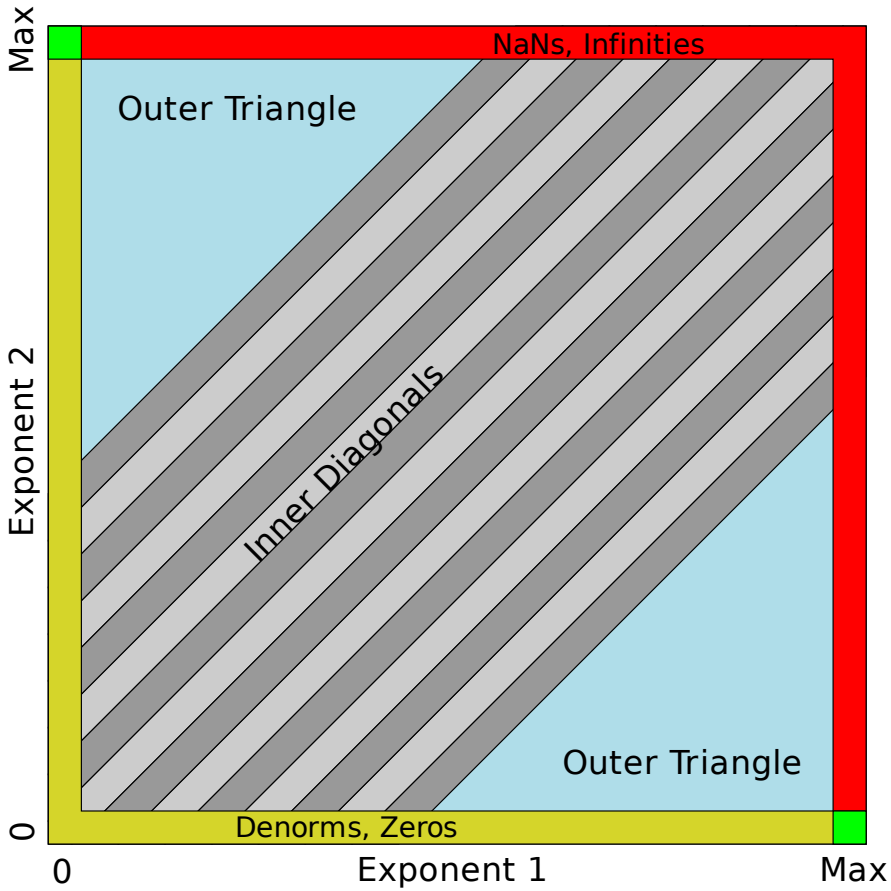
The Verilog-to-E Translator (Jared Davis)



The Centaur Media-Unit, Case-Splitting Approach

Problem too big to verify all at once.

- Case split by exponent differences
- Separately, account for special cases (e.g., NaNs, Infinity)
- For each case, generate symbolic inputs that cover exactly the specified set of inputs
  - BDDs are parametrized
  - Approach used for all FP sizes



Centaur Media-Unit, Verification

We attempted to verify single, double, and extended precision addition/subtraction operations.

We first found an *information flow* bug; the output contained an input signal from a different cycle than the rest of the inputs.

- Single precision (32-bit) results and flags OK.
- Double precision (64-bit) results and flags OK.
- Extended precision (80-bit) results had an error.
  - Exactly one pair of numbers returned an incorrect answer
  - Sort of like a *perfect storm*; a 64-bit cancellation
  - Answer returned was twice as big as it should have been.

A fix was developed, and this bug has been eliminated. We have checked the correctness of the new design – it took less than an hour.

Robert Krug has completed a proof that our Boolean-based adder/subtracter specification is correct.

Conclusion

E-language in everyday commercial use at Centaur Technology.

- Each night, entire design is translated
  - 550,000 lines of Verilog translated to E
  - Unable to translate some modules – working to finish translation
- New ACL2 executable containing all E-based modules is built each day.
  - Verifiers may use newest design version
  - Entire translation and build time about 50 minutes
- Each night we recheck our proofs on the new model

ACL2 is now used to support an industrial verification flow.