Informatikwerkstatt, Foliensatz 1 Einführung bis Bitverarbeitung

G. Kemnitz

19. Oktober 2016

Inhaltsverzeichnis

1 Einleitung

Organisation und Gruppeneinteilung

- Es wird meist einen Teile »Theorie und Experimente unter Anleitung für alle« und einen Teil nur für Studierende ohne Programmiervorkenntnisse geben, während dem die anderen selbstständig arbeiten.
- Sitzordnung: Ohne Vorkenntnisse vorn, die anderen hinten, gegebenenfalls auf Zusatzplätzen und für die praktische Arbeit im Labor.
- Jede Sitzreihe gliedert sich in eine Zweier- und einer Dreiergruppen, die zusammen einen Box mit Hardware bekommen und zusammen arbeiten.
- Später optional Gruppenumsortierung, um die Leistungsfähigkeit innerhalb der Gruppen anzugleichen.

Platzaufteilung. Ausgabe der Boxen mit der Hardware.

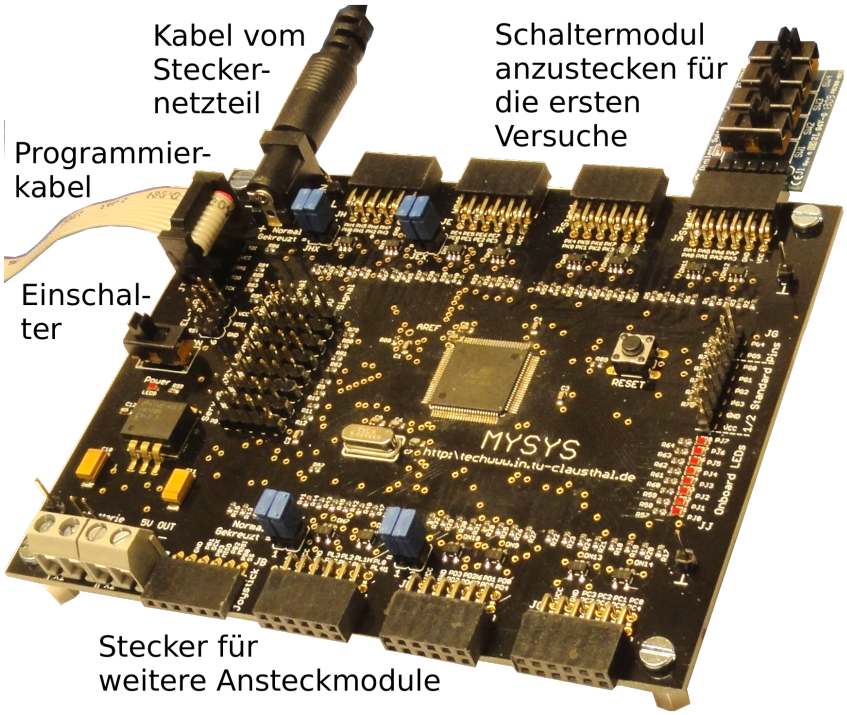
Arbeitsprogramm

- 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, ...
- Projekt: Entwicklung mit selbst zu definiertender Zielfunktion.

Zentrale Angebote für alle Gruppen: Praxisvorträge, ...

1.1 Entwicklungsumgebung

Das Versuchsboard



Inbetriebnahme der Baugruppe

- Anstecken des Programmieradapters.
- Anstecken des Netz- teils (Achtung, nur 5 V- Netzteile verwenden).

- Anstecken des Schalter- moduls an JA (Port A[1]).
- Einschalten, erst wenn die Hardware fertig zusammengesteckt ist.

Kommunikationskontrolle

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

  ```
  techwww.in.tu-clausthal.de/site/Lehre
    /Informatikwerkstatt_2016/
  ```

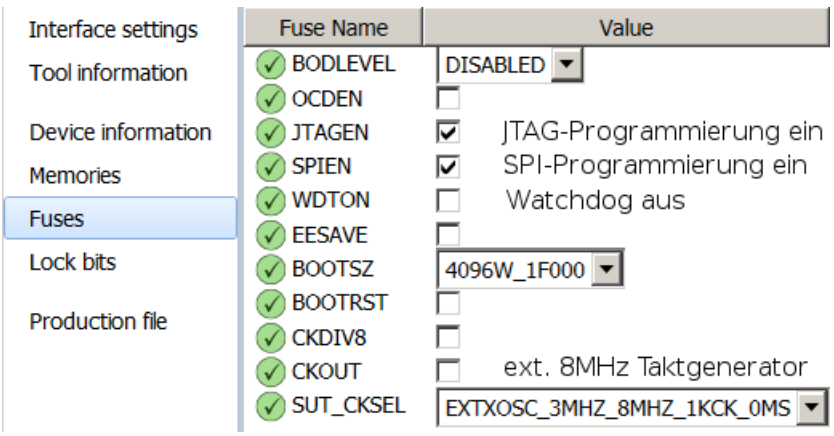
- Atmel Studio starten  .
- Zur Kontrolle, dass der Prozessor richtig angeschlossen und vom System erkannt wird, in Atmel Studio

  ```
  Tools > Device Programming
  ```

  auswählen. Nachfolgende Kontrollen vornehmen

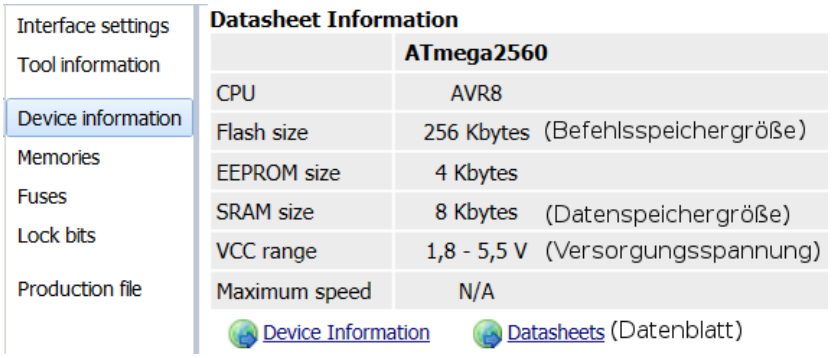
Kontrolle der Sicherungsbits (Fuses)

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



[1] 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.

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



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

1.2 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.
- Hardware zusammenstecken, Programm laden.
- Programm testen.

Projekt einrichten

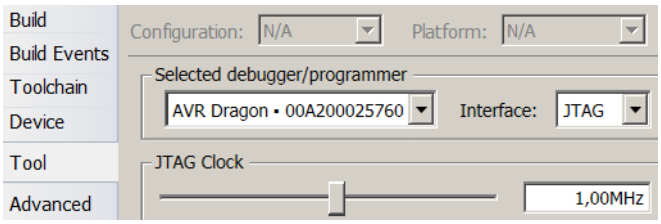
- Neues Projekt anlegen:

  `File > New > Project > GCC C Executable`

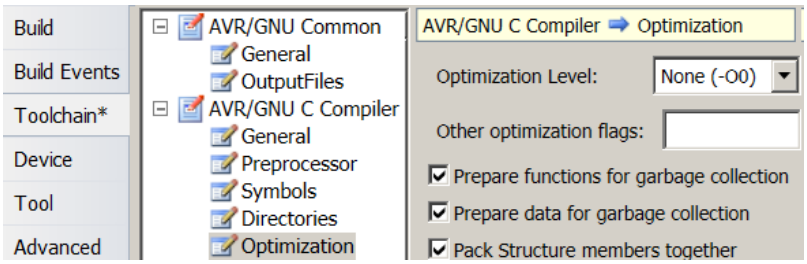
Project Name: »bit_io1«. Location: H:\Informatikwerkstatt. Prozessortyp: Atmega2560.

- Programmier-Tool / Schnittstelle auswählen:

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



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



- Zeilennummern einschalten:

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

- Einstellungen Speichern (Strg + S)

Programm eingeben

- Automatisch erzeugten Programmrahmen vervollständigen[3].

```
#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
  }
}
```

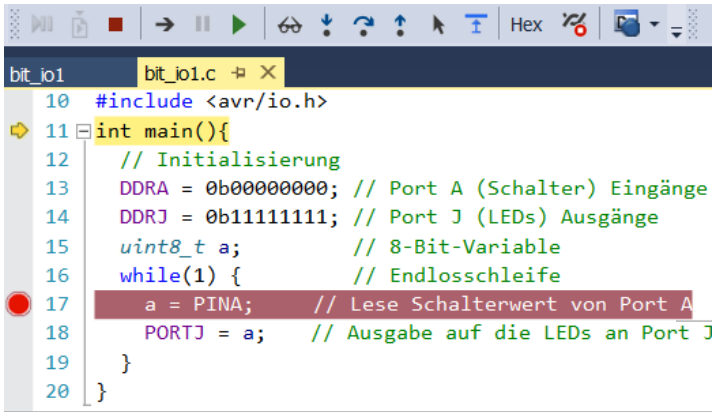
[2] Optimiert sonst für das Ein-/Ausgabeverhalten entbehrliche Schritte weg, z.B. das Zwischenspeichern von Variablenwerten im Datenspeicher und Warteschleifen. Optimierte Programme lassen sich nur eingeschränkt im Schrittbetrieb auf Quellcodeniveau testen.

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

- Speichern.
- Debugger starten:

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

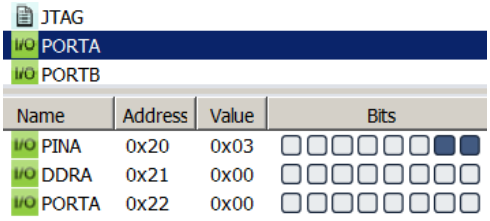
Debugger-Ansicht



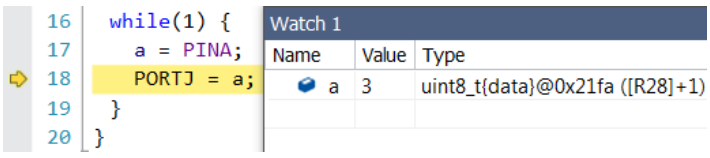

Nächste auszuführende Anweisung.
Unterbrechungspunkt (Mouse-Click grauer Rand davor).
Schritt abarbeiten und halten.
Fortsetzen bis zum nächsten Unterbrechungspunkt.

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[4].
- 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.

2 Bitverarbeitung

2.1 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).

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

- 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
```

---

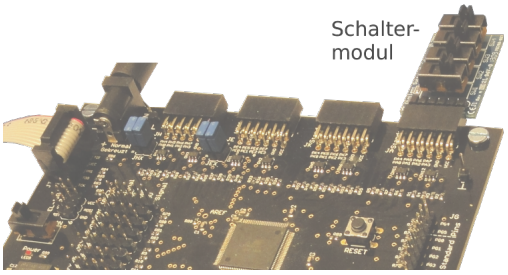
x_1	x_0	$\bar{x}_0$	$x_1 \wedge x_0$	$x_1 \vee x_0$	$x_1 \oplus x_0$
0	0	1	0	0	0
0	1	0	0	1	1
1	0	–	0	1	1
1	1	–	1	1	0

Programmieraufgabe: LD0 = SW1 ∧ SW2

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

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

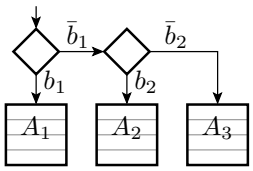
```
File > close solution
```

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

2.2 Binäre Fallunterscheidung

Binäre Fallunterscheidungen mit »if« und »else«

```
if (<Bedingung >){
 <Anweisungsblock 1>
}
else if (<Bedingung >{
 <Anweisungsblock 2>
}
else{
 <Anweisungsblock 3>
}
```



{...} – 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
 }
```

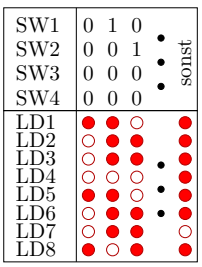
Schalter und Leuchtdioden sind gut zur Prüfung logischer Operationen geeignet.

2.3 Auswahlanweisung

Auswahlanweisung

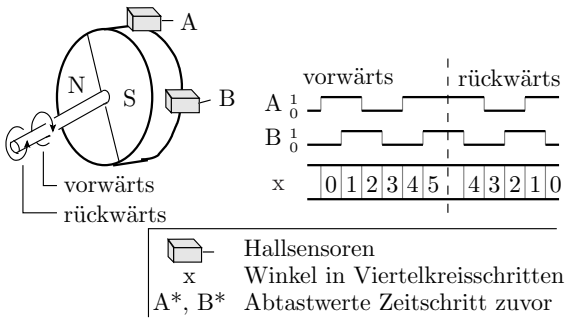
```
switch (PINA & 0b1111){//SW4 bis SW1
  case 0b0000: PORTJ = 0b10010001;
               break;
  case 0b0001: PORTJ = 0b01110111;
               break;
  case 0b0010: PORTJ = 0b11100110;
               break;
  ...
  default: PORTJ = 0b10111111;
}
```

Auswahlausdruck			
w_1	w_2	···	sonst
A_1	A_2	···	A_{sonst}



- Die auszufürende 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

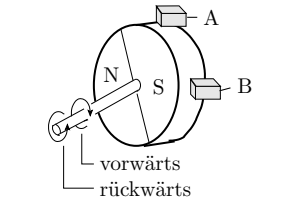


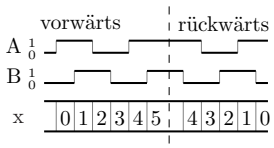
A*	B*	A	B	x
0	0	0	0	—
0	0	0	1	−1
0	0	1	0	+1
0	1	0	1	—
0	1	0	0	+1
0	1	1	1	−1
1	0	1	0	—
1	0	1	1	+1
1	0	0	0	−1
1	1	1	1	—
1	1	1	0	−1
1	1	0	1	+1

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 = (sens<<2) | (PINL & 0b11);
switch (sens){
    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

2.4 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 Berechnung des Drehwinkels auf Folie 10 aus den aktuellen und vorherigen Hallsensorwerten 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;
...
for (Ct=0; Ct<200000; Ct++);
```

Beispielaufgabe Lauflicht

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

Lösung (Beispielprojekt »bit_io3«)

```
#include <avr/io.h>
uint32_t Ct;              //32-Bit-Zähler
uint8_t a=1;              //8-Bit Ausgabewert
int main(void){
 DDRA = 0;                //Schalter-Port: Eingänge
 DDRJ = 0xFF;             //LEDs-Port: Ausgänge
 while(1){                //Endlosschleife
  for (Ct=0; Ct<200000; Ct++); //Warteschleife
  if (PINA & 0b1)         //wenn SW1=1
   a = (a<<1) | (a>>7); //Rotation links
  else                    //sonst
   a = (a>>1) | (a<<7); //Rotation rechts
   PORTJ = a;             //Ausgabe
 }
}
```

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

3 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 8 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 Ausgabe- funktion für einen Automaten. Eingabe: SW1 und SW2.
- Programmierung in Anlehnung an Folie 12.
- 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){
  if (PINA & 1)
    z = 0x31;
  else
    z = (z>>1) ^ (z<<7) ^ ((z<<5)&0x80)
      ^ ((z<<4)&0x80) ^ ((z<<3)&0x80);
  PORTJ = z;
}
```

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

4 Zusatzteil

C-Programmierung

```
// Kommentar bis Zeienende
/*
 Kommentar über mehrere Zeilen
*/

// einfügen der datei io.h aus dem Header-
// Verzeichnis avr. Der Header io.h enthält
// z.B. die Definition von PINA und PORTJ
#include <avr/io.h

int main()
{
 ... // * Anweisusungen, die nacheinander
 ... //   auszuführen sind.
}
```

Zusatzaufgabe 1.1:

```
#include <avr/io.h>
uint8_t a;            // Variablebverenb.
int main(){
 DDRB = 0b01010101;   // Port-Initialisierung
 uint8_t a;
 while(1){            // Endlosschleife
  a =                 // Einleseanweisung
                      // Ausgabeanweisung

 }
}
```

1. Was passiert, wenn die Include-Anweisung fehlt?
2. Welche Pins von Port B sind Ein- und welche Ausgänge?
3. Eränzen Sie das Einlesen der Eingabewerte und die invertierte Ausgabe auf dem Nachbarpins.

Lösung

1. Compiler meldet DDRB, PINB oder PORTB nicht definiert.
2. Gerade Bits sind Ein- und ungerade Ausgänge.
3. Kompletiertes Programm:

```
#include <avr/io.h>
uint8_t a;            // Variablebverenb.
int main(){
 DDRB = 0b01010101;  // Port-Initialisierung
 uint8_t a;
 while(1){           // Endlosschleife
  a = PINB;          // Einleseanweisung
  PORTB = (~a)<<1;   // Ausgabeanweisung
 }
}
```

- Projektanlegen, eingeben, übersetzen und ausprobieren.

Zusatzaufgabe 1.2:

```
#include <avr/io.h>
int main(){
 DDRA =          // Init. als Eingänge
 DDRJ =          // Init. als Ausgänge
 ...     a;      // Variablenvereinbarungen
  while(...){    //
  ...            // Lesen der Eingabe in a
                 // EXOR des gelesen mit dem
                 // nach rechtverschobenen gelesen
                 // Wert
                 // löschen der Bits 1 bis 7
                 // Ausgabe von Bit 0 auf LED4
  }
}
```

Ergänzung, so dass in einer Endlosschleife an PJ.4 die EXOR-Verknüpfung von PA.0 PA.1 ausgegeben wird.

Lösung

```
#include <avr/io.h>
int main(){
 DDRA =          // Init. als Eingänge
 DDRJ =          // Init. als Ausgänge
 uint8_t a;      // Variablenvereinbarungen
  while(1){      //
   a = PINA;     // Lesen der Eingabe in a
   a = (a>>1)^a;// EXOR des gelesen mit dem nach
                 // rechtverschobenen gelesen Wert
   a = a & ~1;   // löschen der Bits 1 bis 7
   PORTJ = a<<4;// Ausgabe von Bit 0 auf LED4
  }
}
```

- Projektanlegen, eingeben, übersetzen und ausprobieren.

Zusatzaufgabe 1.3:

In dem Programmrahmen auf der nächsten Folie ist folgende Funktionalität zu ergänzen:

1. Die Anschlüsse PA.0 und PA.1 seien Ein- und alle Anschlüsse von Port J sowie die restlichen Anschlüsse von Port A Ausgänge.
2. Schrittfunktion:
   - Wenn (PA.0=1) $\wedge$ (PA.1=0): Erhöhung der Ausgabe an Port J um eins.
   - Sonst wenn (PA.1=1): Verringerung der Ausgabe an Port J um eins.
   - Sonst Ausgabe unverändert.
3. Übergangsereignis alle 2 s. Programmierung mit Warteschleife.

```
#include <avr/io.h>
...         Ct=0;      // Zähler von 0 bis 400000

int main(){
 DDRA  = ...        ; // PORT0 und 1 Eingänge
 DDRJ  = ...        ; // PORTJ Ausgänge
 PORTJ = ...        ; // Anfangsausgabewert
 while(1){            // Endlosschleife
  if (...           ) // Wenn PA.1=0 und PA.0=1
   ...                 // PortJ hochzählen
  else if (...         ) // sonst wenn PA.1=1
   ...                // PortJ abwärts zählen
  for (                  ); // Warteschl. 2s
 }
}
```

Lösung

```
#include <avr/io.h>
uint32_t Ct=0;        // Zähler von 0 bis 400000
int main(){
 DDRA = ~0x03;        // PORT0 und 1 Eingänge
 DDRJ = 0xFF;         // PORTJ Ausgänge
 PORTJ = 0;           // Anfangsausgabewert
 while(1){            // Endlosschleife
  if ((PINA&0x3)==1) // Wenn PA.1=0 und PA.0=1
   PORTJ++;           // PortJ hochzählen
  else if ((PINA&0x2)==2) // PA.1=1
   PORTJ--;           // PortJ abwärts zählen
   for (ct=0;ct<400000;ct++); // Warteschl. 2s
  }
}
```

- Projektanlegen, eingeben, übersetzen und ausprobieren.