HW/SW Co-Design of Embedded Systems

Lecture 3
Verification

Timotej Torok

Outline

The Design Flow

- Informal Specification, Constraints
- Arch. Selection
- System Model
- System architecture
- Estimation
- Scheduling
- Functional Simulation & Formal Verification
- Mapper and scheduled model
- Simulation & Formal Verification
- SW model
- Simulation
- HW model
- OK
- Not OK

The Design Levels

- System
  - ... RTL
  - Gate
  - Transistor
  - Electronic

What is the right level for verification?

What is verification?

Design verification involves taking steps to guarantee that a design will perform according to its specification.
The Basic Levels of Verification

1. Design verification – have we designed what we intended to design?
2. Implementation verification – have we actually implemented our design?
3. Manufacturing verification – have we actually manufactured our implementation?

What we want to verify?

We can not verify our intentions about design – vague, imprecise, and can not be written down

Verifiable properties:
- Compiled "c" program reflects source codes
- Design in VHDL reflects implementation in FPGA or ASIC
- "these two" events never happen at the same time
- ...

How we can verify?

- Simulation
- Formal verification methods

Verification by Simulation

- often the main tool for checking the design
- design team usually tries few test cases
- at the end of design phase – simulation over extended period of time
- brute-force manner * distributed simulation * special simulation HW
- despite this the serious design errors often remain undetected

Simulating a 256 bit RAM

- 10^80 possible combinations of initial state and input
- 12^12 test/cases per second on single electron
- Single electron 10^-13 kg
- Our galaxy 10^17 kg
- We started at the time of the Big Bang about 10^10 years ago
- we would just have reached 0.05% mark of competing our task
Pipelined Processor

- It is a good idea to verify what from the possible instructions generate a trap in a processor.
- Because of aggressive pipelining the verification by simulation is often impossible.

Formal Verification

- Tries to overcome the weakness of verification by simulation
- Formal verification methods exploit powerful tools of mathematics rather than brute-force
- Our rescue: Mathematical induction
- Refers rather to hardware than software parts – Why?

SW vs. HW Verification

1. Hardware is often regular and hierarchical
   • Reason:
   Willingness of HW designer to use very small set of primitives; very restrictive rules on how to use them.
   While SW designers want to use anything what is available.

SW vs. HW Verification (cont’d)

2. Re-use of HW designs is common practice
   • Reason:
   Therefore much of the verification effort can be amortized over several products.

SW vs. HW Verification (cont’d)

3. Hardware designers are already familiar with writing detailed specifications
   Widespread use of VHDL, Verilog, ...

SW vs. HW Verification (cont’d)

4. The primitives are simpler
   • Reason:
   It is much more easier to describe a latch ("D") than some semantics of assignment statement in a programming language.
5. The cost of fixing bugs
   - Reason:
     In HW designs a design error can mean
     * a six months delay
     * loss of considerable amount of money

3 basic approaches are:
• theorem proving
• model checking
• symbolic simulation

- supported by CAD tools
- deep mathematical background
- each approach has some strengths and weaknesses

Theorem Proving
Basic idea:
1. Specify the implementation of component
2. Specify a behavioral models of subcomponents
3. Specify the intended behaviour of component
4. Prove that the implementation satisfies its intended behaviour

Underlying theory:
Formal theory S of predicate logic.
1. A finite alphabet. A finite sequence of these symbols is called an expression of S
2. A subset of the expressions of S are called well-formed formulas of S
3. A finite set of axioms for S
4. A finite set of rules of inference is given

Formal proof is a sequence:
f(1), f(2), f(3), ... f(n)
Every f(i) is either an axiom or f(i) can be derived from f(k); k < i

Small example
NAND gate implementation
NAND gate specification
Mapping the function to the architecture
How to proof that the implementation reflects the specification?
**Small example (cont’d)**

**NAND gate specification**

Mapping the function to the architecture

**NAND gate implementation**

Mapping the function to the architecture

Existential quantifier ($\exists$) is used to model the internal connections.

Formula reflecting the specification:
$$o = i_1 \& i_2$$

Formula reflecting the implementation:
$$\exists x \text{ NANDgate}(i_1, i_2, x) \& \text{ NOTgate}(x, o)$$

**Evaluation of Theorem Proving**

- Can be mechanically checked – it is a main advantage
- Computer-based theorem-provers (Boyer-Moore Theorem Prover, Cambridge HOL System, …)
- Deriving a formal proof is overwhelmingly tedious
- It can be proven that formal implementation satisfies the formal specification
- Optimization at RTL can introduce problems because this level does not retain the hierarchy
- Requires large amount of effort (the system must be formally specified at two levels, the theorem-prover must be guided)

---

**The End**