Formal Verification of Elementary Floating-Point Operations with ACL2

David M. Russinoff
Advanced Micro Devices, Inc.

david.russinoff@amd.com
http://www.onr.com/user/russ/david

Floating Point Design Verification at AMD

Focus: elementary (IEEE-compliant) operations

- FDIV, FSQRT, FADD, FSUB, FMUL
- SIMD and 3dnow! instructions

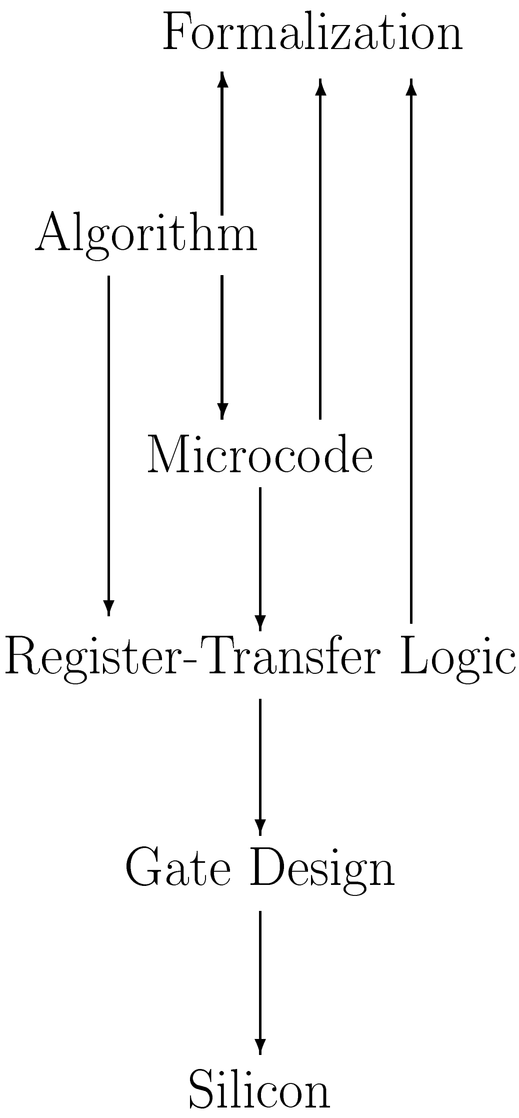
Approach: theorem proving using ACL2

- Executable language with rational arithmetic and S-expressions
- First order logic with recursive functions
- Heuristic prover based on induction

Objectives of Formal Hardware Verification

- Accurate design-level formal model
- Unambiguous abstract specification of behavior
- Convincing mechanically checked proof of correctness

Hardware Models



AMD RTL Language

- Based on Verilog, but intended specifically for RTL modeling of microprocessors
- Simple cycle-based execution model
- Simple iteration construct
- Primitives: logical operations on bit vectors
- Statements: combinational and sequential assignments (wires and registers)

```
sticksum[47:0] = esub ?
                   ina_add[47:0] ^ inb_add[47:0]
                 : ~(ina_add[47:0] ^ inb_add[47:0]);

M4_sticky_co <= M3_sticky_co;
```

Translation to ACL2

- Bit vectors are represented as integers
- RTL primitives correspond to ACL2 functions
- Iteration is translated into recursion
- Signal assignments are translated into ACL2 function definitions:

```
sticksum[47:0] = esub ?
                  ina_add[47:0] ^ inb_add[47:0]
                : ~(ina_add[47:0] ^ inb_add[47:0]);

(DEFUN STICKSUM (N)
    (IF (EQUAL (ESUB N) 0)
      (COMP1 (LOGXOR (BITS (INA_ADD N) 47 0)
                     (BITS (INB_ADD N) 47 0))
             48)
      (LOGXOR (BITS (INA_ADD N) 47 0)
              (BITS (INB_ADD N) 47 0))))

M4_sticky_co <= M3_sticky_co;

(DEFUN M4_STICKY_CO (N)
  (IF (ZP N)
      (UNKNOWN 'M4_STICKY_CO 1)
    (M3_STICKY_CO (1- N))))
```

- Result: a formal model consisting of mutually recursive ACL2 functions

Some Classes of Circuits

Combinational circuit

- Outputs are functions of inputs
- Admits a simpler (and efficiently executable) equivalent formal model

Simple pipeline

- Each signal is accessed only at its associated depth
- The result of removing all registers is an "equivalent" combinational circuit

Conditional pipeline

- Several pipelined operations of different latencies merged in order to share hardware
- Each operation corresponds to a set of input constraints that guarantee pipeline behavior

Analysis of Pipelines

- We have implemented a tool that analyzes conditional pipelines, propagates constraints, assigns signal depths, and establishes equivalences with combinational models.
- The proof effort may then focus on the simpler combinational model, which may also be used for testing
- Theorems pertaining to the combinational model may be "lifted" to equivalent theorems about the primary model

IEEE Compliance

According to IEEE Standard 754, each of the elementary floating point operations,

$$+, -, *, /, \text{ and } \sqrt{\ },$$

> ... shall be performed as if it first produced an intermediate result correct to infinite precision and with unbounded range, and then rounded that result according to one of the [prescribed] modes....

Formal Statement of Correctness

Faithful representation of IEEE compliance requires formal definitions of the following:

- Rational arithmetic operations
- IEEE rounding modes
- Floating-point encoding of rationals

```
(defthm correctness-of-fadd
  (let ((a (Pipe0_FSrc1_11 n))
        (b (Pipe0_FSrc2_11 n))
        (op (EPC_EX1_FPOpcode0_11 n))
        (rc (RQ_SpecCWRC_13) (+ n 2))
        (pc (RQ_SpecCWPC_13) (+ n 2))
        (result (Pipe0_FRes_FPA_15 (+ n 4))))
  (implies (input-conditions a b op rc pc)
           (equal (decode result)
                  (rnd (if (subtractionp op)
                           (- (decode a) (decode b))
                         (+ (decode a) (decode b)))
                       (mode rc)
                       (precision pc)
```

Proving Correctness

The proofs involve thousands of lemmas, which may be classified as follows:

- General floating-point theory
  - Bit vectors and logical operations
  - Floating-point representation
  - Rounding modes
- Special techniques
  - Constant injection
  - Newton-Raphson approximation
  - Booth encoding and compression trees
  - Leading and trailing one prediction
- Applications
  - FADD, FMUL, etc.

Floating-Point Addition: a Naive Approach

(1) Compare the exponent fields of the summands to determine the right shift necessary to align the significands;

(2) Perform the required right shift on the significand field that corresponds to the lesser exponent;

(3) Add (or subtract) the aligned significands, together with the appropriate rounding constant;

(4) Determine the left shift required to normalize the result;

(5) Perform the left shift and adjust the exponent accordingly;

(6) Compute the final result by assembling the sign, exponent, and significand fields.

Can any of these operations be performed in parallel?

Leading One Prediction

$$\begin{array}{rcl} a &=& \mathtt{1\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ \ldots\ 0} \\ b &=& \mathtt{1\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ \ldots\ 0} \\ a-b &=& \mathtt{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ \ldots\ 0} \\ \sigma &=& \mathtt{T\ T\ T\ T\ T\ T\ A\ B\ B\ B\ T\ B\ A\ T\ T\ \ldots\ T} \\ lop &=& \mathtt{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ \ldots\ 0} \end{array}$$

(1) Let $\sigma = \sigma_{n-1} \cdots \sigma_0 \in \{\mathtt{T}, \mathtt{A}, \mathtt{B}\}^n$ be defined by

$$\sigma_i = \begin{cases} \mathtt{T} & \text{if } a_i = b_i \\ \mathtt{A} & \text{if } a_i > b_i \\ \mathtt{B} & \text{if } a_i < b_i. \end{cases}$$

Let $\sigma_{n-1} \cdots \sigma_\ell$ be the maximal prefix of σ of the form $\mathtt{T}^p\mathtt{AB}^q$ or $\mathtt{T}^p\mathtt{BA}^q$. Then $expo(a-b) \in \{\ell, \ell-1\}$.

(2) Equivalently, ℓ is the largest k such that

$$\begin{array}{rl} \sigma_{k+1}\sigma_k\sigma_{k-1} \in & \{\mathtt{TAA}, \mathtt{TAT}, \mathtt{ABA}, \mathtt{ABT}, \mathtt{BBA}, \mathtt{BBT}, \\ & \ \mathtt{TBB}, \mathtt{TBT}, \mathtt{BAB}, \mathtt{BAT}, \mathtt{AAB}, \mathtt{AAT}\}. \end{array}$$

(3) Define lop so that for each k, the above condition holds iff $lop[k] = 1$. Then $expo(lop) = \ell$.

Leading One Prediction (cont.)

The RTL definition of lop (for the case $n = 68$):

```
lop_t[67:0] = a[67:0] ^ ~b[67:0];
lop_a[67:0] = a[67:0] & ~b[67:0];
lop_b[67:0] = ~a[67:0] & b[67:0];
lop[67:0] = {1'b0,
             (lop_t[67:2] & lop_a[66:1] & ~lop_b[65:0]) |
              (~lop_t[67:2] & lop_b[66:1] & ~lop_b[65:0]) |
              (lop_t[67:2] & lop_b[66:1] & ~lop_a[65:0]) |
              (~lop_t[67:2] & lop_a[66:1] & ~lop_a[65:0]),
             ~lop_t[0]};
```

Lemma *If $expo(\mathtt{a}) = expo(\mathtt{b}) = 67$ and $\mathtt{a} \neq \mathtt{b}$, then*

$$expo(\mathtt{lop}) - 1 \leq expo(\mathtt{a} - \mathtt{b}) \leq expo(\mathtt{lop}).$$

Summary

- Verification of the adder was a 20 man-week effort
- Much of the effort was spent in the development of a library of reusable results
- All hardware models, behavioral specifications, and proofs were encoded in the same logical language
- All proofs were mechanically checked and mostly surveyable
- Several design flaws, which had survived extensive testing, were exposed and ultimately corrected