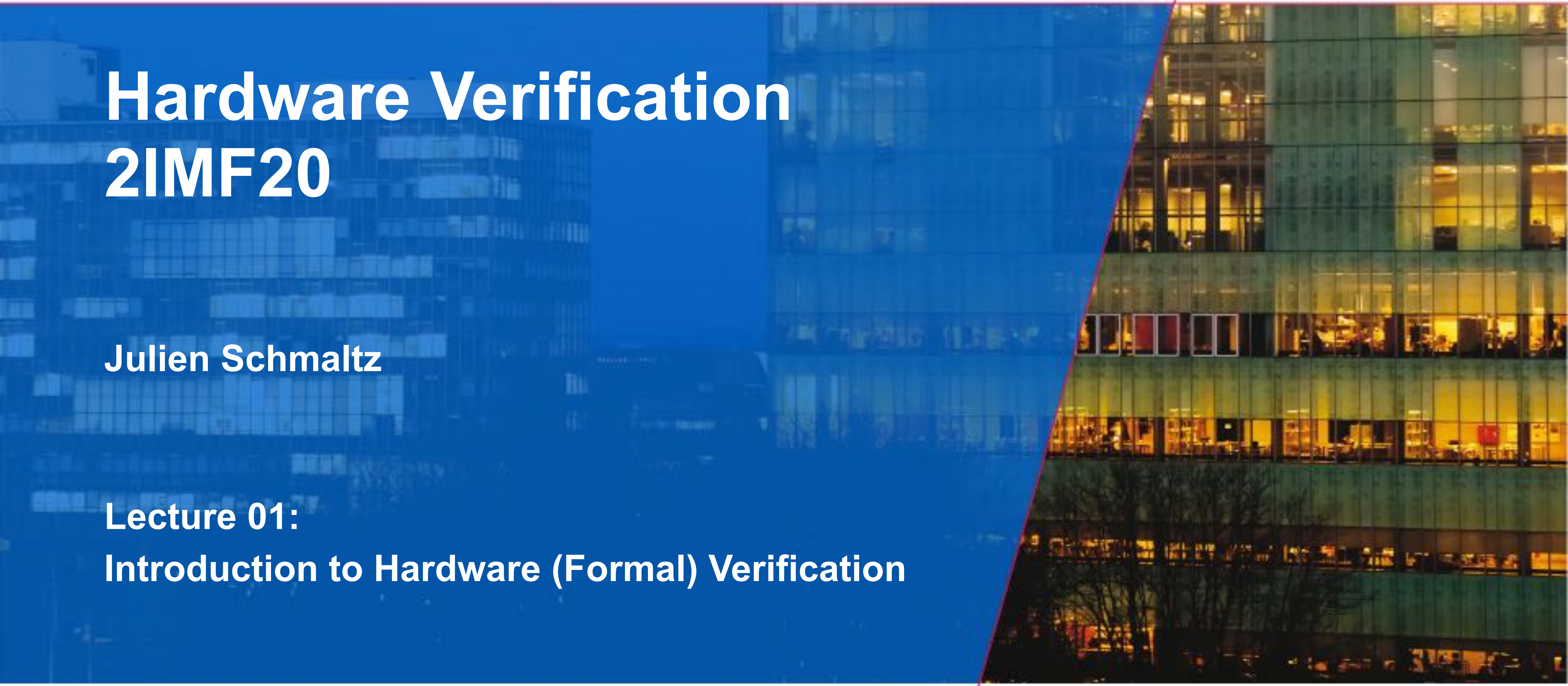
Hardware Verification 2IMF20

Julien Schmaltz

Lecture 01:

Introduction to Hardware (Formal) Verification



Technische Universiteit Eindhoven University of Technology

Where innovation starts

Lectures

- Two blocks every week (w36 to w43)

- Tue 13:45 - 15:30 Room LUNA 1.056
- Thu 08:45 - 10:30 Room LUNA 1.056

- Exam on 31.10.2017 (week 34)
  - 3-hour written exam

Lecturer

- » Julien Schmaltz
- » Room MF6.071b / j.schmaltz@tue.nl / 8691
- » Drop by any time, but might not be available
- » PhD 2006 from Univ. Grenoble (NoC Verification)
- » 2006-2007 Postdoc Saarbruecken, Germany
- » 2007-present Working in the NL at different universities
  - – Radboud University, ESI, Open Universiteit
  - – TU/e since February 2014

Round table - Quick check

- Short introduction about yourself. Where are you from?
- CSE/ES/AT/BIS? Which stream are you in? Pre-master?
- 1st/2nd year?
- Background hardware design or formal methods?

Grading

- 50 % grade of the written exam

- 50% practical assignments
  - groups of 2 students allowed

- Types of assignments
  - make your own circuit equality checker
  - verify a (simple) design using a commercial or academic tool
  - current plan (warning: subject to modifications!)
    - 1st exercise, easy, 20%
    - 2nd exercise, medium, 30%
    - 3nd exercise, small project 50%
    - (alternative might also be 20,40,40)

New 2017: Verify your own design

- Assignments 2 and 3 about using Jasper Gold from Cadence to verify hardware designs
  - simple communication network
  - simple execution unit of a micro-processor

- Possible alternative:
  - verify your own design !
  - something you made for another course, for instance
  - Verilog, SystemVerilog, or VHDL
  - must be synthesizable
  - interested ?
    - check with me / show me your design

Lecture materials

» Slides

» Papers

» Demo’s

» Notes from the lecturer

» Invited lecture from Industry (Dialog Semiconductors)

» Everything available from course website
  – http://www.win.tue.nl/~jschmalt/teaching/2IMF20/2IMF20.html

» Check website for updates!

A possible definition of ‘FV’

Formal Verification:

is the use of tools that ***mathematically analyse the space of possible behaviours*** of a design, rather than computing results for particular values.

Seligman, Schubert, Kumar

Formal verification: A short historical perspective

1950-1970

Birth of AI

1970-1980

Automated reasoning

1980-2010

Early applications

Industrial routine

FDIV bug (1994)

2010-present

Formal SignOff (Oski)
block level
with 100% FV

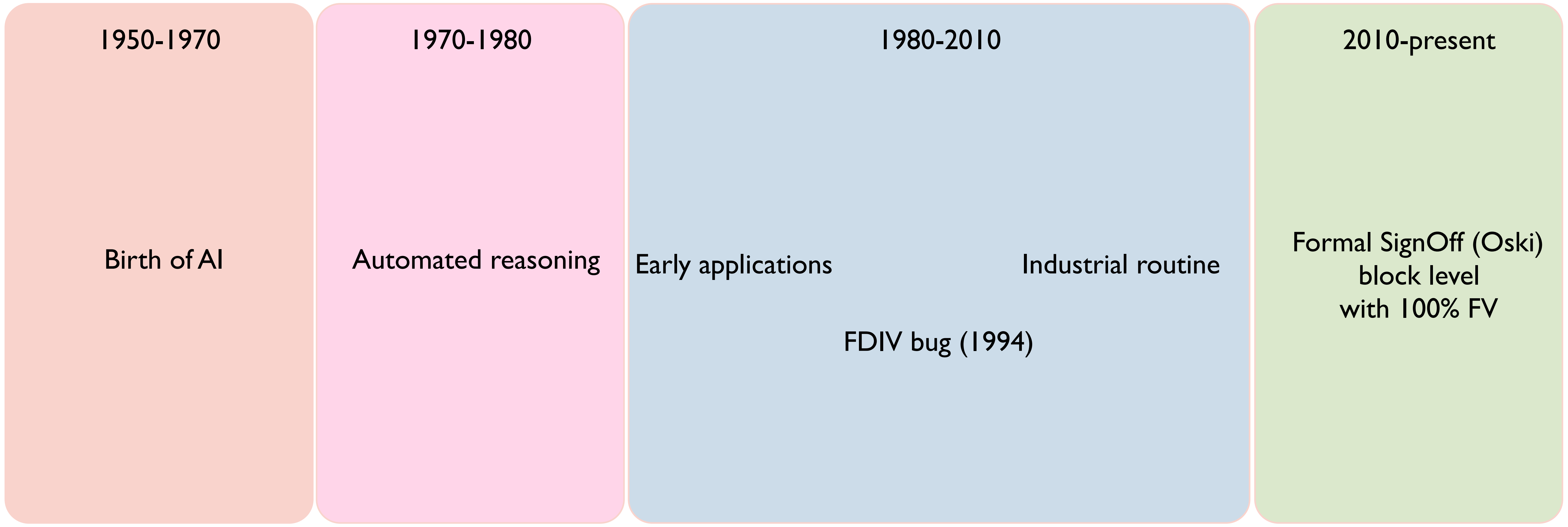
Formal verification: A short historical perspective

1950-1970	1970-1980	1980-2010	2010-present
Birth of AI	Automated reasoning	Early applications FDIV bug (1994) Industrial routine	Formal SignOff (Oski) block level with 100% FV

1956:
- Logic Theorist, Newell Simon and Shaw
- McCarthy coined the term “Artificial Intelligence”

1959: A proof of a large routine by A. Turing

Formal verification: A short historical perspective



1960's:

- Floyd and Hoare
- notion of partial and total correctness

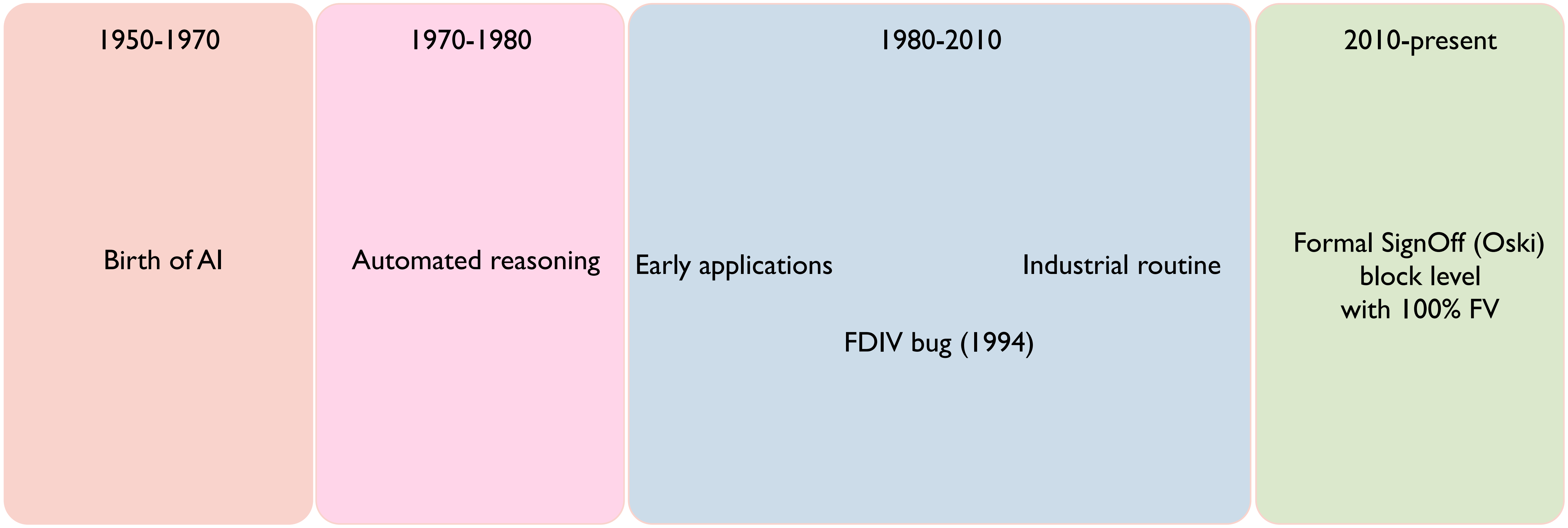
Formal verification: A short historical perspective

1950-1970	1970-1980	1980-2010	2010-present
Birth of AI	Automated reasoning	Early applications Industrial routine FDIV bug (1994)	Formal SignOff (Oski) block level with 100% FV

1970's:

- Boyer-Moore theorem prover (Nqthm, later ACL2)
- Introduction of Temporal Logics by Pnueli

Formal verification: A short historical perspective



1980's:

- Model checking Emerson, Clarke, and Sifakis
- Automatic proofs of temporal properties
- BDD from Bryant

Formal verification: A short historical perspective

1950-1970	1970-1980	1980-2010	2010-present
Birth of AI	Automated reasoning	Early applications FDIV bug (1994) Industrial routine	Formal SignOff (Oski) block level with 100% FV

1990's:

- Symbolic model checking (SMV) McMillan
- FDIV bug at Intel
- First EDA solutions for equivalence checking

Formal verification: A short historical perspective

1950-1970	1970-1980	1980-2010	2010-present
Birth of AI	Automated reasoning	Early applications Industrial routine FDIV bug (1994)	Formal SignOff (Oski) block level with 100% FV

2000's:

- Standards for writing assertions (PSL and System Verilog)
- Efficient SAT solvers
- Bounded-model checking (Biere et al.)

Formal verification: A short historical perspective

1950-1970	1970-1980	1980-2010	2010-present
Birth of AI	Automated reasoning	Early applications FDIV bug (1994) Industrial routine	Formal SignOff (Oski) block level with 100% FV

2010's:

- IC3 algorithm (Bradley)
- Formal Sign-Off (Oski Technologies)

Formal verification: A short historical perspective

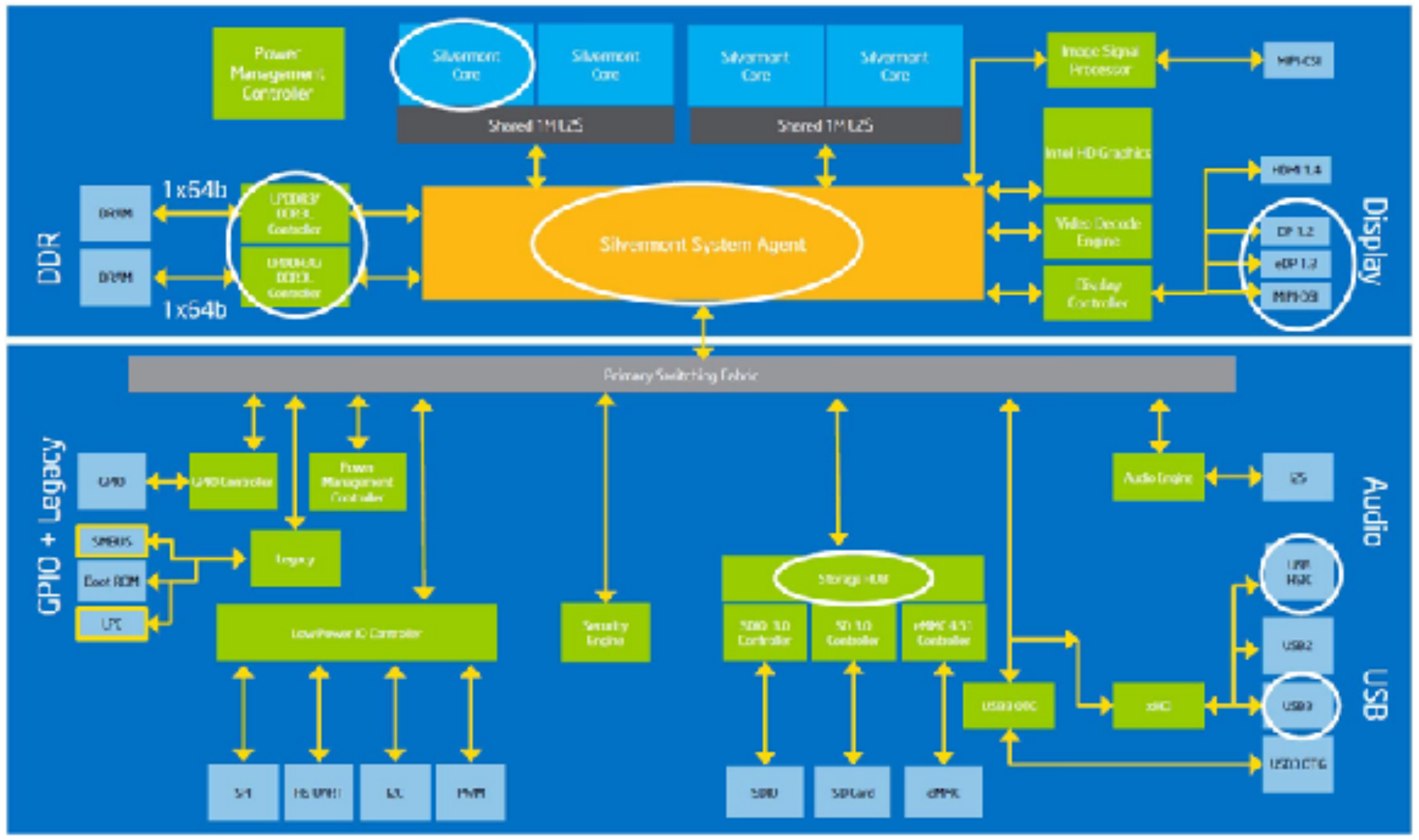
1950-1970	1970-1980	1980-2010	2010-present
Birth of AI	Automated reasoning	Early applications FDIV bug (1994) Industrial routine	Formal SignOff (Oski) block level with 100% FV

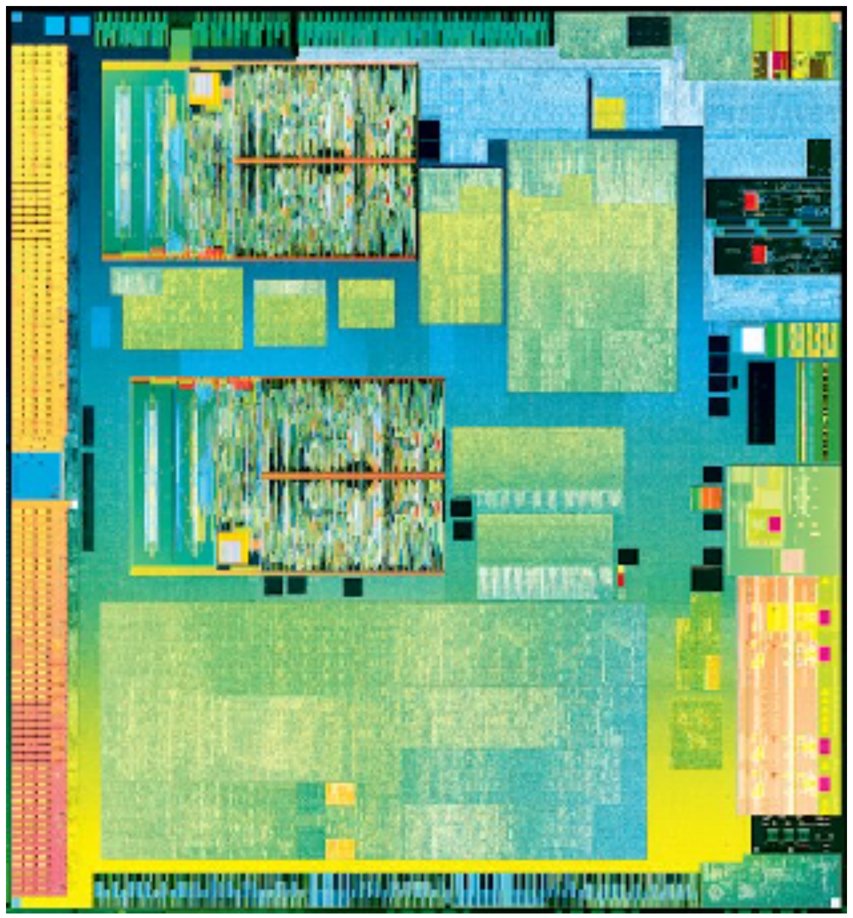
Future: system level formal verification

System Level (Formal) Verification Challenges

- Interconnects (can be seen as small systems)
- Component interactions (Power Management, Cache Coherence, NoC, …)
- Producer-Consumer relations (e.g. peripheral and DMAC)
- Liveness, livelock, deadlock, performance, …

Bay Trail SOC Block Diagram





Limitations and challenges (1)

Kurt Goedel
(1906-1978)

Alan Turing
(1912-1954)

Incompleteness & Undecidability

"This sentence is not provable"
(liar paradox)

Halting problem

Limitations and challenges (2)

Stephen Cook

(1939 - .)

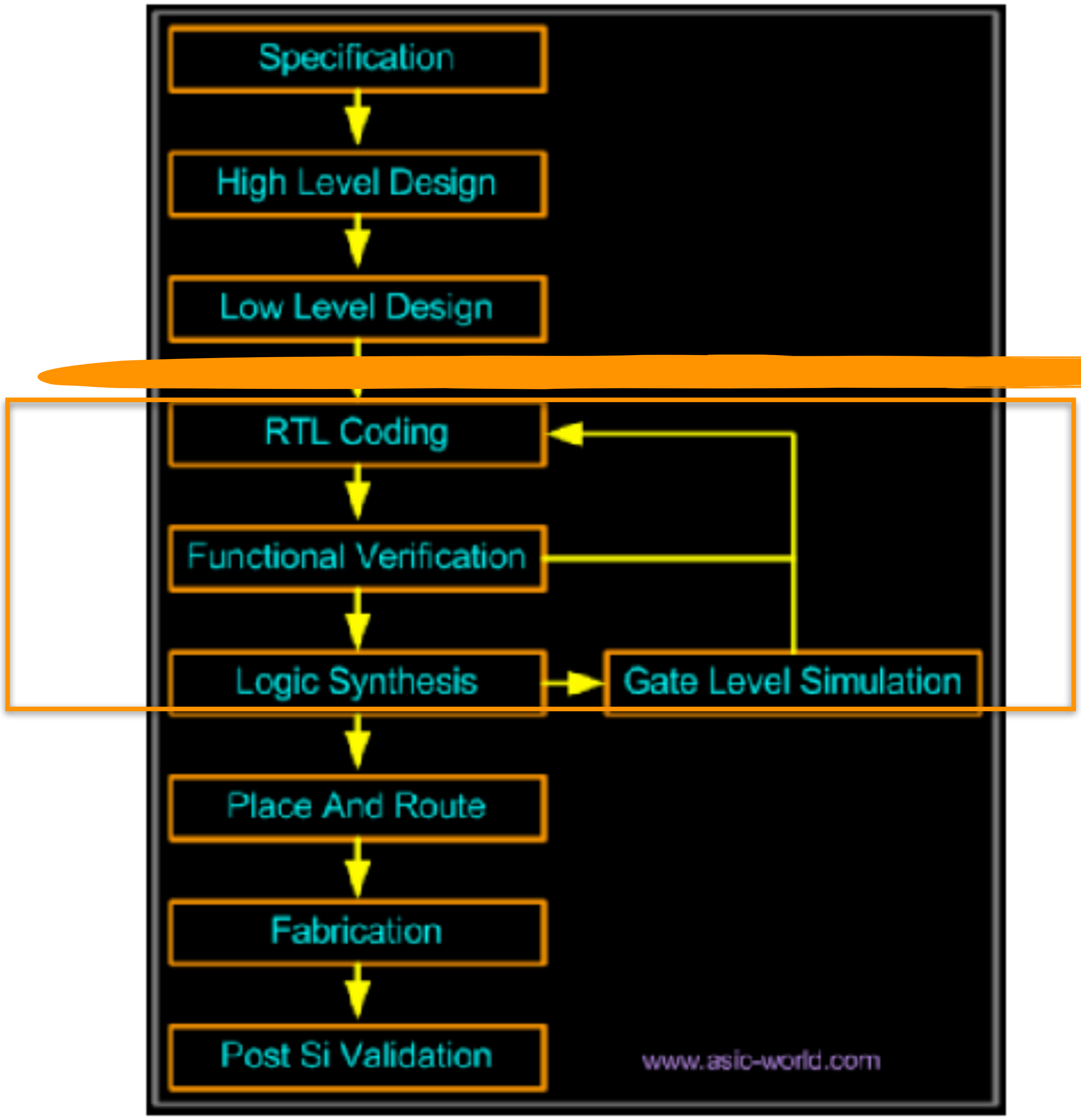
Boolean SAT is NP-Complete

Combatting limitations

- State matching
- Bounded proofs
- Decomposition
- Targeted verification
- Size reduction
- Case splitting
- Design abstractions
- Data abstractions

Some will be covered in this course, not all of them.

Hardware Design Flow Overview

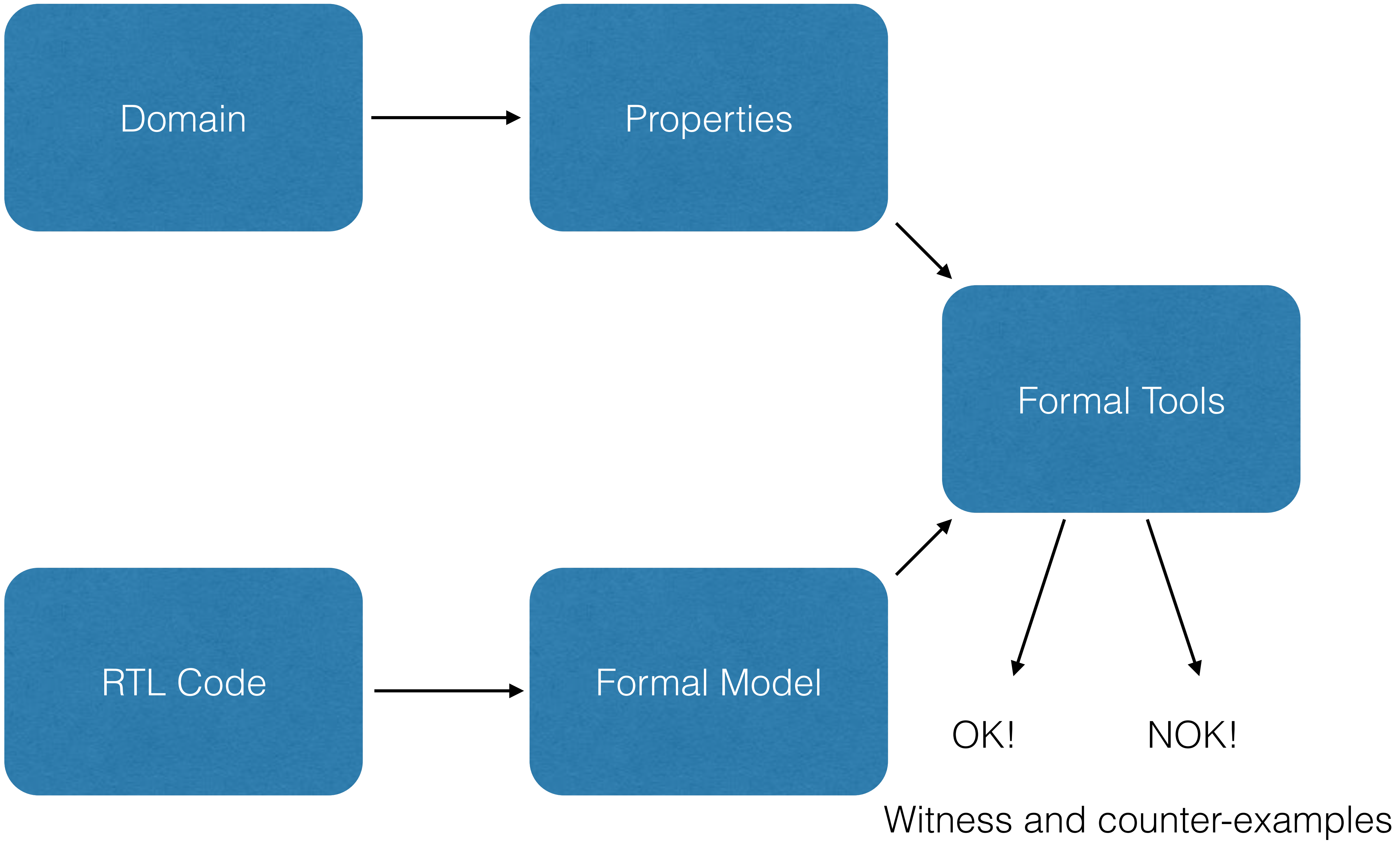


No formal equivalences

All steps proven functionally equivalent to RTL

This is changing.
C to RTL equivalence today feasible

Hardware Formal Verification



Current practice

cādence™

Mentor Graphics®

SYNOPSYS

JASPER design automation

Oski TECHNOLOGY

Why formal?

- Intel FDIV bug (1995)
  - estimated cost about USD 475,000,000

- Similar bug today
  - probable cost about USD 12,000,000,000
  - FYI: net income in 2014 was USD 11.7 billion

- Also, using formal is **cheaper** and **faster** than simulation
  - (sometimes, but more and more often)

Oski decoding formal club

Video from Oski web page
link is the following:

http://www.oskitechnology.com/decodingformals/

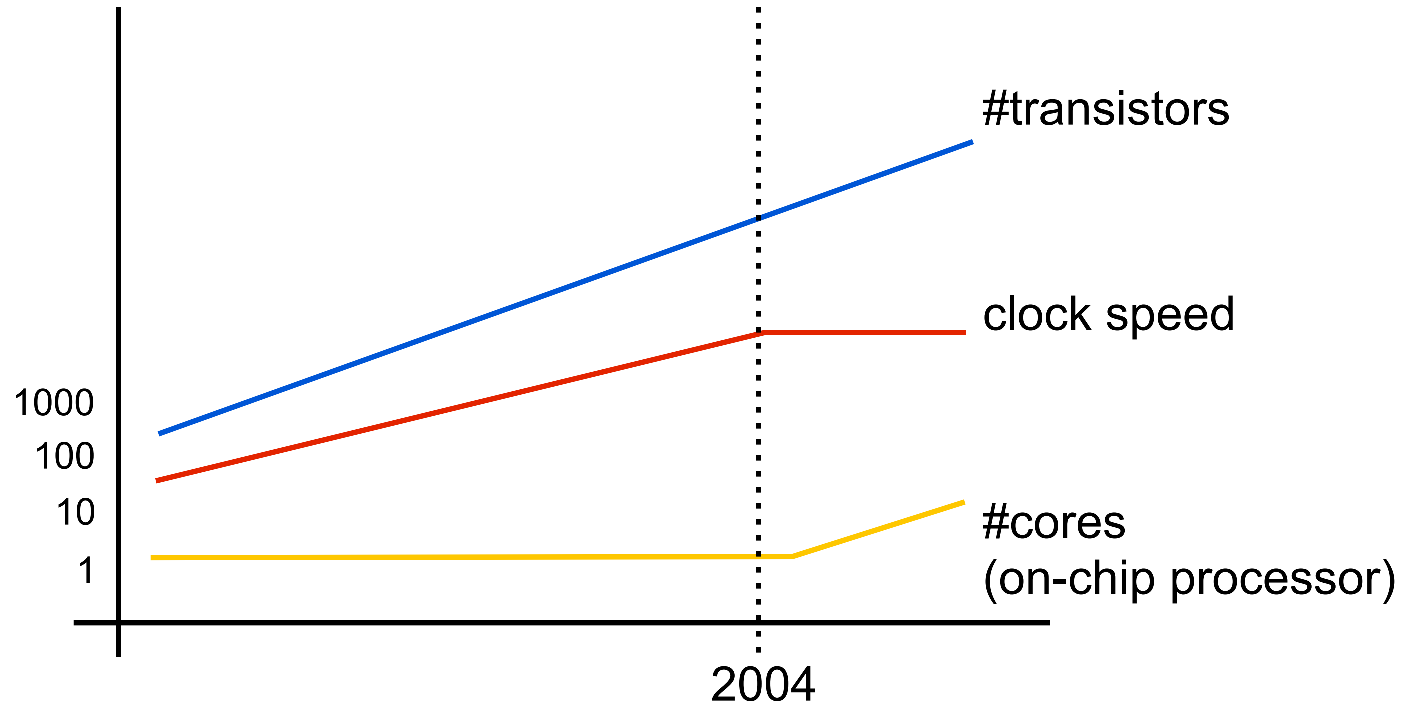
Notes from the video

- Formal is replacing simulation
  - 100% formal, no testing on some blocks
- Ever growing communities
- C to RTL equivalence checking
- Fully automatic solutions, even without writing any properties!
- “Hiding formal” was a key aspect
  - From academic to practical solutions

Impact of Formal

- @Home have a look at the following keynote video
  - http://www2.dac.com/events/videoarchive.aspx?confid=139&filter=Keynote&id=139-121--0&#video
- ARM CTO about scaling to 2020 solutions
- A few points
  - “Formal as a side salad”
  - 1.5 millions of hours to simulate M0 chip
  - (note this is about 170 years!)
  - Need to “hide” formal and integrate at design phase

Moore's law still applies - More cores!



How do we build bug-free multi-core industrial systems?

Scalability Issue

10^{80} atoms in the universe.

more than 10^{300} states in a 5x5 network.

This course and our research

Foundations

General Theory of
On-Chip Network
Architectures

Technological Advances

Languages
&
Verification Technologies

Real World Applications

Intel
ARM
STMicroelectronics

Integration & Tools

Prototype

Environment

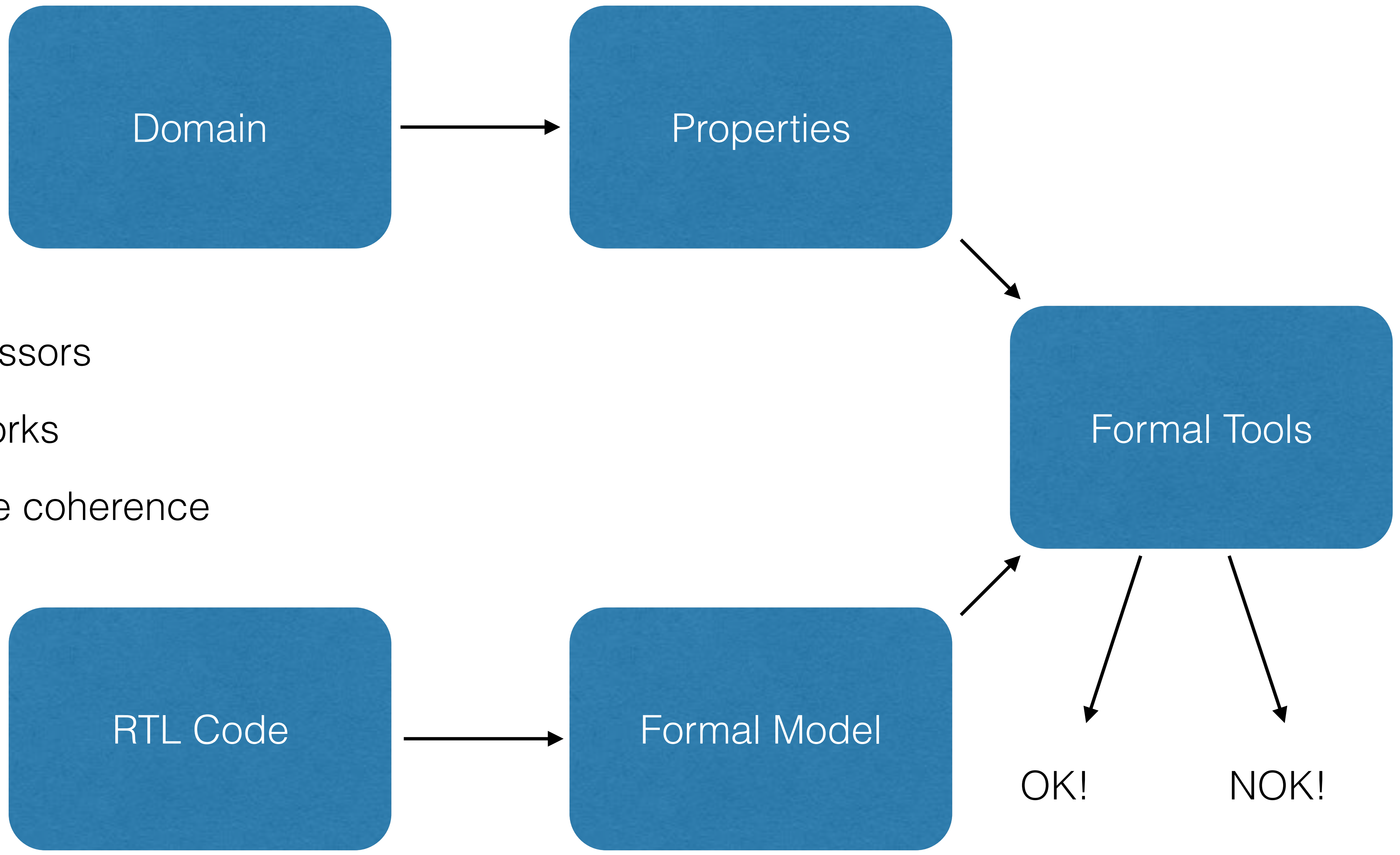
Link with other courses (1)

- System Validation (2IMF30)
  - mu-calculus & process algebra (2IMF10)
  - high-level models
- Automated Reasoning (2IMF25)
  - Details about SAT/SMT
  - Model-checking with NuSMV and BDDs
- **Seminar Formal System Analysis (Q6, next quartile)**
- Algorithms for model checking (2IMF35)
- Proving with computer assistance (2IMF15)
- Program verification techniques (2IMP10)

Link with other courses (2)

- 5SIB0: Electronic Design Automation
  - Synthesis, technology mapping, etc.
- 5LIH0: Digital integrated circuit design
  - Design oriented (low level)
- 5LIF0: Advanced digital circuit design
  - Design at low level
- 2IMN35: VLSI Programming
  - Higher level system layers (e.g. data flows)

Course content - Domain & RTL



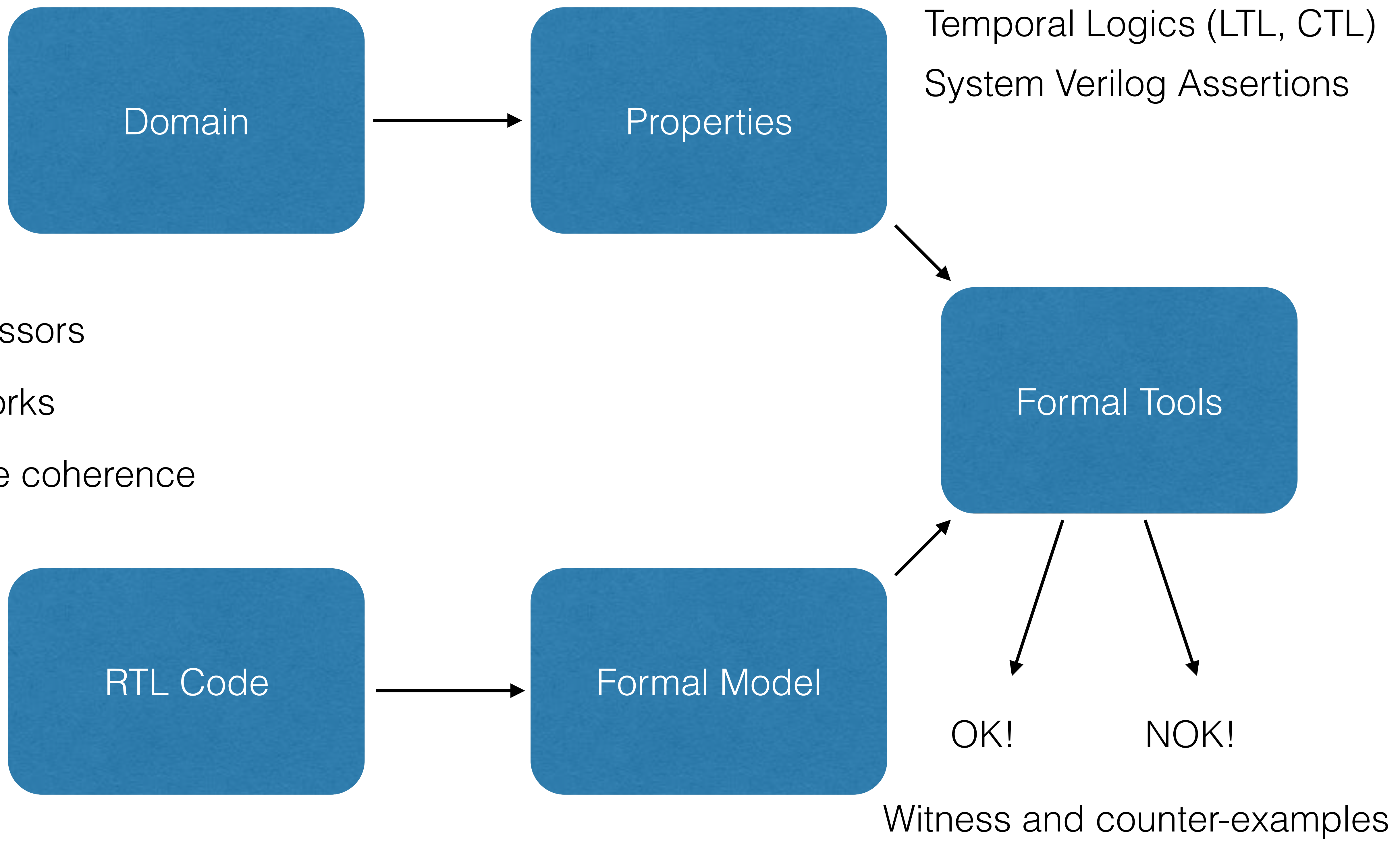
Processors

Networks

Cache coherence

Witness and counter-examples

Course content - Properties

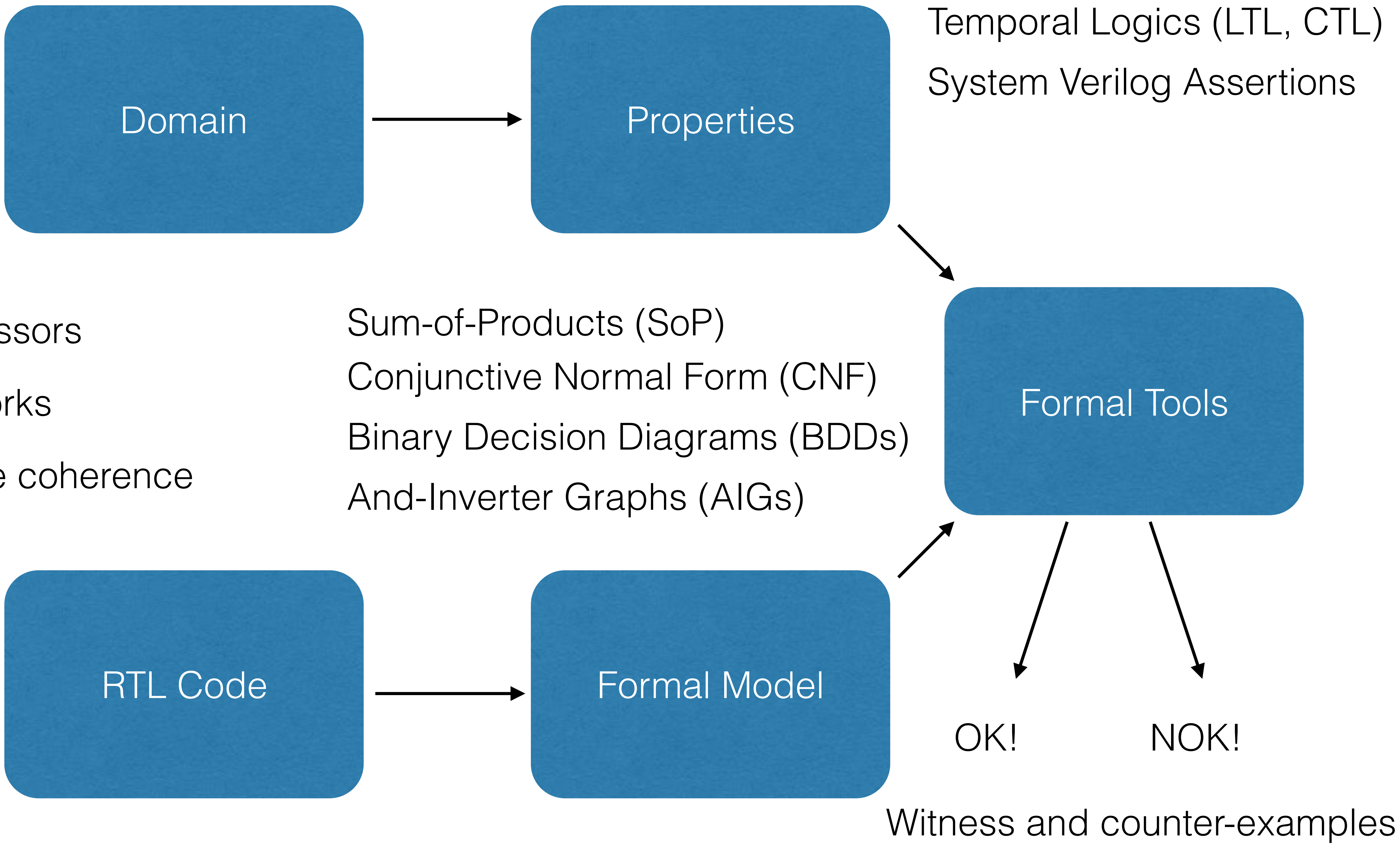

Domain
Properties
Temporal Logics (LTL, CTL)
System Verilog Assertions
Formal Tools
RTL Code
Formal Model
OK!
NOK!
Witness and counter-examples

Processors

Networks

Cache coherence

Course content - Formal models


Domain
Properties
Temporal Logics (LTL, CTL)
System Verilog Assertions
Formal Tools
RTL Code
Formal Model
OK!
NOK!
Witness and counter-examples

Processors

Networks

Cache coherence

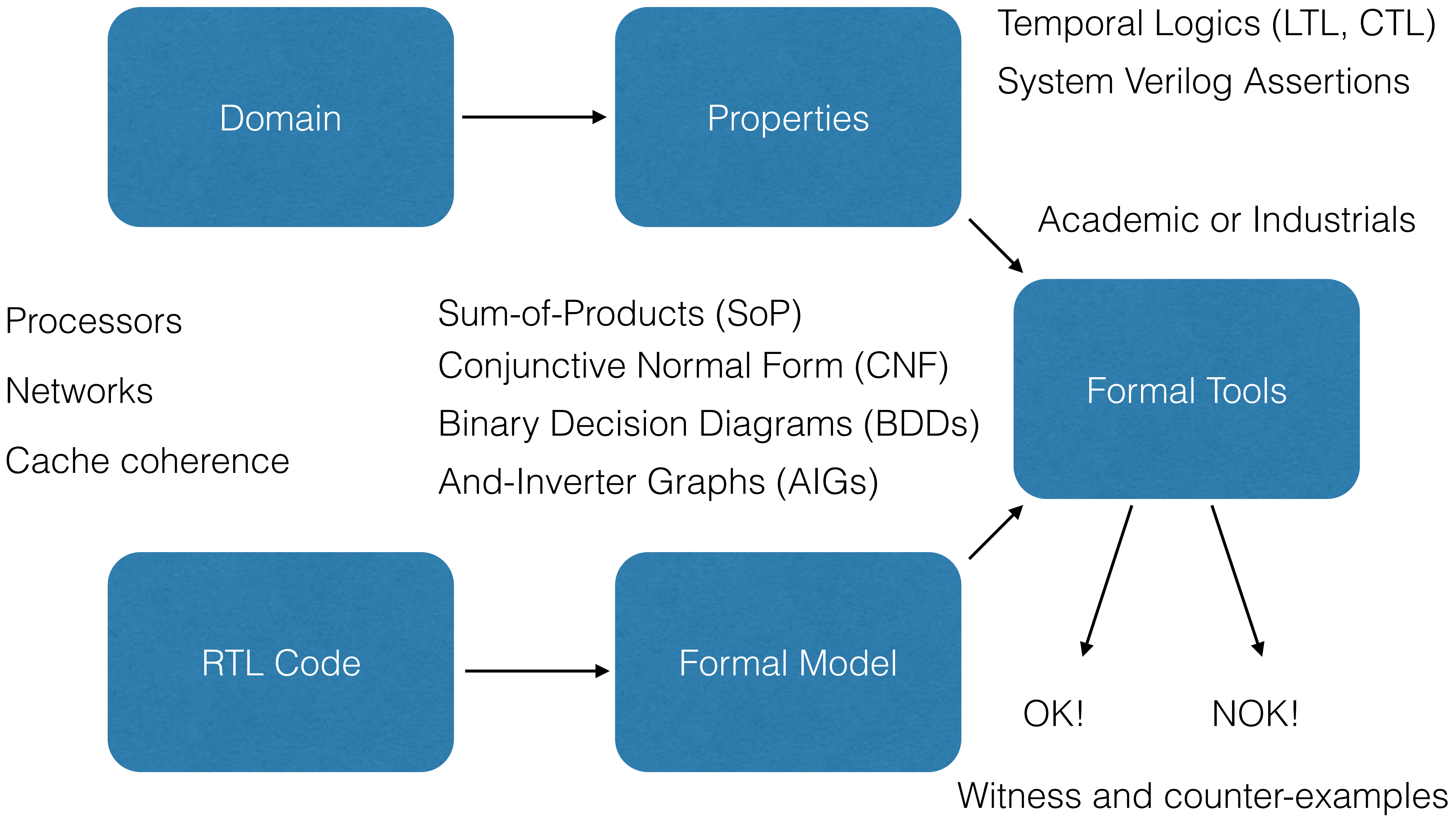
Sum-of-Products (SoP)

Conjunctive Normal Form (CNF)

Binary Decision Diagrams (BDDs)

And-Inverter Graphs (AIGs)

Course content - Formal tools


Domain
Properties
Temporal Logics (LTL, CTL)
System Verilog Assertions
Academic or Industrials
Processors
Networks
Cache coherence
Sum-of-Products (SoP)
Conjunctive Normal Form (CNF)
Binary Decision Diagrams (BDDs)
And-Inverter Graphs (AIGs)
Formal Tools
RTL Code
Formal Model
OK!
NOK!
Witness and counter-examples

Course practical assignments

- During the course you will work on several practical assignments

- The goal is
  - to practice with concrete verification issues
  - (for example, the verification of a processor using Jasper)
  - or to write your own verification “tool”
  - (for example, writing a circuit comparator)

- You will get some help!

Learning Goals

At the end of the course, a student:

» understands the challenges related to hardware verification at levels of abstractions ranging from high-level functional specifications down to bits;

» understands the basic principles of key verification techniques: SAT solving, model checking, equivalence checking;

» can use academic or industrial verification tools on small and moderate-size problems;

» understands and applies abstractions and over-approximations when necessary;

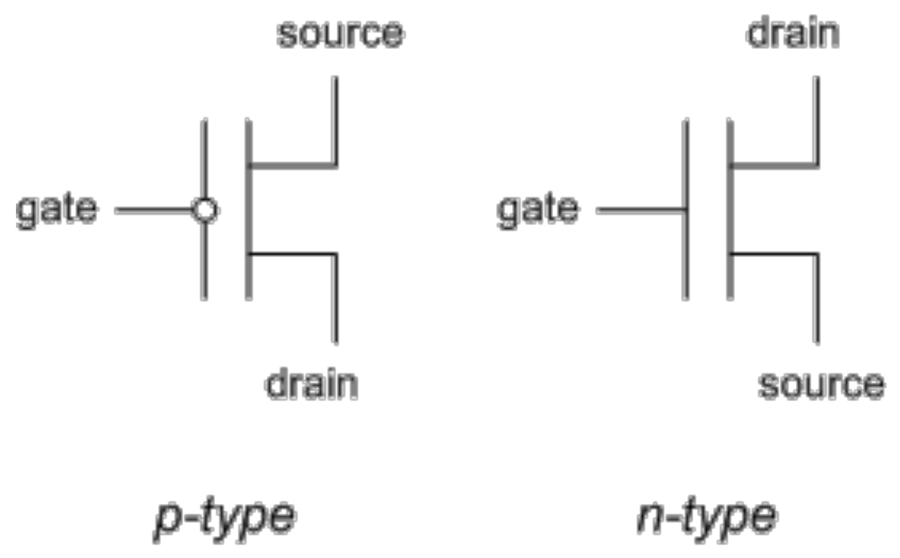
» has a basic knowledge about issues and techniques related to concurrency and multi-core architectures.

Program for today

- Basic Verilog & Hardware

- Reminder about basic UNIX
  - you need UNIX on your machine (Linux and OSX have it)
  - you will need to connect to our server
  - you need to ssh from a terminal with graphic export
  - you will need to use the UNIX file system in command line
    - (mkdir, rm, mv, etc)
  - you also need to use simple text editors from our server
    - vi, vim, emacs
    - evince to read pdf
  - You need make and gcc
  - If you have no idea what I am talking about, please speak up!

CMOS Transistors

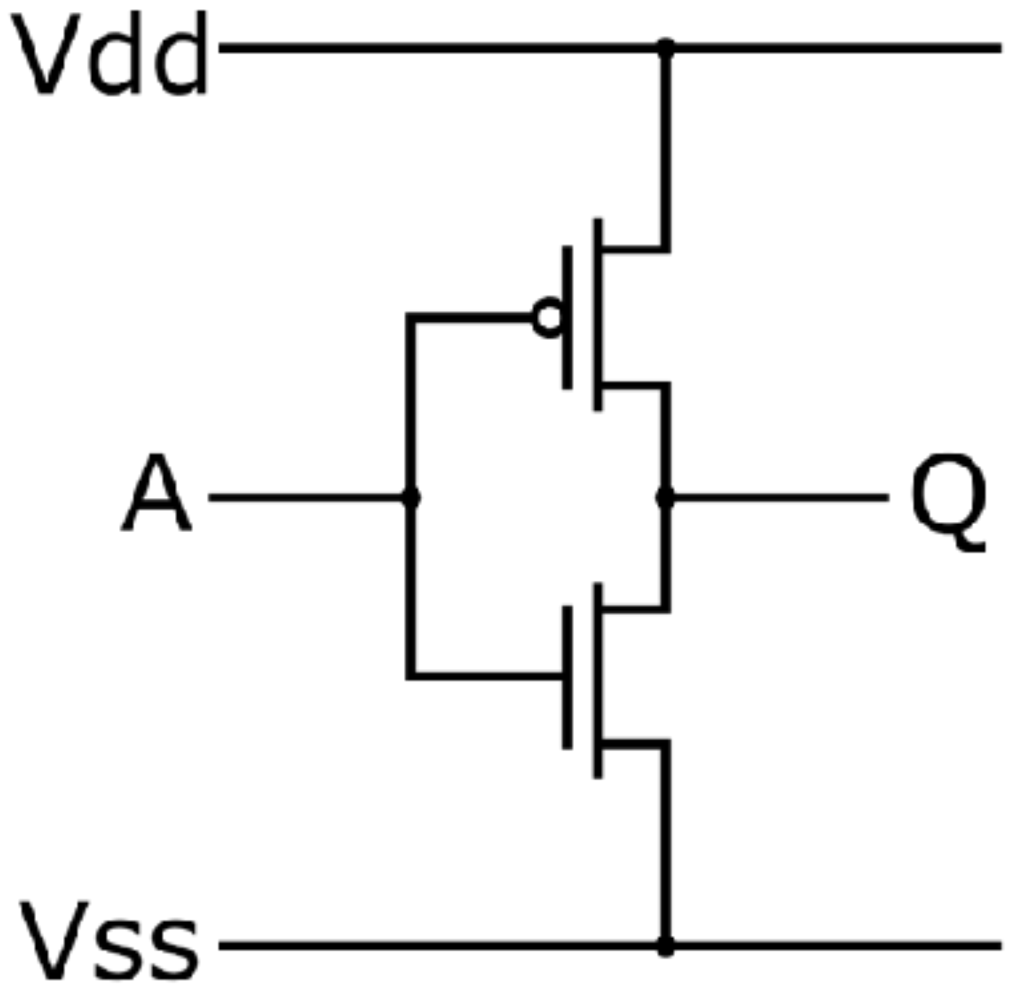


These are CMOS (Complementary Metal-Oxide Semiconductor).

The PMOS transistor is “on” when the gate low.

The NMOS transistor is “on” when the gate is high.

CMOS Gates (1)

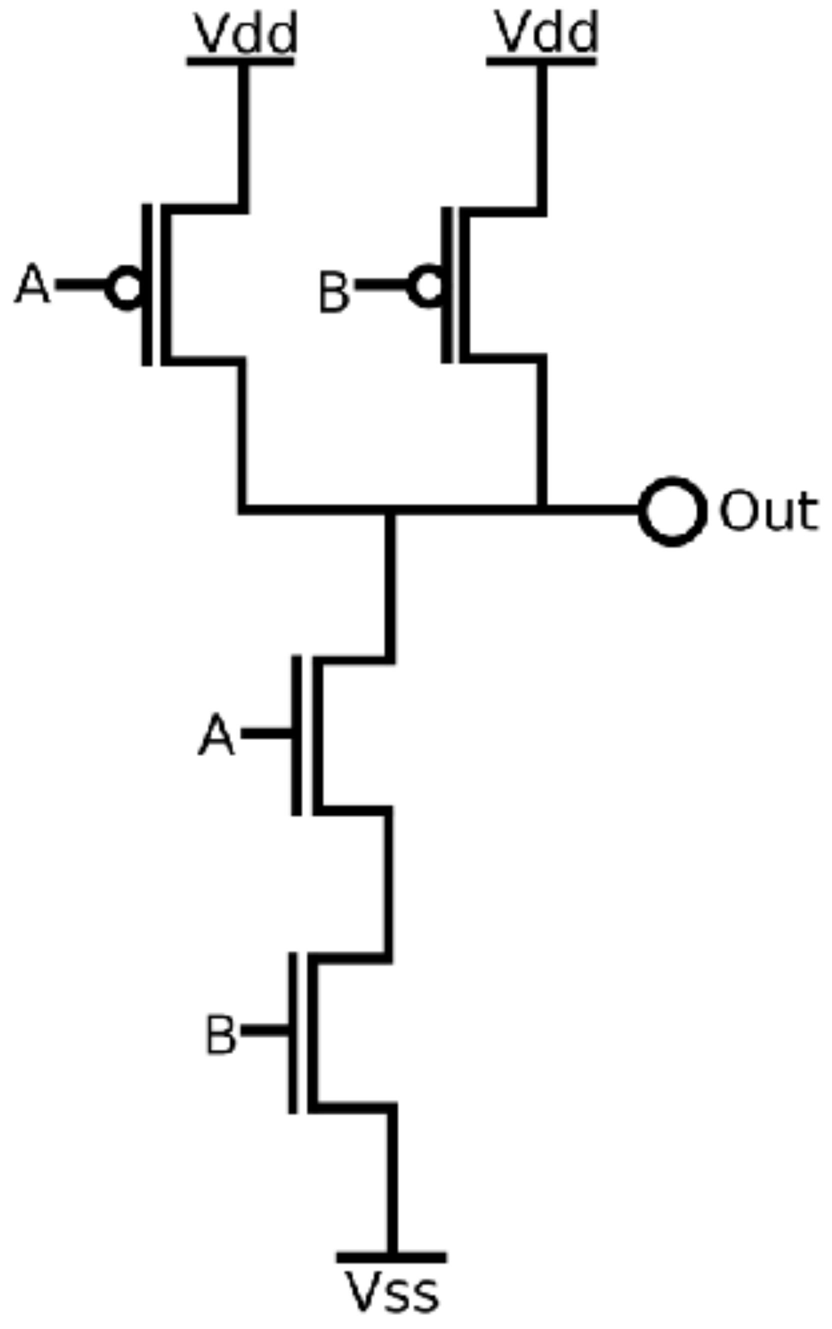


CMOS transistors are combined to create *logic gates.*

What logical (Boolean) function is implemented here?

Remember: Vdd is a logical 1, Vss is a logical 0.

CMOS Gates (2)



What logical (Boolean) function is implemented here?

Remember: Vdd is a logical 1, Vss is a logical 0.

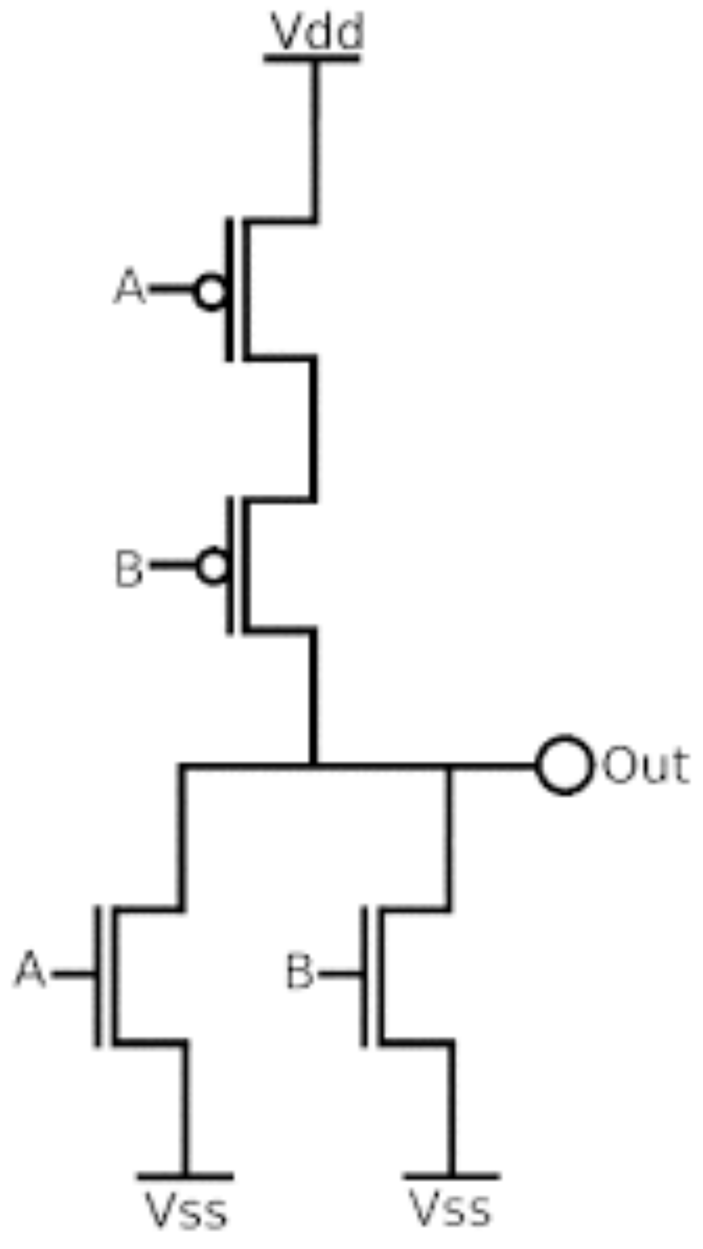
Do you see where the "complementary" is coming from?

CMOS Gates (3)

What logical (Boolean) function is implemented here?

Remember: Vdd is a logical 1, Vss is a logical 0.

Do you see where the “complementary” is coming from?



CMOS Gates (4)

How would you build an AND gate? an OR gate?

Abstracting away from transistors

We will not work at the level of transistors.

We will first abstract to 4-valued logic as follows:

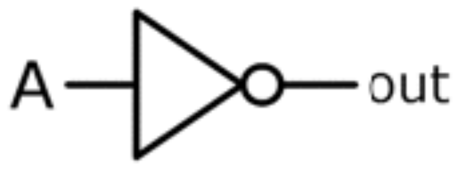
- logical 0 (or “False”) will be a 0 voltage (0 Volt)
- logical 1 (or “True”) will be a 1 voltage (say about 1 Volt)
- X or unknown, e.g. signals need time to propagate. Values cannot be determined yet
- Z for high-impedance or un-driven, for instance when transistors are all blocks then output is not driven (used in practice in a bus with multiple drivers)

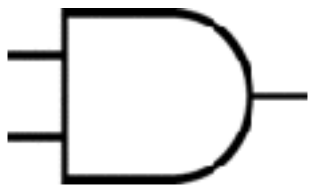
We will then quickly go to Boolean logic with only True and False.

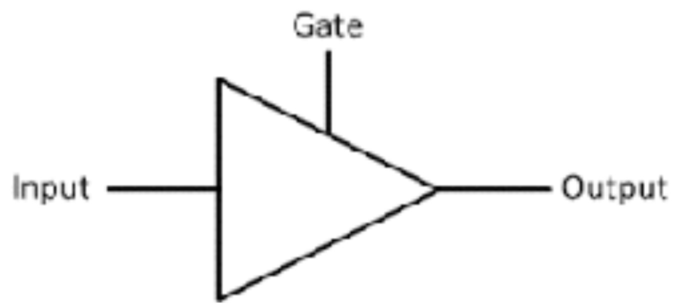
4 valued logic

- 0: false or low voltage
- 1: true or high voltage
- X: unknown, don't care
- Z: un-driven, high impedance

Verilog standard
http://verilog.renerta.com/source/vrg00003.htm





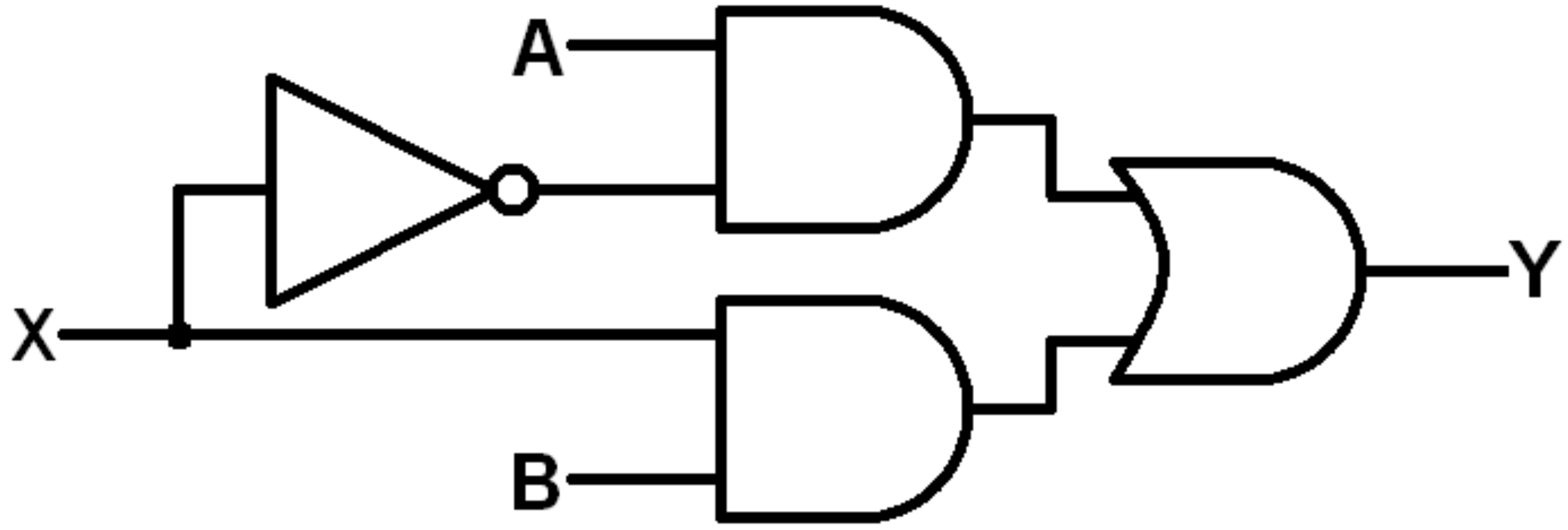


Input	Output
1	0
0	1
X	X
Z	X

	0	1	X	Z
0	0	0	0	0
1	0	1	X	X
X	0	X	X	X
Z	0	X	X	X

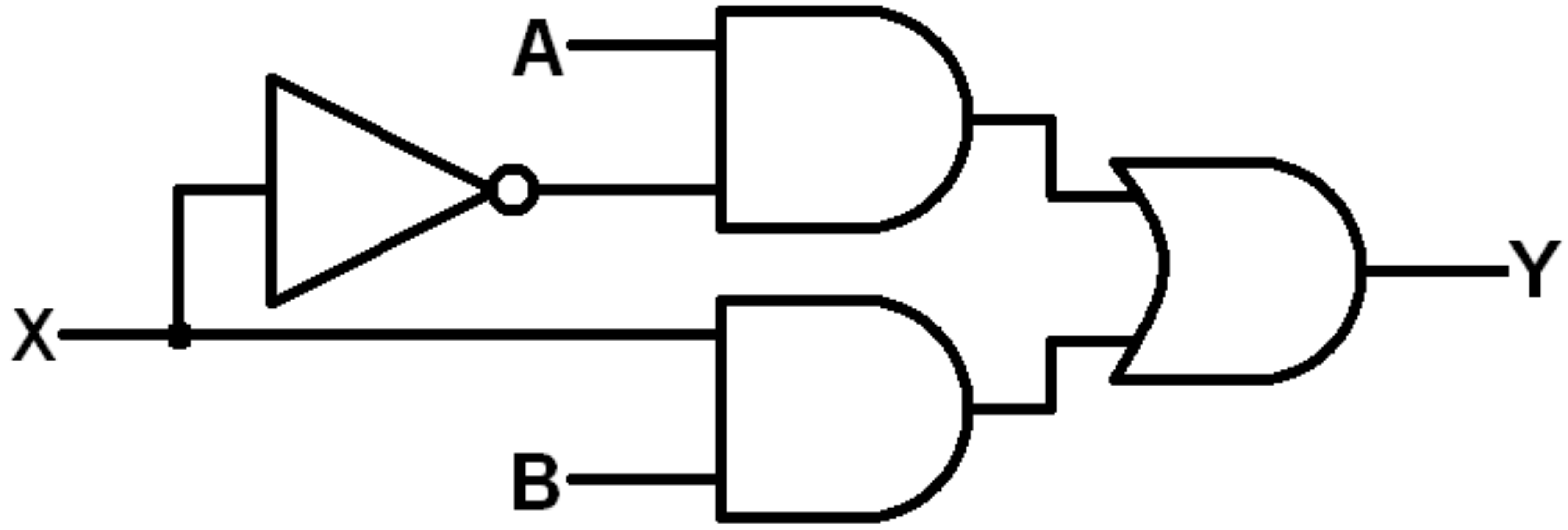
Input	Gate	Output
1	1	1
0	1	0
1	0	Z
0	0	Z

Quizz: What is this?



$Y = (A \cdot \bar{X}) + (B \cdot X)$

Quizz: What is this?



$Y = (A \bullet \bar{X}) + (B \bullet X)$

A 2-to-1 multiplexer described at a structural level.

Levels of abstractions

We can describe hardware at the structural level by connecting gates.

Thanks to Logic Synthesis we can describe hardware *at the behavioural level*.

```
module mux(
  input [7:0] i0,
  input [7:0] i1,
  input sel,
  output [7:0] o);

assign o = sel ? i1 : i0;

endmodule
```

This is our 2-to-1 multiplexer written using the Verilog Hardware Description Language (HDL).

Another popular language is VHDL. We will only use Verilog and its extension System Verilog.

Combinatorial vs. Sequential

A circuit is said to be ***combinatorial*** when its outputs only depend on its current inputs. The circuit has no memory, no *state.*

In contrast, a circuit is said to be ***sequential*** when it has memory, that is, state. Its outputs depend on previous computations. This defines what is commonly known as a Finite State Machine (FSM).

The basic state holding element in this course is a register.

An FSM is a "**Moore**" machine when ***the outputs only depend on the current state***.

An FSM is a "**Mealy**" machine when ***the outputs depend on the current state and the current inputs***.

Verilog HDL - Simple module

```
module sreg4 (clk,rst,ie,in,clr,flags);
input clk, rst,ie;
input [3:0] clr; // clear vector
// clear all flags marked with a 0 in the vector.
output[3:0] flags;
input [3:0] in;

reg [3:0] content;

always @(posedge clk)
   content <= rst ? 4'b0 : ( ie ? in & clr: content & clr);
// content is masked with the clear vector.
// All positions marked with a 0 in clr will also be zero in content.

assign flags = content;

endmodule
```

Verilog HDL - Module instantiation

```
//-------------------------- SREG -------------------------------
// Status register


wire SREG$ie;
wire [3:0] sreg$out, clrf;
sreg4 sreg (.clk(clk),.rst(rst),.ie(SREG$ie),
            .in(flg),.clr(clrf),.flags(sreg$out));
```

Verilog - Some references

- System verilog standard
- Verilog Language Reference Guide
  - http://verilog.renerta.com/source/vrg00000.htm
- Use Verilog to describe some simple designs
- Later in the course
  - System Verilog Assertions
  - Writing properties for Assertion Based Verification (ABV)

Verilog - some more references

- **[1] https://inst.eecs.berkeley.edu/~cs150/sp12/resources/FSM.pdf**
- [2] http://www.asic-world.com/tidbits/verilog_fsm.html
- [3] http://csit-sun.pub.ro/~duca/cn/sinteza/verilog_fsm.pdf
- [4] https://www.xilinx.com/support/documentation/university/ISE-Teaching/HDL-Design/14x/Nexys3/Verilog/docs-pdf/lab10.pdf
- [5] http://electrosofts.com/verilog/fsm.html (with test bench example)
- **[6] http://www-inst.eecs.berkeley.edu/~cs150/fa05/Lectures/07-SeqLogicIIIx2.pdf**
- [7] http://tuline.com/wp-content/uploads/2015/11/Finite-State-Machine-Examples.pdf

Verilog - in this course

- You will need to ***read simple Verilog / System Verilog***. We will use design examples for applying verification techniques. You need to read and understand (a bit) these examples.

- You will need to ***write System Verilog Assertions***. To do this, you need to be able to write some very basic Verilog code. Notions like clocks and assignments are important.

- You are NOT expected to become advanced Verilog or System Verilog experts at the end of this course !!

- In general, Verilog syntax close to C.

- At the end, Verilog is “just” another programming language
  - but be aware that it describes hardware !

Homework before next lecture

- Read papers mentioned in the course website
  - paper about Intel's experience with formal
  - some papers about hardware verification using ACL2

- Check on-line documentation about Verilog and UNIX-like OS
  - (make sure you have a UNIX-like system on your machine)

- Install **ONE** SAT solver to start with assignment 1
  - Glucose (http://www.labri.fr/perso/lsimon/glucose/)
  - Lingeling (http://fmv.jku.at/lingeling/)
  - Z3 (https://github.com/Z3Prover/z3)

- Next lecture will be about
  - representations of Boolean expressions
  - combinational equivalence checking
  - start with assignment 1

Thanks!