Informatikwerkstatt-MR Einführung

G. Kemnitz

Institut für Informatik, TU Clausthal (IW_Einf.pdf)
19. Oktober 2021

Informatikwerkstatt

Lernziele, Inhalt, Bewertung

Lernziele:

- Kennenlernen typische Arbeitsfelder und berufsbezogene Tätigkeiten,
- Grundfertigkeiten für die Software-Entwicklung:
  - Analyse von Problemstellungen,
  - Implementierung, Test, Fehlersuche,
  - Team-Arbeit, Präsentation der Lösungen, ...

Inhalt:

- Einführung in ein bis zwei Programmiersprachen und Programmierumgebungen,
- interaktive Rechnerübungen und Mini-Projekte,
- Abschlussprojekt und Projektpräsentation.

Bewertung:

- unbenoteter Leistungsnachweis

Informatikwerkstatt-MR

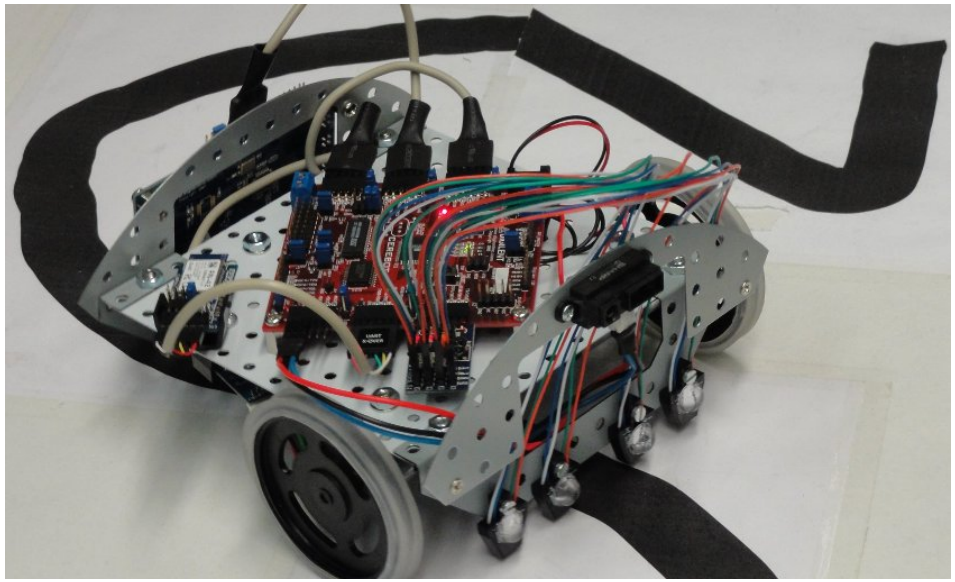
Lern- und Arbeitsziel

- Schrittweise Heranführung an die Entwicklung / Inbetriebnahme / Fehlersuche programmierter MR.
- Konfiguration eines Fahrzeugs mit einer Mikrorechnerbaugruppe und Ansteck- (PMod-) Modulen[1].

[1] Benachbarte Arbeitsplätze bilden eine 2er- bzw. 3er-Gruppe mit je einem Satz Hardware.

Abschlussprojekt

Demonstrator für ein autonomes Fahrzeug, programmiert nach einer selbst gewählten mit dem Betreuer abgesprochenen Spezifikation oder ein andere MR-Projekt.



Kann ich das?

- Kein Hexenwerk. Interesse wichtiger als Vorkenntnisse.

Zeitablauf, Leistungsnachweis

- erste Semesterhälfte: Mo. 15:30 bis 17:00 und Di. 17:15 bis 18:00 BBB-Vorlesung, anschließend bis 19:00 Übung
- ca. ab Mitte Dezember bis vorletzte VL-Woche nur Übung
- letzte Vorlesungswoche: Präsentation der Abschlussprojekte.
- Zu den Übungen ist corona-bedingt je Zweiergruppe alternierend nur eine Person vor Ort und die zweite über BBB verbunden.

Zu erbringende Leistungen:

- Vorführung gelöster Aufgaben. Schwierigkeit nach Vorkenntnissen in Absprache mit Betreuer.
- Abschlusspräsentation.
- Arbeitsumfang incl. Vorlesungsstunden 6 LP[2].

Bei Abwesenheit sind die zu erbringenden Leistungen nachzuholen.

[2] 1 LP = 30 h, 6 LP = 180 h, 14 Vorlesungswochen, ca. 13 Arbeitsstunden, 6 Präsenzzeit + 7 Eigenstudium pro Vorlesungswoche.

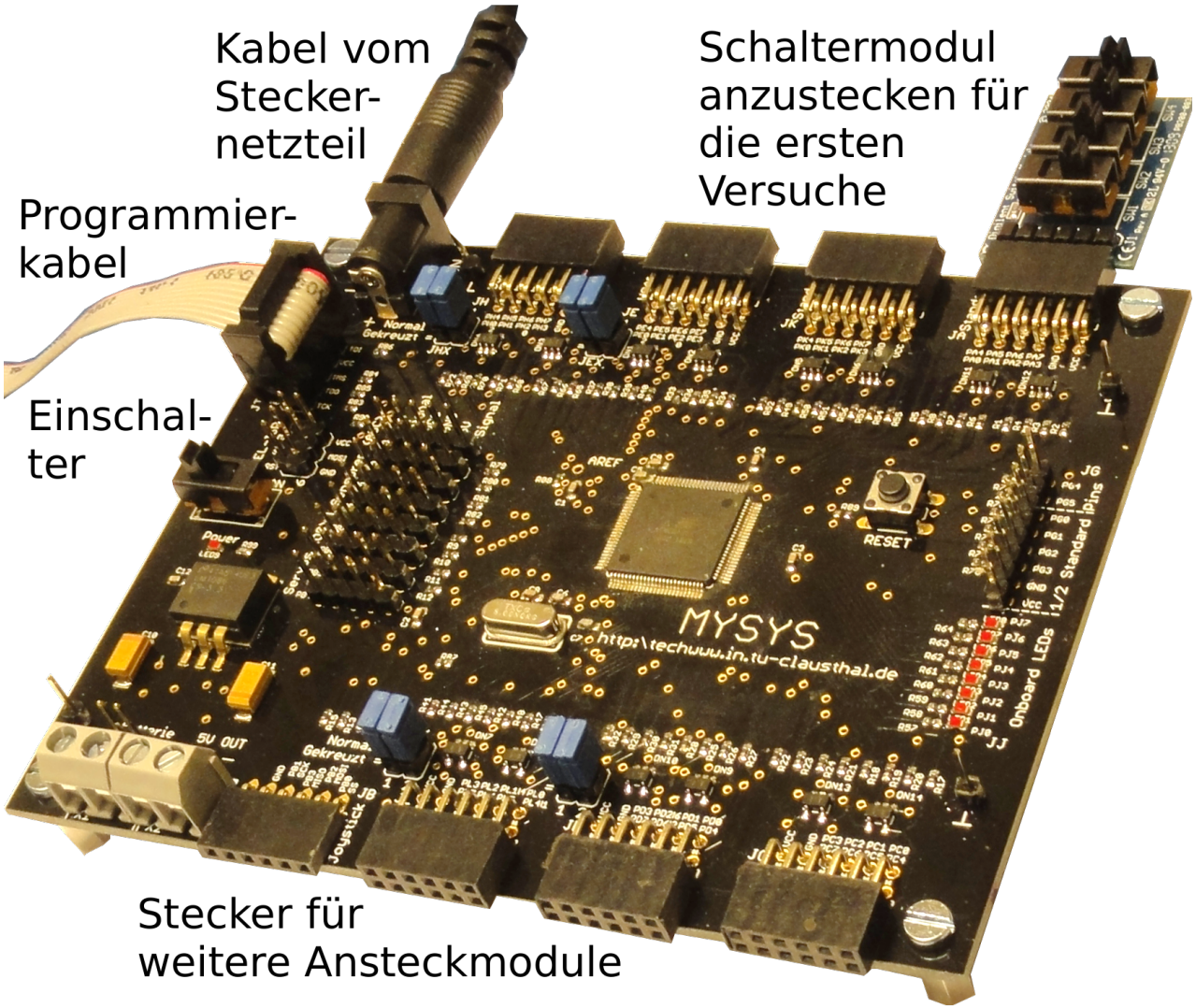
Arbeitsprogramm

- Kennenlernen der Entwickungsumgebung (Atmel-Studio, ...).
- Bitverarbeitung: Einfache Programme mit Eingabe über Schalter und Ausgabe an LEDs, Automaten, ...
- C-Programmierung, Datentypen, Modularisierung, Modultest, ...
- PC als Ein- und Ausgabe. Logikanalyse. Python für Programmtest vom PC aus. ...
- Ansteuerung weiterer Hardware-Einheiten (Ultraschallsensor, LC-Display, Timer, ...).
- Nebenläufigkeit: Treiber, Polling, Interrupts, Überwachung von Zeitabläufen.
- Motorsteuerung: Kennlinienbestimmung, Regelung, ...

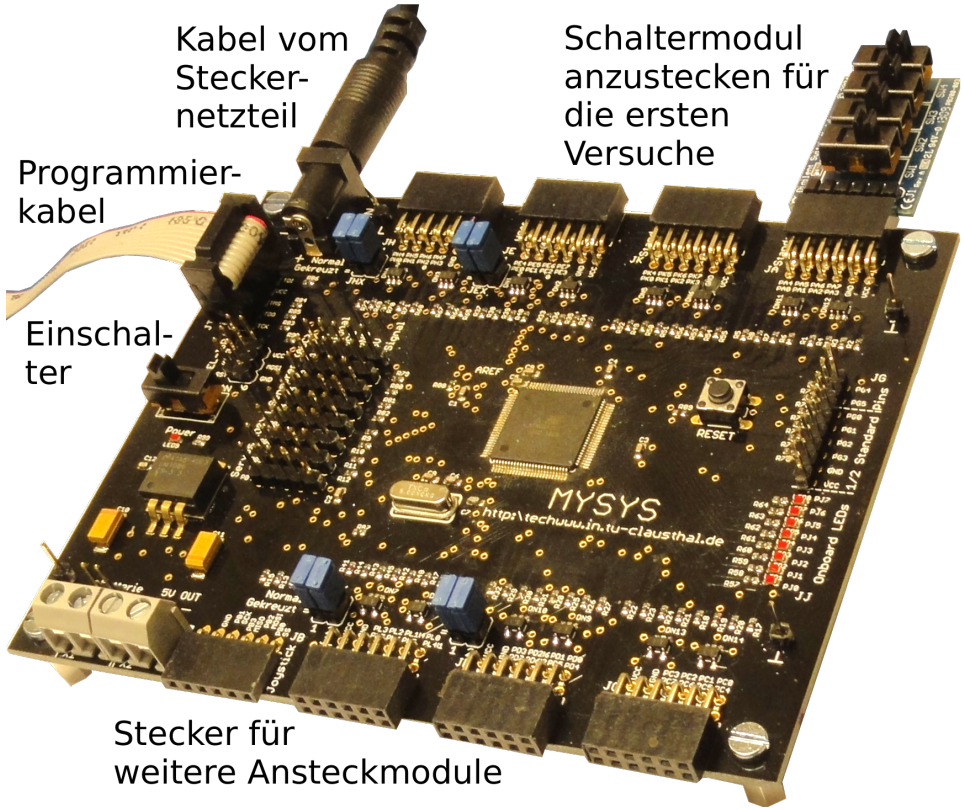
Zu jedem Thema werden zu Beginn funktionierende Programmbeispiele besprochen und gemeinsam getestet.

MR-Programmierung

Das Versuchsboard



Inbetriebnahme der Baugruppe



- Programmieradapter anstecken.
- Netzteil anstecken (Achtung, nur 5 V-Netzteile verwenden).
- Schaltermodul JA (Port A[3]) anstecken.
- Erst wenn Hardware fertig zusammengesteckt, einschalten.

[3] Oben angesteckt: SW1⇒JA.0, SW2⇒JA.1, SW3⇒JA.2, SW4⇒JA.3. Unten angesteckt: SW1⇒JA.4, SW2⇒JA.5, SW3⇒JA.6, SW4⇒JA.7.

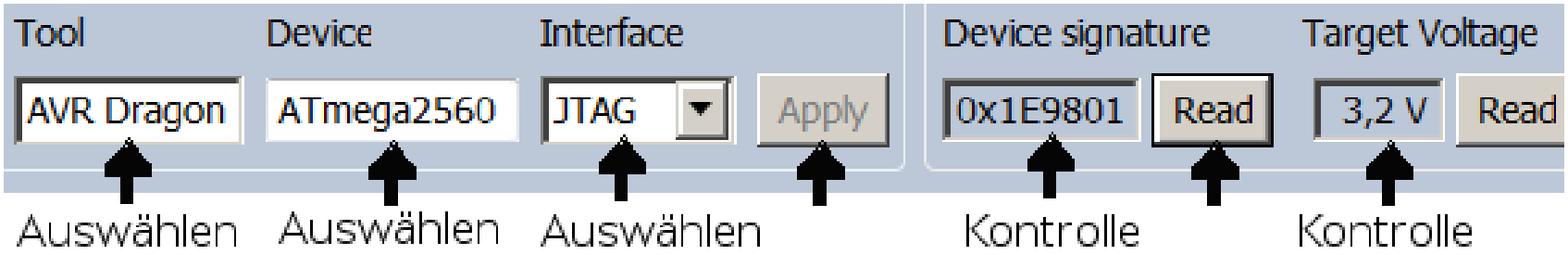
Verbindung mit dem Arbeitsplatzrechner

- Rechner unter Windows starten
- Web-Browser (Firefox) öffnen. Foliensatz zum Mitlesen öffnen:

  `techwww.in.tu-clausthal.de/site/Lehre`
  `/Informatikwerkstatt/`
- Atmel Studio starten  .
- Zur Kontrolle, dass der Prozessor richtig angeschlossen und vom System erkannt wird, in Atmel Studio

  `Tools > Device Programming`

  auswählen. Nachfolgende Kontrollen vornehmen:



Das erste Programm

Das erste Programm

Port A (Schalter) Eingang
Port J (LEDs) Ausgang
Wiederhole immer
lese Byte von Port A
schreibe Byte auf Port J

```
#include <avr/io.h>
int main(){

  DDRA = 0b00000000; // Port A (Schalter) Eingänge
  DDRJ = 0b11111111; // Port J (LEDs) Ausgänge
  uint8_t a;         // Variable, 8-Bit positiv
  while(1) {         // Endlosschleife
    a = PINA;        // Lesen der Schalterwerte
    PORTJ = a;       // Ausgabe auf die LED
  }
}
```

- Programmierprojekt anlegen.
- Programm eingeben und übersetzen.
- Programm laden. (Hardware zusammenstecken.)
- Programm testen.

Projekt einrichten

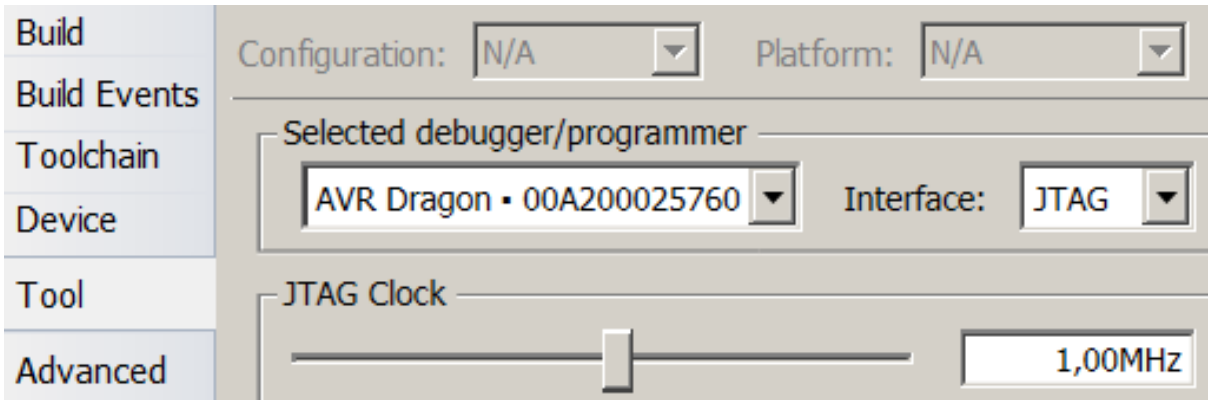
- Neues Projekt anlegen:

  `File > New > Project > GCC C Executable`

  Project Name: »bit_io1«. Location: C:\Informatikwerkstatt.
  Prozessortyp: Atmega2560.
- Programmier-Tool / Schnittstelle auswählen:

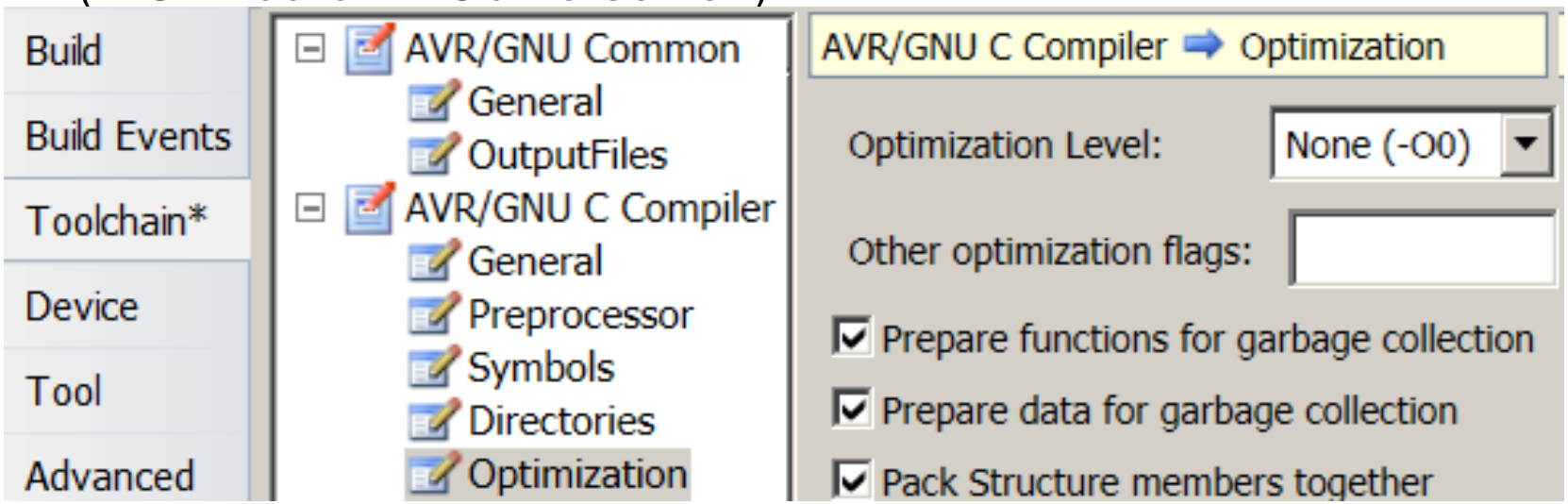
  `Project > bit_io1 Properties (Alt + F7) >`

  `Tool > AVR Dragon ..., JTAG`



4. Das erste Programm

- Unter Toolchain die Optimierung für den Übersetzer ausschalten:[4] (»-O1« durch »-O0« ersetzen)



- Zeilennummern einschalten:

```
Tools > Options > Text Editor > All languages
      > Line numbers√
```

- Einstellungen Speichern (Strg + S).

[4] Optimiert sonst für das Ein-/Ausgabeverhalten entbehrliche Schritte weg, z.B. das Zwischenspeichern von Variablenwerten im Datenspeicher und Warteschleifen. Optimierte Programme lassen sich nur eingeschränkt im Schrittbetrieb auf Quellcodeniveau testen.

Programm eingeben

- Automatisch erzeugten Programmrahmen vervollständigen[5].

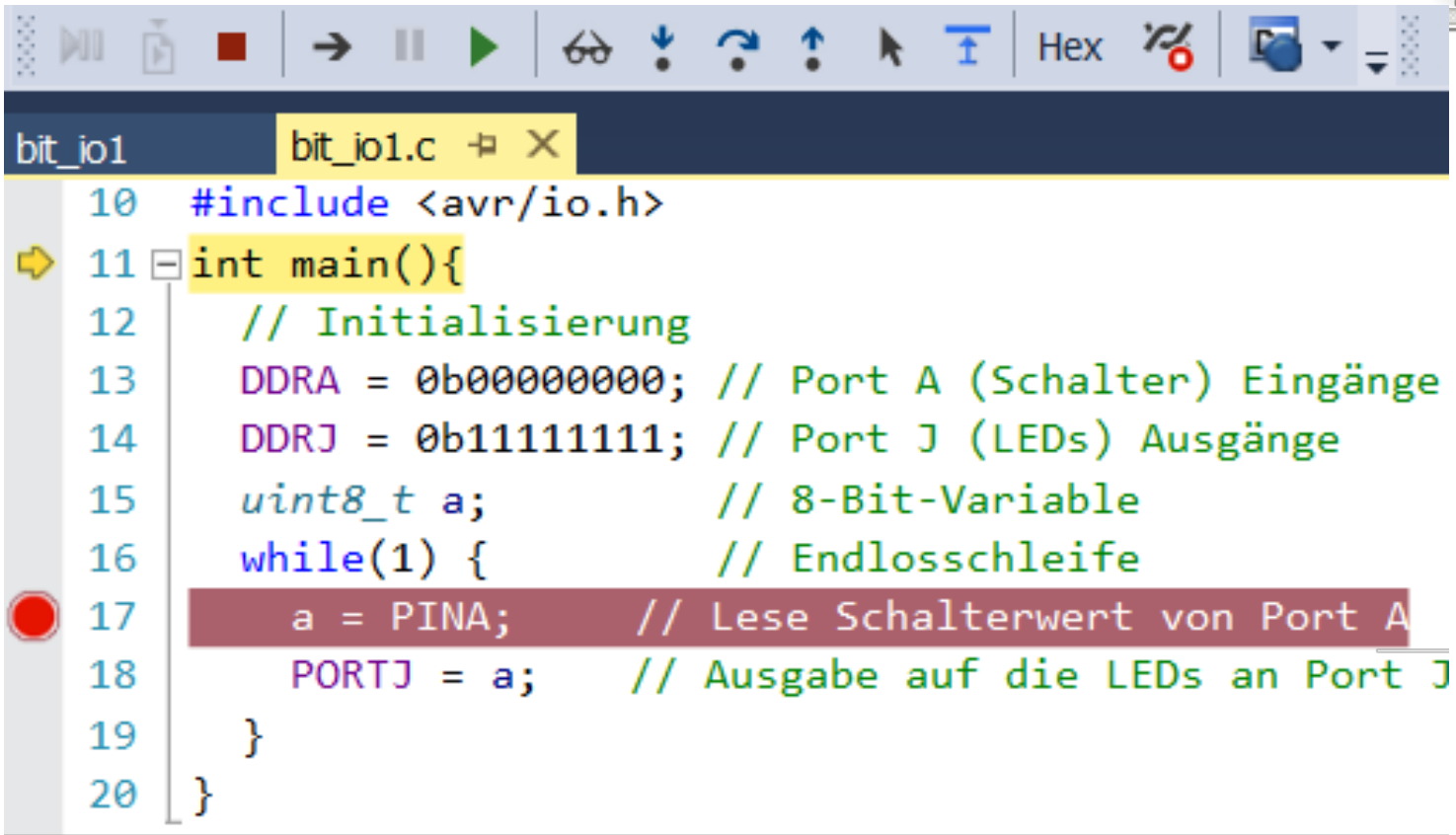
```
#include <avr/io.h>
int main(){
  DDRA = 0b00000000; // Port A (Schalter) Eingänge
  DDRJ = 0b11111111; // Port J (LEDs) Ausgänge
  uint8_t a;         // Variable, 8-Bit positiv
  while(1) {         // Endlosschleife
    a = PINA;        // Lesen der Schalterwerte
    PORTJ = a;       // Ausgabe auf die LED
  }
}
```

- Speichern.
- Debugger starten: 

Debug > Start Debugging and Break (Alt+F5).

[5] Das Beispielprogramm befindet sich mit im zip-File auf der Webseite.

Debugger-Ansicht



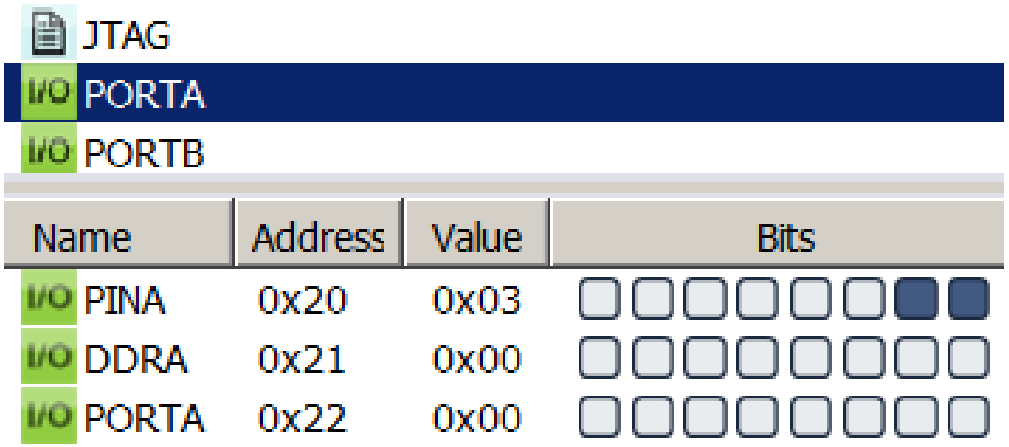
Nächste auszuführende Anweisung.
Unterbrechungspunkt (Mouse-Click grauer Rand davor).

Schritt abarbeiten und halten.

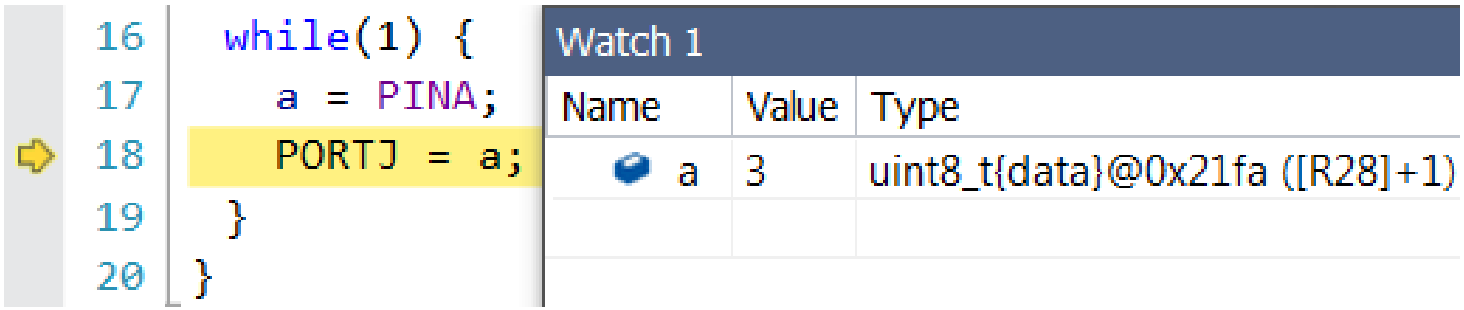
Fortsetzen bis zum nächsten Unterbrechungspunkt.

Beobachtungsfenster öffnen

`Debug > Windows > IO`



`Debug > Windows > Watch > Watch1`



Programm Testen

Schrittbetrieb:

- Schritt abarbeiten und halten ( ).
- Werte in »IO« und »Watch 1« kontrollieren.

Test mit Unterbrechungspunkt:

- Unterbrechungspunkt setzen[6].
- Start/Programmfortsetzung mit .
- Werte in »IO« und »Watch 1« kontrollieren.
- Schaltereingabe ändern.

Test ohne Unterbrechung:

- Unterbrechungspunkt löschen.
- Start/Programmfortsetzung mit .
- Schaltereingabe ändern und LED-Ausgabe kontrollieren.

[6] Mouse-Click auf den grauen Rand vor der Anweisung

Python

Python

Mikrorechner haben weder Tastatur, noch Bildschirme, noch ein Dateisystem, ... Bedienung und Test genau wie Programmierung über eine serielle Verbindung vom PC aus. Python ist eine gute Programmiersprache für die Entwicklung von Demonstratoren und Testrahmen:

- viel weniger Anweisungen als C für dieselbe Funktion
- viel einfacher zu testen, ...

Python hier im Kurs:

- Steuerung und Test der Fahrzeuge.
- Graphische Darstellung der Sensor- und Bewegungsdaten.

Ausprobieren von Programmanweisungen

Python eignet sich u.a. deshalb gut zum Programmierenlernen und für Test-Scripte, weil sich Programmzeilen einzeln auf der Konsole testen lassen. Start der Programmierkonsole »Idle« von

```
> Start > Python 3.5 > IDLE (Python 3.5 ...)
```

Ausgabefunktion:

```
print(<Obj>{,Obj})
```

Ausprobieren von Anweisungen:

```
>> s = 'Hallo'
>> print(3, 5==7, s) #Zahl, Wahrheitswert, Text
>> 3 False Hallo
>> print(type(s), len(s))
>> <type 'str'> 5
```

Textverarbeitung

Beim Testen sind gut lesbare Textdarstellungen wichtig. :

- unformatierte Textkonvertierung: str(<OBj>)
- formatierte Textkonvertierung: '<Formatstring>'%(<Wertetupel>)
- Verketten von Texten: Text1 + Text 2

Zum Ausprobieren auf der Konsole:

```
>> a=5; b=37.7; s='Hallo'
>> >print(str(a) + str(b) + s) #print(a, b, s)
>> 537.7Hallo
>> print('a=%2i , b=%4.2f : %s'%(a,b,s))
>> a= 5, b=37.70: Hallo
```

("%s" – Text (string); "%ni" – ganze Zahl (int), Darstellung mit mindestens n Zeichen; "%$n.m$f" – Gleitkommazahl (float), Darstellung mit mindestens n Zeichen und m Nachkommastellen).

Sequenzobjekte: Zeichenkette, Tupel

Zeichenkette: Sequenz von Zeichen bzw. Bytes. Auswahl von Elementen und Teilzeichenketten.

```
>> s='Hallo Welt!'
>> print('s[3]="%s", s[4:9]="%s"'%(s[3],s[4:9]))
>> s[3]="l", s[4:9]="o Wel"
```

Tupel: Sequenz beliebiger Objekte.

```
>> t = ('abc', 3, 4.2)
>> print(t, t[1], t[1:2])
>> (('abc', 3, 4.2), 3, (3,))
```

Iteratoren: Funktionen zur Generierung allg. beschreibbarer Folgen.

```
>> print(range(5))
# Erzeugt Folge: 0, 1, 2, 3, 4
>> print(range(-4,2,1))
# Erzeugt Folge: -4, -3, -2, -1, 0, 1
```

Wiederholschleifen

Wiederholschleifen iterieren über Sequenzobjekte (Zeichenketten, Tupel, Listen):

```
>> s='Hallo'
>> for c in s: # für alle Zeichen in s
...  print(c)   # Einrücktiefe erhöhen
...             # Einrücktiefe zurücksetzen
H
a
l
l
o
```

Auch hier nach »:« (für den Schleifenkörper) Einrücktiefe erhöhen und nach der Schleife zurücksetzen.

Zusätzliche Fallunterscheidung im Schleifenkörper:

```
>> i=0;
>> for c in s:
... if c=='l': # Einrücktiefe erhöhen
...  print(i)  # Einrücktiefe erhöhen
    i=i+1      # Einrücktiefe zurücksetzen
...            # Einrücktiefe zurücksetzen
2
3
```

Programme besser als Dateien eingeben und Testen. In der »Idle«:

`File > New File (Ctrl+N)`

Programmeingabe:

```
s='Hallo'; i=0;
for c in s:
 if c='l': # Einrücktiefe erhöhen
  print(i) # Einrücktiefe erhöhen
 i=i+1;    # Einrücktiefe zurücksetzen
```

Programm Speichern

`File > Save (Ctrl+S), C:\Informatikwerkstatt\test.py`

und starten

`Run > Run Modul (F5).`

Anwendungssysteme oder Mikrorechner?

- Anwendungssysteme / Python:
  - Python bietet ein schlankes System von Funktionsbausteinen, die zum Teil sehr komplex sind (Wörterbücher, Netzwerkkommunikation, Threads, graphische IO, ...).
  - Wenige Zeilen Python-Code ersetzen mehrere Seiten C-Code. Preis wesentlich schnellere und leistungsfähigere Rechner.
  - Schneller Prototyp-Entwurf, Testskripte, Skripte zum Trainieren von neuronalen Netzen auf Servern, ...
  - Informatikausbildung: Übungen High-Level-Konzepte, ...
- Mikrorechner / C:
  - Programmierung fast so, wie der Prozessor arbeitet.
  - Hilfreich für Verständnis der Funktionsweise von Rechnern und der internen Programmabarbeitung.
  - Hardware-nahe Programmierung, Entwicklung von Gerätetreibern, Betriebssystemen und Steuersystemen (z.B. für Maschinen und Fahrzeuge).
  - Vorbereitung auf praktische Arbeiten in der Technischen Informatik.
  - Wer gut programmiert, braucht keine 32-Bit MCU.