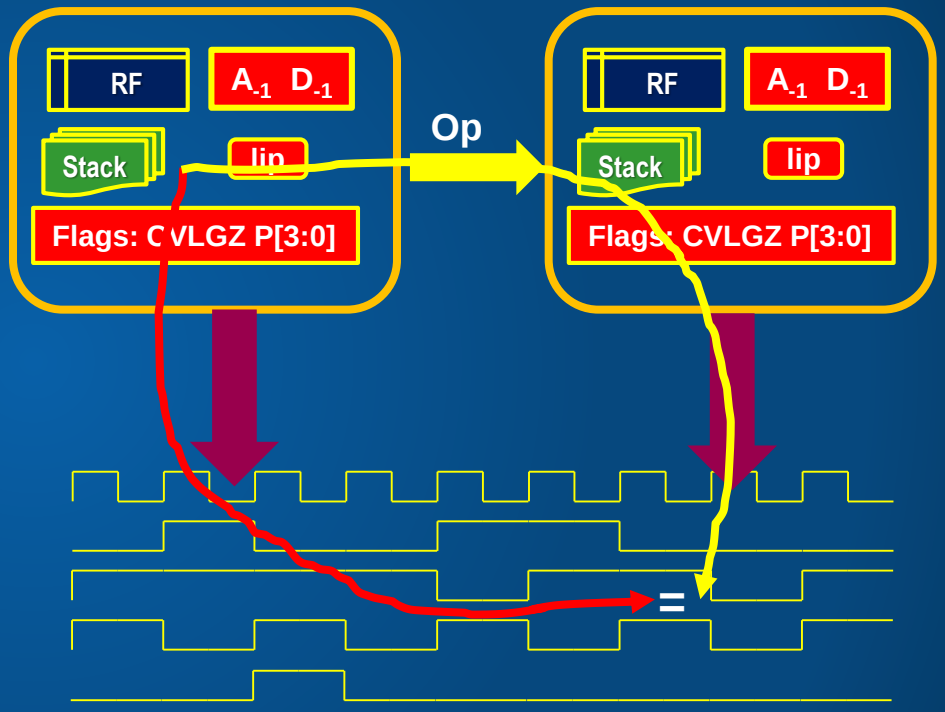
# Practical Formal Verification using BDDs & SAT

Carl Seger

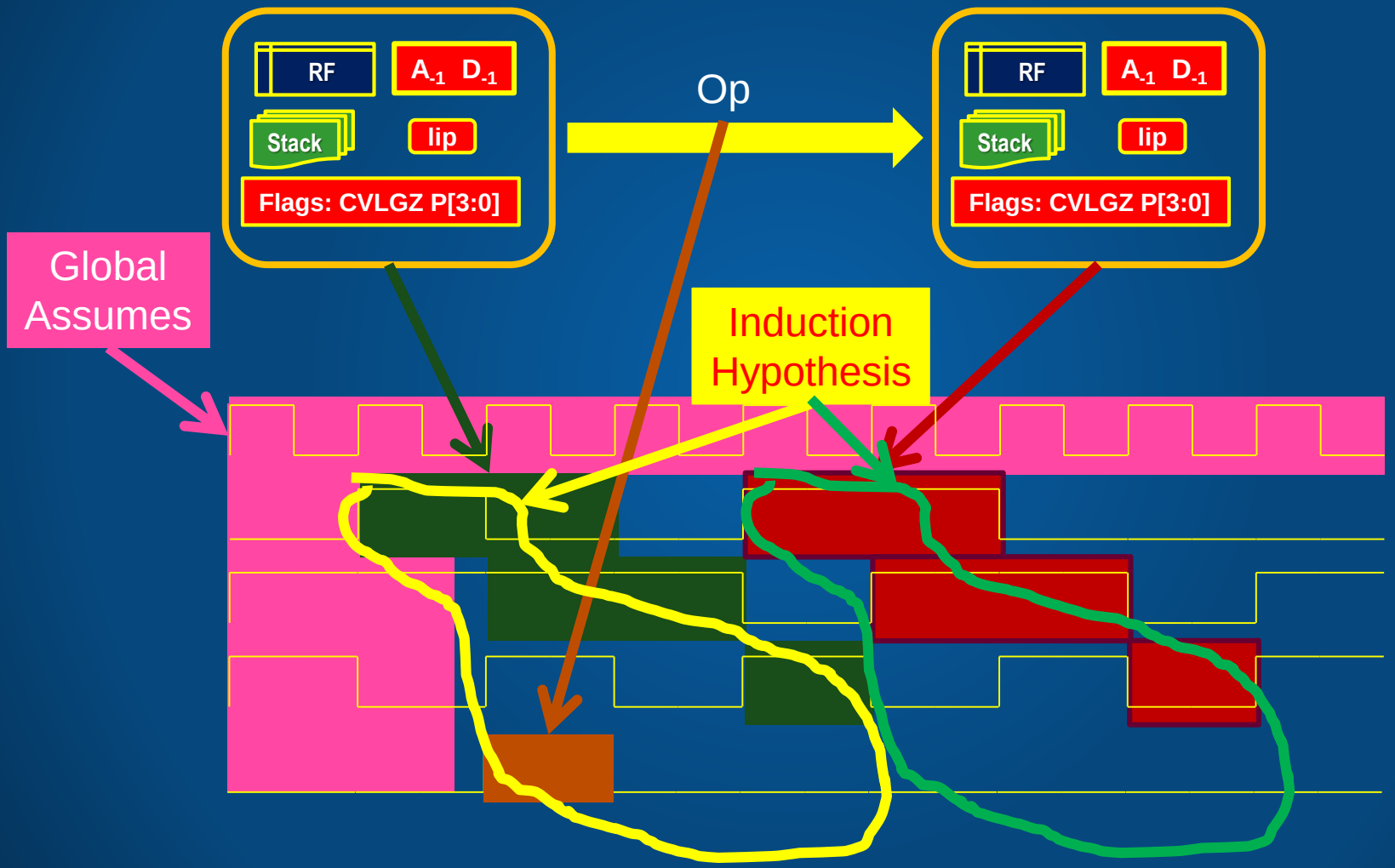
September 21, 2016

# Formal Verification in Theory

- Reference model verification:
  - Create an abstract state representation
  - Define an abstract next-state function
  - Define a mapping from abstract state to circuit state
  - Verify commutativity:
    For every instance in time and for every possible value in the machine, if the signal values match the mapped "before" state, then they will also match the "after" state.



# Formal Verification in Practice



# Real Life Example of State

- Register file:
  - 32 registers, 32 bit wide
- Bypass state:
  - Last register write address (if any)
  - Last data written/about to be written into register file
- Call stack
  - Stack pointer
  - 16 entries of 20bit addresses
- Logic instruction pointer
  - 20 bit instruction pointer
- Memory
  - RAM, ROM and IO space (1 distinct address in each selected by “control register”)
- Flags:
  - C: (unsigned) carry/overflow
  - V (signed) overflow
  - L: less-than-or-equal to zero
  - G: greater-than-or-equal to zero
  - Z: equal to zero
  - P[3:0]: flags for parallel byte-wide operation

# Specification: What Instruction Does

- Specification derived from C++ reference model
  - Translated specification very similar to C++ code
    - C++ microcode simulator: ~4k lines
    - FL specification: ~5k lines
  - Bugs in fl spec pointed directly to bugs in uc.cpp
    - Some minor translation bugs did creep in but were easy to find.
- Evaluation speed very good:
  - Complete ISA next state function for any uop: ~40 seconds
  - Complete ISA next state function for a particular uop: ~6 seconds

# Example of Specification

- C++ run function:

```
int FoxsimMicroController::
run_instruction(instruction_t instr) {
  pcode_inst_vopcode vopcode = decode_vopcode(instr);
  int imm = decode_immediate(vopcode, instr);
  int regA = (PROJECT_UC_EAS_VERSION < 4.0) ?
                (instr & 0x0f) : (instr & 0x1f);
...
  switch (vopcode) {
...
    case PCODE_INST_FFSB:
      mcs.isw.invalidate();
      if (mcs.R[regB].hasValue()) {
        int v = FFSB(mcs.R[regB].getValue());
        mcs.R[regA] = v;
        mcs.set_fv(v == -1 ? 1 : 0);
      } else {
        mcs.R[regA].invalidate();
      }
      break;
...
```

- FL specification:

```
let run_instruction mcs {instr::instruction_t} =
  let young_EAS = (PROJECT_UC_EAS_VERSION < 4.0) in
  let vopcode = decode_vopcode instr -->
  let imm = decode_immediate vopcode instr -->
  let regA = young_EAS? (instr & '0x0f) :: (instr & '0x1f) -->
...
  switch (vopcode) [
...
    case [PCODE_INST_FFSB] (
      let mcs = mcs.isw_invalidate() -->
      if( (mcs.R.read[regB]).hasValue() ) then (
          let v = FFSB((mcs.R.read[regB]).getValue()) -->
          let mcs = mcs.Rwrite(regA, Xint v) -->
          mcs.set_fv(v == '-1 ? '1 :: '0)
      ) else (
          mcs.Rapply(regA, (\r. invalidate r ()))
      )
    ),
```

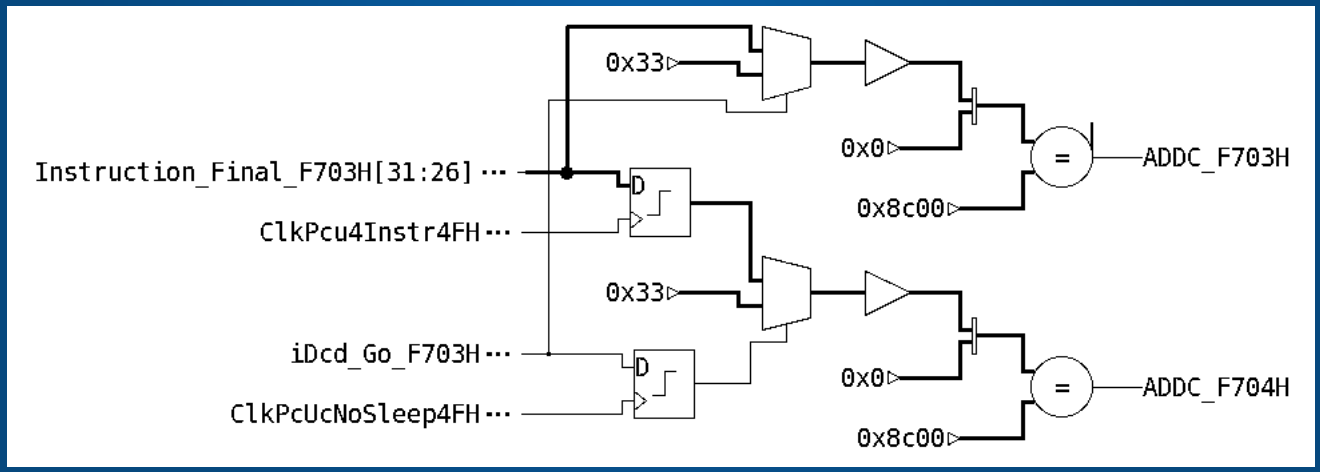
# Induction Hypothesis

- Predicate that essentially says: “machine is ready for next instruction”
- Almost 200 signals.
  - Most deal with redundant uopcode decoding.
    - See next slide
  - A mixture of manually derived and automatically computed using backwards simulation.
- The correctness is ensured by checking it as part of the verification of each uop

- NOTE: this predicate is potentially very fragile if major changes of the control parts of the machine is needed due to timing, area or power requirements.

# Inefficient Logic in RTL

- The same instruction is often decoded in more than one pipestage.
- Due to clock gating (different clocks) synthesis is unlikely to remove the redundancy.
- For example:




- There are > 60 cases of this type of duplicated logic

# Actual Formal Verification

- The abstract state is made fully symbolic
- The “distinct” memory addresses are made fully symbolic
- The abstract next state is computed by evaluating the fl program using “run_instruction” on the symbolic state
- The mappings are defined using STE’s verification engine
- The circuit is then symbolically simulated using either BDDs or SAT.
  - BDDs uses a manually defined (“obvious”) variable order.
  - First attempts are (usually) SAT based since it is far faster for weeding out silly mistakes.
  - Final verification is (usually) BDD based.
- Total number of Boolean variables: >1500

# Finding a Needle in a Haystack vs Finding a HW bug


vs.

Finding a single pair of values for a double precision floating point divide operation that fails.

**For probability to be the same, how big should the haystack be? (Assume half-sphere haystack)**

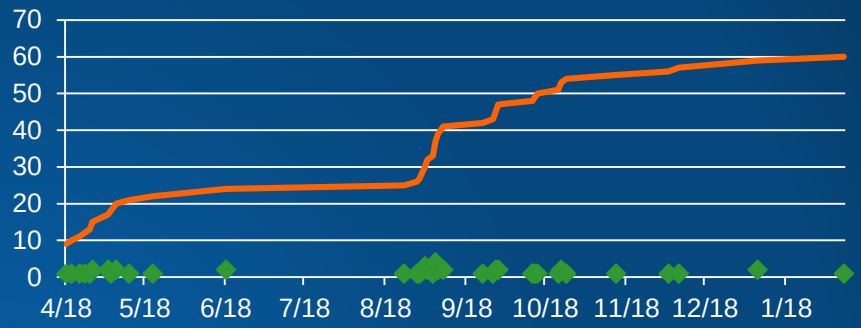
**Answer: Radius ~550 light years!**

# Snapshot Example of Results

- Total verification time for one uop typically ~5 minutes
  - This includes creating the specification property
- Complete regression takes ~4 hours on ~30 machines each with 24 threads and 256GByte of memory.

# Bug Analysis Summary

- 60 tickets/bugs filed:
  - 34 bugs in uc.cpp
  - 37 bugs in RTL
  - 8 bugs in EAS

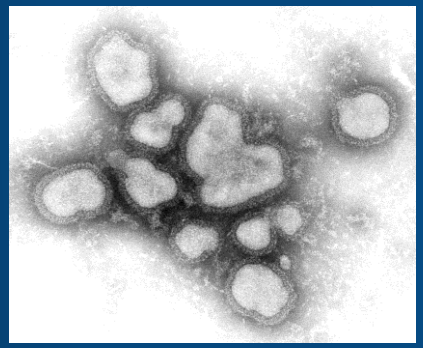



- At least one ticket caused a change in all three models!
- It appears 2-3 bugs were already present in existing HW!
- Most bugs related to setting of flags and/or missing assumptions, but a few affected main results.
- All filed tickets have been fixed
  - Some required more than one spin to get right.
- Most complex bug required a program with 71 instruction and carefully selected program layout to split cache lines + suitable cache misses
  - “Friday the 13th bug”

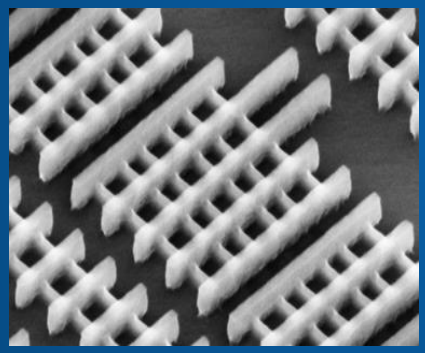
Questions?

# Quiz 1 – Small Numbers

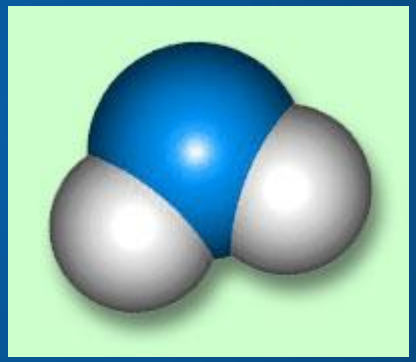
Order the following in order of size (smallest first)



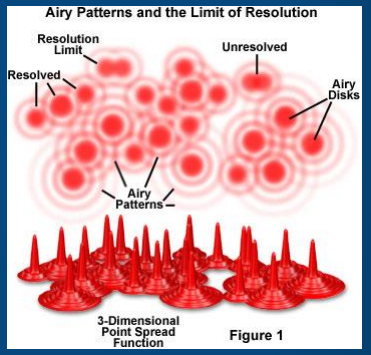
Influenza A virus



Transistor in microprocessor as of June 2015



Water molecule

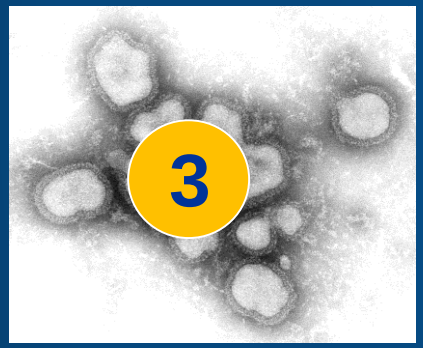



Resolution of optical microscope

# Answer Quiz 1

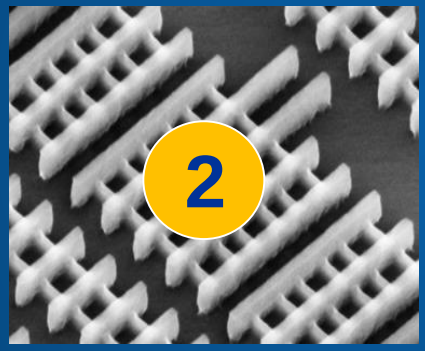
## Order the following in order of size (smallest first)

~100nm



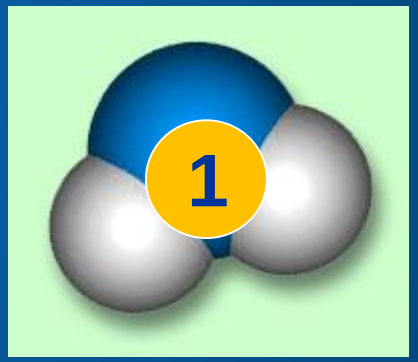

Influenza A virus

~14nm



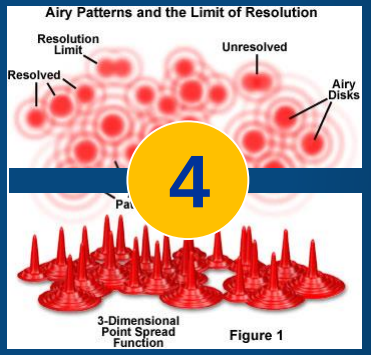

Transistor in microprocessor as of June 2015

~0.3nm




Water molecule

~300nm




Resolution of optical microscope

# Quiz 2 – Large Numbers

Order the following in order of size (largest first)

Number of light bulbs in the world

Number of atoms in the Empire State Building

Number of transistors in a 2014 cell phone

Number of patterns needed to simulate all possible inputs to one AVX instruction (two 256-bit inputs)

# Answer Quiz 2

Order the following in order of size (largest first)

$\sim 10^{10}$


Number of light bulbs in the world

$\sim 10^{31}$


Number of atoms in the Empire State Building

$\sim 10^{11}$


Number of transistors in a 2014 cell phone

$\sim 10^{154}$


Number of patterns needed to simulate all possible inputs to one AVX instruction (two 256-bit inputs)