Hardware description and verification

http://www.cse.chalmers.se/edu/course/TDA956/

Getting hardware designs right

[using ideas from computer science]

Mary Sheeran

Verification

Design verification is the task of establishing that a given design accurately implements the intended behaviour.

In current projects, verification engineers outnumber designers, with this ratio reaching two or three to one for the most complex designs. Design conception and implementation are becoming mere preludes to the main activity of verification...

Without major breakthroughs, verification will be a non-scalable, show-stopping barrier to further progress in the semiconductor industry.

Int. Technology Roadmap for Semiconductors, 2006

Current state?

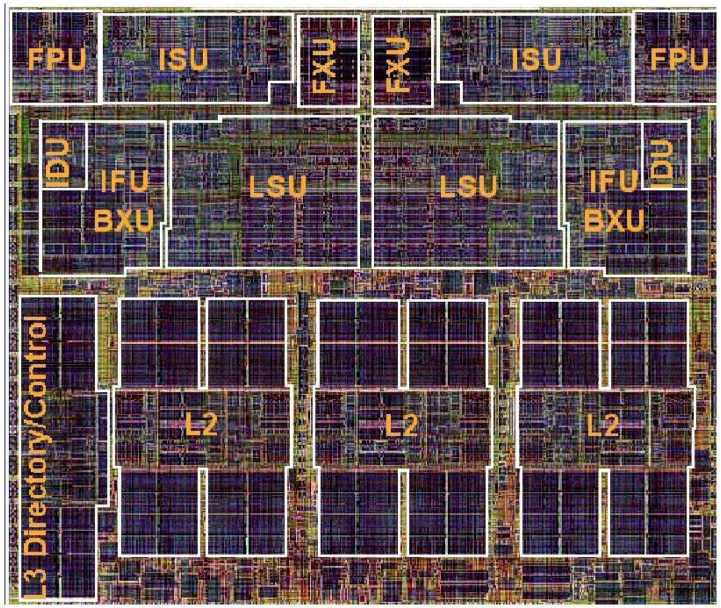
Simulation is the work-horse of verification.

The following 3 slides show the view from IBM
(i.e. the high frequency domain)
[Jacobi’s talk is on the FMCAD06 web-site. See links]

BUT, formal verification is coming on strong

(see particularly the 2010 paper from IBM that is on the
schedule as well as tomorrow’s presentation by
Magnus Björk from Jasper)

Verification Steps



Unit-sim is the work-horse of verification

- unit is LSU, ISU, FXU, FPU, ...
- most bugs are found on unit-level

Once stable, units integrated to *core-level simulation*

- can execute real programs here
- special „test program generators“ tailored to test interesting scenarios [AVPGEN; GPro]

→verification on architecture level (reusable!)

Multiple cores/chips/IO/memory etc. integrated into *system sim*

- can verify MP effects with real cores/memory
- can verify „power-on & boot“ on this model
  - including config scan, bootstrap init, PLL ...
  - (back to „all the details there“)
- only few bugs slip into system sim → mostly „power on stuff“

(slide by C. Jacobi, IBM)

Macro- vs. Unit-level Verification

- Macros are blocks with 100-1000 registers
  - cover a certain functionality, and tie them together as one PD-entity
  - unit comprises dozens of macros
- Many macros heavily interact to achieve a certain functionality
  - FPU: typical macros are multiplier, shifter, adder, exponent macros, etc.
  - large interaction between macros for datapath control (shift-amount, carry‘s, etc.)
  - cache: fetch controller, address queue, directory compare, data access, ECC, ...
- Macro I/Os change late due to timing & bugs

- Unit is the lowest „transactional level“
  - perform multiply-add, fetch, store, ...
- Relatively stable & well-documented interfaces, which eases verification
  - usually a unit has ~200 I/O-signals and busses
  - a macro also has ~200 I/Os, and a unit has dozens of macros

- ➔ attempts made, but macro level too much overhead as main verif target

(Slide due to C. Jacobi, IBM)

Benefits and Drawbacks of Simulation

- It scales: from unit level to system level, always working on the real VHDL
  - nearly linear time / model-size
- Find most bugs: the simple ones immediately, the complex ones after some „cooking time“.
- Proven methodology → first hardware usually works amazingly well
- We know how it works
  - huge investment in training: re-use concepts, lessons-learned, sometimes code from previous project
  - want to verify a new unit design: „there‘s always somebody around who‘s done something similar before“.
  - project manageability: predictable technology

Drawbacks:

- some bugs found very late, never sure you got all
- some bugs not found at all before tape-out

(Slide due to C. Jacobi, IBM)

Drawbacks

some bugs found very late, never sure you got all

some bugs not found at all before tape-out

BUGS are BAD

Bugs cost HUGE amounts of money, both by delaying the product and (worse still) often causing respin (= new set of masks)

The grand-daddy of them all was

Nov. '94 Intel FPU bug

824633702441.0 times (1/824633702441.0) = 0.999999999$\underline{274709702}$

Fault in look-up table

COST about $500.000.000

BUGS are BAD

Bugs cost HUGE amounts of money, both by delaying the product and (worse still) often causing respin (= new set of masks)

The grand-daddy of them all was

Nov. '94 Intel FPU b

824633702441.0 times (1/8246337

Fault in look-up table

COST about $500.000.000

Intel's response was to hire many experts in formal verification and develop the Forte system (see links page)

Pentium 4 was first processor verified with FV on a wide scale
Schubert’s DAC’03 paper showed this chart:

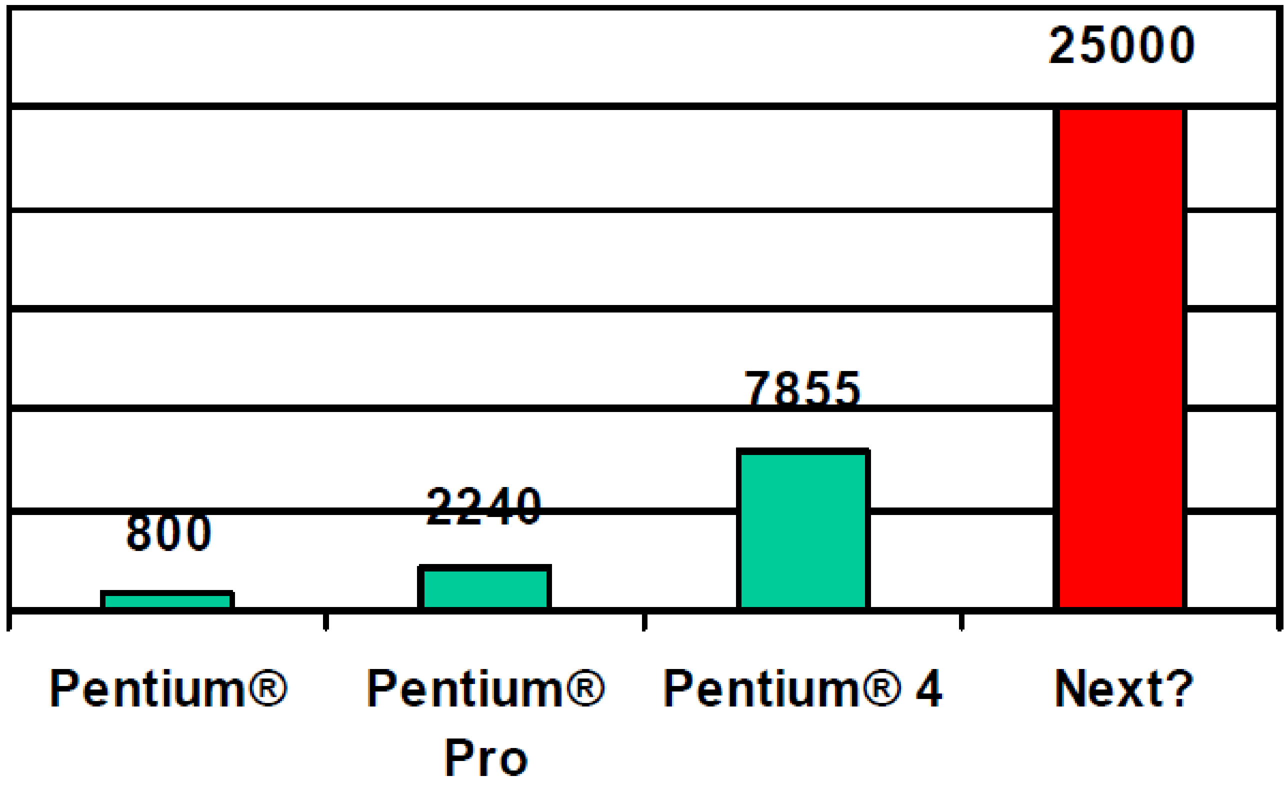


Figure 1. Pre-silicon logic bugs per generation.

Source: Tom Schubert : High Level Formal Verification of Next-Generation Microprocessors, DAC’03

Formal Verification

Based on mathematical or logical methods

Used either for bug-hunting or proof of properties (or both)

Aim to increase confidence in the riktighet of the system

In practice often combined with other methods
and then called hybrid or semi-formal
(for example look at talk about IBM's SixthSense tool
at FMCAD 2006, see links)

Some fundamental facts

Low level of abstraction, Finite state systems

=> automatic proofs possible

High level of abstraction, Fancy data types, general programs

=> automatic proofs IMPOSSIBLE

Two main approaches (1)

Use powerful interactive theorem provers and highly trained staff

for example Harrison’s work at Intel on floating point algorithms

VERY COOL. But not covered in this course.

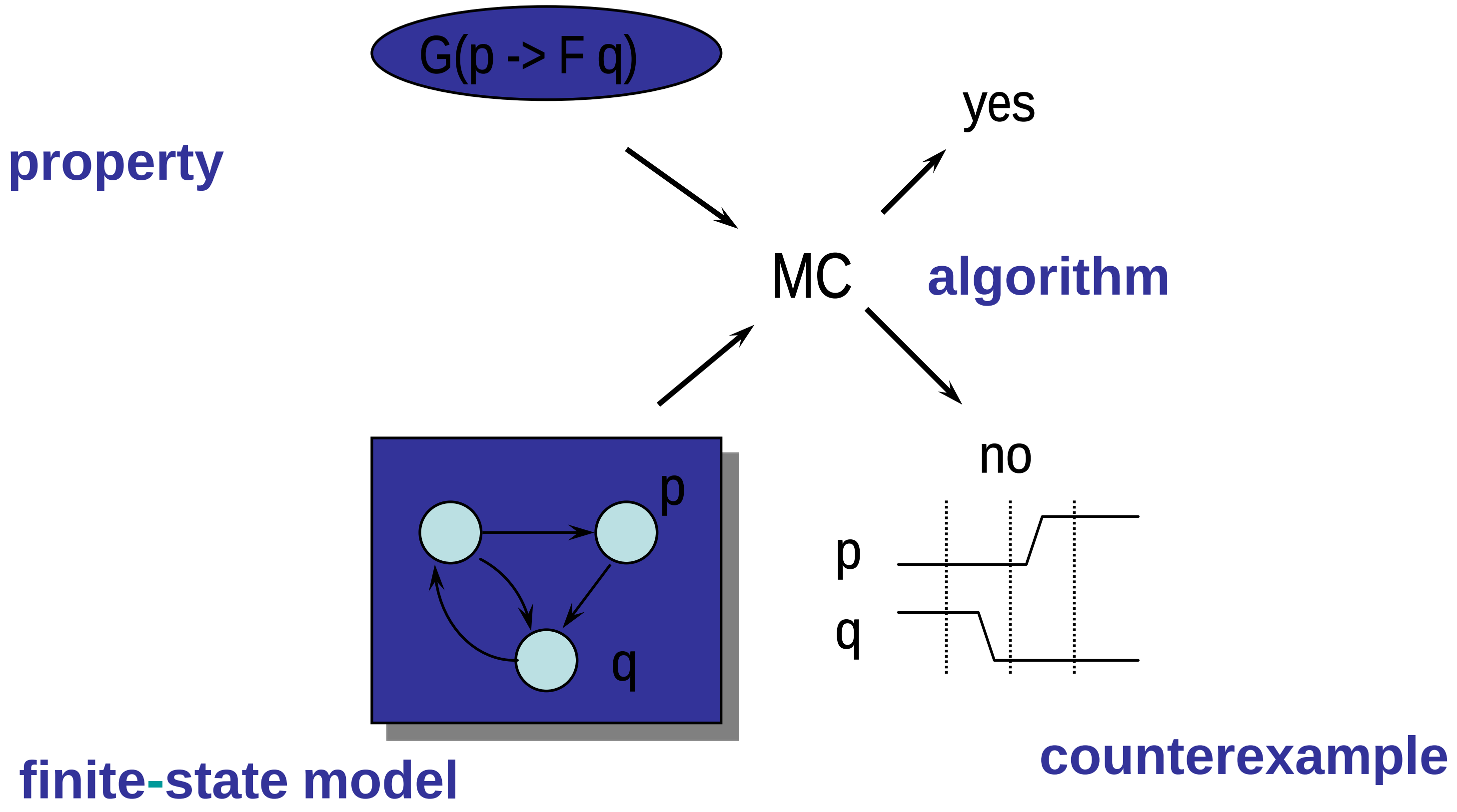
Two main approaches (2)

Squeeze the problem down into one that can be handled automatically

- reason about Finite State Machines (FSMs)
- works on fixed size circuits (not generic)
- hard part is writing the specs (but sometimes that can be automated too)

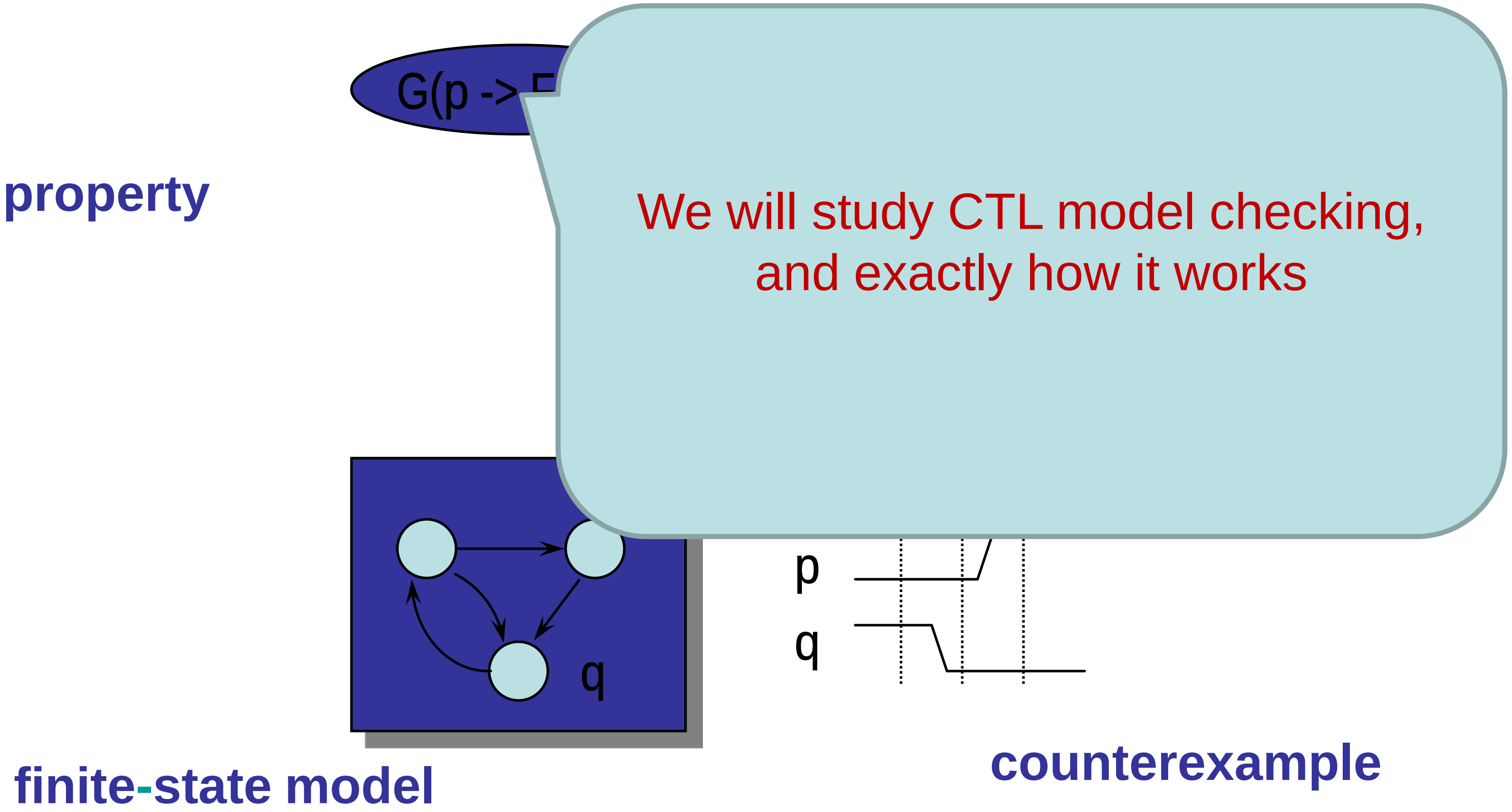
[Equivalence checking is very important but not covered in this course.]

Model Checking



(Ken McMillan)

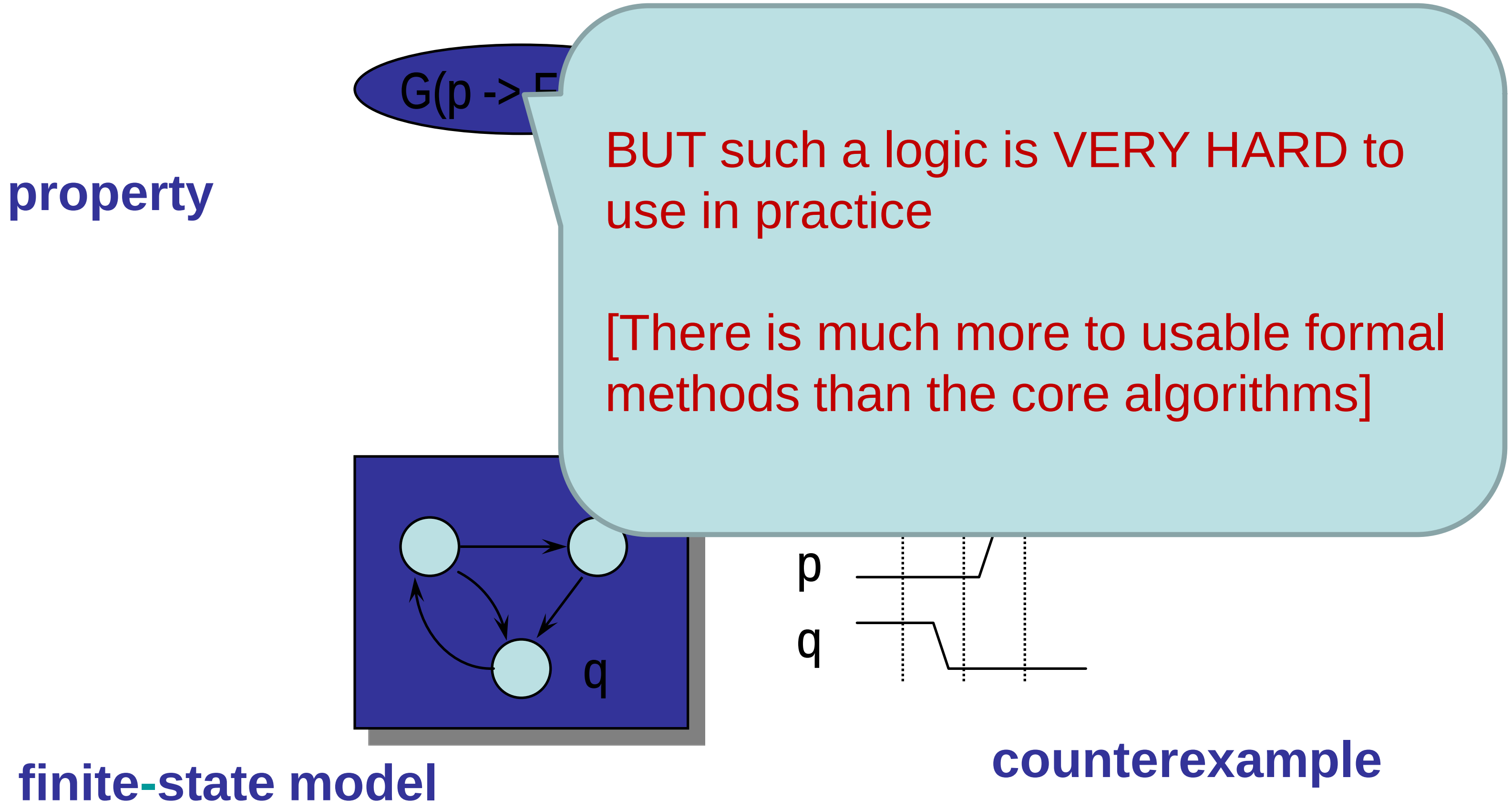
Model Checking



We will study CTL model checking, and exactly how it works

(Ken McMillan)

Model Checking

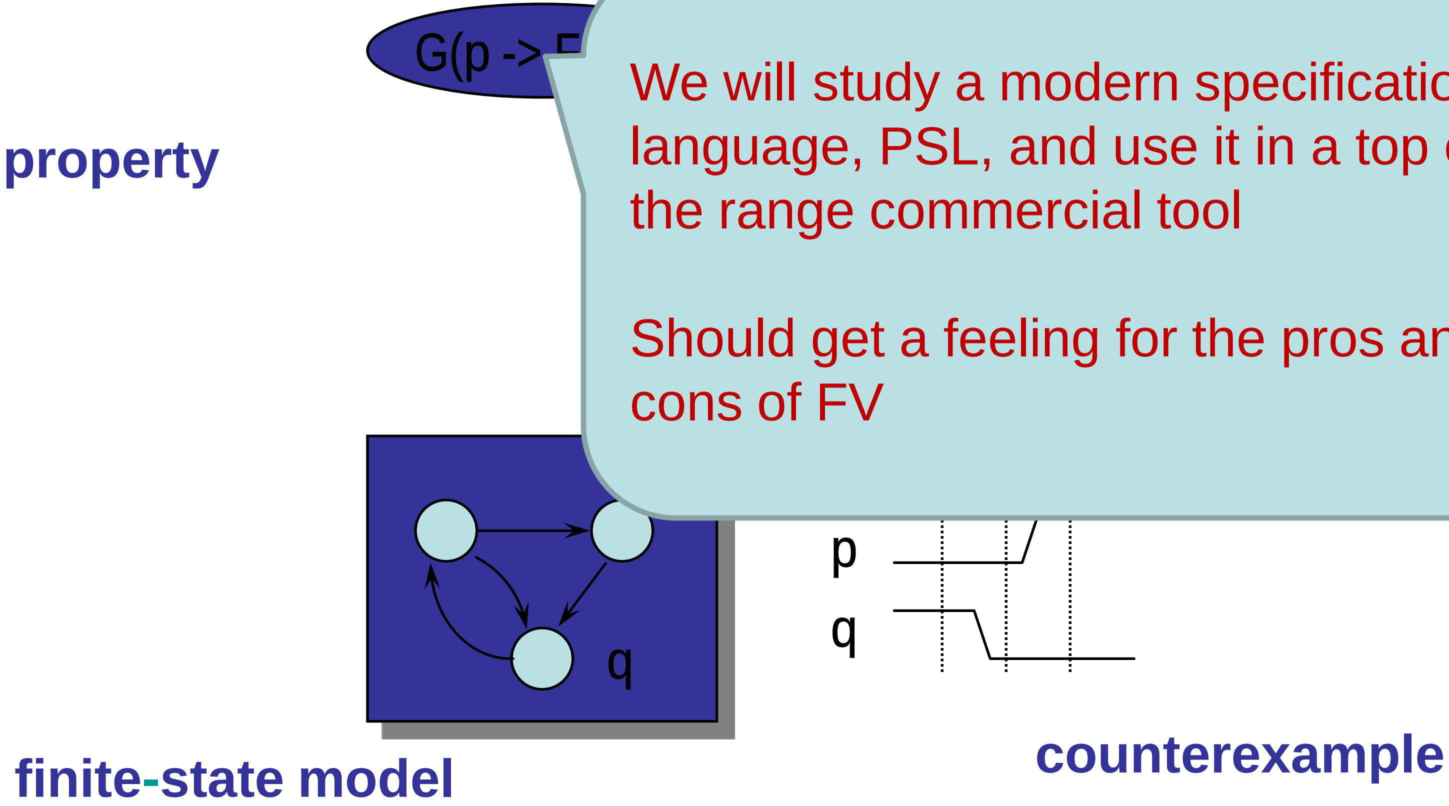


BUT such a logic is VERY HARD to use in practice

[There is much more to usable formal methods than the core algorithms]

(Ken McMillan)

Model Checking

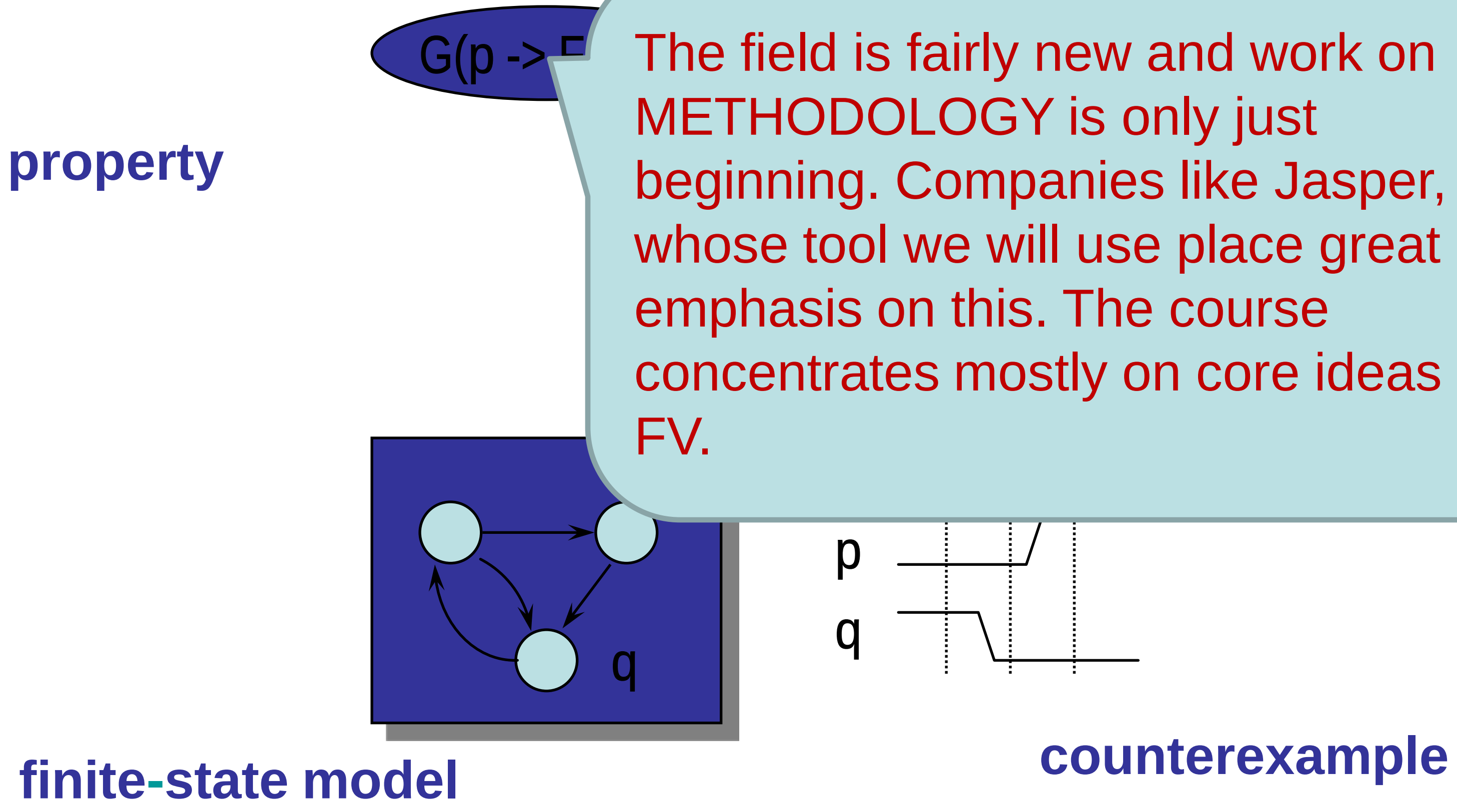


We will study a modern specification language, PSL, and use it in a top of the range commercial tool

Should get a feeling for the pros and cons of FV

(Ken McMillan)

Model Checking



The field is fairly new and work on METHODOLOGY is only just beginning. Companies like Jasper, whose tool we will use place great emphasis on this. The course concentrates mostly on core ideas in FV.

(Ken McMillan)

Industry and academia

The success of formal methods for hardware has depended on a very close collaboration between these two worlds.

A good example is the use of SAT-solving in hardware verification (which we have worked on here with success).

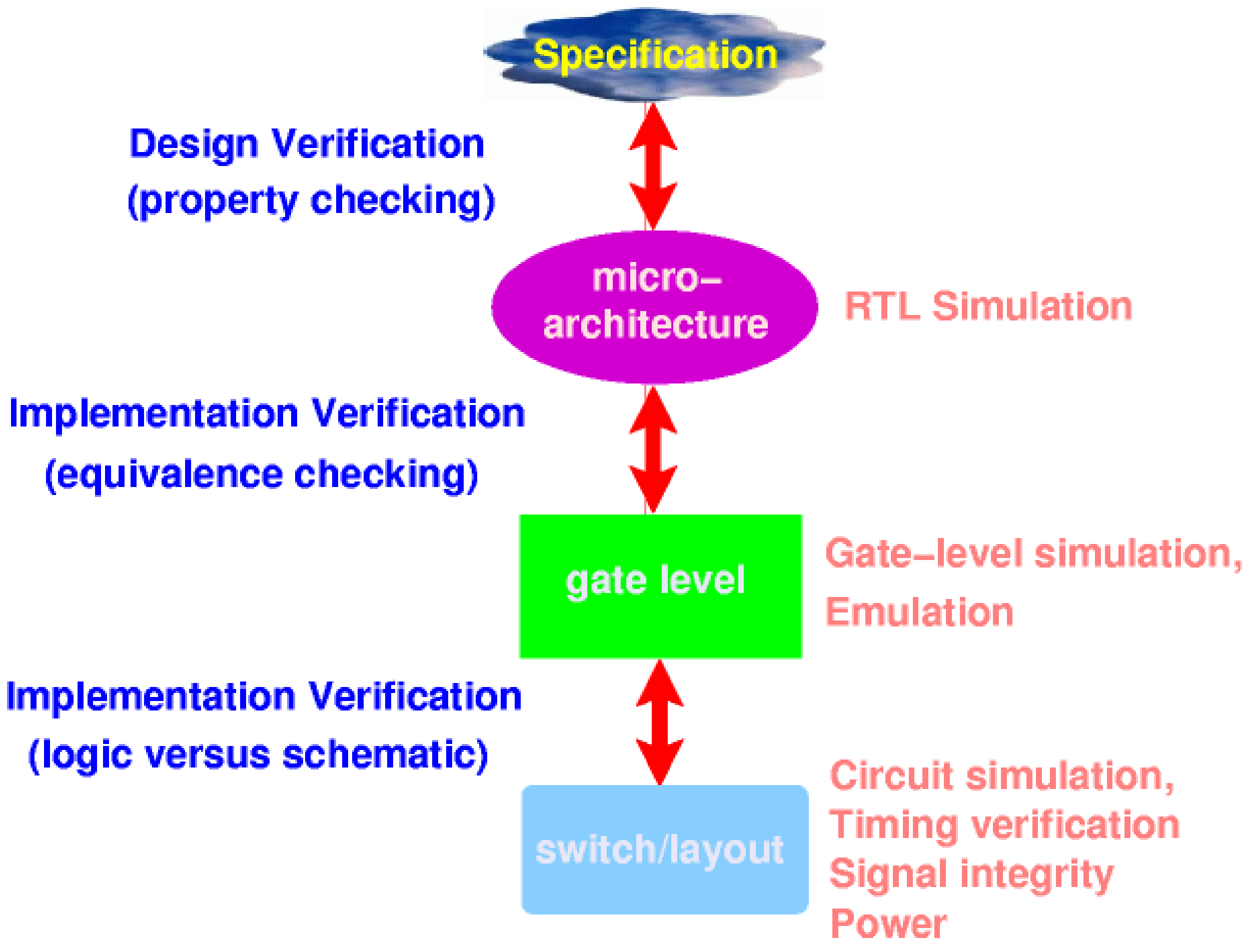
Steps in Designing Hardware

(according to slides by Jacob Abraham)

Designer	Tasks	Tools
Architect	Define Overall Chip C/RTL Model Initial Floorplan	Text Editor C Compiler
Logic Designer	Behavioral Simulation Logic Simulation Synthesis Datapath Schematics	RTL Simulator Synthesis Tools Timing Analyzer Power Estimator
Circuit Designer	Cell Libraries Circuit Schematics Circuit Simulation Megacell Blocks	Schematic Editor Circuit Simulator Router
Physical Designer	Layout and Floorplan Place and Route Parasitics Extraction DRC/LVS/ERC	Place/Route Tools Physical Design and Evaluation Tools

Design and Implementation Verification

(according to slides by Jacob Abraham)



FV has had great success BUT

What we can design and build far outstrips what we can verify and the gap is widening!

We are in a double whammy:
Pulled upwards in abstraction levels by the need to speed design and verification
Pulled downwards by the need to take account of physical properties (like power) at today's process nodes

We need to THINK OUT OF THE BOX as current hardware design and verification methods are running out of steam
=> this is a very interesting research field ☺

Course outline

Part 1:

Languages: VHDL and PSL

Tools: ModelSim, Jasper Gold, ...

Underlying ideas: BDDs, CTL model checking

Lab 1

Take home exam 1

Part 2:

Language: Lava

Tools: Lava, SMV, ...

Underlying ideas: SAT solving, temporal induction, synchronous observers

Lab 2

Take home exam 2

Guest Lectures exploring high level synthesis

Haskell to hardware, Singh, MSR Cambridge

Bluespec, Augustsson

At the end of the course

Regular written exam

Part 1:

Languages: VHDL and PS

Tools: ModelSi

Underlying id
model che

Lab 1

Take home e

Part 2:

Language: Lava

y, ...

ng, temporal
hronous

2

Guest Lecture
level synth

At the end of the course

Regular written exam

I also want to place these topics in the context of the industrial and research field of (formal) hardware verification. This may give some insight into the process of research and its way out into reality.

History lesson for this lecture: Compare the Schubert paper from 2003 (Intel) with the Paruthi paper from 2010 (IBM). See schedule.

Part 1:

Languages: VHDL and PS

Tools: ModelS

Underlying id
model che

Lab 1

Take home e

Part 2:

Language: Lava

V, ...

ing, temporal
chronous

n 2

It is important to note that hardware verification is NOT a solved problem!
Much remains to be done.
(See the links page for a snapshot.)

Guest Lectures
level synthesis

At the end of the course

Regular written exam

Details

3 slots (always ES51)

Tuesday 13.15
Wednesday 10.00
Friday 10.00

(not all slots will be used)

Supervised lab
Friday 08.00 Use this!

People

Mary Sheeran
(course responsible)

Emil Axelsson (TA)

My background

First degree in EE, never intended to become an academic!

Final year: theoretical CS, software engineering and electronics

MSc in formal methods (mostly course work about sofware verification)

DPhil (use of functional language to describe and reason about hardware)

Heavily involved with the research community in formal hardware verification (see FMCAD conference, for instance; key point is that the community is half industrial, half academic) Worked with Prover Technology (a startup) and changed my view of research!

Currently working with Ericsson on a language for DSP algorithm design (software!) and also on GPU programming. Next grant proposal is about approaches to programming FPGAs that also have several processors on them.

Your background?

How many are reading IESD / EESD ?

How many are reading Alg., Lang. Logic ?

Where do the rest of you come from?

Are you experienced in using VHDL? PSL?

Have you taken a course on Functional Programming?

Have you taken the SE using FM course?

Are you comfortable with logic and the idea of writing logical specifications?

Have you used a model checker or theorem prover?

What is your main interest relevant to this course? H/w design? Prog. Langs? Formal methods? Something else?

Conclusion

If you plan to work with electronic design, you must view verification as a central problem, and be well aware of modern approaches and their limitations.

You must also be equipped to learn about new methods as they arise. That’s why we also look at some theory and some current research.

Have fun, and feel free to give us feedback!
[Need 2 student reps. to make that work. Volunteer!]