Informatikwerkstatt, Foliensatz 1
Einführung bis Bitverarbeitung

G. Kemnitz

Institut für Informatik, Technische Universität Clausthal

13. November 2013

Ziel der Veranstaltung

Die Veranstaltung »Informatikwerkstatt« soll

- ein Gefühl vermitteln, wie die praktische Arbeit eines Informatikers aussieht
- und helfen, später den praktischen Bezug zum Lehrstoff z.B. über Datenstrukturen, Automaten, verteilte Systemen, Rechnernetzen zu finden.

Inhalt des Foliensatzes

Zielstellung

Aufbau eines mikrorechnergesteuerten Fahrzeuges:

1. Phase: Vermittlung von Grundkenntnissen
   - Kennenlernen einen Mikrorechners
   - seiner Programmierumgebung in C und
   - wie man Hardwareeinheiten (Schalter, Leuchtdioden, Motoren, ...) darüber ansteuert.
2. Phase: Beispielprogramme
   - Einlesen von Sensordaten
   - Drehzahlsteuerung von Motoren
   - Datenaustausch mit dem PC
   - Datenverarbeitung auf dem PC mit Matlab
3. Phase: Spezifikation, Entwicklung, Montage und Test eines eigenen Fahrzeugs. Ergebnispräsentation

Zu erbringende Leistungen

Teilnahme an 80% der Veranstaltungen (12 Veranstaltungen).
Teilnahmebestätigung gibt es:

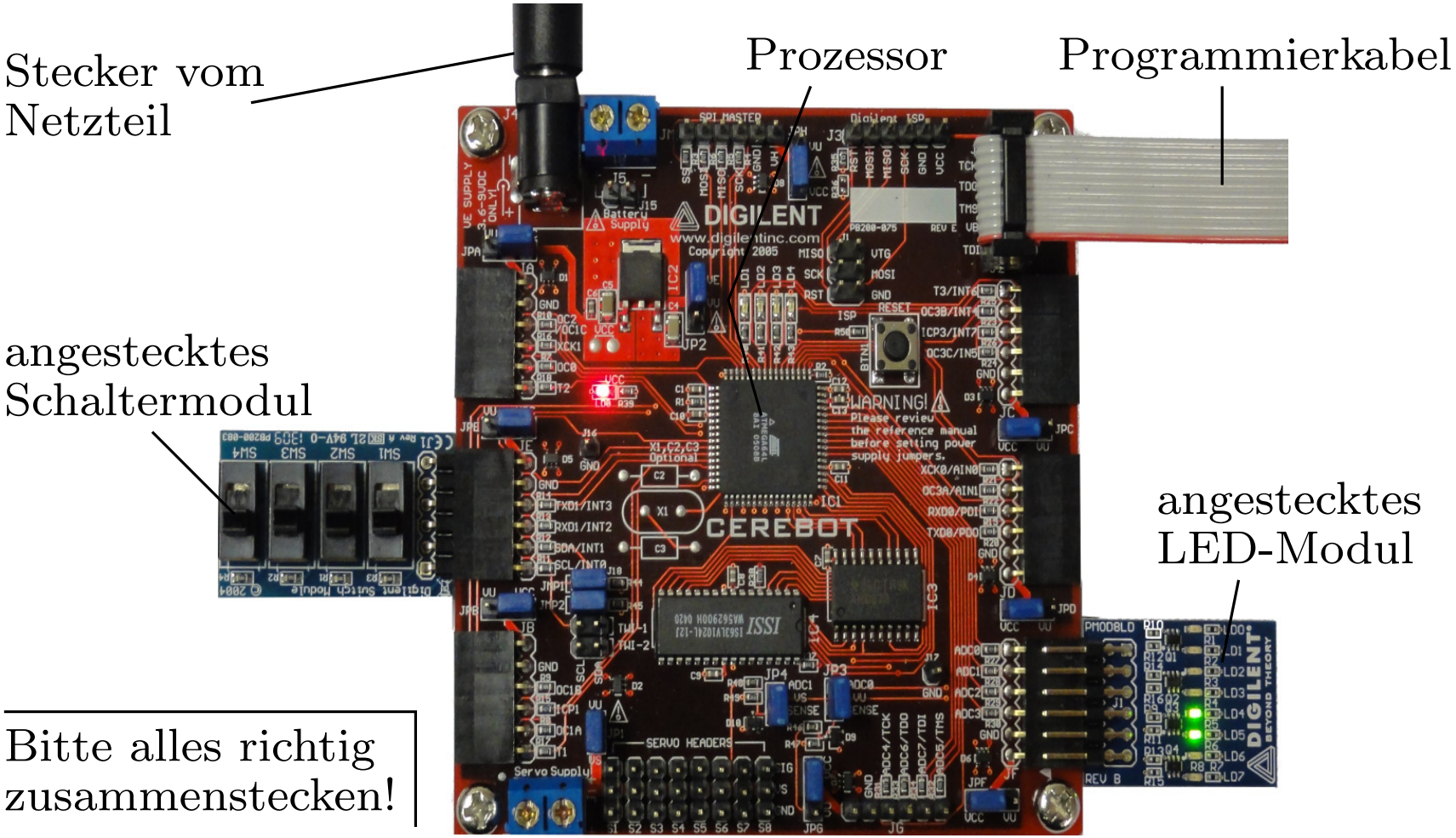
- Für die ersten beiden Veranstaltungen: Anwesenheit
- Abschlusveranstaltung: Präsentation einer Lösung
- Laborübungen: Anwesenheit und Bestätigung des Übungsleiters, das aktiv an der Lösung von Aufgabenstellungen gearbeitet wurde.

Je nach Vorkenntnis kann bei jedem Aufgabenkomplex zwischen unterschiedlich schweren Aufgaben gewählt werden.

Leistungsstarke Studierende können in Abstimmung mit dem Übungsleiter auch Hardware ausleihen und zu Hause arbeiten. Die Teilnahmebestätigung gibt es dann für die Vorführung der Lösungen für die abgesprochenen Aufgaben.

Kennenlernen Mikrorechner

Die Mikrorechnerbaugruppe »Cerebot«



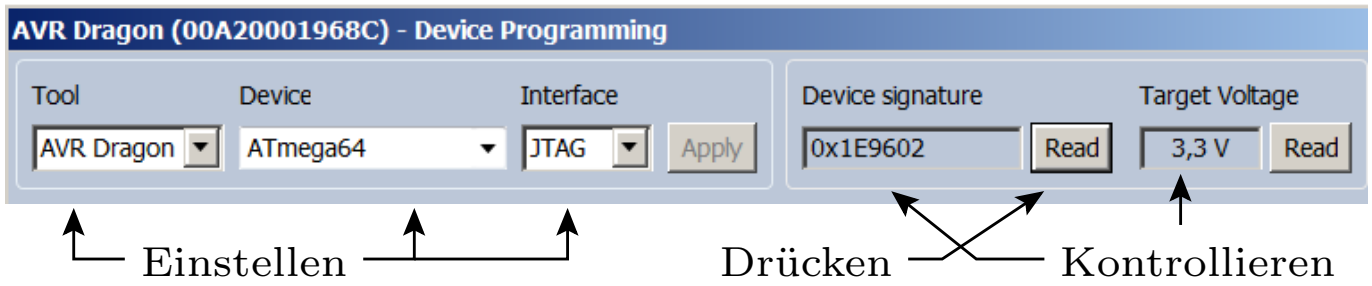
Verbindung mit dem PC

- Rechner unter Windos starten
- Web-Browser öffnen. Foliensatz zum Mitlesen öffnen:

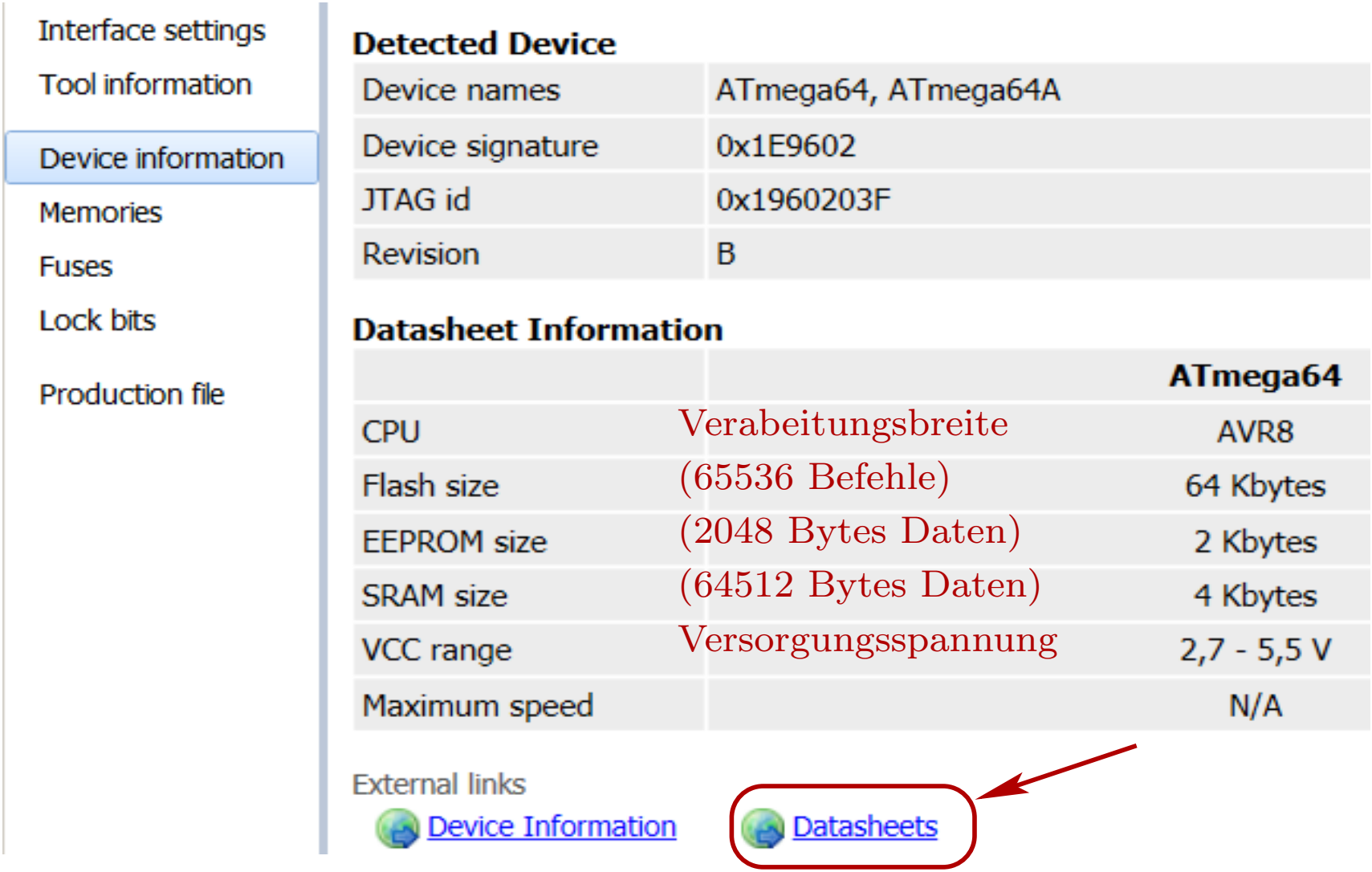
```
techwww.in.tu-clausthal.de/site/Lehre/Informatikwerkstatt/
```

- Atmel Studio starten 
- Zur Kontrolle, dass der Prozessor richtig angeschlossen und vom System erkannt wird, in Atmel Studio:

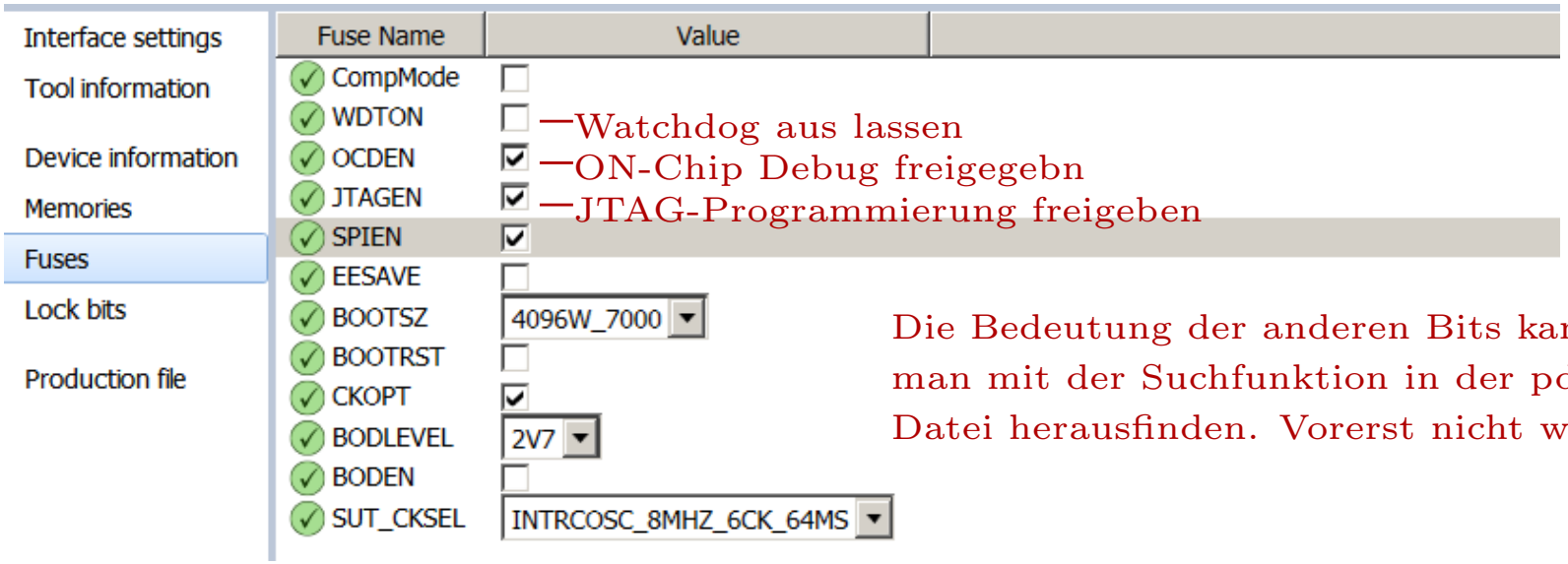
```
Tools > Device Programming
```



- Unter »Device Information« findet man auch das Datenblatt (Datasheet)



- Kontrolle der Sicherungsbits (Fuses, Grundeinstellungen)



Die Bedeutung der anderen Bits kann man mit der Suchfunktion in der pdf-Datei herausfinden. Vorerst nicht wichtig.

Hallo LED

Das erste Programm

Port D (Schalter) Eingang
Port F (LEDs) Ausgang
Wiederhole immer
lese Port E in einen Variable
schreibe Wert auf Port F

```
#include <avr/io.h> [1]
uint8_t tmp;  // 8-Bit-Variable
int main(){
 DDRD = 0;    // DDR... Bit(i)=0: Eingabebit
 DDRF = 0xff; // DDR... Bit(i)=1: Ausgabebit
 while(1) {   // wiederhole immer
  tmp = PIND; // Lesen von Port D
  // hier kann der Wert von tmp bearbeitet werden
  PORTF = tmp; //
 }
}
```

[1] Definiert Registeradressen und Datentypen wie uint8_t

Arbeitsschritte

- Projekt einrichten: Programmiersprachen, Art des Projekts, Arbeitsverzeichnis, Prozessortyp, Programmiergerät und Optionen für die Programmerstellung festlegen.
- Programm eingeben, übersetzen und Übersetzungsfehler beseitigen.
- Programm im Debugger starten. Schrittweise Abarbeitung und Kontrolle der Zwischenergebnisse.
- Programm normal starten und testen der Funktion in Echtzeit.

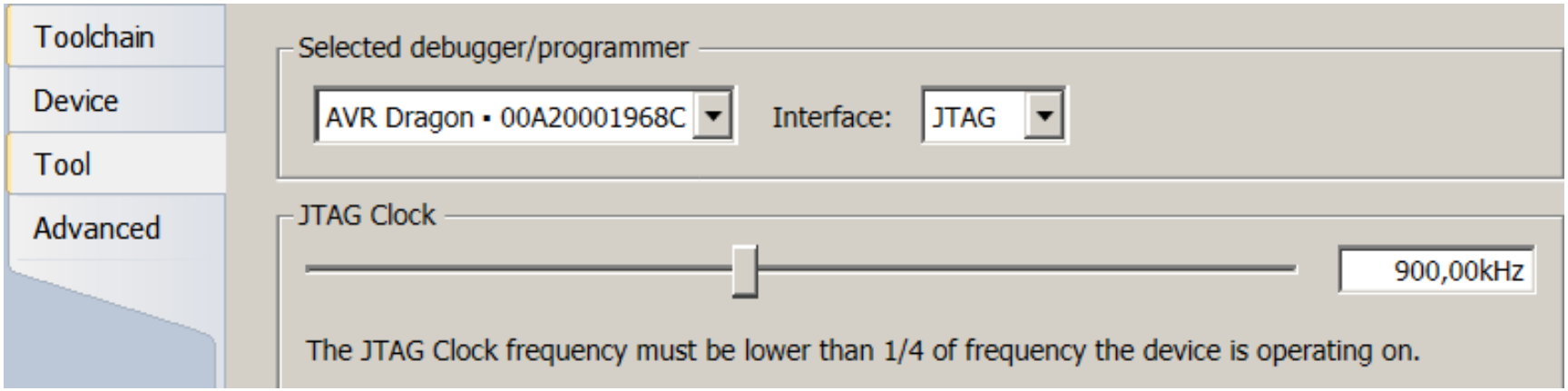
Projekt einrichten

- Neues Projekt anlegen:

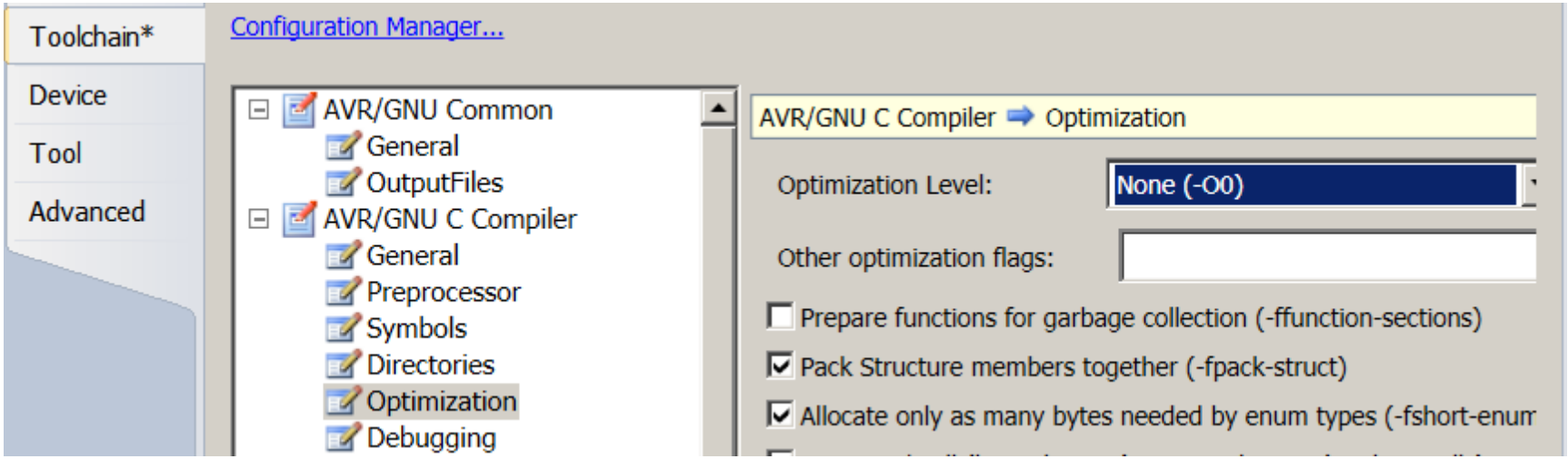
  `File > New > Project > GCC C Executable`

  Project Name: HalloLED. Location: H:\Werkstatt.
  Prozessortyp: Atmega64.
- Doppelklick auf 

.
- Unter Tool wie im nachfolgenden Bild Dragon und JTAG auswählen:



- Unter Toochain die Optimierung für den Übersetzer ausschalten[2] (O1 durch O0 ersetzen)



- Zeilennummern einschalten:

  `Tools > Options > Text Editor > All languages > General: Display on line numbers`√

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

Programm eingeben

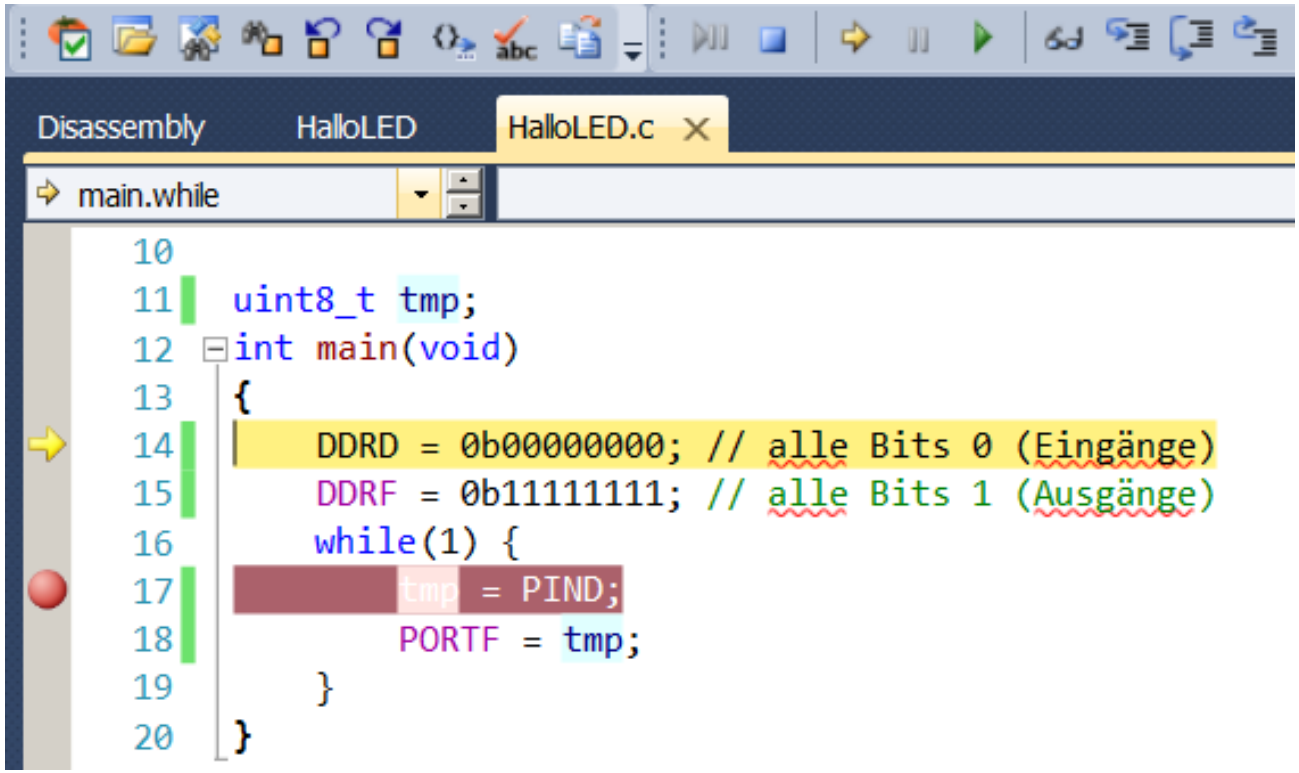
- Gegebenen Programmrahmen vervollständigen und Speichern:

```
 9  #include <avr/io.h>
10
11  uint8_t tmp;
12  int main(void)
13  {
14      DDRD = 0b00000000; // alle Bits 0 (Eingänge)
15      DDRF = 0b11111111; // alle Bits 1 (Ausgänge)
16      while(1)
17      {
18          //TODO:: Please write your application code
19          tmp = PIND;
20          PORTF = tmp;
21      }
22  }
```

Im Schrittbetrieb abarbeiten

- Debugger starten:

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

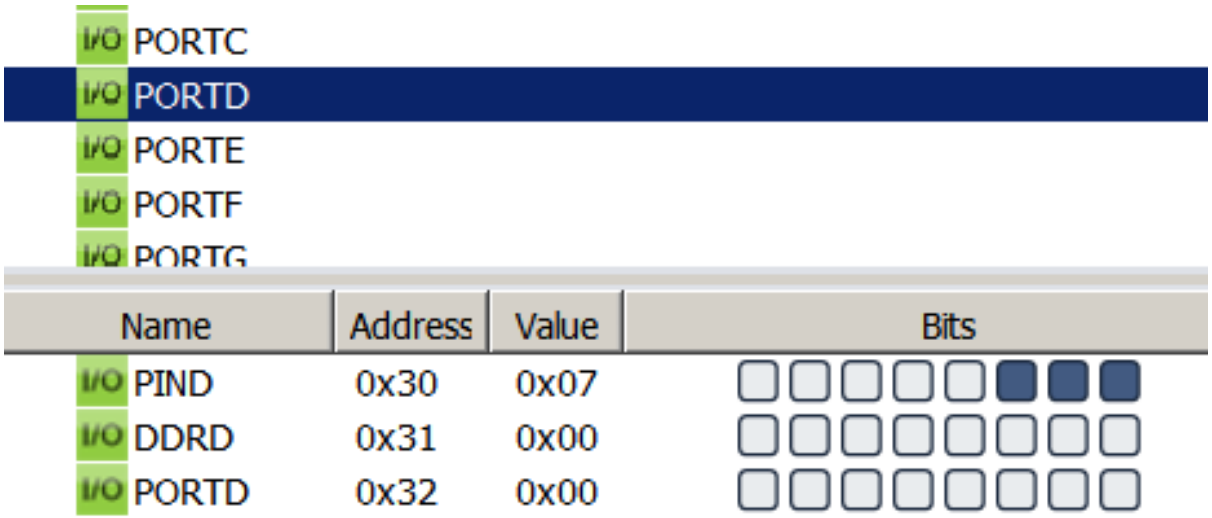


– nächste Anweisung; – Unterbrechungspunkt; – Schritt abarbeiten und halten; – Fortsetzen

Vor der Ausführung Beobachtungsfenster öffnen

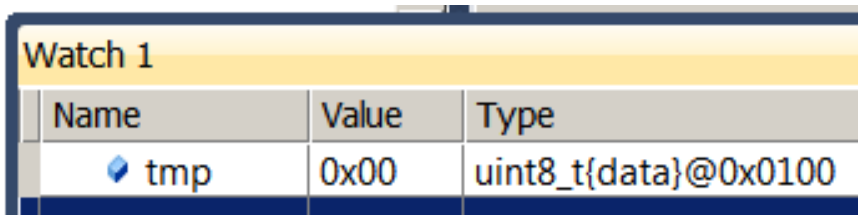
- Fenster zur Beobachtung der Ein- und Ausgabewerte:

  `Debug > Windows > IO View`



- Fenster zur Beobachtung der (globalen) Variablen:

  `Debug > Windows > Watch > Watch1`



Programm in Echtzeit abarbeiten

- Debug > Stop Debugging (Ctrl+Shift+F5) 
- Debug > Start without Debugging 
- Änderung der Schalterwerte und Kontrolle der LED-Ausgabe

Bits verarbeiten

Verarbeitung von Bits

- Mikrorechner verarbeiten vielfach einzelne Bits: Schalter, LED, Motor, ... ein/aus
- Die Bits werden in unserem Prozessor immer zu Bytes (8 Bit) zusammengefasst.

Verarbeitungsmöglichkeiten für Bitvektoren:

```
uint8_t a, b; // zwei 8-Bit-Variablen
```

- kopieren

```
a = b;
```

- nach rechts oder links verschieben

```
a = 0b10110111;
b = a >> 2; // b: 00101101
a = b << 3; // a: 01101000
```

- bitweise Negation

```
a = 0b10110111;
a = ~a; // a: 0b01001000
```

- bitweises UND (Ergebnis 1, wenn beide Operanden 1 sind)

```
a = 0b10011111 & 0b00111101; // a: 0b00011101
```

- bitweises ODER (Ergebnis 1, wenn mindests ein Operand 1 ist)

```
a = 0b10011111 | 0b00111101; // a: 0b10111111
```

- bitweises EXOR (Ergebnis 1, wenn genau ein Operand 1 ist)

```
a = 0b10011111 ^ 0b00111101; // a: 0b10100010
```

Beispiel: LE4 = SW1 UND SW2

Schalter / LED	SW1	SW2	LD4
Port, Bitnummenr	D0	D1	F0

Mögliche Lösung mit zwei 8-Bit-Variablen a und b:

```
...
while(1) {
 // Lesen aller Schalterwerte
 a = PIND;
 // nach b kopieren und alle Bits außer SW1 löschen
 b = a & 0b00000001;
 // in a alle Bits außer SW2 löschen
 // und Ergebnis in Bit 0 verschieben
 a = (a & 0b00000010) >>1;
 // bitweise UND verknüpfen und ausgeben
 PORTF = a & b;
}
```

Ein etwas anderes Programm mit derselben Funktion:

```
 9  #include <avr/io.h>
10
11  uint8_t a, b;
12  int main(void)
13  {
14      DDRD = 0b00000000; // alle Bits 0 (Eingänge)
15      DDRF = 0b11111111; // alle Bits 1 (Ausgänge)
16      while(1) {
17          a = PIND & 0b00000001;
18          b = PIND & 0b00000010;
19          b = b>>1;
20          PORTF = a & b;
21      }
22  }
```

- Wie funktioniert dieses Programm?

Aufgabe 1.1: Für Programmieranfänger

- Anlegen eines neuen Projektes mit dem Namen Logiktest und ansonsten denselben Einstellungen wie »HalloLED«.
- Eingabe des Beispielprogramms der Folie zuvor.
- Test im Schrittbetrieb.
- Test in Echtzeit.
- Veränderung des Programms. Andere Verarbeitung oder etwas andere Funktion.
- Ausprobieren, was die Veränderungen bewirken.

Aufgabe 1.2: Berechnung der Parität der 4 Schalter

Die Parität ist die bitweise Summe der Eingabevariablen (Schalter) unter Vernachlässigung des Übertrags:

SW(3)	SW(2)	SW(1)	SW(0)	LED(4)
0 (aus)	0 (aus)	0 (aus)	0 (aus)	0 (aus)
0 (aus)	0 (aus)	0 (aus)	1 (ein)	1 (ein)
0 (aus)	0 (aus)	1 (ein)	0 (aus)	1 (ein)
0 (aus)	0 (aus)	1 (ein)	1 (ein)	0 (aus)
...	...	...	...	...

Identisch mit:

```
LED(4) = SW(3) EXOR SW(2) EXOR SW(1) EXOR SW(0)
```

Bitte programmieren und testen!

Aufgabe 1.3: Logik für Tüftler

- LD4 soll leuchten, wenn mindestens einer der vier Schalter eingeschaltet ist.
- LD5 soll leuchten, wenn mindestens zwei der vier Schalter eingeschaltet sind.
- LD6 soll leuchten, wenn mindestens drei der vier Schalter eingeschaltet sind.
- LD7 soll leuchten, wenn alle vier Schalter eingeschaltet sind.

Weiterer Ports

Nutzung weiterer Ports

- Wenn man mit einem Adapterkabel den Stecker JF mit der oberen statt wie bisher mit der untereren Reihe verbindet, leuchten LD0 bis LD3 statt LD4 bis LD7.
- Man kann auch mit zwei Kabeln die Bits von zwei Ports auf die acht LEDs des Moduls ausgeben.
- Der Mikrorechner hat sieben 8-Bit-Ports (A bis G)
- Die Baugruppe hat sechs identische Stecker (JA bis JG und JM), auf denen jeweils vier Port-Bits herausgeführt sind. Zuordnung siehe CEREBOT-rm.pdf, Table 1 (pdf-Datei ist auf techwww oder mit google auffindbar).

Aufgabe 1.4: Bestimmung Stecker-Port-Zuordnung

Vervollständigen Sie unter Nutzung der Dokumentation folgende Zuordnungstabelle zwischen Steckern und Ports:

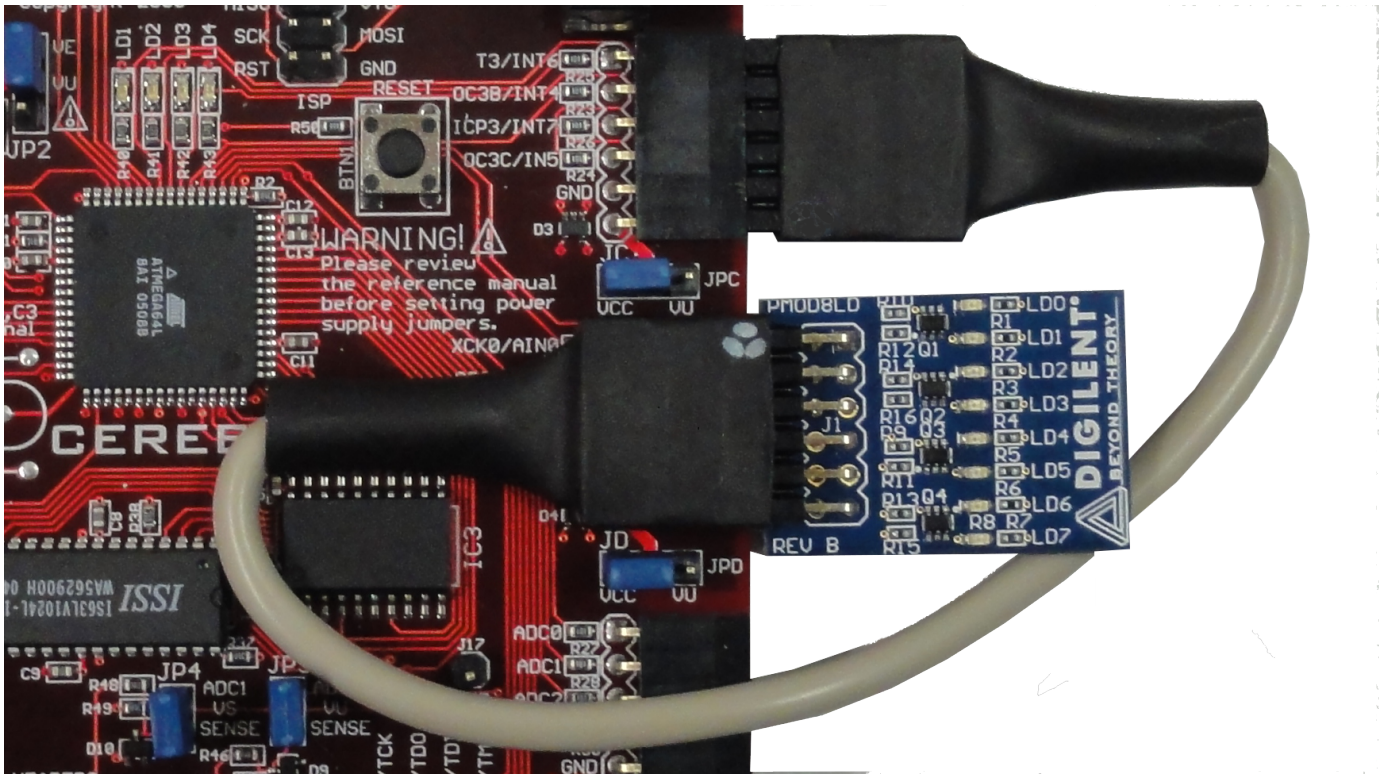
Stecker	Pin 1	Pin2	Pin3	Pin 4
JA				
JB				
JC				
JD				
JE	PD 0	PD 1	PD 2	PD 3
JF	PF 0	PF 1	PF 2	PF 3
JG				
JM				

Aufgabe 1.5: LED-Ausgabe auf der Cerebot-Baugruppe

- Schaltermoduls am Stecker JE lassen.
- Die Schalterwerte sollen auf den vier LEDs direkt über dem Mikrorechner ausgegeben werden.
- Umstecken des Schaltermoduls an JA und Änderung des Programms, so dass auch jetzt noch der Eingabewert von Schalter i auf Leuchtdiode LDi der Cerebot-Baugruppe ausgegeben wird.

Aufgabe 1.6: LED-Ausgabe über JC und JD

- Anstecken von JC an die obere und JD an die untere Anschlussreihe des LED-Moduls.
- Auf LD0 bis LD3 soll SW1 bis SW4 direkt und auf LD4 bis LD7 soll SW1 bis SW4 negiert ausgegeben werden.



Warteschleifen

Warteschleifen

- Ziel sei, dass die Leuchtdioden sichtbar blinken. Dazu muss zwischen Ein- und Ausschalten > 100ms Zeit vergehen.
- Der Prozessor arbeitet in 100 ms bis etwa eine halbe Millionen Maschinenbefehle bzw. viele Tausend C-Anweisungen ab. Lösung Warteschleife:

```
uint8_t a=0; // Ausgabewert
uint16_t b;  // Zähler WB: 0 bis 2^16 - 1 ≈ 65.000
...
while(1){
 PORTF = a;
 for (b = 0; b<65000; b++); // Warteschleife
 a++;
}
```

- Was tut das Programm?
- Warum lässt es sich nicht im Schrittbetrieb testen?

Aufgabe 1.7: Wandernder Leuchtpunkt

Entwickeln Sie ein Programm, bei dem eine Leuchtpunkt auf den 4 LEDs auf der Cerebot-Baugruppe hin und her wandert.

Aufgabe 1.8: Wandernder Leuchtpunkt für Fortgeschrittene

- LED-Modul so anstecken und Programm so ändern, dass der Leuchtpunkt über alle 8 LEDs des LED-Moduls hin und her wandert.
- Einbeziehung des Schaltermoduls zur Steuerung des Lichteffektes.