

Applying Model Checking to Real Microcontroller Code

Dipl.-Inform. Bastian Schlich

`schlich@cs.rwth-aachen.de`

Dagstuhl – 05.03.2007

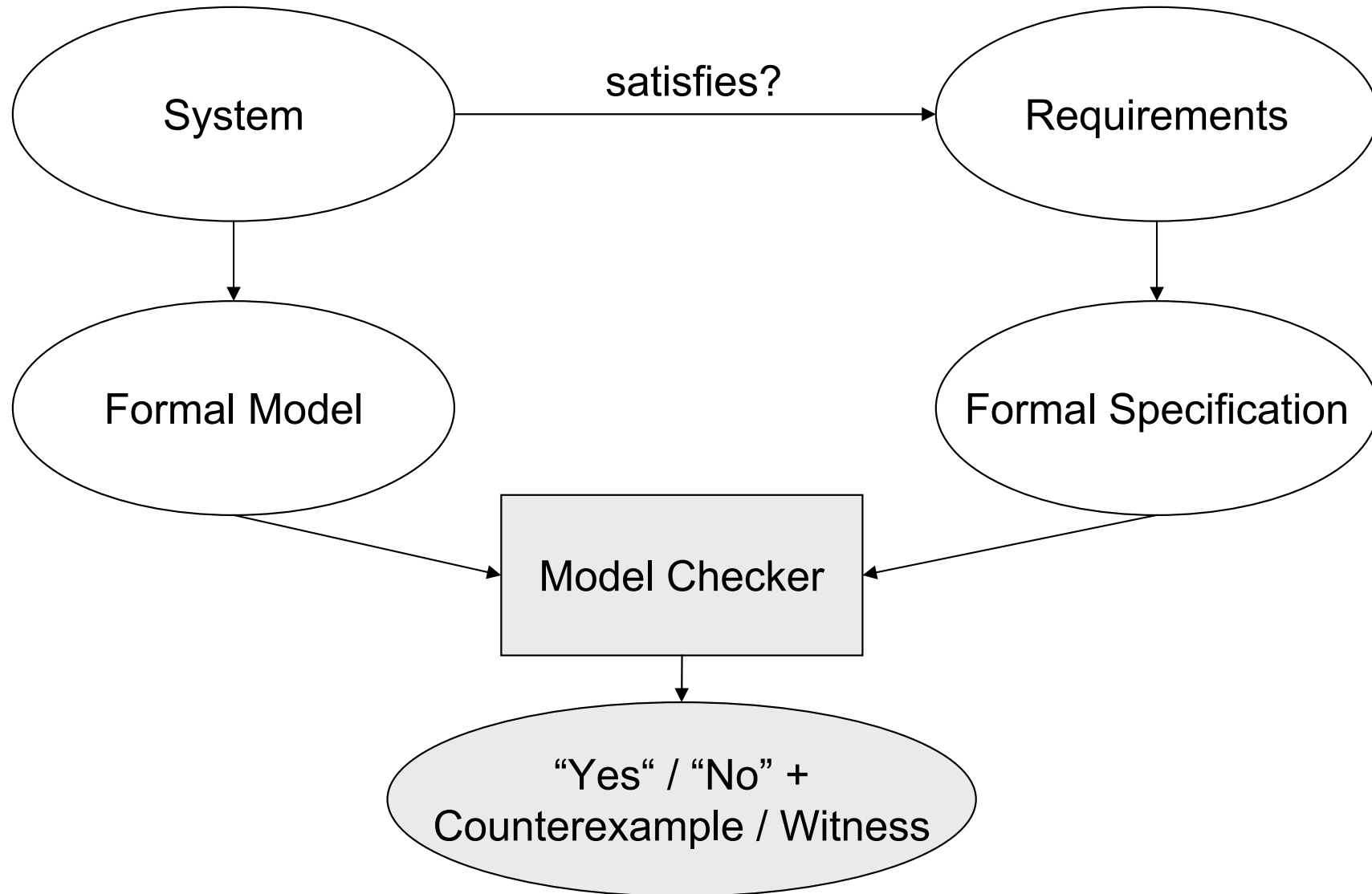
Outline

- Short Summary: Model Checking
- Excursus: Microcontroller

- Motivation
- Model: C Code vs. Assembly Code
- [mc]square
- Two Case Studies
- Related Work
- Conclusion & Future Work

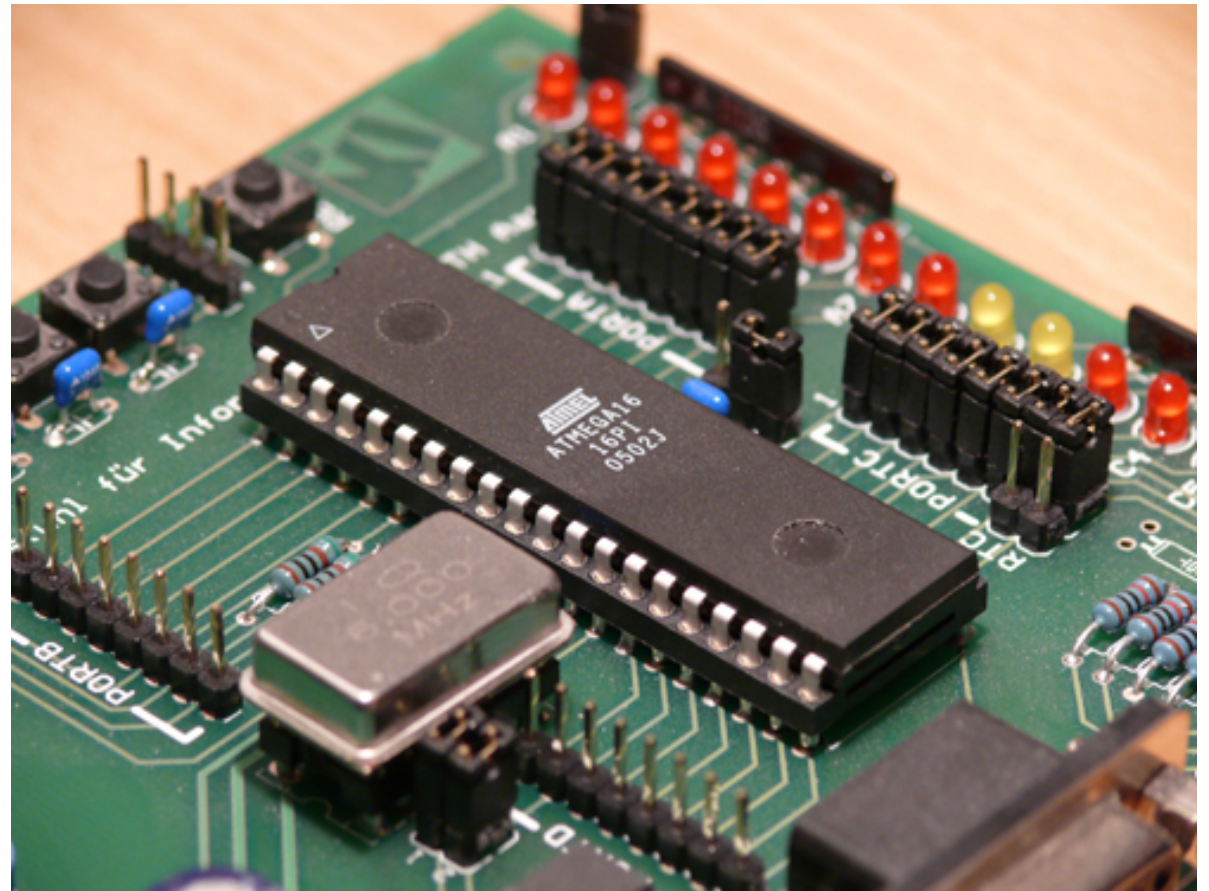


Model Checking



Microcontroller: ATMEL ATmega16

- 8-bit microcontroller
- 16 KB flash memory
- 1K Byte internal SRAM
- 512 bytes EEPROM
- 3 Timer/Counter Units
- 4 I/O Ports 8-bit
- 20 vectorized interrupts
- ...

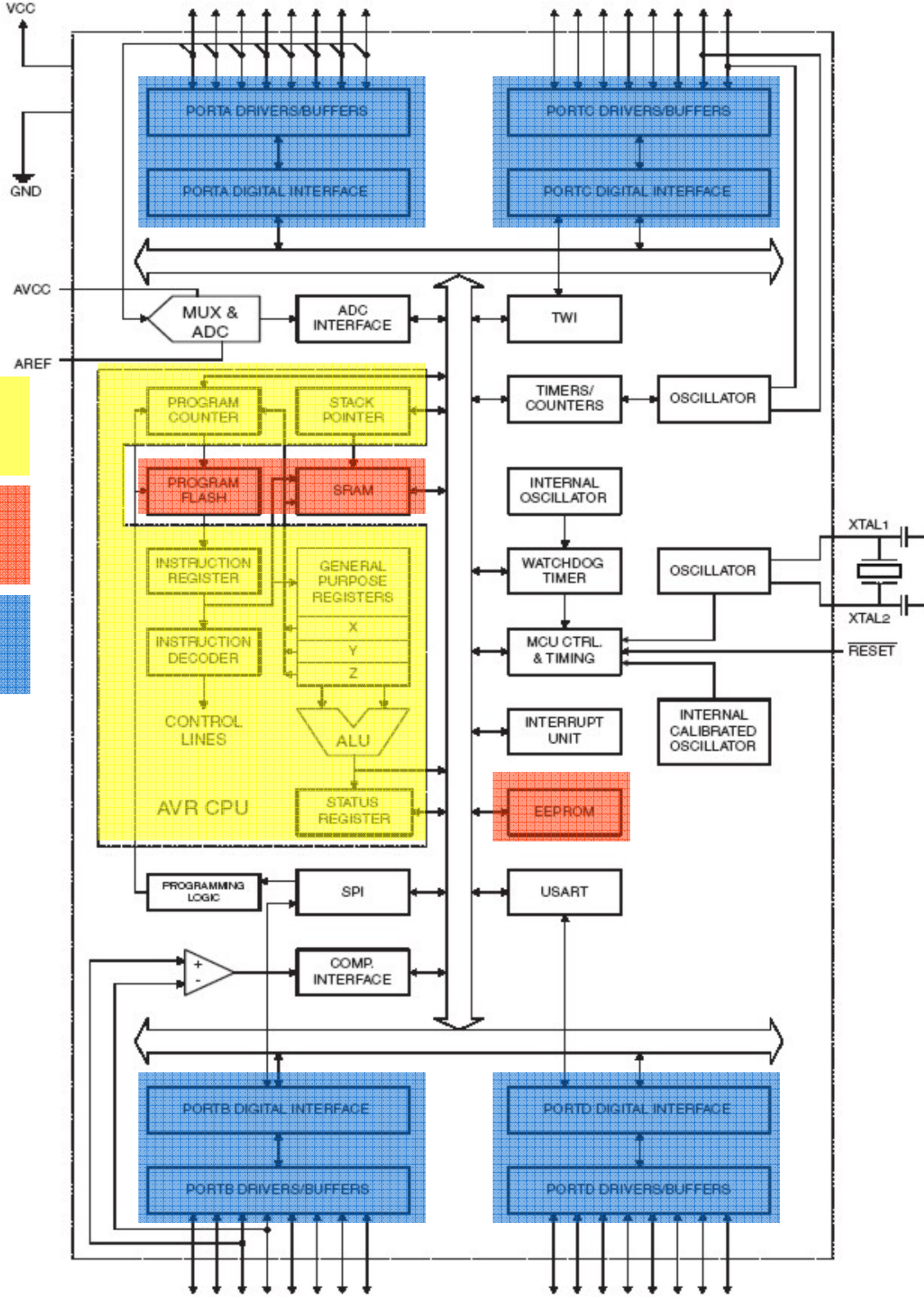


ATmega16

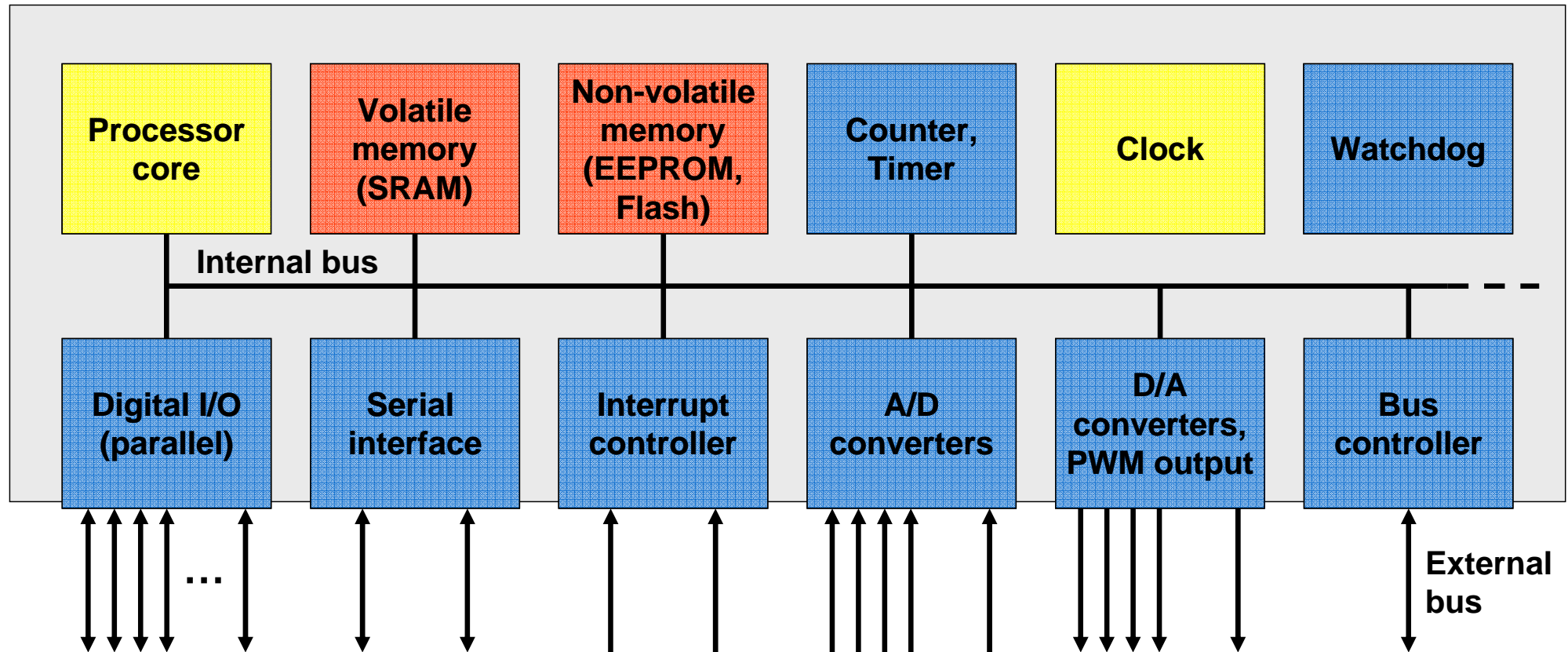
8bit CPU

Memory

I/O Ports



Basic Structure of a Microcontroller



Digital I/O and on-chip peripherals are accessed by dedicated registers.

- different memory types are mapped into different address ranges

(if you use C, the compiler handles most of it)

Outline

- Short Summary: Model Checking
- Excursus: Microcontroller

- Motivation
- Model: C Code vs. Assembly Code
- [mc]square
- Two Case Studies
- Related Work
- Conclusion & Future Work

Motivation

- Microcontrollers are frequently used in devices that we use every day (including safety-critical systems).
 - Full testing is often not possible due to:
 - Uncertain / vague environments
 - Fast time to market
 - Etc.
- ➔ Therefore model checking is considered as a promising future tool for analyzing embedded software.

Model: C Code vs. Assembly Code

- Most microcontroller programs are written in C
 - There are many model checkers that work on C (ANSI C) programs (e.g. Blast, Magic, CBMC)
 - Some restrict the set of supported ANSI C constructs:
 - No statements with side-effects
 - No function calls via pointers
 - No recursions
 - Only integer variables
 - Etc.
 - But: Microcontroller programs include features that are not part of ANSI C, e.g.:
 - Direct hardware accesses
 - Embedded assembly language statements
 - Interrupts
- ➔ Too many restrictions on the C source code.
[Schlich et al. 2005]

Program before Preprocessing

```
int main (void) {
    init();// call initialization
    sei();
    while(1) {
        inputs = PINA & 0x0F;
        cli();
        if (direction != 5) {
            if (inputs & (1 << 1)) { // down
                if (direction != 1 && direction != 2) {
                    TCCR1B = 0x00;
                    TIFR = 0xFF;
                    TCNT1 = 0x00;
                    TIMSK = (1<<OCIE1A);
                    TCCR1B = 0x05;
                    direction = 1;
                }
            }
        }
    }
}
```

...

Program after Preprocessing

```
int main (void) {
    init();
    __asm__ __volatile__ ("sei" :::);
    while(1) {
        inputs = (*(volatile uint8_t *)((0x19) + 0x20)) & 0x0F;
        asm__ __volatile__ ("cli" :::);
        if (direction != 5) {
            if (inputs & (1 << 1)) {
                if (direction != 1 && direction != 2) {
                    (*(volatile uint8_t *)((0x2E) + 0x20)) = 0x00;
                    (*(volatile uint8_t *)((0x38) + 0x20)) = 0xFF;
                    (*(volatile uint16_t *)((0x2C) + 0x20)) = 0x00;
                    (*(volatile uint8_t *)((0x39) + 0x20)) = (1<<4);
                    (*(volatile uint8_t *)((0x2E) + 0x20)) = 0x05;
                    direction = 1;
                }
            }
        }
    }
    ...
}
```

Assembly Code

- Microcontroller C programs include embedded assembly language statements
- C programs are compiled into assembly language
- Examples from industry showing errors which:
 - Passed all tests and reviews
 - Only observable on assembly level
 - Caused by e.g.:
 - Change of compiler version
 - Forgotten interrupt disabling
 - Reentrance problems

➔ Use the Assembly code for model checking

Advantages of Using Assembly Code [Schlich et al. 2005, 2006]

- Assembly constructs are easy to handle, they have a clean and well documented semantics.
- Code is checked that is deployed to the microcontroller
- All errors that are introduced during the development can be found.
 - Compiler errors
 - Errors not visible in the C code (e.g. reentrance problems)
 - Errors in handling the hardware
- This comes at the cost of hardware dependency and bigger state spaces.

Outline

- Short Summary: Model Checking
- Excursus: Microcontroller

- Motivation
- Model: C Code vs. Assembly Code
- [mc]square
 - Goals
 - Features
 - Procedure
 - Example of Handling Nondeterminism
- Two Case Studies
- Related Work
- Conclusion & Future Work

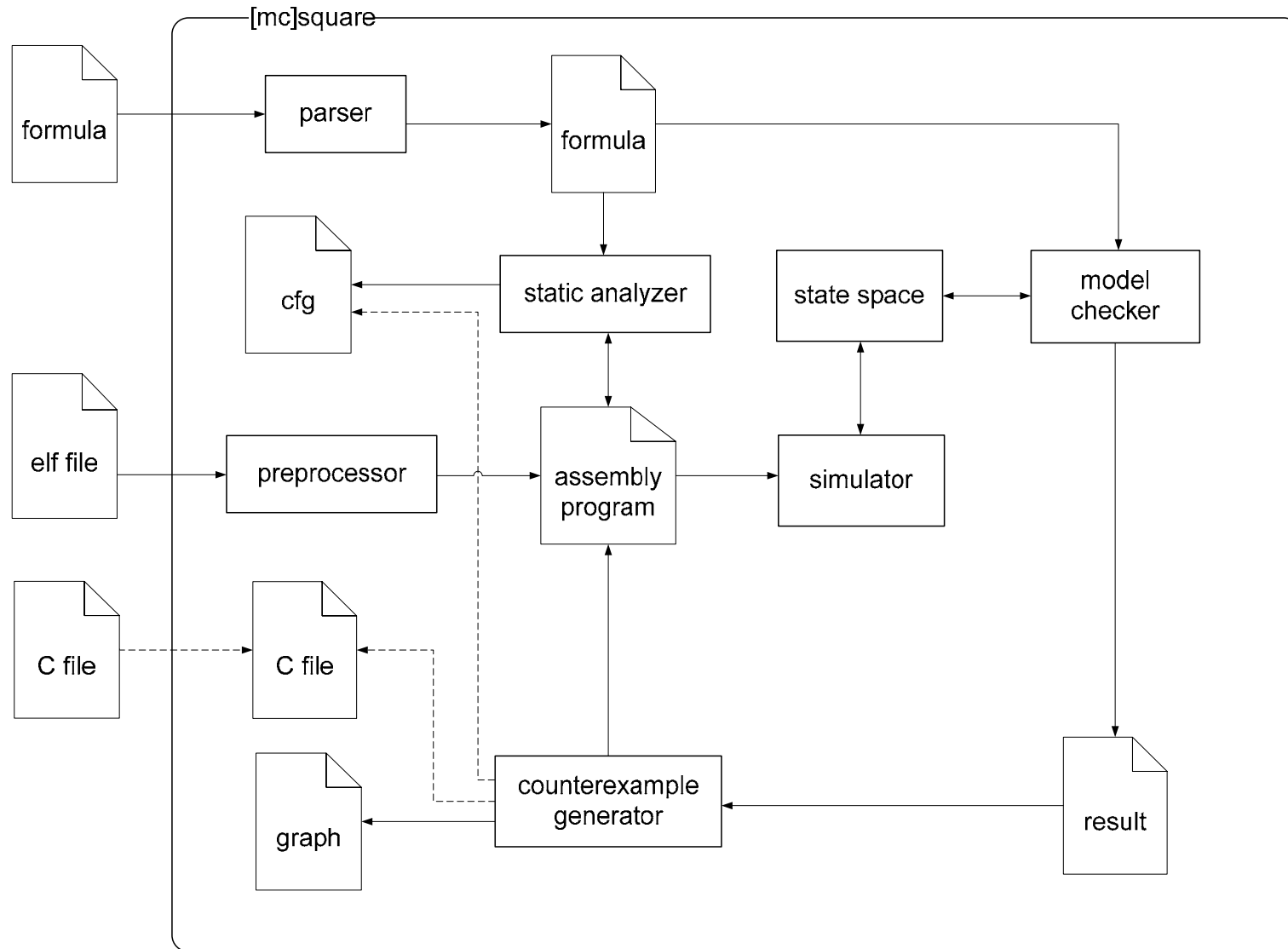
Goals

- No manual effort should be needed for preparation.
- All instructions and hardware features should be supported
- The model checking tool should be extendable in two directions:
 - New microcontrollers
 - New model checking algorithms
- Graphical User Interface that helps the user to model check the code

[mc]square - Features

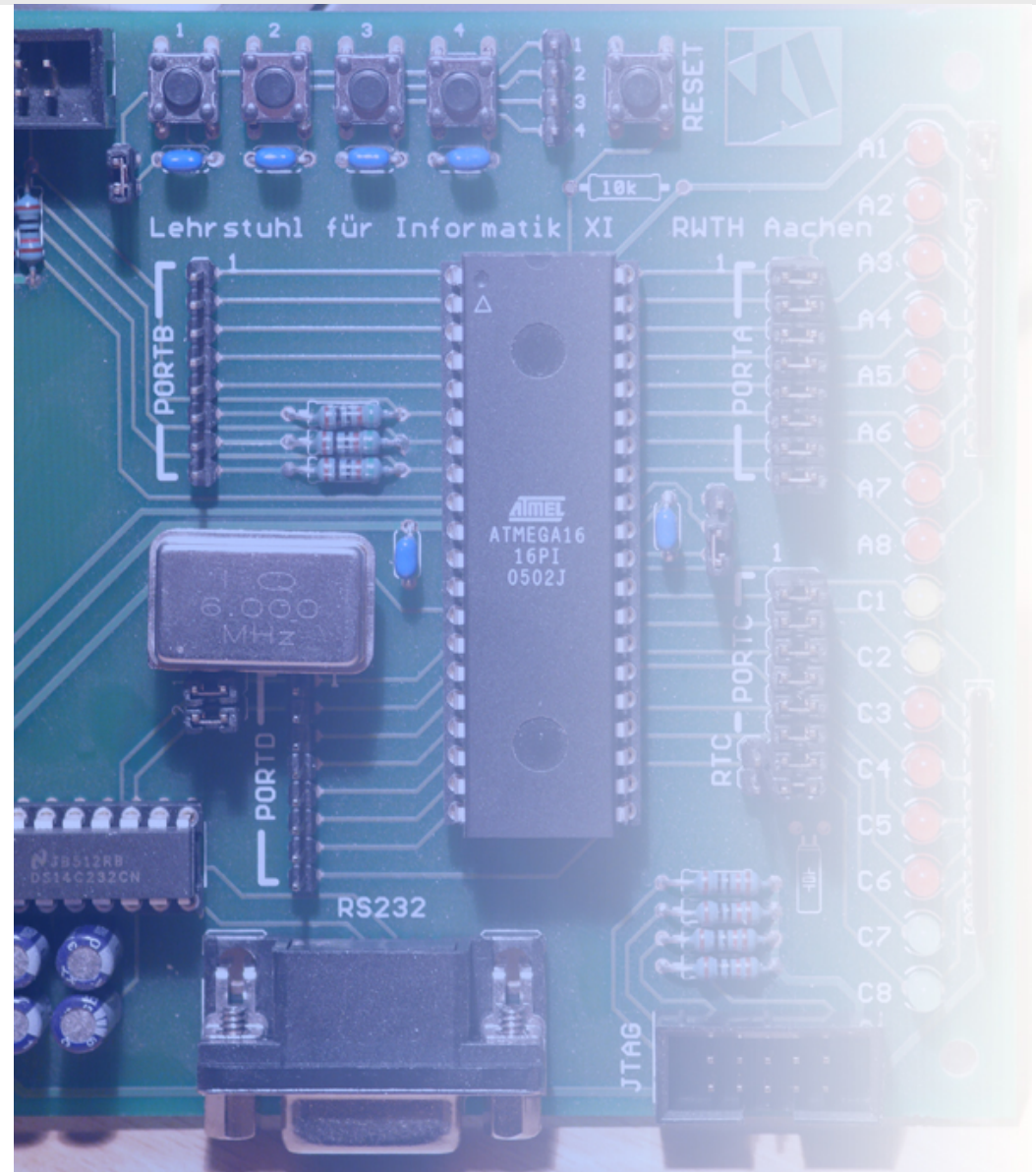
- Model: File in Executable and Linking Format (ELF)
- Specification: Full Computational Tree Logic (CTL)
- Supported microcontrollers:
 - ATMEL ATmega16, 32, 128
 - Infineon XC167 (under work)
- Algorithms (explicit):
 - Global model checking algorithm [Clarke et al. 1999]
 - Local model checking algorithm (on-the-fly) [Vergauwen et al. 1993, Heljanko 1997]
- Options:
 - Static analysis (e.g. dead variable reduction)
 - Delayed nondeterminism
 - Different compression algorithms
 - Hard disk model checking
 - Etc.
- Counterexamples presented:
 - C code and Assembly code
 - Control Flow Graph (CFG)
 - State space graph

Procedure

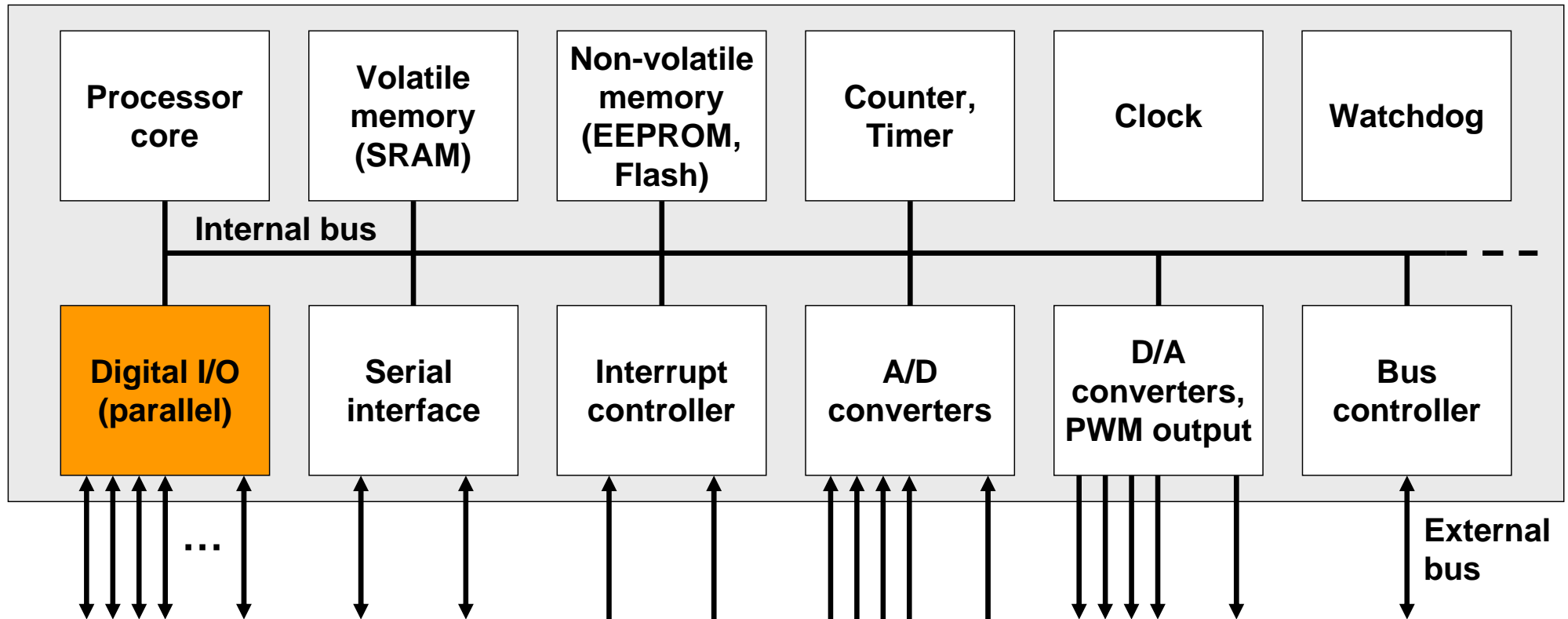


Sources for Nondeterminism in ATmega16

- I/O ports
- SPI
- Timer
- TWI
- USART
- Etc.

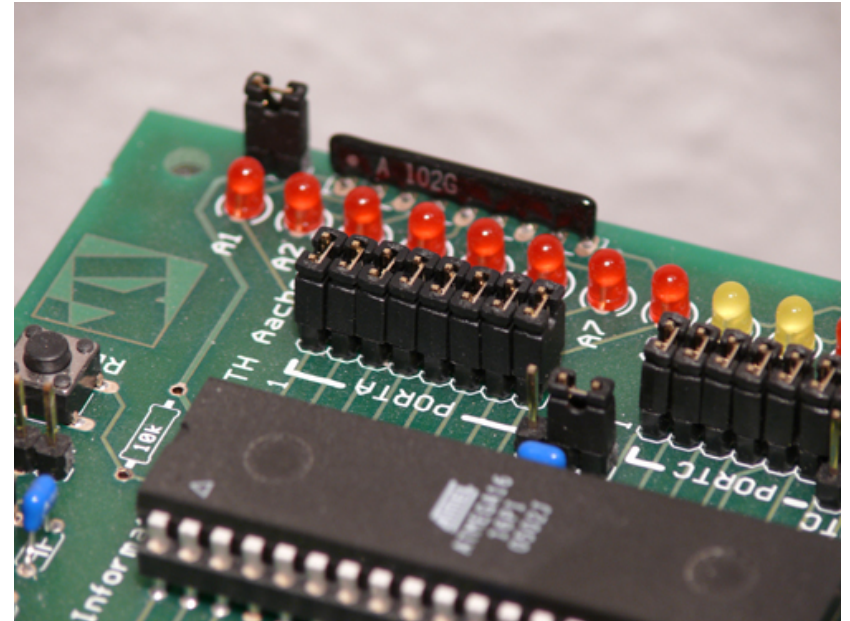


Nondeterminism of I/O Ports



I/O Ports

- Basic means to monitor and control external hardware.
- Usually, I/O ports
 - consist of **8 I/O pins** (byte access)
 - are **bidirectional** (i.e., can be used as input or output pins)
 - can have **alternate** functions (i.e., can be used for purposes different than digital I/O, e.g., as analog I/O pins)
- Monitoring, access and control of I/O ports is done via **three special registers** for each port:
 - **Data Direction Register (DDR)**
 - **Port Register (PORT)**
 - **Port Input Register (PIN)**



Control of I/O Ports via Registers

- **Data Direction Register (DDR):**
 - read/write
 - specifies for each bit of the corresponding port whether it is an **input or an output** bit
- **Port Register (PORT):**
 - read/write
 - specifies for the **output** pins whether the output value is **high or low**
 - ATmega16: also used for controlling **pull-up resistors** for **input** pins
- **Port Input Register (PIN):**
 - read only
 - contains the current value (high or low) of all pins (input and output)
 - usual purpose: reading values of input pins

Examples

DDRA

1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---

PORTA

1	0	0	0	1	0	0	1
---	---	---	---	---	---	---	---

OUTPUT

PINA

1	0	0	0	1	0	0	1
---	---	---	---	---	---	---	---

DDRA

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

PORTA

1	1	1	0	1	0	0	1
---	---	---	---	---	---	---	---

INPUT

PINA

N	N	N	N	N	N	N	N
---	---	---	---	---	---	---	---

DDRA

1	1	1	1	1	1	0	0
---	---	---	---	---	---	---	---

PORTA

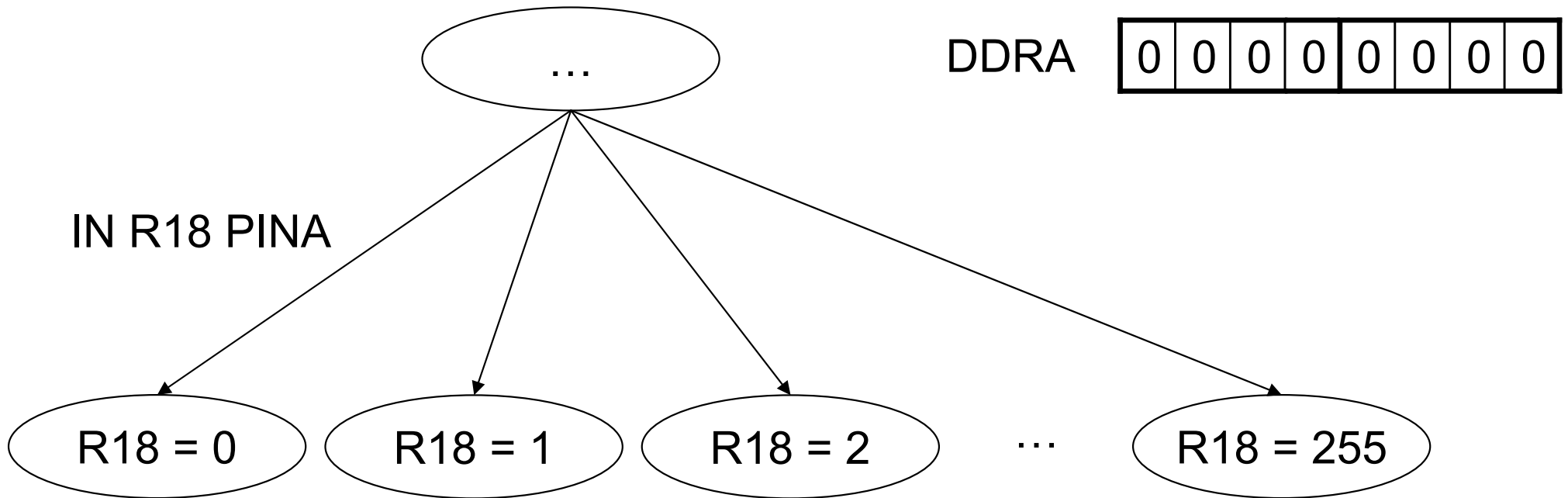
1	0	1	0	1	0	0	1
---	---	---	---	---	---	---	---

INPUT/OUTPUT

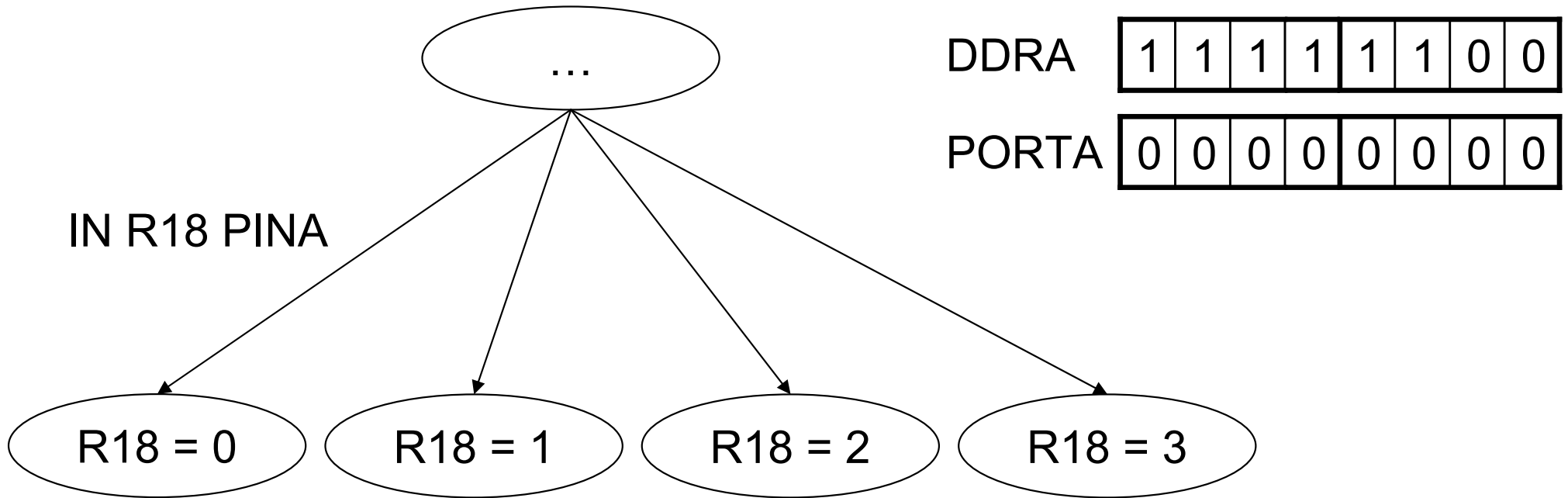
PINA

1	0	1	0	1	0	N	N
---	---	---	---	---	---	---	---

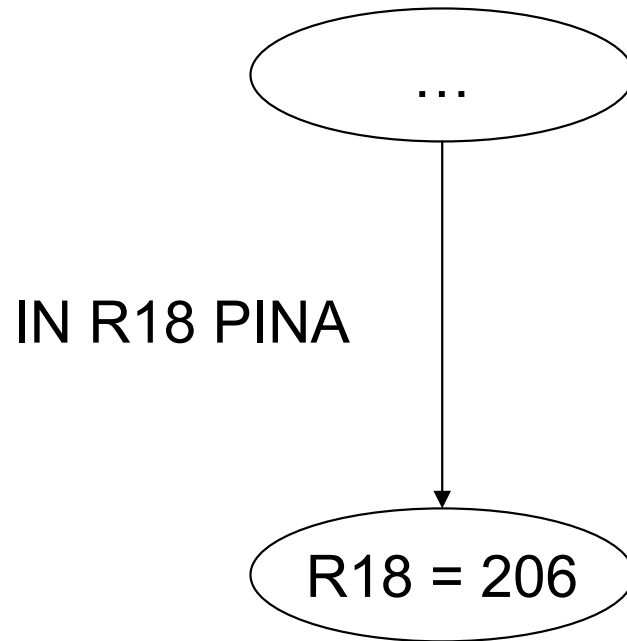
Impact on State Space 1/3



Impact on State Space 2/3



Impact on State Space 3/3



DDRA

1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---

PORTA

1	1	0	0	1	1	1	0
---	---	---	---	---	---	---	---

Nondeterminism of I/O Ports - Summary

- PORTA, DDRA, PINA
- DDRA controls “grade of nondeterminism”:
 - Input
 - Output
 - Input/Output
- Similar methods used for:
 - Timers
 - Interrupts
 - Etc.
- Delayed nondeterminism can be used to minimize state space even more.



Outline

- Short Summary: Model Checking
- Excursus: Microcontroller

- Motivation
- Model: C Code vs. Assembly Code
- [mc]square
- Two Case Studies
- Related Work
- Conclusion & Future Work

Window Lift

- Program that operates a window lift
- Two buttons (up and down) and one motor are used.
- When a button is pressed for more than 1 second, the window is moved in the corresponding direction until it is opened/closed completely or the other button is pressed.
- When an object is stuck, the window is completely opened before normal operation is possible again.

- Implementation:
 - 114 lines of C code + 3 additional libraries (1664 lines of assembly code)
 - 3 interrupts used

Example Specification

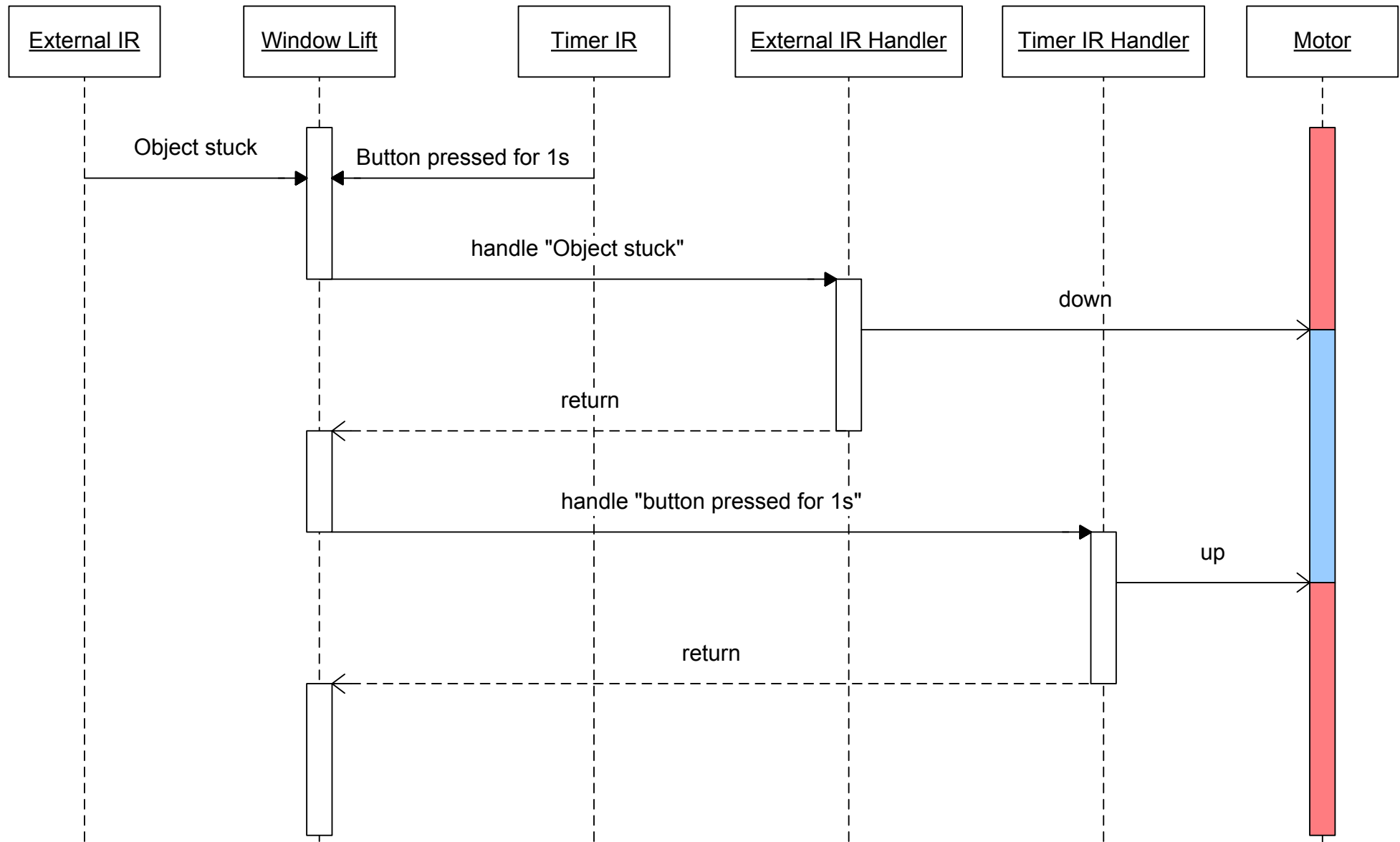
- When an object is stuck, the window is opened completely before normal operation is possible again:

$AG (mode = 5 \Rightarrow \neg E \text{ mode} = 5 \ U (mode \neq 5 \wedge mode \neq 6))$

➔ Not satisfied

- Cause: Interaction between two interrupts. Two events occurred at the same time (object stuck and „up“ button pressed for 1 second).

Sequence that leads to Error



Solution to this Error

Reset flags of pending interrupts in interrupt handler of external interrupt (object stuck).

→ Satisfied



Next Program: Non-Atomic Write

- In the main program the variable i (16 bit integer) is incremented.
- In the interrupt handler an output is created depending on the variable i .
- The variable i is only allowed to stay in interval $0 \leq i \leq 300$:
 $AG (i \geq 0 \wedge i \leq 300)$
- If i is outside this interval, an error occurs.

Evaluation

- Window Lift:
 - Standard: 2,313,660 states
 - Using all options: 10,699 states (stored)
- Non-Atomic Write:
 - Standard: 107,649 states
 - Using all options: 6,631 states (stored)
- [mc]square can deal with state spaces of up to 80,000,000 states in memory (16 GB main memory).
- When using the hard disk model checker, [mc]square can store up to 682,740,000 states.

Related Work

- StEAM [Leven et al. 2004]:
 - Assembly code for the Internet C Virtual Machine (ICVM)
 - Main focus: hardware independent, parallel C++ programs
- ESTES [Mercer et al. 2005]:
 - Assembly code for the Motorola 68hc11
 - Uses discrete time
 - User has to provide an environment written in C++
 - Invariants
- MCESS (diploma thesis done at CWI and our institute) [Rohrbach 2006]:
 - Assembly code written for the ATMEL ATmega16
 - Translates assembly code into bytecode for a virtual machine

Conclusion

- [mc]square is able to model check assembly code written for specific microcontrollers.
- C source code is not needed.
- A simulator is used to build an over-approximation of the real state space.
 - Simulator can be exchanged.
 - Over-approximation can be adjusted via options.
- Hardware-dependency is used to:
 - Minimize state space
 - Accurately represent behavior of the microcontroller
- Some non-trivial programs can be checked.
- Size of state spaces is influenced by different factors.
 - Some are obvious, others not.
 - Lines of Code are no direct indication on the size of the state space.

Future Work

- Additional microcontroller: Infineon XC167 (under work)
- Improve simulator (new options)
- Integrate other simulators (e.g. Abstract State Machines and Instruction List)
- Implement assume-guarantee approach
- Combine explicit and symbolic approach