Overview for Viorel Preoteasa’s defence

John Harrison
Intel Corporation
`johnh@ichips.intel.com`

Åbo Akademi

10th November 2006

The correctness problem

It's fundamentally difficult to produce correct hardware, software, protocols etc.

Bugs are commonplace, almost routine.

Quality issues are becoming a major preoccupation in the hardware and the software industry.

Many software companies devote more resources to quality/validation than to writing the software in the first place.

In hardware, validation is often the crucial bottleneck.

Famous floating-point bugs

- Patriot missile failure during first Gulf War in 1991
- Explosion of Ariane 5 rocket on maiden flight
- Faulty transcendental functions in early HP-35 calculators
- Bug in floating-point division (FDIV) instruction on some early Intel®Pentium® processors

Things are not getting easier

The FDIV bug caused Intel to set aside $475M to cover relacement.

The environment is becoming even less benign.

- The overall market is much larger, so the potential cost of recall/replacement is far higher.
- New products are ramped faster and reach high unit sales very quickly.
- Competitive pressures are leading to more design complexity (recently around 4x per generation)

Limits of testing

Bugs are usually detected by extensive testing.

- Too many possibilities to test them all
- In hardware, it's slow to test under simulation

The solution: *prove* correctness instead of just testing: formal verification.

A spectrum of formal techniques

There are various possible levels of rigor in correctness proofs:

- Programming language typechecking
- Lint-like static checks (uninitialized variables . . .)
- Checking of loop invariants and other annotations
- Complete functional verification

FV in the software industry

Some recent success with partial verification in the software world:

- Analysis of Microsoft Windows device drivers using SLAM
- Non-overflow proof for Airbus A380 flight control software

Much less use of full functional verification. Very rare except in highly safety-critical or security-critical niches.

Problems with formal software verification

There is an elegant theoretical basis for software verification, but:

- Theory much uglier with decomposed state space
- Often for toy language without procedures or recursion
- No theory to support dynamic data structures, pointers
- Inadequate mechanized support makes proofs impractical

Exactly these limitations are addressed by this work.