Einführung

Bevor wir starten: Haben Sie schon einmal darüber nachgedacht, wie aus einem Hochsprachen-Programm ein tatsächliches Verhalten eines Computers wird? Der Idee liegt darin, die Komplexität dieser Frage in handhabbare Schichten zu unterteilen. Wie diese Schichten genau aussehen, werden wir in dieser Vorlesung systematisch erarbeiten.

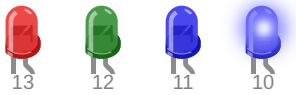
Parameter	Kursinformationen
Veranstaltung:	`Digitale Systeme / Eingebettete Systeme`
Semester:	`Wintersemester 2025/26`
Hochschule:	`Technische Universität Freiberg`
Inhalte:	`Motivation der Vorlesung "Eingebettete Systeme" und Beschreibung der Organisation der Veranstaltung`
Link auf GitHub:	https://github.com/TUBAF-IfI-LiaScript/VL_EingebetteteSysteme/blob/master/00_Einfuehrung.md
Autoren:	Sebastian Zug, André Dietrich & `fjangfaragesh`, `FnHm`, `gjaeger`, `ShyFlyGuy`, `Lalelele`

Fragen an die Veranstaltung

- Welche Abstraktionsebenen durchlaufen wir vom Arduino-Code bis zur Hardware-Ebene?
- Warum ist das Verständnis eingebetteter Systeme für Informatiker wichtig?
- Welche Rolle spielen Mikrocontroller in modernen digitalen Systemen?
- Wie unterscheiden sich eingebettete Systeme von herkömmlichen Computersystemen?
- Welche praktischen Anwendungsgebiete finden Sie für eingebettete Systeme in Ihrem Alltag?
- Was bedeutet „Real-Time" im Kontext eingebetteter Systeme?
- Welche Herausforderungen ergeben sich beim Debugging auf Hardware-Ebene?
- Wie wirken sich Ressourcenbeschränkungen auf die Programmierung aus?

Worum geht es in dieser Vorlesung?

Starten wir mit etwas Code ...



Simulation time: 00:07.000 (76%)

BlinkLEDs.ino

```
byte leds[] = {13, 12, 11, 10};

void setup() {
  Serial.begin(115200);
  for (byte i = 0; i < sizeof(leds); i++) {
    pinMode(leds[i], OUTPUT);
  }
}

int i = 0;
void loop() {
  Serial.print("LED: ");
  Serial.println(i);
  digitalWrite(leds[i], HIGH);
  delay(250);
  digitalWrite(leds[i], LOW);
  i = (i + 1) % sizeof(leds);
}
```

```
Sketch uses 2238 bytes (6%) of program storage space. Maximum is 32256
bytes.
Global variables use 200 bytes (9%) of dynamic memory, leaving 1848
bytes for local variables. Maximum is 2048 bytes.


LED: 0
Sketch uses 2238 bytes (6%) of program storage space. Maximum is 32256
bytes.
Global variables use 200 bytes (9%) of dynamic memory, leaving 1848
bytes for local variables. Maximum is 2048 bytes.


LED: 0
LED: 1
LED: 1
LED: 2
LED: 2
LED: 3
LED: 3
LED: 0
LED: 0
LED: 1
LED: 1
LED: 2
LED: 2
LED: 3
LED: 3
LED: 0
LED: 0
LED: 1
LED: 1
LED: 2
LED: 2
LED: 3
LED: 3
LED: 0
LED: 0
LED: 1
LED: 1
LED: 2
LED: 2
LED: 3
```

```
LED: 3
LED: 0
LED: 0
LED: 1
LED: 1
LED: 2
LED: 2
LED: 3
LED: 3
LED: 0
LED: 0
LED: 1
LED: 1
LED: 2
LED: 2
LED: 3
LED: 3
LED: 0
LED: 0
LED: 1
LED: 1
LED: 2
LED: 2
LED: 3
LED: 3
```

Frage: Welche Funktion hat der Code?

Die Funktion des Codes besteht darin, nacheinander vier LEDs an den Pins 10 bis 13 des Arduino-Boards ein- und auszuschalten. Dabei wird jede LED für 250 Millisekunden aktiviert, bevor die nächste LED eingeschaltet wird. Schlagen Sie ggf. die Dokumentation der Befehle unter https://www.arduino.cc/reference/en/ nach. Interessant an diesem Code: Er ist nur 20 Zeilen lang, aber dahinter verbergen sich tausende von Zeilen Arduino-Bibliothekscode, Compiler-Optimierungen, und am Ende Millionen von Transistor-Schaltoperationen. Die `digitalWrite()`-Funktion allein ruft über sieben weitere Funktionen auf, bevor sie einen einzigen Pin schaltet.

Frage: Was ist notwendig, damit dieses Programm auf einem Mikrocontroller ausgeführt werden kann?

Der Code muss zunächst übersetzt und in eine für den Mikrocontroller verständliche Form gebracht werden. Dies geschieht in mehreren Schritten: 1. **Kompilierung:** Der C++-Code wird in Assembler-Code übersetzt, der spezifisch für die AVR-Architektur des Mikrocontrollers ist. 2. **Assemblierung:** Der Assembler-Code wird in Maschinencode umgewandelt, der aus binären Instruktionen besteht, die der Mikrocontroller direkt ausführen kann.

```
:100000000C945D000C9485000C9485000C94850084
:100010000C9485000C9485000C9485000C9485004C
:100020000C9485000C9485000C9485000C9485003C
:100030000C9485000C9485000C9485000C9485002C
:100040000C9412020C9485000C9482020C945C02B5
:100050000C9485000C9485000C9485000C9485000C
:100060000C9485000C9485000000000024002700FB
:100070002A0000000000250028002B0004040404CE
:1000800004040404020202020202030303030342
:10009000010204081020408001020408102001021F
...
```

Am Ende des Semesters können Sie jeden Schritt dieser Transformation verstehen und erklären!

Das Faszinierende: Wenn Sie diese Hex-Datei verstehen können - und das werden Sie am Ende des Semesters - dann können Sie jeden Computer der Welt verstehen. Die Grundprinzipien sind universell, egal ob es sich um einen Arduino, einen Smartphone-Prozessor oder einen Supercomputer handelt. Schauen Sie genau hin: Die erste gepunktete Zeile beginnt mit "0C 94 72 00". Das sind vier Bytes, die einen einzigen Maschinenbefehl darstellen - einen Jump-Befehl zur Adresse 0x0072. Dort steht der Interrupt-Vektor für den Reset.

Die Reise: 16 Stationen zum Verständnis

Vorlesung	Was lernen Sie hier?
00 Einführung	Die große Vision
01 Historie	Von Leibnitz zu Arduino
02 Boolesche Algebra	0 und 1 verstehen
03 Minimierung	Effizienz in Logik
04 Schaltnetze	Logik wird Hardware
05 Standardschaltnetze	Decoder & Multiplexer
06 FlipFlops	Speicher entsteht
07 Schaltwerke	Zustandsmaschinen
08 Standardschaltwerke	Counter & Register
09 Rechnerarithmetik	Addition in Hardware
10 CPU-Basis	Der erste Prozessor
11 Modell-CPU	CPU-Simulation
12 Pipeline	Geschwindigkeit
13 AVR-CPU	**IHR Arduino-Chip!**
14 ADC	Analog trifft Digital
15 Timer & Interrupts	Multitasking
16 Aktoren	Hardware ansteuern

Wir haben nominell 21 Vorlesungen, an einigen Stellen werden wir aber länger verweilen und an anderen Stellen schneller vorankommen. Diese Aufzählung ist also eher ein Leitfaden, denn ein strikter Plan.

Was steht am Ende?

Ein kleiner Reality-Check: Diese acht Fragen würden viele Kommmilitonen der höheren Semester zum Schwitzen bringen. Nach diesem Kurs gehören Sie zu den wenigen, die wirklich verstehen, was unter der Haube passiert. Das macht Sie unglaublich wertvoll auf dem Arbeitsmarkt.

Sie können diese 8 Fragen fundiert beantworten:

1. **"Wie wird `pinMode(13, OUTPUT)` zu Hardware-Konfiguration?"** *Von Arduino-Funktion über AVR-libc zu Assembler zu DDRB-Register zu Tri-State-Logik*
2. **"Was bedeutet, das `Serial.begin(9600)`?"** *UART ist eine asynchrone Kommunikationsschnittstelle, wir vereinbaren zu Beginn die "Sprechgeschwindigkeit".*
3. **"Ist eine `delay(1000)` tatsächlich genau 16.000.000 Taktzyklen lang?"** *Taktfrequenz, Befehlszyklen und Timer-Hardware verstehen*
4. **"Was passiert im ATmega328P während `analogRead(A0)`?"** *ADC-Wandlung, Sampling, Quantisierung und Speicherung*
5. **"Wie führt die CPU den Maschinenbefehl `0C 94 72 00` aus?"** *Instruction Decode, Fetch-Execute-Cycle, Register-Operationen*
6. **"Warum ist `if (digitalRead(2) && digitalRead(3))` als Schaltung effizienter?"** *Boolesche Algebra, Gatter-Optimierung, Hardware-Software-Grenze*
7. **"Wie kann ein 8-bit-Mikrocontroller 16-bit-Zahlen addieren?"** *Multi-Precision-Arithmetik, Carry-Flag, ALU-Design*

Der Unterschied zu einem "Arduino-Kurs"

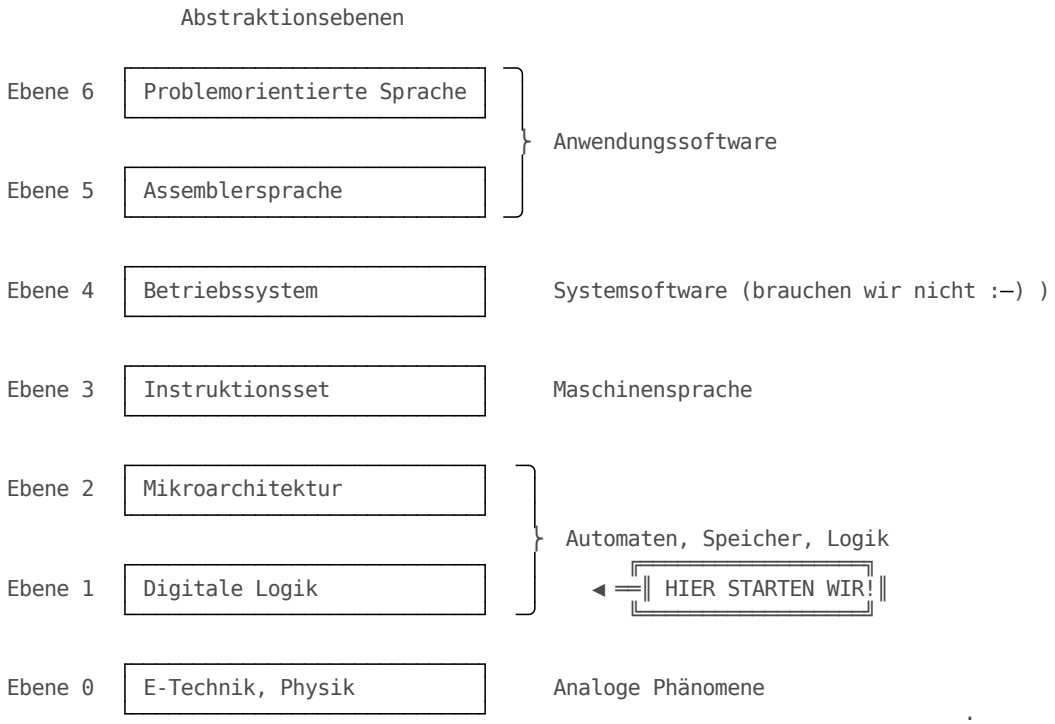
Hier ist ein wichtiger Punkt: Nach einem typischen Arduino-Workshop können Sie LEDs blinken lassen. Nach diesem Kurs können Sie erklären, warum die `digitalWrite()`-Funktion exakt 12 Taktzyklen benötigt und welche vier Hardware-Register dabei beschrieben werden. Der Unterschied zwischen "es funktioniert" und "ich verstehe warum es funktioniert".

Arduino-Kurs: "Drücke diesen Button, LEDs blinken" ✨ *Magie!*

Unser Kurs: "Warum passiert das und wie funktioniert es bis hinunter zum Transistor?"

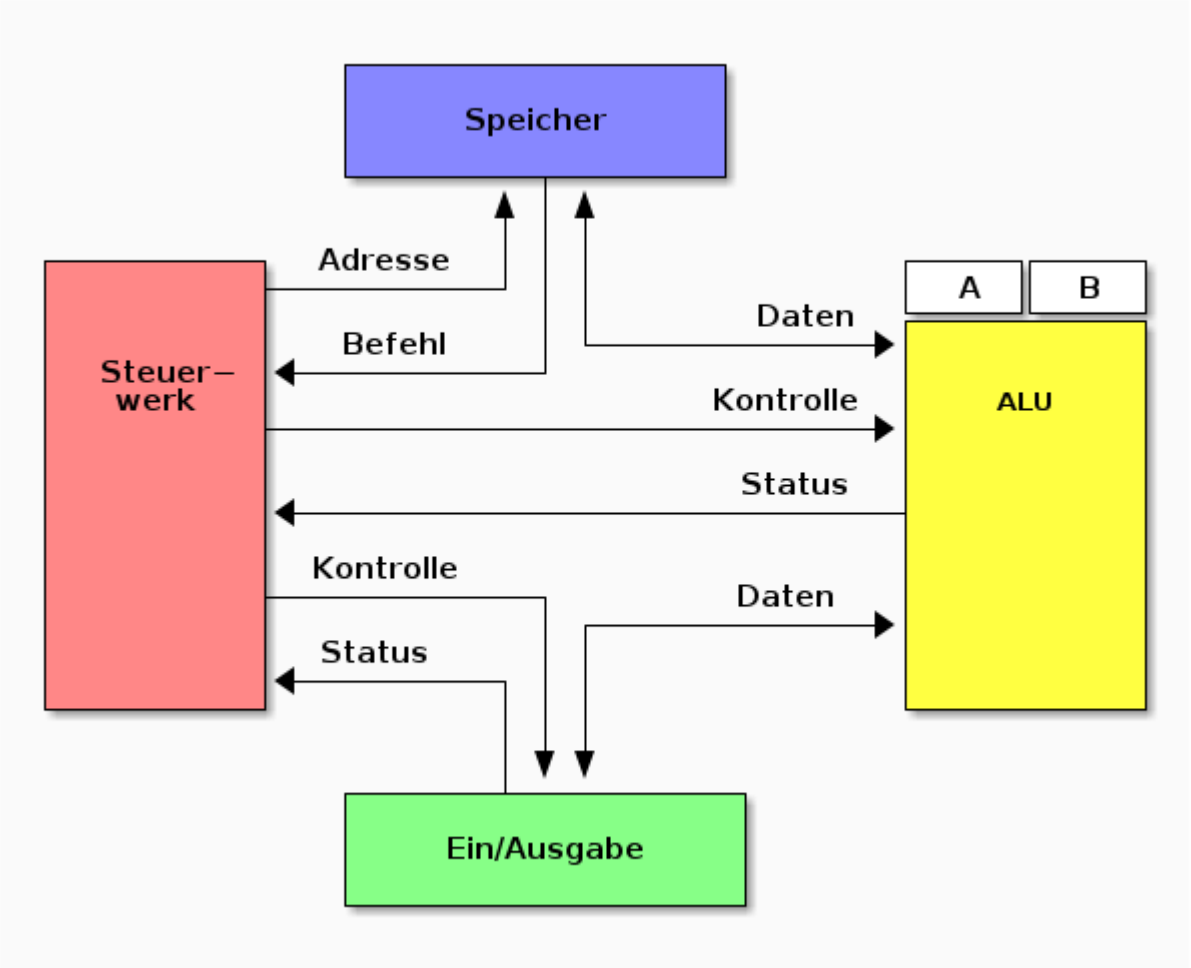
Unsere Bottom-Up-Reise

Schauen Sie sich diese Abstraktionsebenen an: Die meisten Programmierer arbeiten nur auf Ebene 6 und 5. Wir steigen bis auf Ebene 1 hinab und arbeiten uns systematisch nach oben.



Beispiel 1: Mikroarchitektur

Ein Rechner ist eine ziemlich komplizierte Maschine. Moderne CPUs oder GPUs umfassen Milliarden von Transistoren! Wir brauchen ein methodisches Verständnis und mehrere Abstraktionsebenen, um das zu verstehen.



Der ATmega328P hat "nur" etwa 100.000 Transistoren. Aber auch das ist schon gewaltig - stellen Sie sich vor, Sie müssten 100.000 Schalter von Hand verkabeln! Fun Fact: Diese Struktur wurde 1945 von John von Neumann beschrieben und ist heute noch gültig. Ihr Mobiltelefons hat die gleiche Grundarchitektur wie dieser Arduino - nur mit ein paar Milliarden Transistoren mehr. Die Kunst liegt darin, diese Komplexität in verständliche Module zu unterteilen.

Beispiel 2: Quer über alle Ebenene

Hier wird's richtig interessant: Wir verfolgen eine einzige Arduino-Zeile durch vier Abstraktionsebenen. Das ist wie ein CSI für Programmierer - wir verfolgen jeden Beweis, bis wir den echten Täter finden: den Transistor, der die Arbeit macht.

Wir bauen von den Grundlagen nach oben! Jede Ebene erklärt die nächste, bis Sie verstehen, wie Ihr `pinMode()` am Ende einen Transistor schaltet.

Schauen wir uns konkret an, was in diesen 4 Abstraktionsebenen passiert:

Ebene 6 - Ihr Arduino-Code:

Ebene sechs sieht harmlos aus, aber diese eine Zeile triggert eine Kaskade von 50+ Funktionsaufrufen in der Arduino-Laufzeitumgebung. Pin 13 ist übrigens kein Zufall - das ist der eingebaute LED-Pin auf den meisten Arduino-Boards.

```
pinMode(13, OUTPUT);  // Pin 13 als Ausgang konfigurieren
```

Ebene 5 - AVR-libc Implementation (echte Arduino-Version):

Hier wird's professionell: Die Arduino-Entwickler müssen Interrupts abschalten, weil ein anderer Interrupt-Handler zur falschen Zeit das gleiche Register manipulieren könnte. Das nennt man Race Condition - ein klassisches Problem in der Embedded-Programmierung.

```
void pinMode(uint8_t pin, uint8_t mode) {
  uint8_t bit = digitalPinToBitMask(pin);    // Pin 13 → Bit 5
  uint8_t port = digitalPinToPort(pin);      // Pin 13 → PORTB
  volatile uint8_t *reg = portModeRegister(port); // → &DDRB

  if (mode == OUTPUT) {
    uint8_t oldSREG = SREG;  // ⚠️ INTERRUPT-SCHUTZ!
    cli();                   // Interrupts AUS
    *reg |= bit;            // DDRB |= (1 << 5)
    SREG = oldSREG;         // Interrupts wieder AN
  }
  // INPUT und INPUT_PULLUP analog...
}
```

Ebene 3 - AVR-Assembler Code:

Jetzt sind wir bei der Maschinensprache angekommen. "CLI" schaltet global alle Interrupts ab - ein drastischer Schritt! "OUT 0x04, r24" schreibt direkt ins Hardware-Register DDRB. Diese 0x04 ist eine magische Zahl - die Speicheradresse des Data Direction Register B im AVR-Chip.

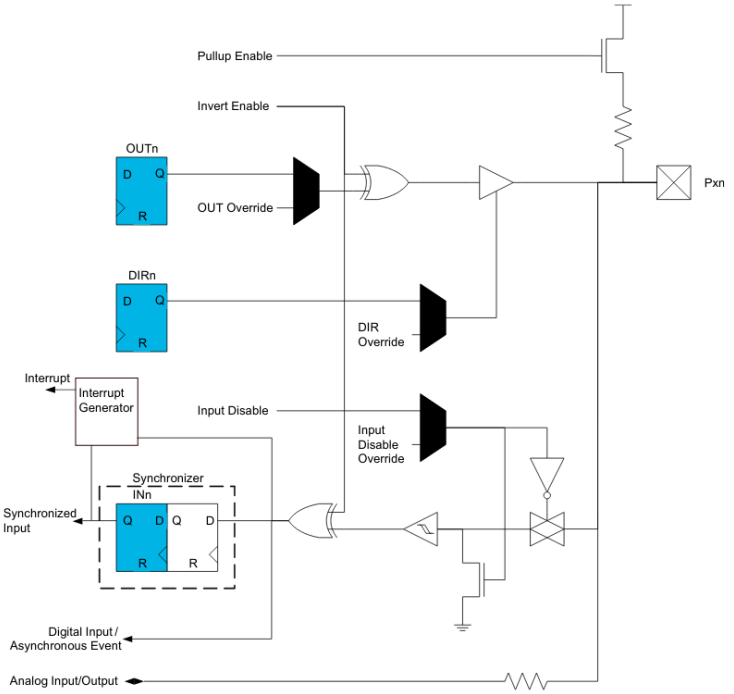
```
; uint8_t oldSREG = SREG; cli();
in   r25, 0x3F    ; Lade SREG (Status Register)
cli               ; Clear Interrupt Flag → Interrupts AUS

; *reg |= bit;  (DDRB |= (1 << 5))
in   r24, 0x04    ; Lade DDRB Register (I/O-Adresse 0x04)
ori  r24, 0x20    ; OR mit 0x20 (Bit 5 setzen)
out  0x04, r24    ; Schreibe zurück zu DDRB

; SREG = oldSREG;
out  0x3F, r25    ; Restore SREG → Interrupts wieder AN
```

Ebene 1 - Hardware-Konfiguration:

Endlich am Ziel: echte Hardware! Das DDRB-Register besteht aus acht Flip-Flops, jedes steuert einen Pin. Wenn Sie Bit 5 setzen, öffnet sich ein Transistor-Paar am Pin 13. Der obere Transistor kann dann +5V durchschalten, der untere 0V. Das ist der Moment, wo Software zu Physik wird.

16.2.1 **Block Diagram**

Figure 16-1. PORT Block Diagram



Darstellung der Input/Output Beschaltung eines Microcontrollers

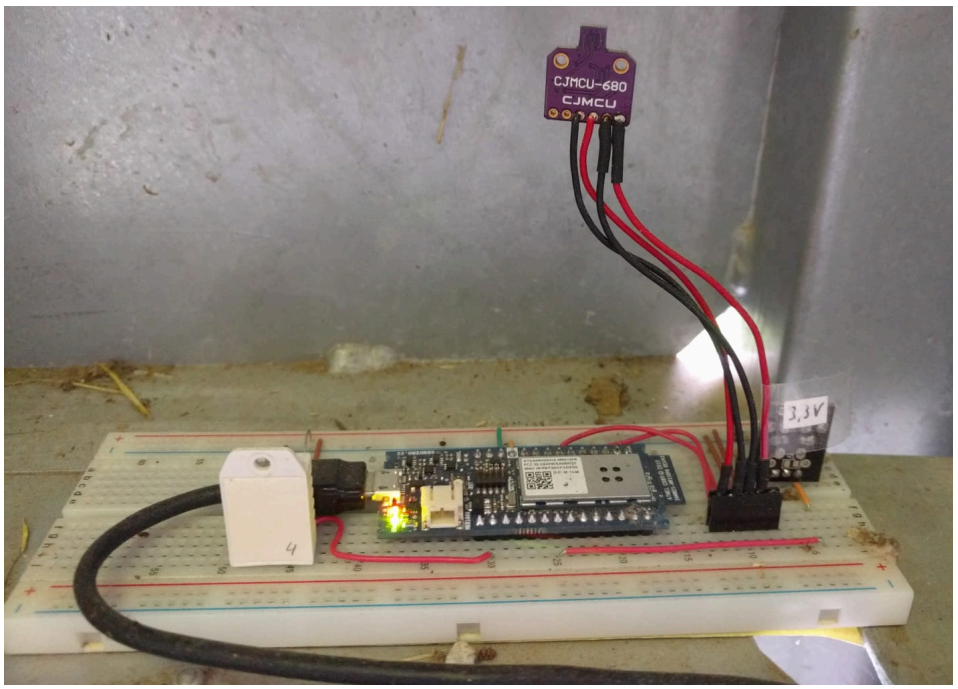
Das `DDRB`-Register steuert direkt die **Tri-State-Logik** des I/O-Pins:

- **DDRB[5] = 1**: Ausgangstreiber wird aktiviert → Pin kann HIGH/LOW ausgeben
- **DDRB[5] = 0**: Hochohmig → Pin kann Eingangssignale lesen

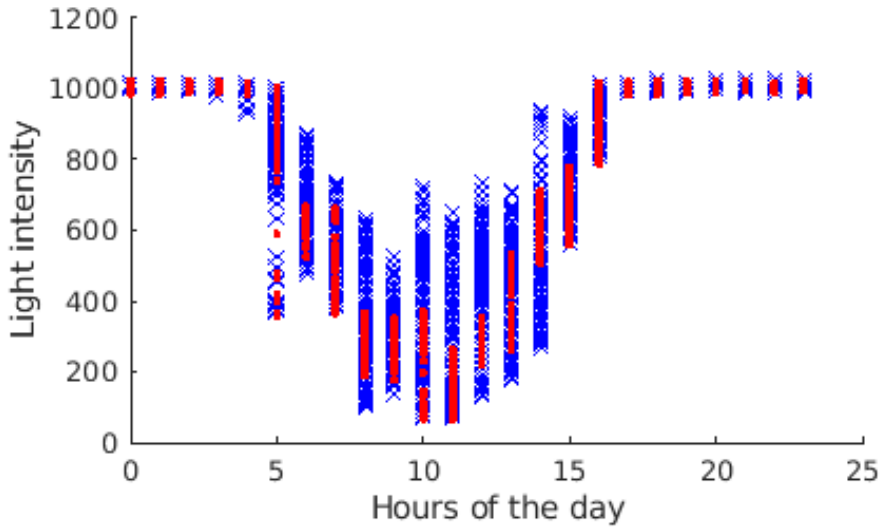
Was können Sie damit bauen?

Mit diesem tiefen Verständnis können Sie Systeme optimieren, die andere nur benutzen können. Sie wissen, warum der ADC-Wandler manchmal "rauscht", wie Sie Timer-Interrupts für präzises Timing einsetzen, und warum manche Sensoren 5V brauchen und andere mit 3.3V auskommen.

Von einfachen LEDs zu echten Projekten:



Nach diesem Kurs wissen Sie nicht nur, wie man Sensordaten ausliest, sondern **warum** der ADC-Wandler 1023 als Maximum liefert und wie die Timer-Interrupts das Sampling steuern.



Sie verstehen, warum die Messwerte (1023 = dunkel, niedrige Werte = hell) genau so aussehen und können die ADC-Referenzspannung für präzisere Messungen optimieren.

"Aber ich will doch Webentwickler/Data Scientist/KI-Entwickler werden..."

Hier ist ein Geheimnis der Tech-Industrie: Die wertvollsten Entwickler sind die, die eine Schicht tiefer schauen können als ihre Kollegen. Wenn Ihre Webanwendung langsam ist und andere raten, wissen Sie genau, welche CPU-Zyklen verschwendet werden. Das macht den Unterschied zwischen Senior- und Junior-Entwickler aus.

Perfekt! Denn Sie werden ein BESSERER Entwickler in JEDEM Bereich:

- **Webentwickler**: Sie verstehen Performance bis zur Hardware-Ebene
- **Data Scientist**: Sie wissen, wie Ihre Daten WIRKLICH entstehen
- **KI-Entwickler**: Edge-AI auf Mikrocontrollern? Sie können es!
- **Cloud-Architekt**: Sie verstehen die Hardware unter Ihren VMs
- **Game-Developer**: Low-Level-Optimierung wird Ihr Superpower

Das tiefe Hardware-Verständnis macht Sie in JEDER Tech-Rolle wertvoller!

Organisation

Name	Email
Prof. Dr. Sebastian Zug	sebastian.zug@informatik.tu-freiberg.de
Adrian Köppen	adrian.koeppen@student.tu-freiberg.de

Bitte melden Sie sich im OPAL unter Digitale Systeme für die Veranstaltung an. Dort finden Sie auch die aktuellen Informationen zur Veranstaltung.

Zeitplan

Die Veranstaltung wird sowohl für die Vorlesung als auch die Übung in Präsenz durchgeführt.

Veranstaltungen	Tag	Zeitslot	Ort	Bemerkung
Vorlesung I	Mittwoch	16:15 - 17:45	FOR-0270	wöchentlich
Vorlesung II	Donnerstag	09:45 - 11:15	FOR-0270	gerade Wochen

Die zugehörigen Übungen starten im Dezember und werden dann wöchentlich durchgeführt. Dort haben Sie dann insbesondere ab Januar Gelegenheit anhand spezifischer Mikrocontrollerschaltungen Ihre Kenntnisse praktisch zu vertiefen.

Remote-Labore werden etwa Dezember bereitstehen, um einzelne Aspekte der Lehrveranstaltung zu vertiefen.

Prüfungsmodalitäten

Prüfungsform: Die Veranstaltung wird mit einer schriftlichen Prüfung abgeschlossen. Diese wird als **Open Book Klausur** entworfen. Sie dürfen dazu alle schriftlichen, nicht-digitalen Materialien verwenden.

Im Laufe der Übungen werden wir "alte" Übungsaufgaben durchspielen, die Ihnen einen Eindruck von der Prüfung vermitteln sollen.

Literaturempfehlungen

1. Umfassende Lehrbücher
    - David A. Patterson, John L. Hennessy: Computer Organization & Design
    - B. Becker, R. Drechsler, P. Molitor: Technische Informatik - Eine Einführung, Pearson Studium, 2005
    - Hoffmann, D. W.: Grundlagen der technischen Informatik, Hanser Verlag
2. Videos
    - Youtube – „How a CPU Works“ Link

Bei den jeweiligen Vorlesungsinhalten werden zusätzliche Materialien angegeben.

Engagement und Motivation

Credit-Points: 6

Arbeitsaufwand: Der Zeitaufwand beträgt 180h und setzt sich zusammen aus **60h Präsenzzeit** und **120h Selbststudium**. Letzteres umfasst die Vor- und Nachbereitung der Lehrveranstaltung, die eigenständige Lösung von Übungsaufgaben sowie die Prüfungsvorbereitung.

Open Educational Resources

Die Lehrmaterialien finden Sie unter GitHub, einer Webseite für das Versionsmanagement und die Projektverwaltung.

https://github.com/TUBAF-IfI-LiaScript/VL_EingebetteteSysteme

Die Unterlagen selbst sind in der Auszeichnungsprache LiaScript verfasst und öffentlich verfügbar.

Markdown ist eine Auszeichnungssprache für die Gliederung und Formatierung von Texten und anderen Daten. Analog zu HTML oder LaTex werden die Eigenschaften und Organisation von Textelementen (Zeichen, Wörtern, Absätzen) beschrieben. Dazu werden entsprechende "Schlüsselworte", die sogenannten Tags, verwendet.

Markdown wurde von John Gruber und Aaron Swartz mit dem Ziel entworfen, die Komplexität der Darstellung so weit zu reduzieren, dass schon der Code sehr einfach lesbar ist. Als Auszeichnungselemente werden entsprechend möglichst kompakte Darstellungen genutzt.

HelloWorld.md

```
# Überschrift

__eine__ Betonung __in kursiver Umgebung__

+ Punkt 1
+ Punkt 2

Und noch eine Zeile mit einer mathematischen Notation $a=cos(b)$!
```

Überschrift

eine Betonung in kursiver Umgebung

- Punkt 1
- Punkt 2

Und noch eine Zeile mit einer mathematischen Notation $a = cos(b)$!

Eine gute Einführung zu Markdown finden Sie zum Beispiel unter:

- MarkdownGuide
- GitHubMarkdownIntro

Mit einem entsprechenden Editor und einigen Paketen macht das Ganze dann auch Spaß.

- Wichtigstes Element ist ein Previewer, der es Ihnen erlaubt "online" die Korrektheit der Eingaben zu prüfen
- Tools zur Unterstützung komplexerer Eingaben wie zum Beispiel der Tabellen (zum Beispiel für Atom mit markdown-table-editor)
- Visualisierungsmethoden, die schon bei der Eingabe unterstützen
- Rechtschreibprüfung (!)

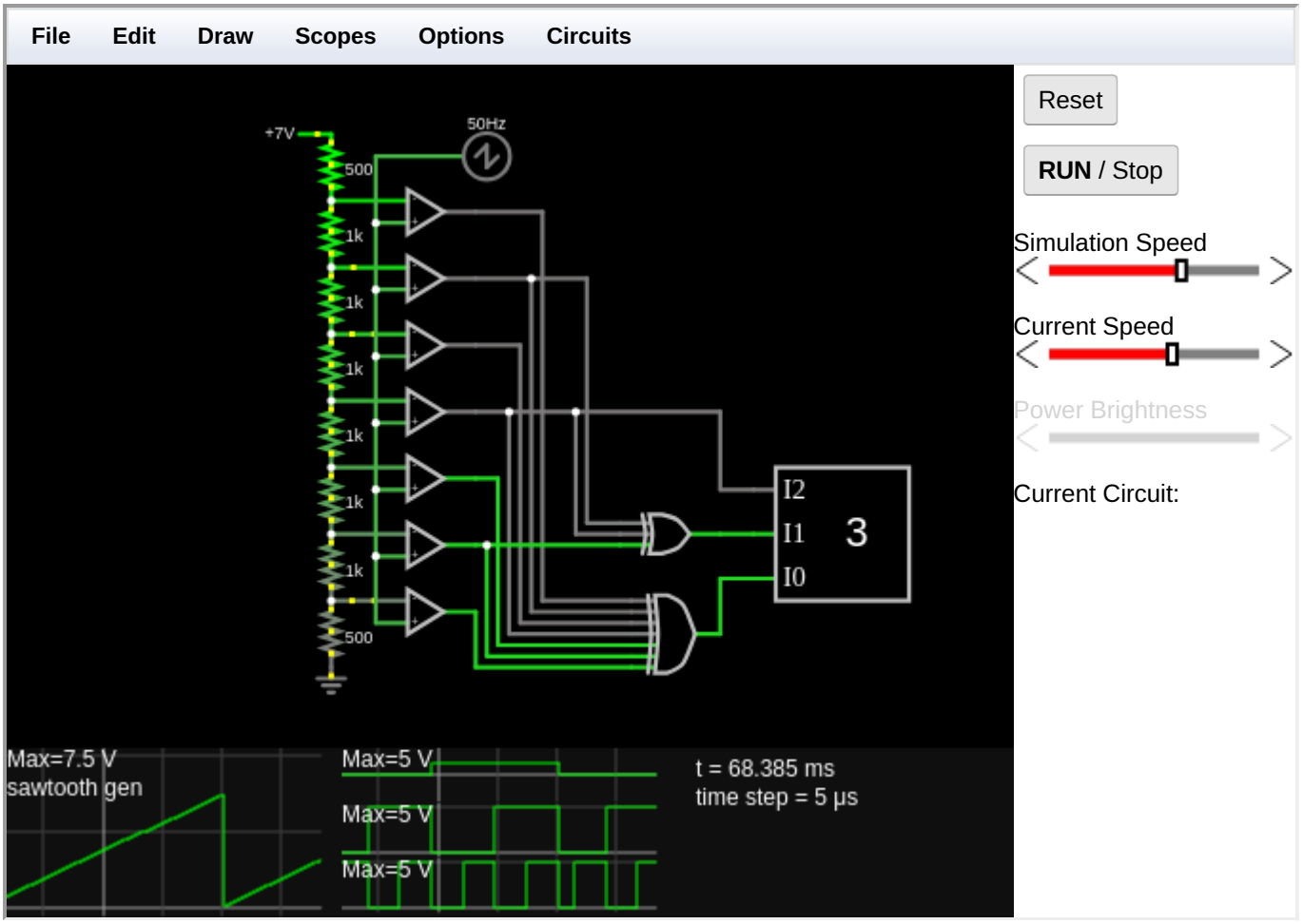
Allerdings vermisst Markdown trotz seiner Einfachheit einige Features, die für die Anwendbarkeit in der (Informatik-)Lehre sprechen:

- Ausführbarer Code
- Möglichkeiten zur Visualisierung
- Quizze, Tests und Aufgaben
- Spezifische Tools für die Modellierung, Simulationen etc.

ArduinoSimulator.ino

```
1  void thing(char i) {
2    switch(i) {
3    case 0: Serial.println("a pear"); break;
4    case 1: Serial.println("an apple"); break;
5    case 2: Serial.println("an elephant"); break;
6    case 3: Serial.println("an arduino"); break;
7    }
8  }
9
10 void setup() {
11   Serial.println("Hello stuff.");
12
13   for (int i=0; i<4; i++) {
14     Serial.print("here's ");
15     thing(random(4));
16   }
17 }
18
19 void loop() {
20 }
```

```
Sketch uses 2070 bytes (6%) of program storage space. Maximum is 32256
bytes.
Global variables use 252 bytes (12%) of dynamic memory, leaving 1796
bytes for local variables. Maximum is 2048 bytes.


Hello stuff.
here's an arduino
here's an apple
here's an apple
here's an elephant
Sketch uses 2070 bytes (6%) of program storage space. Maximum is 32256
bytes.
Global variables use 252 bytes (12%) of dynamic memory, leaving 1796
bytes for local variables. Maximum is 2048 bytes.


Hello stuff.
here's an arduino
here's an apple
here's an apple
here's an elephant
```



https://www.falstad.com/circuit/circuitjs.html?ctz=CQAgzCAMB0l3BWEA2aB2ATAFi2TXIBODMARjUJAQA4Q0rIqBTAWlNICgBDEUyLFAPLJBvRkkZ94cFNCyEFipUqixk0uBlKF1yMGGpCN3MUNIZeaEewt8qUMRqpzlrhfRiR1GrTq-7DR3gTPiF1S2twuwkg+BAsFzdlWk9vaV9dAKNpEP4QDEhaYXzCsXtJDUYwRKTFag81Soz-A2zgnlD8hGsrLutxBylpfJrahVUvJu1M1tjIXIEwPOKlo3K5yVGxkVSpvz1ZobgF8ARbXrAzspijmRh5MZVdn2mWwNuOACc6C1pMKh2DHm33+3R+pkGwRBtjy-1IVwqx2hvCu-wKjERwPB6OxWBSQS+uL+FgwOkhSOxZP+YBw5Kx1Np1OIDgQxwASuDaGBfg5GAQ6LzVAgOAB3cFgzpg+ZiuHLOUCaXg+EwoQI0XYmSdHGKtF4iHYFLqtFkrVknUWGlGRa0xWdWVCApQdWdcwVMKGsUu5BFPJaHbO32kfoCDDIBUBkNhiFkCy2vIx6PhOMCe35cMcrVCQLYBwCVngdBIapoGIwYUAc3BYGZ-wIGKd8JD1C53tOFtbpAF4mdFyuK2ZtouzIwzfAA57IlWvECU9tM+WM8Cc8W2dHs0HIgwEuHUudO+szLw-s9h96I65vQ3JVoOIKHpKjDvD-ArcVOLArfPL-vxSDj9Kf5OmKT6AU+H4YuqYHhF+4FAREvDmPkQYIbGkHITYSGTmqwHIZcJJrth8FRhgOZRm+pEhjmlpwSR1pNlyNocI2+TUI+ZJ4fkJrgCyzrzk2VR5G+rGcZueqkseLGPqOW6bqOb5iWSBrXjRYmlLepRvmCX4yShNESohfq6fJN6lBxT6KhxH5clc3L3mZplXKG4ZYNo4BnlGWBaWR4DqmZUZHvkZG+Y57FnmawUWiObmTlF8wAPbgFUsbxEQPoTPAhBWN0W4OBaVQcAlyBFslBCELQCCwHANKKGgDxlSklWQJ2nYWEVPmFTFuapeUjVEFlyA5S1KCTgVKD0NyXVlT10iZUVA1IENyDjRwQA

Eine Reihe von Einführungsvideos findet sich unter Youtube. Die Dokumentation von LiaScript ist hier verlinkt

Trotz Simulation und Virtuellem ...

... braucht es aber auch immer etwas zum anfassen.

Blick hinter eine Arduino-Anwendung

Generelles Engagement

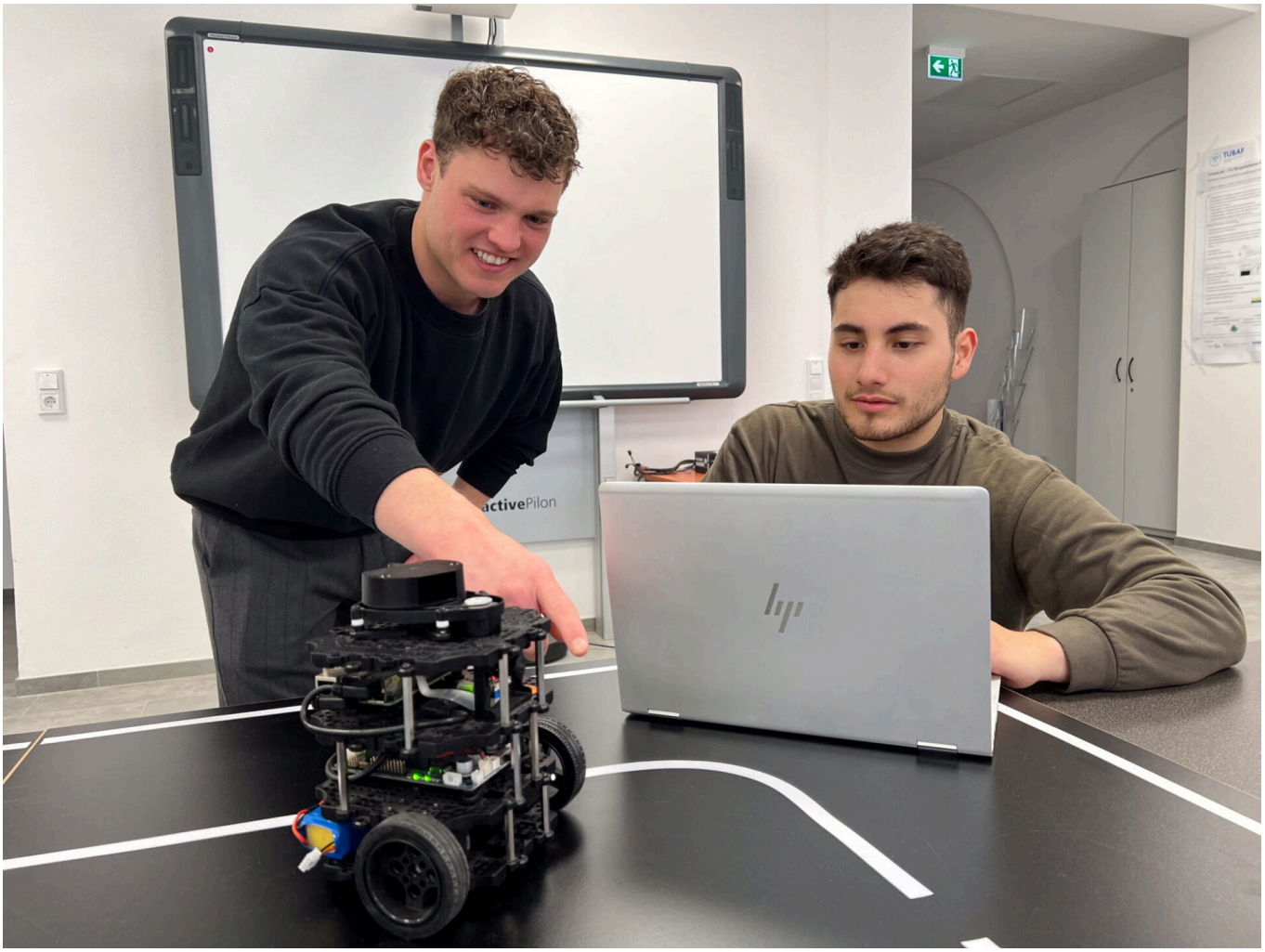
- Stellen Sie Fragen, seien Sie kommunikativ!
- Organisieren Sie sich in Arbeitsgruppen!
- Bringen Sie sich mit Implementierungen als Vortragende in die Veranstaltung ein.

Und wenn Sie dann immer noch programmieren wollen ...

Dann wartet das **racetech** Team auf Sie ... autonomes Fahren im Formula Student Kontext.

Schauen Sie im RoboLab vorbei!

Lion und Caio warten auf Sie ...



Hausaufgabe

- Legen Sie sich einen GitHub Account an ... und seien Sie der Erste, der einen Typo in den Unterlagen findet und diesen als Contributor korrigiert 🙂
- Organisieren Sie sich alle in einer Chatgruppe! Niemand verlässt den Raum, bevor er dort nicht Mitglied ist.