Lecture Series with Laboratory

Formal Verification of Digital Systems

Wolfgang Kunz ***Dominik Stoffel***

University of Kaiserslautern
Department of Electrical and Computer Engineering
Electronic Design Automation Group

Phone: +49 631 205-2603
Email: kunz@eit.uni-kl.de
stoffel@eit.uni-kl.de

Motivation for this lecture

Formal verification: guarantee functional correctness by exact mathematical proofs

Drivers for ***formal verification*** in SoC design:

- Complexity: increasing number of modules and processors on a single chip
- IP-core based design styles
- Better trade-off between design productivity and quality

Motivation for this lecture

Formal verification: guarantee functional correctness by exact mathematical proofs

Drivers for ***formal verification*** in SoC design:

- Complexity: increasing number of modules and processors on a single chip
- IP-core based design styles
- Better trade-off between design productivity and quality

Objective of this lecture

What to expect from this short course in formal verification:

- Understand some of the basic technology underlying today's formal verification tools
  - computational models
  - algorithms
- Gain hands-on experience with a commercial FV tool
- Gain a first understanding of a SoC verification methodology using property checking

Further Reading

Books

E. Clarke, O. Grumberg, D. Peled: Model Checking, The MIT Press, 1999, ISBN 0-262-03270-8.

G. Hachtel, F. Somenzi: *Logic Synthesis and Verification Algorithms*, Kluwer Academic Publishers, 1996, ISBN 0-7923-9746-0.

K.L. McMillan: *Symbolic Model Checking*, Kluwer Academic Publishers, 1993, ISBN 0-7923-9380-5.

S. Hassoun and T. Sasao (Eds.) *Logic Synthesis and Verification*, Kluwer Academic Publishers, 2002. ISBN- 0-7923-7606-4.

Outline

1. Introduction
2. Boolean Decision Making by Satisfiability Solving (SAT)
3. Interval Property Checking

Chapter 1

Introduction

The first „hardware bug“

Photo # NH 96566-KN First Computer "Bug", 1945

9/9

0800 Arctan started
1000 " stopped - arctan ✓
13"uc (032) MP - MC
(033) PRO 2
correct
{ 1.2700 9.037 847 025
9.037 846 795 correct
~~1.9826000~~ ~~2.13047645~~) 4.615925059(-2)
2.13047645
2.13067645
Relays 6-2 in 033 failed special speed test
tm relay " 11.00 test.
Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Multy Adder Test.
1545 
Relay #70 Panel F
(moth) in relay.
First actual case of bug being found.
1700 closed down.

Slide 1-8

Detection of hardware errors

Testing:

Detection of fabrication defects and faults that appear during operation

Verification:

Detection of errors introduced in the design phase

Basic approaches to verification

- **Simulation**

  Exploration of the design's behaviour by simulation. Stimuli are chosen specifically to expose a certain behaviour, or they are generated randomly.

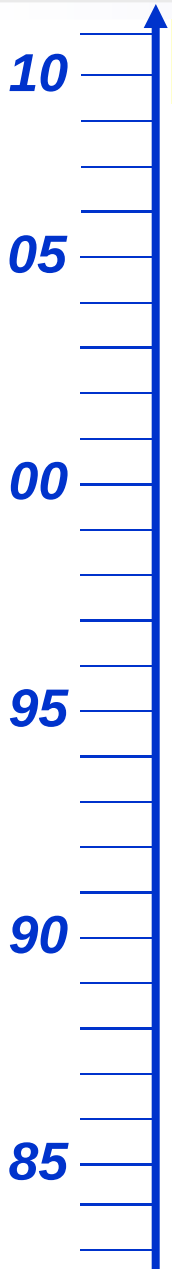
- **Emulation**

  Construction of a prototype of the circuit, for example using programmable logic (*field programmable gate arrays* (FPGAs))

- **Formal verification**

  Application of *exact* mathematical proving methods (performed automatically by software) to check circuit properties

Design verification - History



high-level equivalence and property checking

Gap-free formal property checking

SystemVerilog

SystemC

Hardware Verification languages E, VERA …

Assertion checking, PSL

Bounded model checking

Formal equivalence checking

Symbolic model checking

VHDL simulation

VERILOG simulation

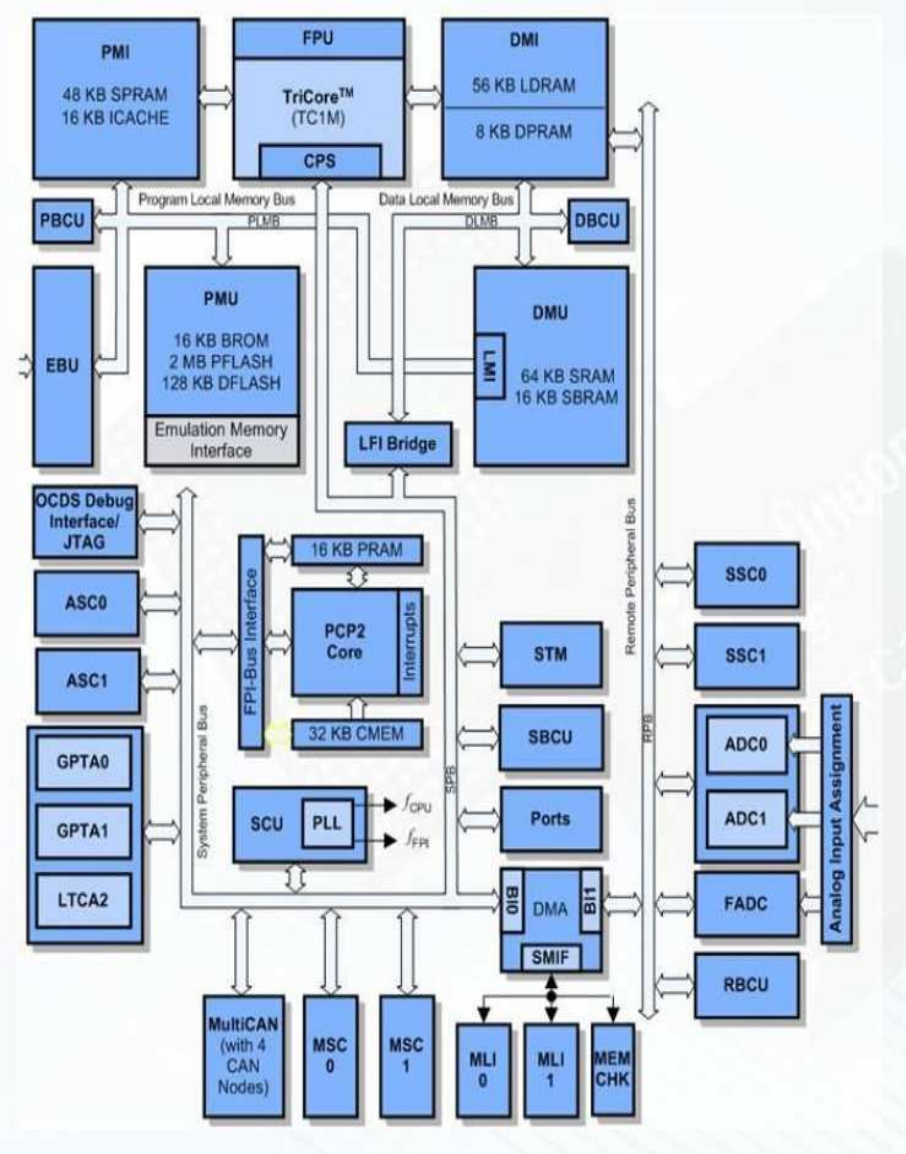
Model checking

The main focus of the lecture

formal / semi-formal

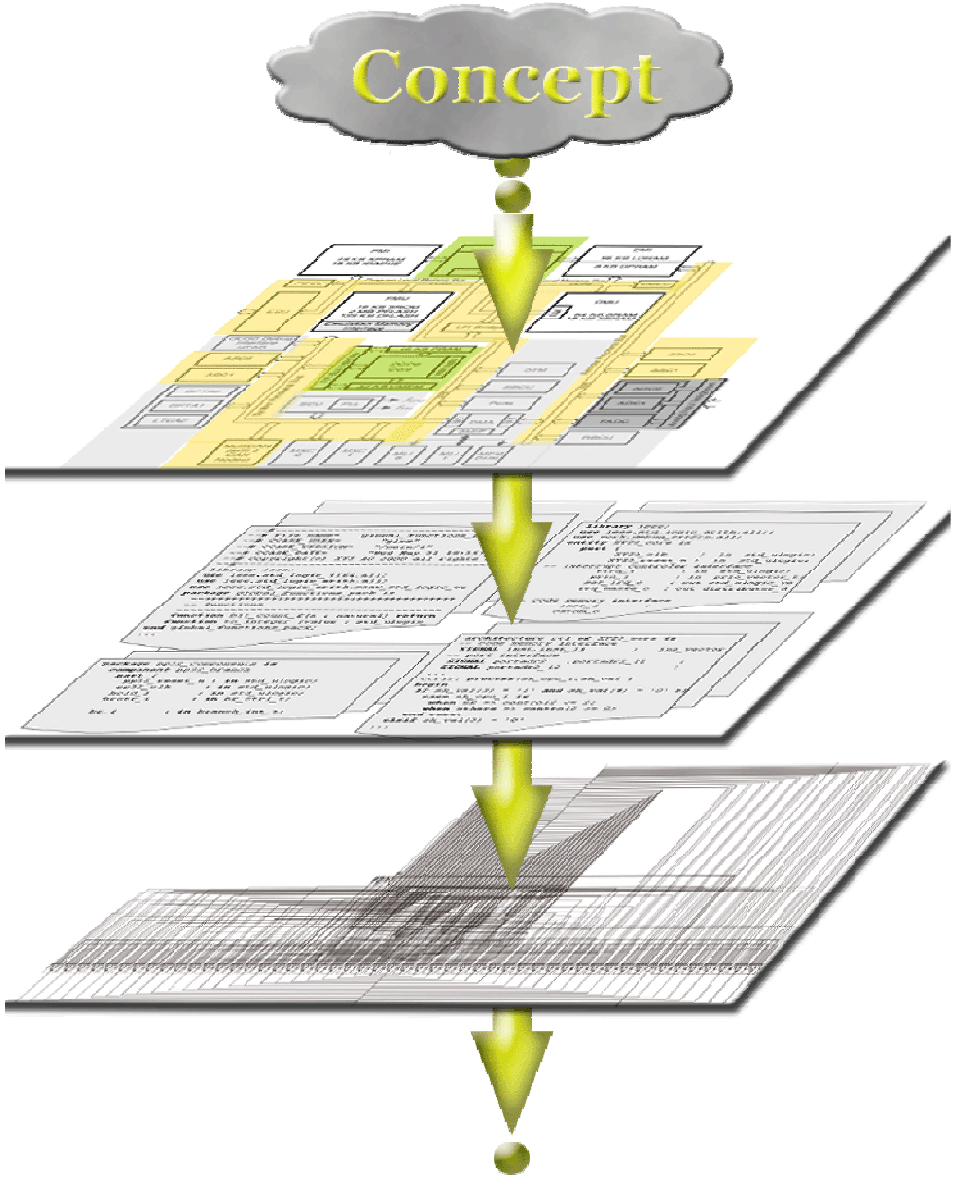
not formal (simulation)

Example: SoC for automotive application



- processors
- hardware accelerators
- memories
- I/O controllers
- mixed signal blocks
- communication structures

SoC Design Flow



Early phase

- set up and assess functional prototypes

Architecture

- model and explore architectural choices
- specify modules and communication for target architecture

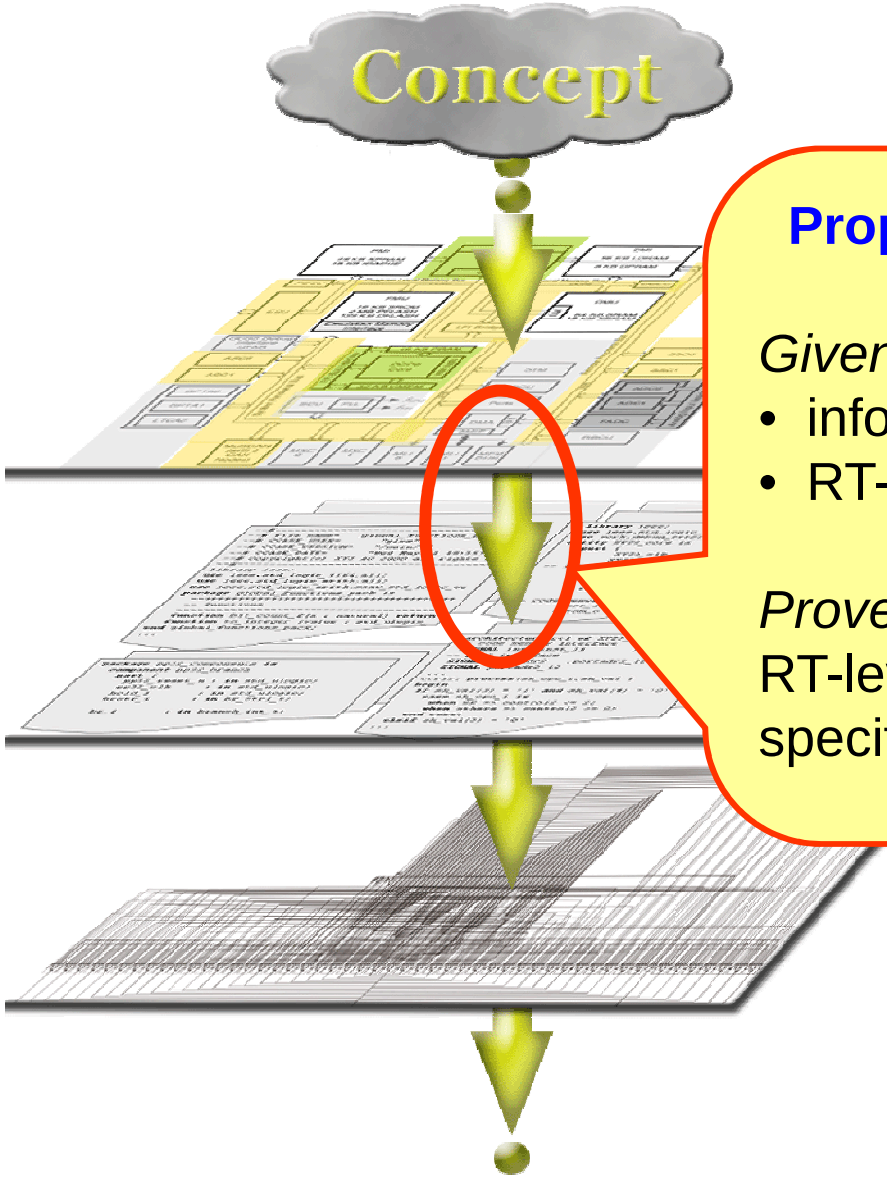
Design (RT)

- Register-Transfer (RT) description of modules
- system integration, communication structures

Implementation

- Synthesis and optimization
- test preparation

SoC Design Flow



Early phase

- set up and assess functional

Property Checking

Given:

- informal specification
- RT-level circuit description

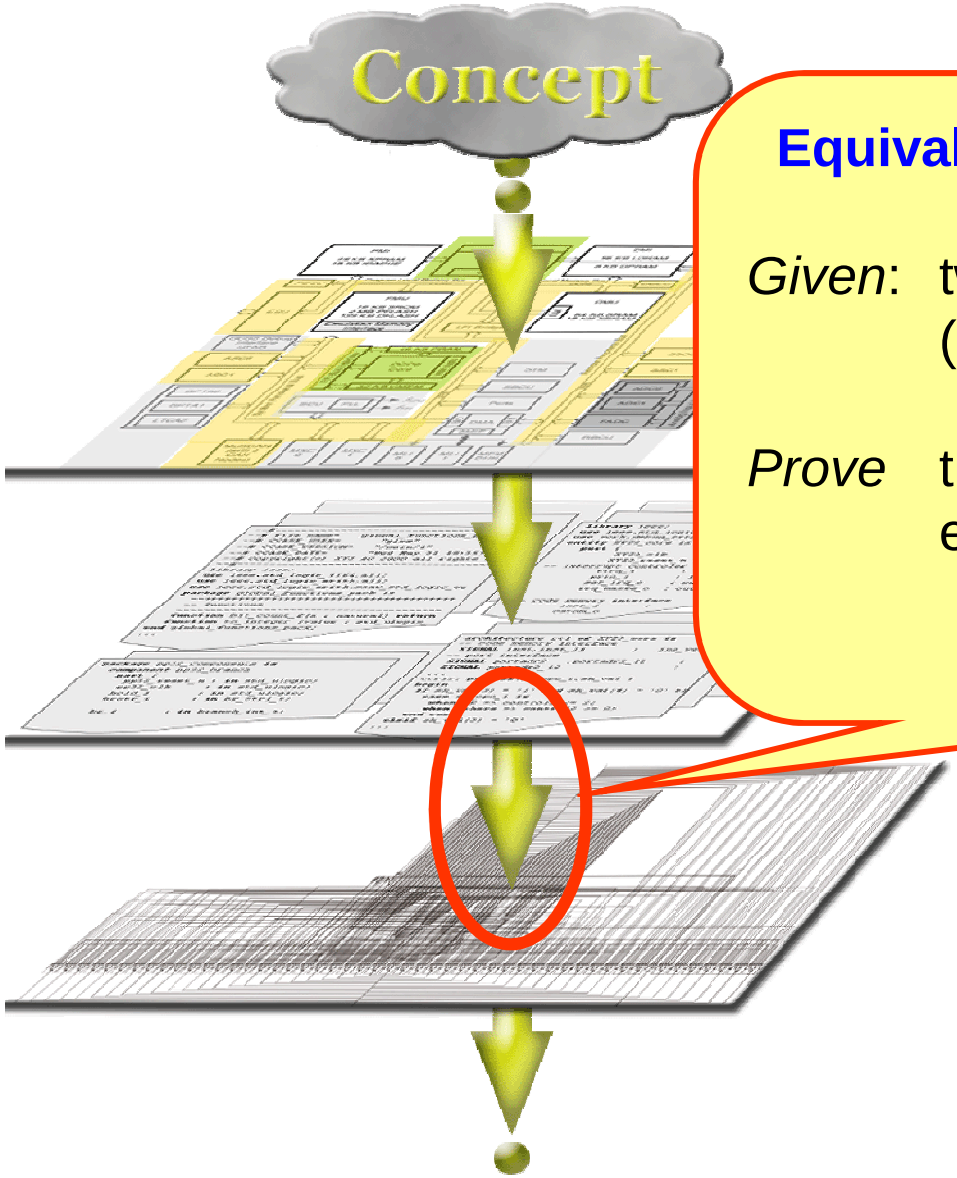
Prove (by checking properties) that the RT-level design description fulfills the specification

structures

Implementation

- Synthesis and optimization
- test preparation

SoC Design Flow



Early phase

Equivalence Checking

Given: two design descriptions (e.g. 1x RTL, 1x Gatelevel)

Prove that both designs are functionally equivalent

of modules

- system integration, communication structures

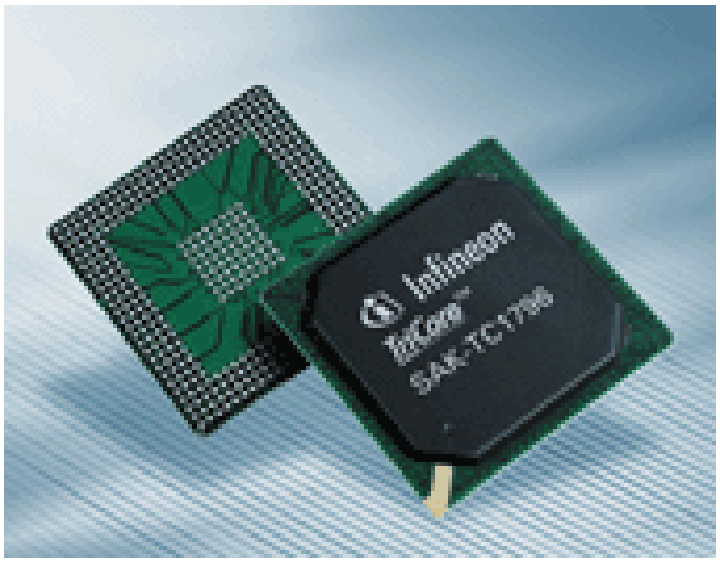
Implementation

- Synthesis and optimization
- test preparation

Infineon Tricore 2 project – Example

Every instruction of the processor is verified by formulating a property (or set of properties) describing its behavior

MAC Unit: multiply, multiply/add, multiply/subtract saturation, rounding, shift-bits



e.g.

- MUL.H
  - packed multiply
  - 2 parallel multiplications
  - 8 variants +12 special cases
  - 16 bit operands
  - 64 bit result

- MADD(S).Q
  - multiply/add in Q-format
  - 40 variants + 24 special cases
  - 32/16 bit operands
  - 64/32 bit results
  - some variants with saturation

Property Checking of processor pipeline

Goal Prove that instructions are performed correctly

Example

Property in ITL (InTerval Language): "assumption + commitment"

"assumptions"

"commitments"

```
property mul;     // packed half word
                     multiplication
assume:
  at t: command_dec(MUL,op1,op2);
  during[t,t+3]: no_reset;
  during[t,t+3]: no_cancel;
  …
prove:
  at t+3: ip_res[31:0]
              == op1[15:0]*op2[15:0];
  at t+3: ip_res[63:32]
              == op1[31:16]*op2[31:16];
end
```

Basic models for property checking

Automata

Definition:

The quadruple $H = (I, S, S_0, \delta)$ is a deterministic, finite *state transition structure*. Here,

I is a finite set of allowed *input symbols* ("input alphabet")

S is a finite set of states

$S_0 \subseteq S$ is a finite set of allowed initial states

$\delta: S \times I \rightarrow S$ is a *transition function.*

Basic models for property checking

Synchronous sequential circuits are typically modeled by *Mealy*- or *Moore*-machines.

Definition:

A *Mealy-Machine* $M = (I, O, S, S_0, \delta, \lambda)$ is a finite, deterministic state transition structure H extended by:

- a finite set O of *output symbols* ("output alphabet")
- an output function $\lambda : S \times I \rightarrow O$.

Definition:

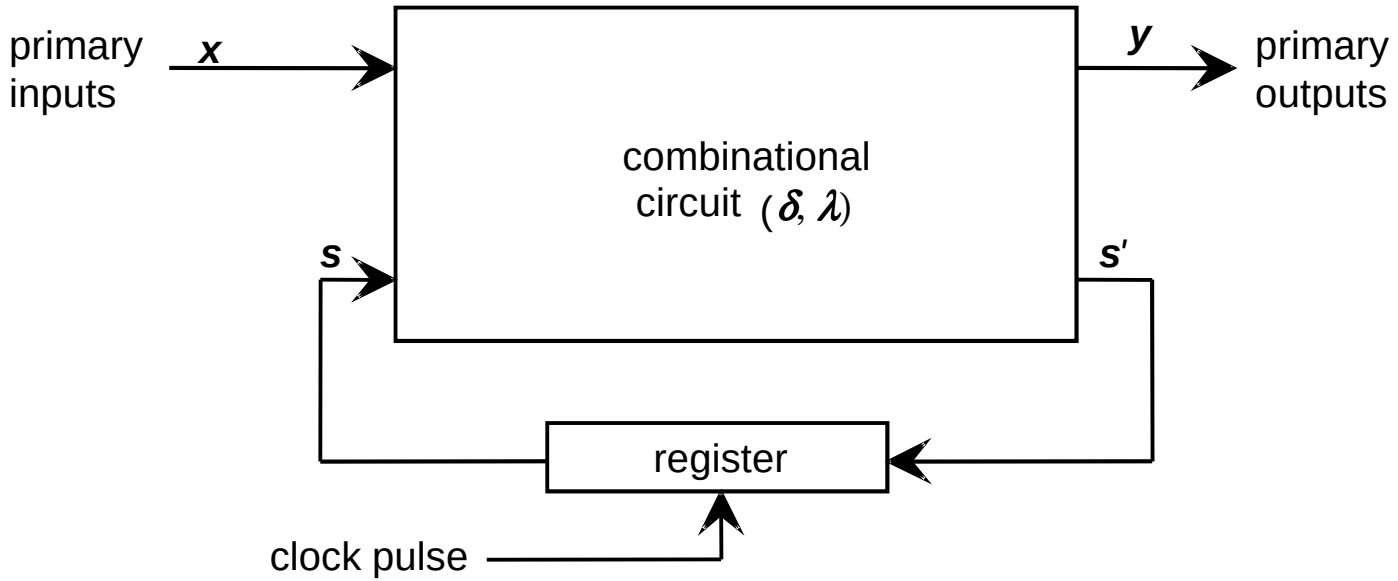
A *Moore-Machine* $M = (I, O, S, S_0, \delta, \lambda)$ is a finite, deterministic state transition structure H extended by:

- a finite set O of *output symbols* ("output alphabet")
- an output function $\lambda : S \rightarrow O$.

Basic models for property checking

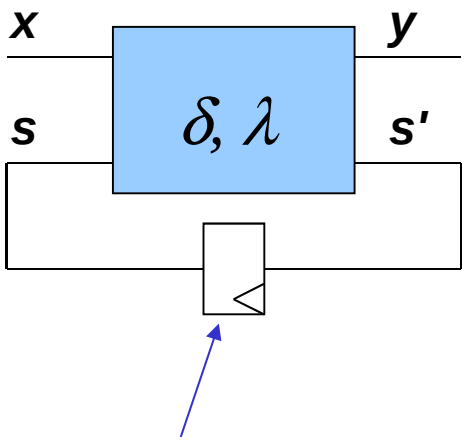
Modelling of a synchronous sequential circuit by *Mealy*- (*Moore*-) machine:

x: input variables
y: output variables
s: variables for the current state
s′: variables for the next state



Block diagram for synchronous sequential circuit

Classical Model Checking



δ: transition function
λ: output function

Reachability analysis: what states are reachable in the design starting from the initial state?

Reachability

Central problem in formal verification of properties of finite state machines:

***Reachability analysis*:**

given: finite state machine *M* with initial states S_0

task: find the set of all states *R* which are possible ("reachable") in *M* through arbitrary input sequences starting from S_0

“Fixed point iteration” for reachability analysis

reach(S_0): procedure to compute all states R reachable from S_0
xreach(A): procedure to compute all immediate successors (next states) for a set of states A

```
reach(S0)
{
    Rit := ∅;
    do
    {
        R := Rit;
        Rit := S0 ∪ xreach(R);
    } until (R == Rit)
    return R;
}
```

Procedure to compute R

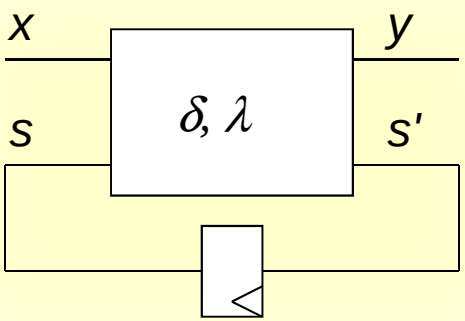
„State explosion“: computation and representation of state sets are very hard problems!

Property Checking

Basic Approaches

The „unbounded“ paradigm:

❖ (Classical) Model Checking



the world of

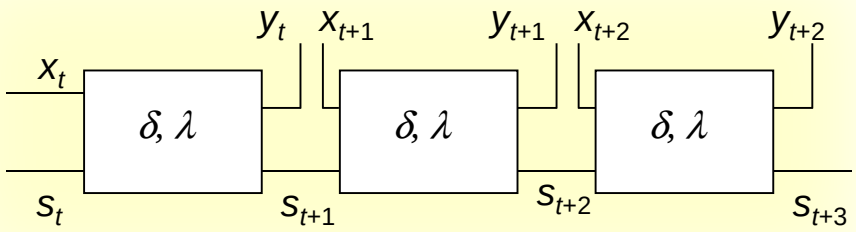
- FSMs and related structures
- state space exploration
- fixed point characterizations of temporal operators
- automatic abstraction/refinement techniques
- handling systems with a few hundred state variables

Property Checking

Basic Approaches

The „bounded“ paradigm:

- Bounded Model Checking
- Interval Property Checking
- K-Step-Induction

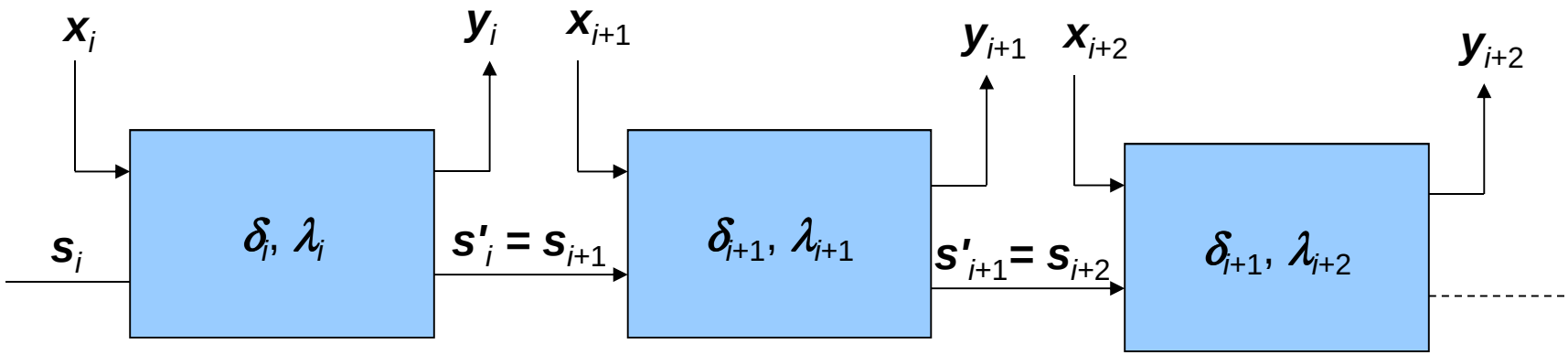


the world of

- unrolled FSMs (next slides)
- SAT (satisfiability solving) (Chap. 2)
- intuitive invariants (Chap. 3)
- sophisticated methodology (project?)
- handling systems with thousands of state variables

Bounded model for property checking

Unrolling the finite state machine



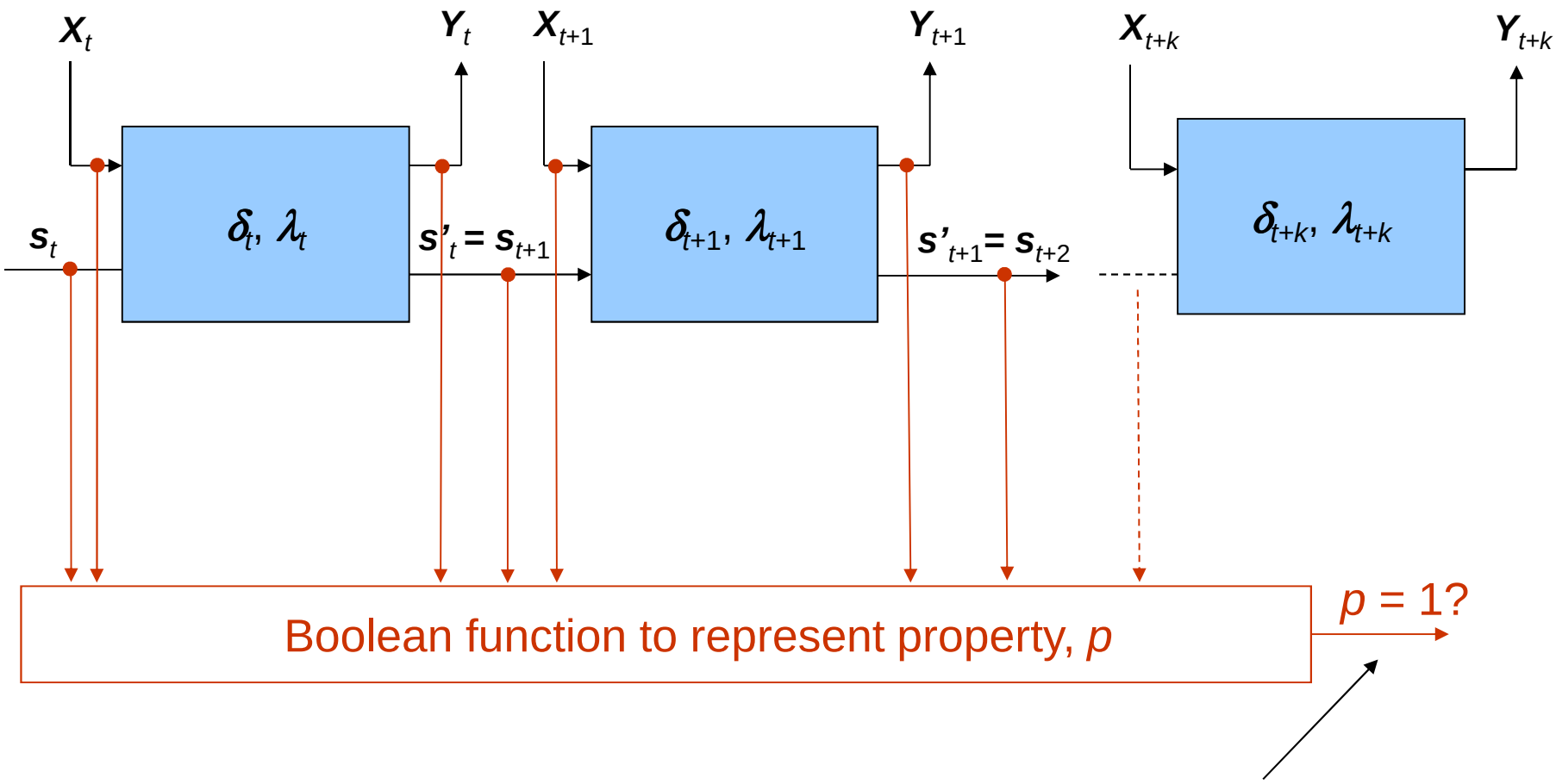
"iterative circuit model", "bounded circuit model"

Concatenate *k* copies of combinational logic for δ and λ („time frames")

$\Rightarrow$ no feedback loops, combinational model

Property Checking by SAT

"Iterative Circuit Model" from $i = t$ to $i = t + k$



„Boolean Satisfiability Problem (SAT)"