SYNOPSYS®
Predictable Success

Embedded software verification

an EDA perspective

Per Bjesse
Synopsys

Caveat

There are few established truths as of yet in this field.

Caveat

There are few established truths as of yet in this field.

This is part of why this is an exciting area to be looking at!

Why does this matter from an EDA perspective?

- Violent agreement:
  - Embedded software is a great pain point for our customers
  - There are large potentials for money in solving (the right) problem.
  - The problems are very hard.
- Less clear:
  - What is the right methodology for low level software verification in general?
    - Formal verification in particular.
  - What are the right problems to solve?

This is a pain point for our customers

- DVCON 2010 survey:
  - More than 35% of respondents spent more development and verification effort on embedded software than on hardware.
  - Hw/sw coverification fourth biggest pain point for respondents
    - overall complexity, full system validation and chip-level verification preceded
  - The problem is growing.

- VDC report on embedded software, 2004:
  - 24% of projects cancelled due to slipped schedules.
  - 54% behind schedule (average 3.9 months)
  - 80% of development effort on finding late bugs.
  - Old data, but things have not improved.

This is a pain point for our customers

- Customers are on a vicious treadmill to get things out the door.
  - Life cycles for devices are short on average.
- For embedded devices, bugs that can't be patched remotely can mean the scrapping of a whole model
  - Stakes are high!

Why does this matter from an EDA perspective

- It is strategic
  - Adjacent market of large size, that is growing.

- We may have to care: Software as hardware paradigm:
  - Implement algorithms in software, just add more processors to get speed
  - This way resources can be shared:
    - Hardware is additive!

What are some interesting research problems?

Unifying thread: Verification where you need both software and hardware models

- More about why later.

What are some interesting research problems?

- Hardware is often used to accelerate software
  - How do you prove equivalence?
  - More complicated than stock C-to-RTL verification.

- How do you model hardware so you can apply software verification methods?
  - Already exists languages for modeling hardware as software components
  - How do you build an efficient model from these, and is it enough?

- Raksha security processor, ARM trust zone.
  - Hardware support for ensuring secure running of software.
  - How do you prove these kinds of system correct?

What are some interesting research problems?

- Hardware is often verified today by software running on embedded processor.
  - Usable throughout development all the way to post-silicon debug
  - How do you leverage formal techniques in this environment?

- Automated debug in a low-level hardware/software environment

- Static analysis:
  - Super linting of combined hardware/software models.
  - What is the cone of influence of a property of low-level software with RTL?
  - ….

The commercial tool point of view

- Lots of money spent on embedded software development.

- However: The further the software is from the hardware, the less money there is in there.
  - Most money spent on real time operating systems, processors.
  - Development tools come for free with these.
  - What money are you spending on software development for general software?

The commercial tool point of view

Some sweet spots:

1. Tools aimed at hardware engineers that have to deal with software
   - They are used to hardware tool license costs.

2. Tools that help find bugs and debug the software that interface to hardware.
   - Debug help #1 concern for customers buying verification environments.

3. Model based development
   - Matlab is a good example---it provides formal verification tools on a model from which both code and RTL generation can be done.

Acknowledgements

Thanks to Badri Gopalan and Tom Borgstrom for thoughtful discussions on this subject.