Informatikwerkstatt, Foliensatz 1
Einführung bis Bitverarbeitung

G. Kemnitz

Institut für Informatik, TU Clausthal (IW-F1)

16. November 2015

Inhalt Foliensatz 1

Einleitung

Ziel und Arbeitsprogramm

Ziel und Methodik der Informatikwerkstatt

Fakt 1

Einführung in die praktische Arbeit eines Informatikers.

Thematischer Schwerpunkt in der Gruppe Mikrorechner:

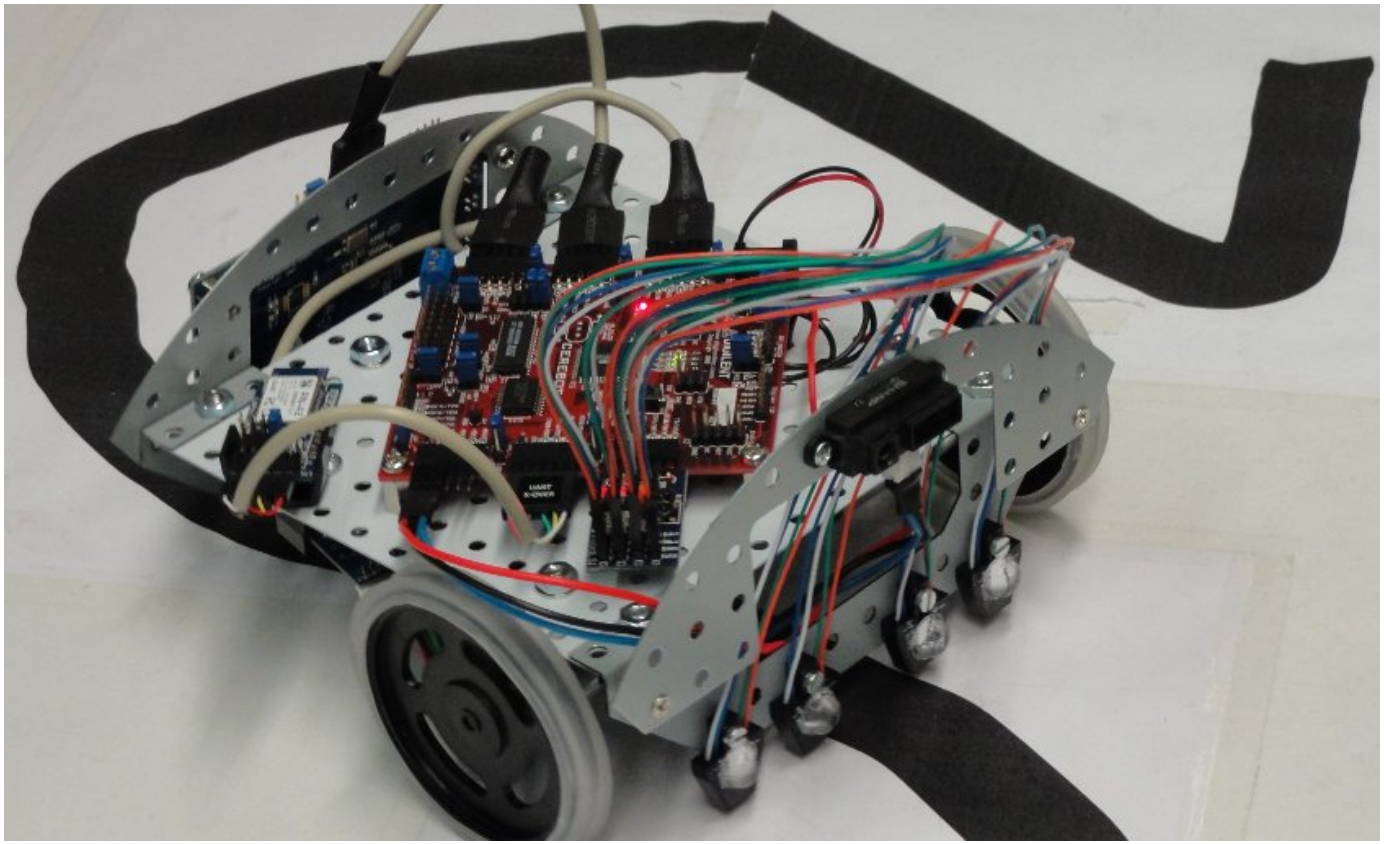
- Programmieren, Programmierumgebung,
- Beschreibung von Abläufen (sequentiell, nebenläufig),
- Testen (»printf-Methode, Debugger, Logikanalysator, Testscripte, prüfgerechter Entwurf, Echtzeittest für Motorreglung und Kommunikation, ...)[1],
- Teamarbeit bei der Aufgabenformulierung und Fehlersuche.

Es werden Lehrinhalte des späteren Studiums mit praktischen Beispielen vorweggenommen, z.B. die Bitverarbeitung mit LEDs und Schaltern, Automaten zur Sensordatendecodierung, ...

[1] Themen, die im späteren Studium zu kurz behandelt werden.

Gegenstand der Arbeit

Aufbau und Programmierung eines Modellfahrzeugs mit einer Mikrorechnersteuerung, Motoren, Sensoren, ...



Organisation und Bewertung

- Wöchentlich Mo. 15-19 Uhr. Vorlesungsteil mit Rechnerübungen, anschließend betreute Übungszeit mit Aufgaben.
- Teilnehmer finden sich in Zweier- und Dreiergruppen mit ähnlichen Vorkenntnissen zusammenzufinden.
- Für eine erfolgreiche Teilnahme ist eine angemessene Anzahl von gelösten Programmieraufgaben und einer selbst gewählten Abschlussaufgabe erforderlich.
- Mikrorechnerprogrammierung in C; Steuer- und Testscripte auf PC mit HTerm und Python-XY; Test und Steuerung über Handy ist möglich[2].
- In Zweifelsfällen, z.B. bei mangelnder Anwesenheit (auch krankheitsbedingt) gibt es die Möglichkeit einer mündlichen Kenntnisprüfung am Rechner.

[2] Wer es nutzen will, muss selbst herauszufinden, wie es geht.

Wer sollte teilnehmen?

- Unbenotete Veranstaltung. Kann von Teilnehmern mit sehr unterschiedlichen Vorkenntnissen erfolgreich belegt werden.
- Es gibt für jeden Themenblock Aufgaben in einem weitgestaffelten Schwierigkeitsbereich. Die einfachsten sind für Programmieranfänger. Die schwierigsten sind auch für Mikrorechner-Freaks eine Herausforderung.
- Programmieranfänger finden in der Veranstaltung einen praktischen Zugang zu Grundlagen der Informatik: Arbeit mit Programmierumgebungen, Bits, Bytes, Logik, ...
- Freaks erhalten einen praktischen Vorgeschmack auf fortgeschrittene Themen: Automaten, Softwarearchitekturen, Treiber, Regelkreise, Übertragungsprotokolle, ...
- Empfohlen für Studierende, die später technisch orientierte Software entwicken, testen, einsetzen, ... wollen.

Arbeitsprogramm

- Kennenlernen der Entwickungsumgebung (Atmel-Studio, ...).
- Bitverarbeitung: Einfache Programme mit Eingabe über Schalter und Ausgabe an LEDs.
- C-Programmierung, Modularisierung, Simulation, ...
- PC als Ein- und Ausgabe. Programmtest vom PC aus.
- Ansteuerung weiterer Hardware-Einheiten (Ultraschallsensor, LC-Display, Timer, ...).
- Nebenläufigkeit: Treiber, Polling, Interrupts, Überwachung von Zeitabläufen.
- Motorsteuerung: Kennlinienbestimmung, Regelung, ...
- Entwicklung und Aufbau eines Fahrzeugs mit selbst definierter Zielfunktion.

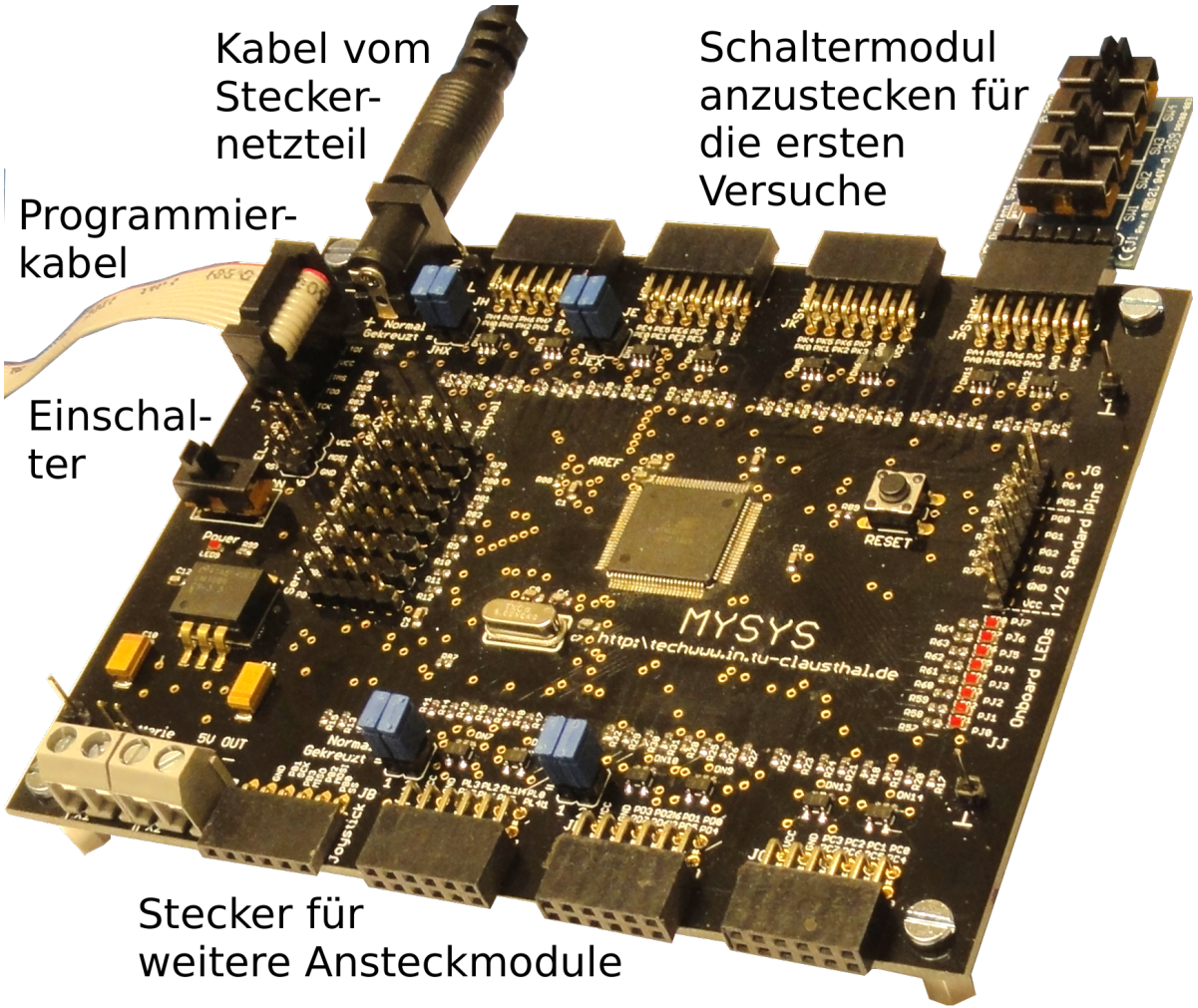
Hinzu kommen zentrale Angebote für alle Gruppen.

Ein wenig Basteln gehört dazu

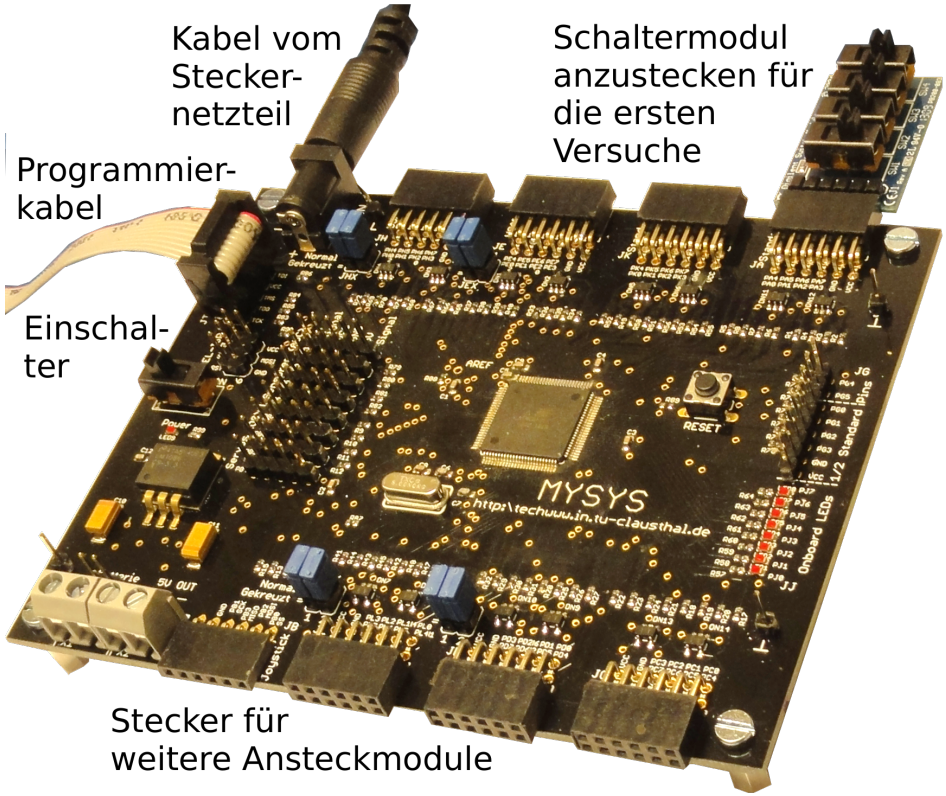
Die Gruppe Mikrorechner trifft sich im Anschluss an diese Einführungveranstaltung im Labor Technische Informatik im Institut für Mathematik, Raum 010 (rechter Eingang, Keller).

Entwicklungsumgebung

Das Versuchsboard



Inbetriebnahme der Baugruppe



- Anstecken des Programmieradapters.
- Anstecken des Netzteils (Achtung, nur 5 V-Netzteile verwenden).
- Anstecken des Schaltermoduls an JA (Port A[3]).
- Einschalten, erst wenn die Hardware fertig zusammengesteckt ist.

[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.

Kommunikationskontrolle

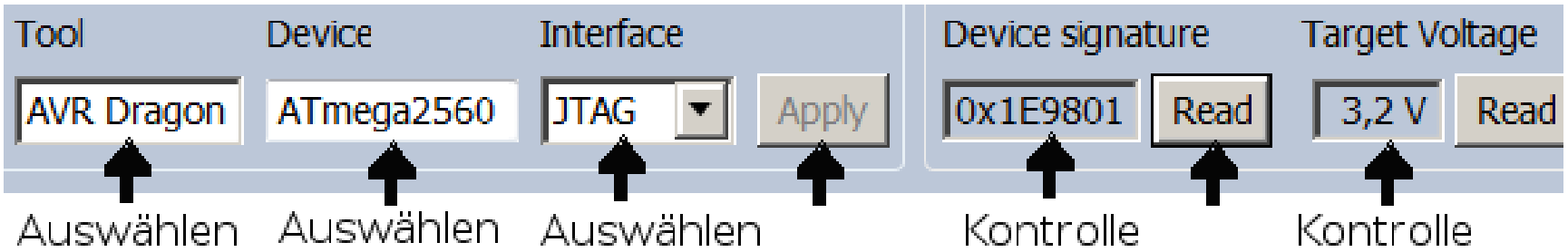
- Rechner unter Windows starten
- Web-Browser (Firefox) öffnen. Foliensatz zum Mitlesen öffnen:

  `techwww.in.tu-clausthal.de/site/Lehre/Informatikwerkstatt_2015/`

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

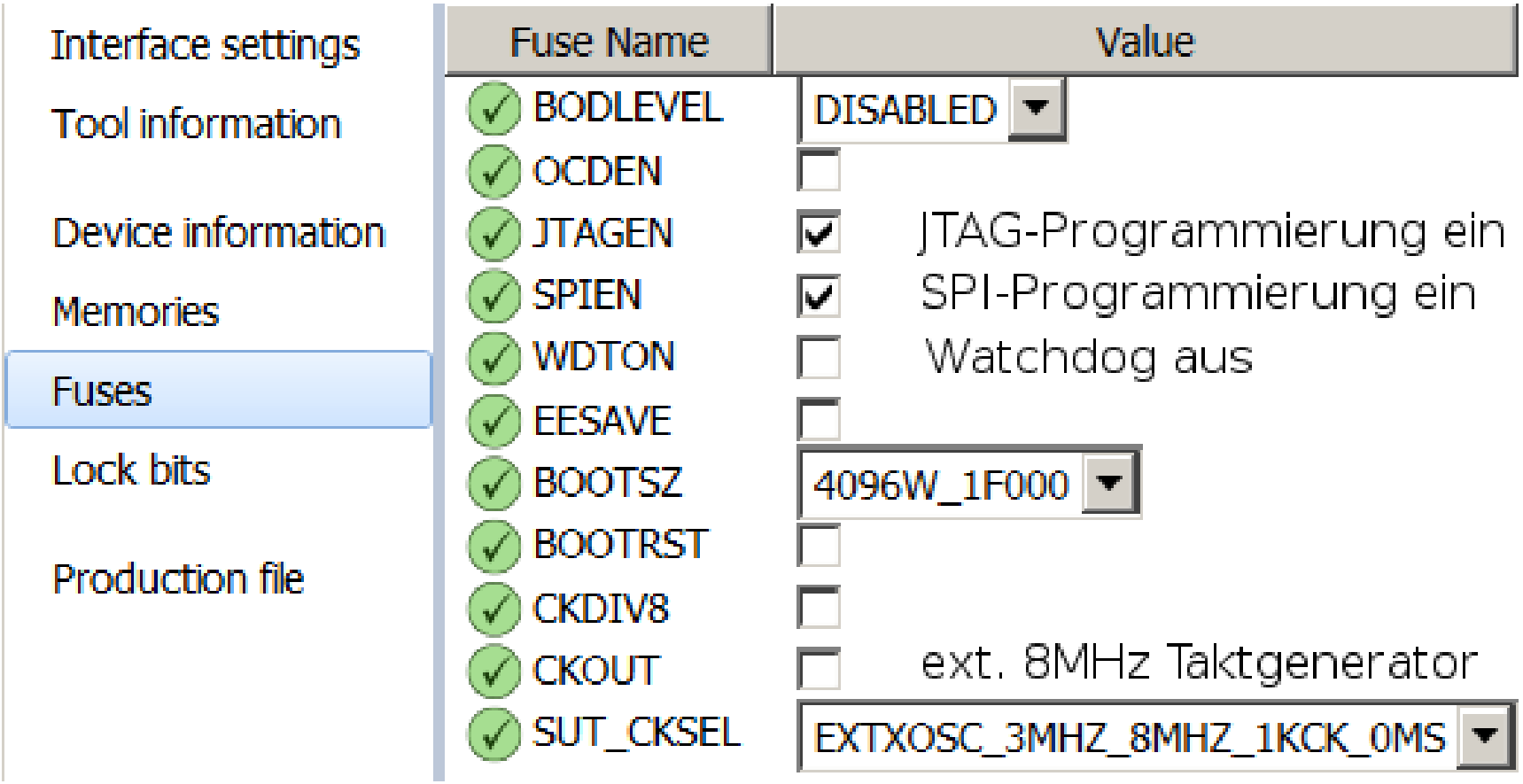
  `Tools > Device Programming`

  auswählen. Nachfolgende Kontrollen vornehmen



Kontrolle der Sicherungsbits (Fuses)

- Die Sicherungsbits aktivieren Grundfunktionen, z.B. Programmierschnittstellen, Kopierschutz, ...
- Bei Einstellungsfehlern lässt sich der Mikrorechner nicht programmieren, die Programme funktionieren nicht, ...



- Unter »Device Information« findet man außer einer Kurzübersicht auch das Datenblatt (Datasheet) des Mikrorechners

Interface settings
Tool information
Device information
Memories
Fuses
Lock bits
Production file

Datasheet Information

	ATmega2560	
CPU	AVR8	
Flash size	256 Kbytes	(Befehlsspeichergröße)
EEPROM size	4 Kbytes	
SRAM size	8 Kbytes	(Datenspeichergröße)
VCC range	1,8 - 5,5 V	(Versorgungsspannung)
Maximum speed	N/A	

Device Information Datasheets (Datenblatt)

Das Menü »Tools > Device Programming« wird nur zur Kontrolle benötigt, ob der Prozessor über den Programmer erreichbar ist, Spannung hat, der Prozessortyp stimmt, ...

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(){
  // Initialisierung
  DDRA = 0b00000000; // Port A (Schalter) Eingänge
  DDRJ = 0b11111111; // Port J (LEDs) Ausgänge
  uint8_t a;         // 8-Bit-Variable//
  while(1) {         // Endlosschleife//
    a = PINA;        // Lese Schalterwert
    PORTJ = a;       // Ausgabe auf die LEDs
  }
}
```

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

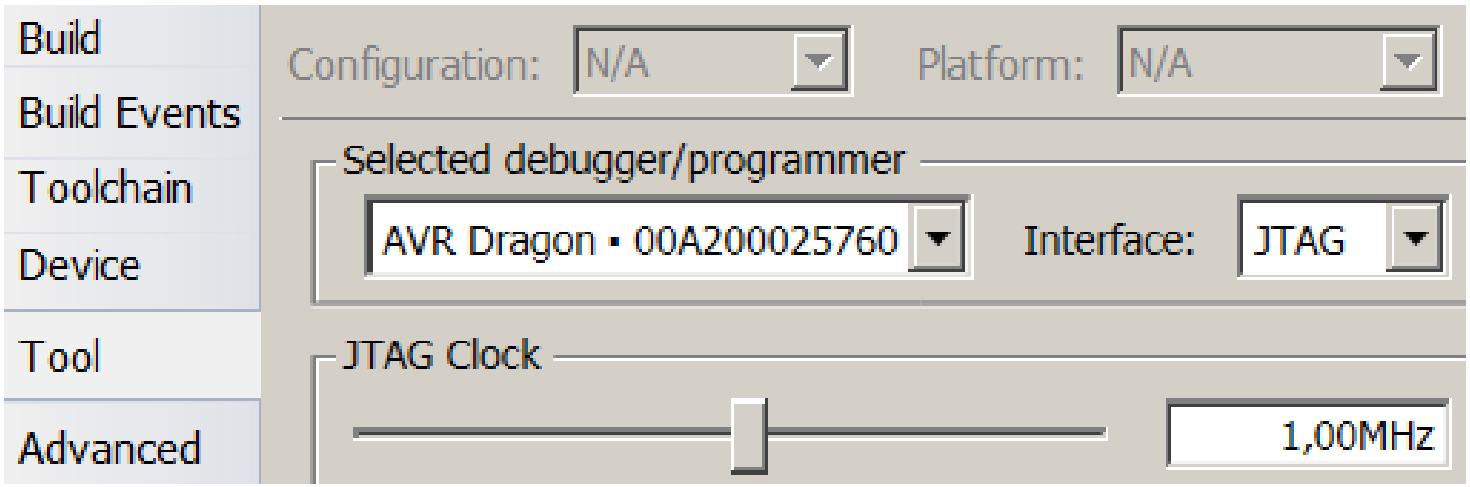
Projekt einrichten

- Neues Projekt anlegen:

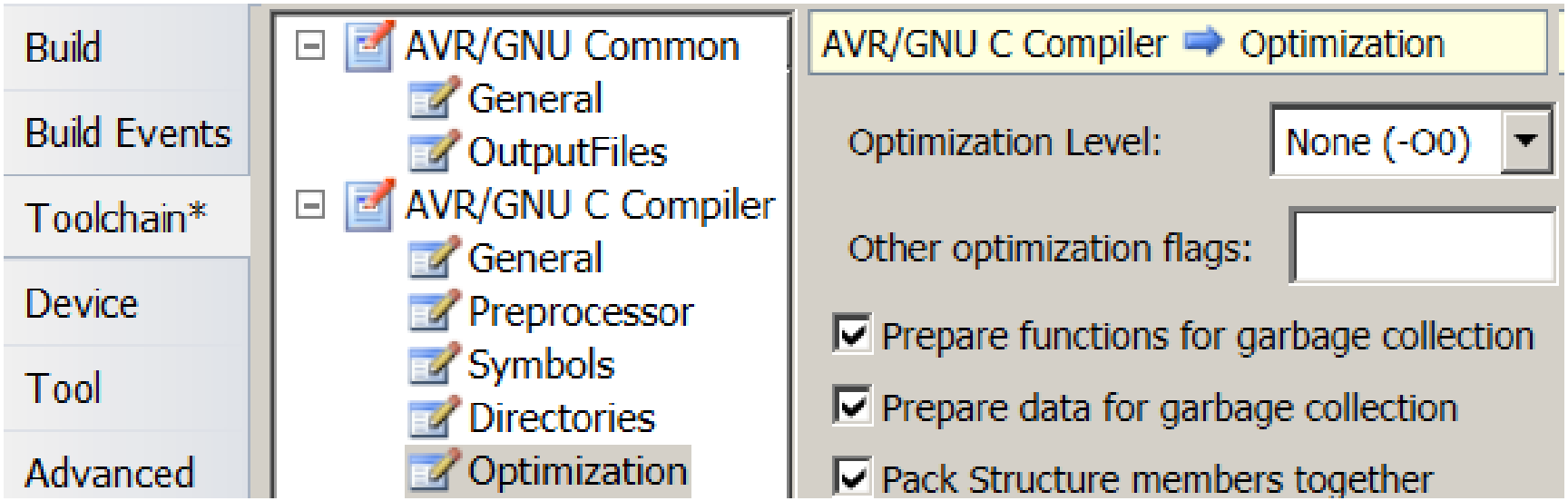
  `File > New > Project > GCC C Executable`

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

  `Project > bit_io1 Properties (Alt + F7) >`
  `Tool > AVR Dragon ..., JTAG`



- 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].

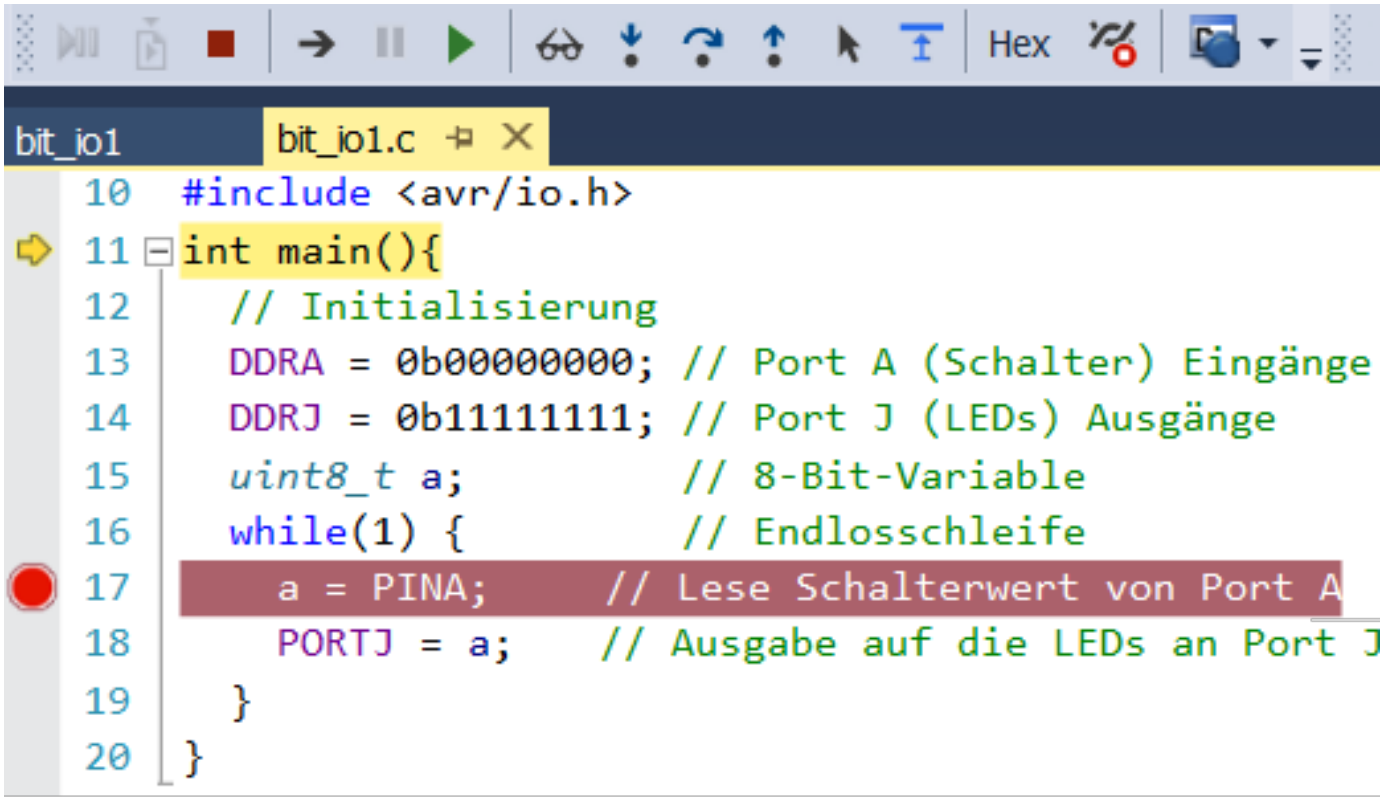
```
10  #include <avr/io.h>
11  int main(){
12    // Initialisierung
13    DDRA = 0b00000000; // Port A (Schalter) Eingänge
14    DDRJ = 0b11111111; // Port J (LEDs) Ausgänge
15    uint8_t a;         // 8-Bit-Variable
16    while(1) {         // Endlosschleife
17      a = PINA;        // Lese Schalterwert von Port A
18      PORTJ = a;       // Ausgabe auf die LEDs an Port J
19    }
20  }
```

- 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`

JTAG
PORTA
PORTB

Name	Address	Value	Bits
PINA	0x20	0x03	
DDRA	0x21	0x00	
PORTA	0x22	0x00	

`Debug > Windows > Watch > Watch1`

```
16  while(1) {
17    a = PINA;
18    PORTJ = a;
19  }
20 }
```

Watch 1

Name	Value	Type
a	3	uint8_t{data}@0x21fa ([R28]+1)

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

Bitverarbeitung

Bitoperationen

Bitoperationen

Mikrorechnerprogramme verarbeiten oft einzelne Bits:

- Schaltereingaben,
- LED-Ausgaben,
- Motor ein/aus, ...

Die Bits sind für die Verarbeitung im Prozessor zu Bytes zusammengefasst. C-Vereinbarung für 1-Byte-Variablen:

```
uint8_t a, b;          // zwei 1-Byte-Variablen
```

- Byte-Werte kopieren:

```
a = b;
```

- Byte nach rechts oder links verschieben:

```
a = 0b10110111; // a: 0b10110111 = 0xB7
b = a >> 2;     // b: 0b00101101 = 0x2D
a = b << 3;     // a: 0b01101000 = 0x68
```

(0b... – Binärdarstellung; 0x...–Hexadezimaldarstellung).

- bitweise Negation:

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

- bitweises UND (Ergebnis 1, wenn beide Operandenbits 1 sind):

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

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

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

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

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

---

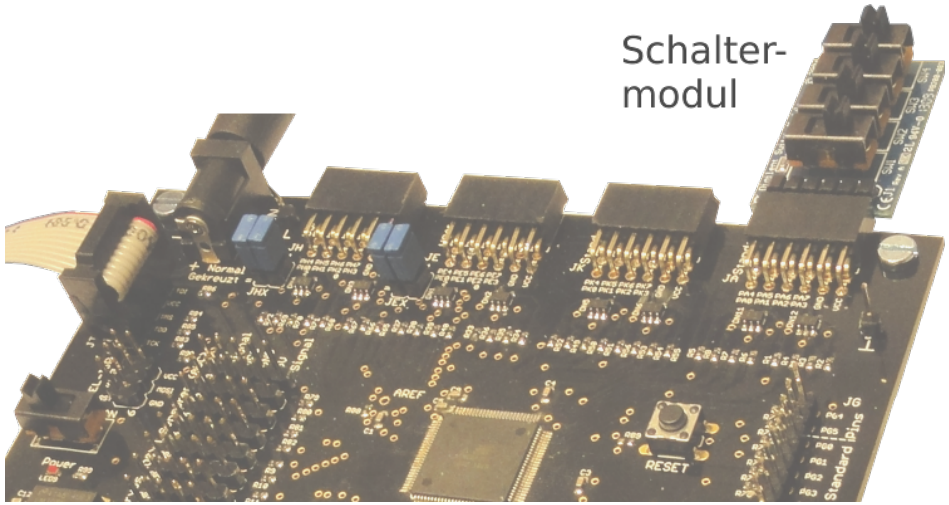
x_1	x_0	$\bar{x}_0$	$x_1 \wedge x_0$	$x_1 \vee x_0$	$x_1 \oplus x_0$
0	0	1	0	0	0
0	1	0	0	1	1
1	0	–	0	1	1
1	1	–	1	1	0

Programmieraufgabe: LD0 = SW1 ∧ SW2

Schalter / LED	SW1	SW2	LD0
Port, Bit	A0	A1	J0

```
#include <avr/io.h>
int main(void){
  // Initialisierung
  DDRA = 0;            // Port A (Schalter) Eingänge
  DDRJ = 0xFF;         // Port J (LEDs) Ausgänge
  uint8_t a, b, c;     // 8-Bit-Variablen
  while(1){
    a = PINA & 0b01; // a(0)<= SW1
    b = PINA & 0b10; // b(1)<= SW2
    c = b >> 1;      // c(0)<= b(1)
    PORTJ=a & c;     // LD(0)<=SW1 & SW2
  }
}
```

Ausprobieren



- Spannung abschalten, Schaltermodul an JA belassen.
- Projekt schließen

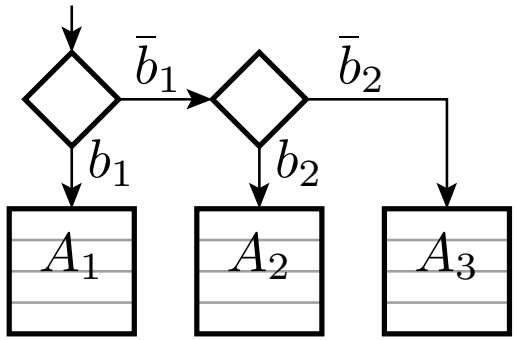
  ```
  File > close solution
  ```

- Archiv »IW.zip« von der Webseite laden und auf Laufwerk H: im neu anzulegenden Unterverzeichnis »Informatikwerkstatt« entpacken.
- Projekt »bit_io2« öffnen, übersetzen, laden, ausprobieren.

Binäre Fallunterscheidung

Binäre Fallunterscheidungen mit »if« und »else«

```
if (<Bedingung b1>){
 <führe Anweisungsblock A1 aus>
}
else if (<Bedingung b2>{
 <führe Anweisungsblock A2 aus>
}
else{
 <führe Anweisungsblock A3 aus>
}
```



{...} – Zusammenfassung von Anweisungen zu einem Block.
$b_i \in$ {falsch, wahr} – Bedingung, Darstellung durch C-Variablen:

Wahrheitswert	falsch	wahr
Bitvektorwert	0	$\neq 0$

Operatoren mit Wahrheitswerten als Ergebnis:

- Vergleichsoperatoren: >, =>, ==, !=, >=, > und
- logische Operatoren für Wahrheitswerte: || (logisches ODER), && (logisches UND) und ! (logische Negation).

Beispielprogramm für »LD0 = SW1 ∧ SW2« (PJ.0=PA.0∧PA.1):

```
while(1){
  if ((PINA & 1) && (PINA & (1<<1)))
    PORTJ |=  1;     // LD0 einschalten
  else
    PORTJ &= ~1;     // LD0 ausschalten
 };
```

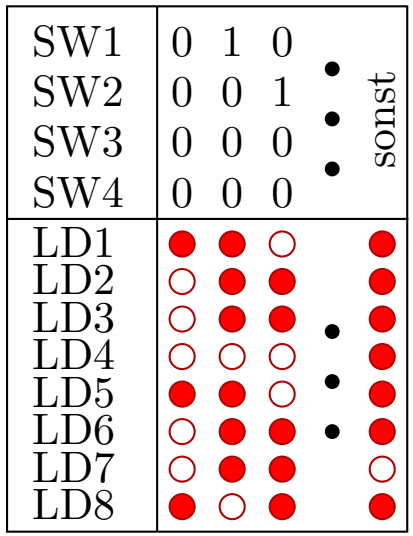
Schalter und Leuchtdioden sind gut zur Prüfung logischer Operationen geeignet.

Auswahlanweisung

Auswahlanweisung

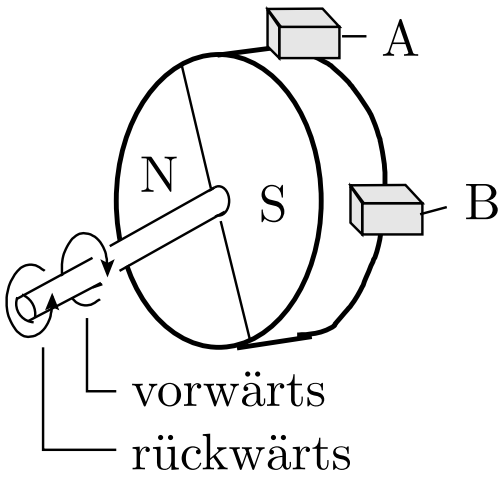
```
switch (PINA & 0b1111){ // SW4 bis SW1
  case 0b0000: PORTJ = 0b10010001;
               break;
  case 0b0001: PORTJ = 0b01110111;
               break;
  case 0b0010: PORTJ = 0b11100110;
               break;
  ...
  default: PORTJ = 0b10111111;
}
```

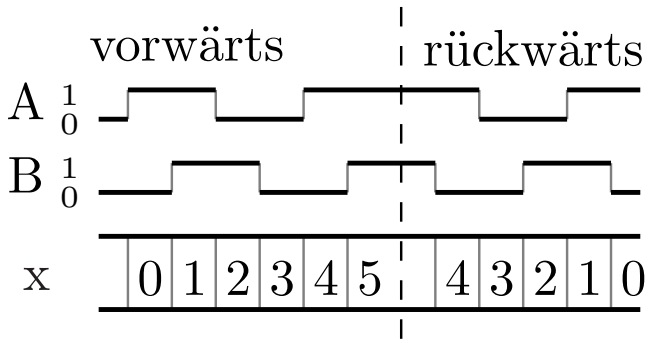
Auswahlausdruck			
w_1	w_2	$\cdots$	sonst
A_1	A_2	$\cdots$	A_{sonst}

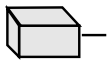


- Die auszuführende Anweisungsfolge reicht von »:« bis »break«.
- Ohne »break« werden auch die Anweisungen des nächsten Auswahlfalls mit abgearbeitet.
- »default« steht für alle anderen Werte.

Winkelmessung mit Wertetabelle





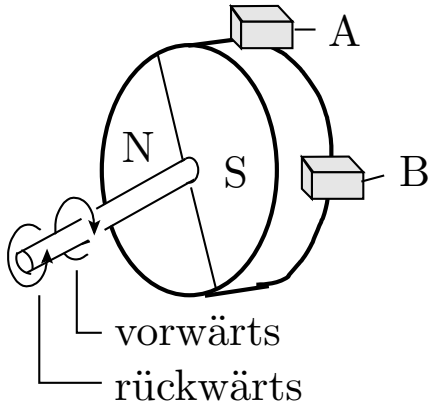
	Hallsensoren
x	Winkel in Viertelkreisschritten
A*, B*	Abtastwerte Zeitschritt zuvor

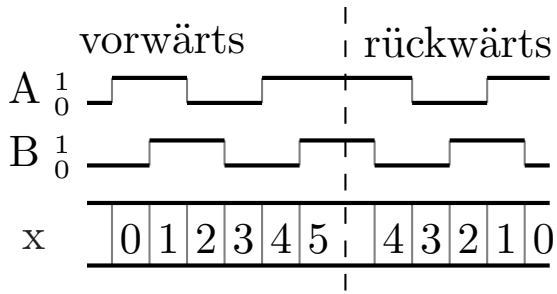
A*	B*	A	B	x
0	0	0	0	—
0	0	0	1	−1
0	0	1	0	+1
0	1	0	1	—
0	1	0	0	+1
0	1	1	1	−1
1	0	1	0	—
1	0	1	1	+1
1	0	0	0	−1
1	1	1	1	—
1	1	1	0	−1
1	1	0	1	+1

Bei einer Rotation von max 90° je Abtastintervall lassen sich aus zwei aufeinanderfolgenden Abtastwerten (4 Bits) bestimmen, ob sich der Magnet um +1, 0 oder -1 Winkelschritt gedreht hat. Nach diesem Prinzip arbeitet die Wegmessung der Fahrzeugmodelle.

- Sensorwert »A« sei PL0 und »B« PL1:

```
sens_A = (sens_A<<2) | (PINL & 0b11);
switch (sens_A){
    case 0b0010:
    case 0b0100:
    case 0b1011:
    case 0b1101:
    Ct++; break;
    case 0b0001:
    case 0b0111:
    case 0b1000:
    case 0b1110:
    Ct--; break
    case 0b0011:
    case 0b0110:
    case 0b1100:
    case 0b1001:
    Ct_Err++;
}
```





A*	B*	A	B	x
0	0	0	0	—
0	0	0	1	−1
0	0	1	0	+1
0	1	0	1	—
0	1	0	0	+1
0	1	1	1	−1
1	0	1	0	—
1	0	1	1	+1
1	0	0	0	−1
1	1	1	1	—
1	1	1	0	−1
1	1	0	1	+1

Automaten und Warteschleifen

Funktion und Automat

- Eine Funktion berechnet eine Ausgabe y aus Eingaben x:

$$y = f(x)$$

  z.B. die LED-Ausgabe aus Schaltereingaben.

- Ein Automat ist ein Berechnungsmodell mit einem zusätzlichen Zustand z, einer Übergangsfunktion

$$z_{n+1} = f_z(z_n, x_n)$$

  und einer Ausgabefunktion:

$$y_{n+1} = f_y(z_n, x_n)$$

  (n – Nummer des Berechnungsschritts).

Ein Beispiel für einen Automaten ist die Berechnnung des Drehwinkels auf Folie 36 aus den aktuellen und vorherigen Hallsenorwerten und dem Ist-Drehwinkel.

- Der Test eines Automaten verlangt, dass nach der Anfangsinitialisierung für jeden Schritt Eingaben bereit gestellt und Ausgaben ausgewertet werden.
- Für den Test mit Schaltern und Leds ist hierzu eine Möglichkeit, die Dauer der Berechnungsschritte mit einer Warteschleife in den Sekundenbereich zu verlängern.
- Eine Wartescheife ist eine Zählschleife, die nichts tut, als Zeit zu verbrauchen, z.B.:

```
uint32_t Ct;  // 32-Bit-Zähler
...
for (Ct=0; Ct<200000; Ct++);
```

Aufgabe:

- bei (SW1==1) soll ein Leuchtpunkt auf den LEDs an Port J nach rechts
- und sonst nach links rotieren.
- Dauer je Ausgabe- (Berechnungs-) Schritt $\approx 250\,\mathrm{ms}$

Lauflicht (Beispielprojekt »bit_io3«)

```
#include <avr/io.h>
uint32_t Ct;          // 32-Bit-Zähler
uint8_t a=1;          // 8-Bit Ausgabewert
int main(void){       // Initialisierung
  DDRA = 0;           // Port A (Schalter) Eingänge
  DDRJ = 0xFF;        // Port J (LEDs) Ausgänge
  while(1){           // Warteschleife
    for (Ct=0; Ct<200000; Ct++);
    if (PINA & 0b1) // Rotation links
      a = (a<<1) | (a>>7);
    else              // Rotation rechts
      a = (a>>1) | (a<<7);
    PORTJ = a;        // Ausgabe
  }
}
```

- Laden, Übersetzen und Testen von Projekt »bit_io3« aus dem Archiv auf der Web-Seite.

Aufgaben

Aufgabe 1.1: Bitverarbeitung

Definieren Sie sich eine Verarbeitungsfunktion zur Abbildung der vier Schalterbits auf die acht Leuchtdioden.

1. Tragen Sie die Funktion in die nachfolgende Wertetabelle ein.
2. Schreiben Sie angelehnt an Folie 29 ein entsprechendes Programm und testen Sie es.

SW1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
SW2	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
SW3	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
SW4	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
LD1	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
LD2	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
LD3	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
LD4	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
LD5	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
LD6	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
LD7	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
LD8	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○

Aufgabe 1.2: Automat

- Definieren Sie eine Übergangs- und eine Ausgabefunktion für einen Automaten. Eingabe: SW1 und SW2.
- Programmierung in Anlehnung an Folie 41.
- Ausfüllen der nachfolgenden Tabelle für Beispieleingaben.

SW1 SW2						
Zustand						
LD1	○ ○ ○ ○	○ ○ ○ ○	○ ○ ○ ○	○ ○ ○ ○	○ ○ ○ ○	○ ○ ○ ○
LD2	○ ○ ○ ○	○ ○ ○ ○	○ ○ ○ ○	○ ○ ○ ○	○ ○ ○ ○	○ ○ ○ ○
LD3	○ ○ ○ ○	○ ○ ○ ○	○ ○ ○ ○	○ ○ ○ ○	○ ○ ○ ○	○ ○ ○ ○
LD4	○ ○ ○ ○	○ ○ ○ ○	○ ○ ○ ○	○ ○ ○ ○	○ ○ ○ ○	○ ○ ○ ○
LD5	○ ○ ○ ○	○ ○ ○ ○	○ ○ ○ ○	○ ○ ○ ○	○ ○ ○ ○	○ ○ ○ ○
LD6	○ ○ ○ ○	○ ○ ○ ○	○ ○ ○ ○	○ ○ ○ ○	○ ○ ○ ○	○ ○ ○ ○
LD7	○ ○ ○ ○	○ ○ ○ ○	○ ○ ○ ○	○ ○ ○ ○	○ ○ ○ ○	○ ○ ○ ○
LD8	○ ○ ○ ○	○ ○ ○ ○	○ ○ ○ ○	○ ○ ○ ○	○ ○ ○ ○	○ ○ ○ ○

Aufgabe 1.3: Psedo-Zufallszahlengenerator

- Programmieren Sie einen Pseudo-Zufallsgenerator (Automat) mit dem Zustand

  ```
  uint8_t z;
  ```

  als Eingabe Bit 0 an Port A (SW1) und nachfolgenden Übergangs- und Ausgabefunktionen:

```
...
while(1){          // Endlosschleife
  if (PORTA & 1) // Wenn Schalter ein
    z = 0x31;    // Anfangsinitialisierung
  else           // sonst pseudo-zuf. Übergang
    z = (z>>1) ^ (z<<7) ^ ((z<<5)&0x80)
      ^ ((z<<4)&0x80) ^ ((z<<3)&0x80);
  PORTJ = z;     // Ausgabe auf die LEDs an Port J
}
```

- Füllen Sie die nachfolgende Tabelle aus:

SW1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Zustand	0x01																					
LD1	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
LD2	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
LD3	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
LD4	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
LD5	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
LD6	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
LD7	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
LD8	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○