

Informatikwerkstatt, Foliensatz 1 Einführung bis Bitverarbeitung

G. Kemnitz

23. Oktober 2023

Inhalt:

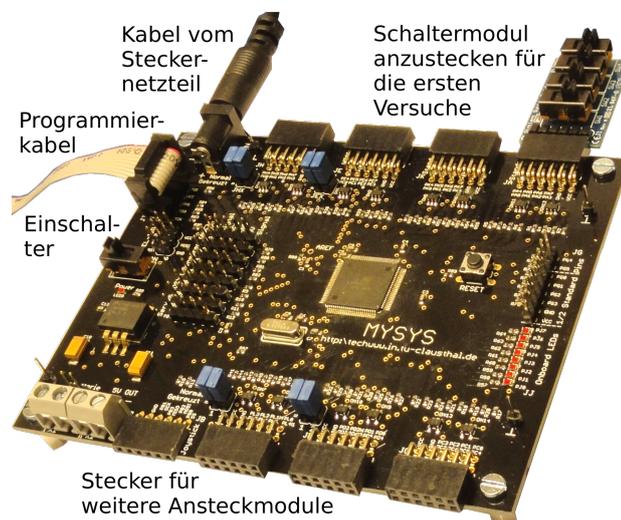
Contents	5	Auswahlweisung	8
1 Entwicklungsumgebung	1	6 Automaten und Warteschleifen	9
2 Das erste Programm	3		
3 Bitverarbeitung	6	7 Aufgaben	11
4 Fallunterscheidung	8	8 Zusatzmaterial	14

Schritt-für-Schritt-Anleitungen auf dem Foliensatz:

- Kommunikationskontrolle on the following page
- Das erste Programm on page 3 (bit_io1).
- Beispielprogramm mit Bitverarbeitung on page 7 (bit_io2).

1 Entwicklungsumgebung

Das Versuchsboard



Inbetriebnahme der Baugruppe

- Programmieradapter anstecken.
- Netzteil anstecken (Achtung, nur 5 V- Netzteile verwenden).
- Schaltermodul JA (Port A¹) anstecken.
- Erst wenn Hardware fertig zusammengesteckt, einschalten.

Kommunikationskontrolle

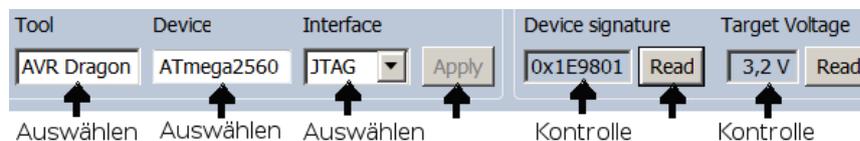
- Rechner unter Windows starten
- Web-Browser (Firefox) öffnen. Foliensatz zum Mitlesen öffnen:

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:



Kontrolle der Sicherungsbits (Fuses)

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

Interface settings	Fuse Name	Value
Tool information	<input checked="" type="checkbox"/> BODLEVEL	DISABLED
Device information	<input checked="" type="checkbox"/> OCDEN	<input type="checkbox"/>
Memories	<input checked="" type="checkbox"/> JTAGEN	<input checked="" type="checkbox"/> JTAG-Programmierung ein
Fuses	<input checked="" type="checkbox"/> SPIEN	<input checked="" type="checkbox"/> SPI-Programmierung ein
Lock bits	<input checked="" type="checkbox"/> WDTON	<input type="checkbox"/> Watchdog aus
Production file	<input checked="" type="checkbox"/> EESAVE	<input type="checkbox"/>
	<input checked="" type="checkbox"/> BOOTSZ	4096W_1F000
	<input checked="" type="checkbox"/> BOOTRST	<input type="checkbox"/>
	<input checked="" type="checkbox"/> CKDIV8	<input type="checkbox"/>
	<input checked="" type="checkbox"/> CKOUT	<input type="checkbox"/> ext. 8MHz Taktgenerator
	<input checked="" type="checkbox"/> SUT_CKSEL	EXTXOSC_3MHZ_8MHZ_1KCK_0MS

¹Oben 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:

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, ...

2 Das erste Programm

Das erste Programm

```
#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
    }
}
```

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

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

Projekt einrichten

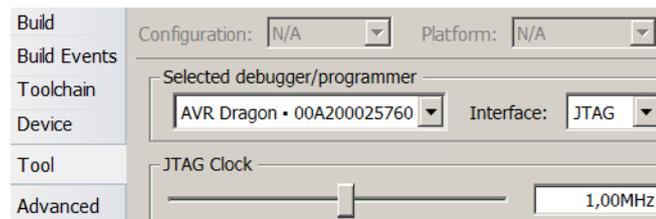
- Neues Projekt anlegen:

```
File > New > Project > GCC C Executable
```

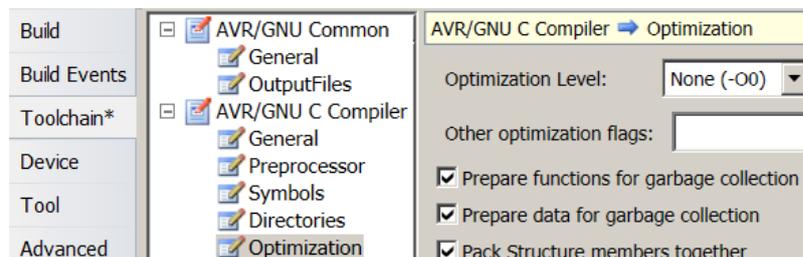
Project Name: »bit_io1«. Location: H:\Informatikwerkstatt. Prozessortyp: Atmega2560.

- 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 vervollständigen³.

```
#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
    }
}
```

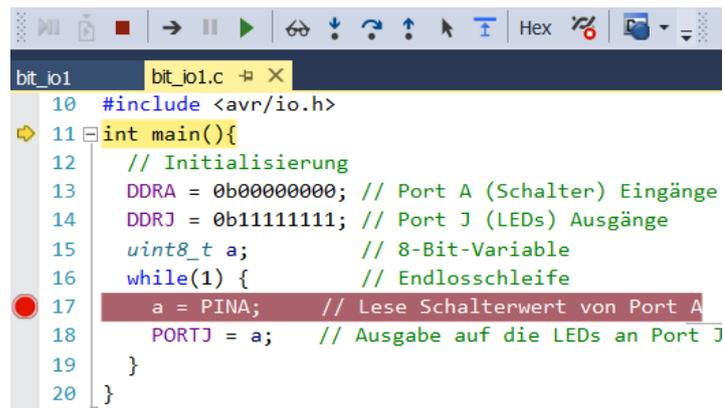
- Speichern.
- Debugger starten: 

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

²Optimiert 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.

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

Debugger-Ansicht



```

bit_io1 bit_io1.c
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 }

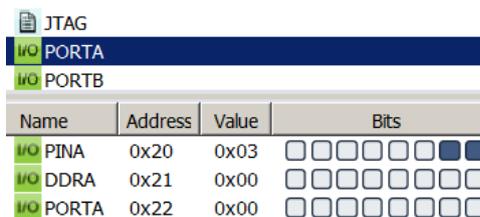
```



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

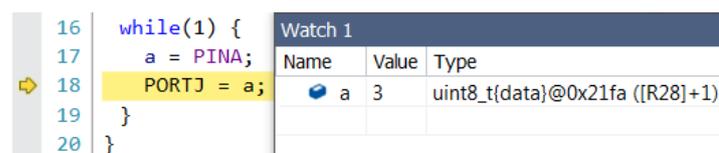
Beobachtungsfenster öffnen

Debug > Windows > IO



Name	Address	Value	Bits
PINA	0x20	0x03	<input type="checkbox"/> <input checked="" type="checkbox"/>
DDRA	0x21	0x00	<input type="checkbox"/>
PORTA	0x22	0x00	<input type="checkbox"/>

Debug > Windows > Watch > Watch1



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 📍 setzen⁴.
- Start/Programmfortsetzung mit ▶.

⁴Mouse-Click auf den grauen Rand vor der Anweisung

- Werte in »IO« und »Watch 1« kontrollieren.
- Schaltereingabe ändern.

Test ohne Unterbrechung:

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

3 Bitverarbeitung

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
```

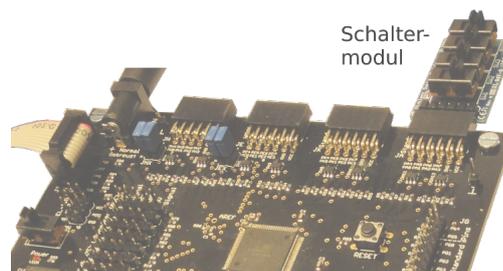
x_1	x_0	\bar{x}_0	$x_1 \wedge x_0$	$x_1 \vee x_0$	$x_1 \oplus x_0$
0	0	1	0	0	0
0	1	0	0	1	1
1	0	1	0	1	1
1	1	0	1	1	0

Programmieraufgabe: $LD0 = SW1 \wedge SW2$

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 »Programme.zip« von der Webseite laden und auf Laufwerk H: im neu anzulegenden Unterverzeichnis »Informatikwerkstatt« entpacken.
- Projekt »F1-bit_io2« öffnen, übersetzen, laden, ausprobieren.

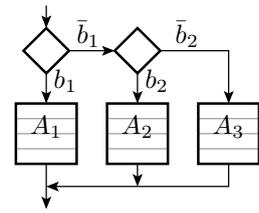
4 Fallunterscheidung

Binäre Fallunterscheidungen mit »if« und »else«

```

if (<Bedingung  $b_1$ >){
  <Anweisungsblock  $A_1$ >
}
else if (<Bedingung  $b_2$ >){
  <Anweisungsblock  $A_2$ >
}
else{
  <Anweisungsblock  $A_3$ >
}

```



{...} – Zusammenfassung von Anweisungen zu einem Block. $b_i \in \{\text{falsch, wahr}\}$ – 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 \wedge SW2« (PJ.0=PA.0 \wedge 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.

5 Auswahanweisung

Auswahanweisung

```

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

```

SW4	0	0	0	0
LD1	●	●	○	●
LD2	○	●	●	●
LD3	○	○	○	●
LD4	○	○	○	●
LD5	●	●	○	●
LD6	○	●	○	●
LD7	○	●	●	○
LD8	○	○	●	○

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

6 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

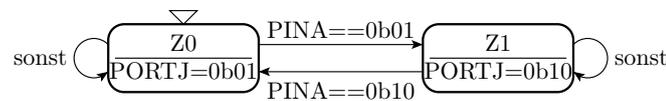
$$z_{n+1} = f_z(z_n, x_n)$$

und einer Ausgabefunktion:

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

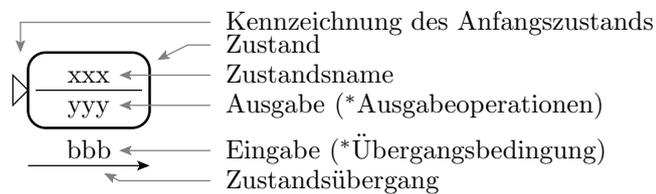
(n – Nummer des Berechnungsschritts).

- Beispielautomat als Graph:



Automaten- und Operationsablaufgraphen

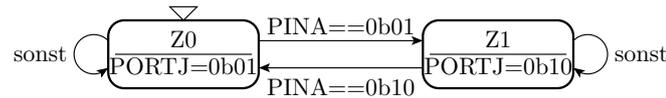
- Ein Automatengraph beschreibt die Zustände durch Knoten und die Zustandsübergänge durch Kanten.
- Die Ausgabe können den Zuständen oder, wenn sie von der Eingabe abhängen, den abgehenden Kanten zugeordnet sein.



* Erweiterung zur Steuerung von Operationsabläufen

- Bei einem Operationsablaufgraphen können die »Ausgaben« auch gesteuerte Operationen und die Übergangsbedingungen aus Operationsergebnissen gebildet werden.

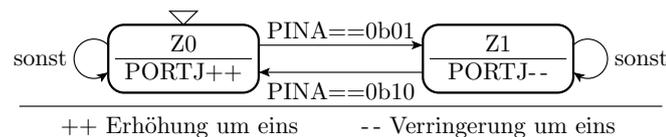
Vom Automatengraph zum Programm



```
#include <avr/io.h>
int main(void){
  DDRA = 0; DDRJ = ~0;    // Ports initialisieren
  PORTJ = 0b01;          // Anfangswert zuweisen
  while (1){
    switch (PORTJ & 0b11){ // Unterscheidung Zustand
      case 0b01:
        if (PINA==0b01) PORTJ = 0b10; break;
      case 0b10:
        if (PINA==0b10) PORTJ = 0b01; break;
      default: PORTJ = 0b01; // bei unzulässigem Zustand
    }
  }
}
```

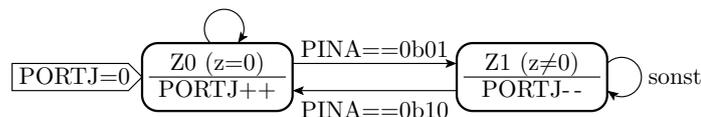
Warteschleifen

- Ein anderer Operationsablauf:



- Die Zeit zwischen zwei Zustandsübergängen beträgt wenige μ s.
- Für den Test mit Schaltern und LEDs ist die Dauer der Berechnungsschritte mit einer Warteschleife in den Sekundenbereich zu verlängern.
- Eine Warteschleife ist eine Zählschleife, die Zeit verbraucht und sonst nichts tut, z.B.:

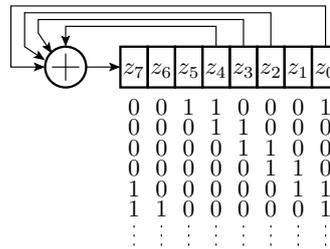
```
for (uint32_t Ct=0; Ct<200000; Ct++); //ca. 250ms
```



```
#include <avr/io.h>
int main(void){
  DDRA = 0; DDRJ = ~0;    // Ports initialisieren
  uint8_t z = 0;         // Zustandsvariable
  while (1){
    switch (z){           // Unterscheidung Zustand
      case 0:
        PORTJ ++;
        if (PINA==0b01) z = 1; break;
      default:            // auch für unzul. Zustände
        PORTJ --;
        if (PINA==0b10) z = 0; break;
    }
    for (uint32_t Ct=0; Ct<200000; Ct++); //ca. 250ms
  }
}
```

Pseudo-Zufallszahlengenerator

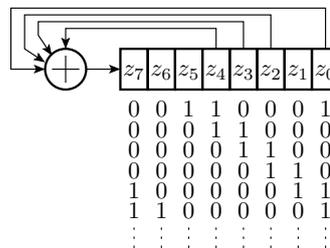
- Pseudo-Zufallsgenerator: Automat, der vom Startzustand aus zylich eine pseudo-zufällige Zustandsfolge durchläuft.
- Beispiel 8-Bit-Rückgekoppeltes-Schieberegister (LFSR Linear Feedback Shift Register):



- Übergangsfunktion:

$$z_7 = z_4 \oplus z_3 \oplus z_2 \oplus z_0$$

$$z_i = z_{i+1} \text{ für } i \in \{0, 1, 2, \dots, 6\}$$



```

...
uint8_t z = 0b00110001; // 0x31;
while(1){
    z = (z>>1) | (((z<<7)^(z<<5)^(z<<4)^(z<<3)) & (1<<7));
    PORTJ = z; // Ausgabe
    ... // Warteschleife
}

```

Pseudo-Zufallszahlen dienen z.B. als Testeingaben für den Programmtest.

7 Aufgaben

Aufgaben

Für alle:

- Handout zum aktuellen Foliensatz noch mal lesen.
- Schritt-für-Schritt-Anleitungen ausprobieren.
- Vorbereitung auf die Beantwortung der Wiederholungsfragen auf dem nächsten Handout.

Browser starten: »google techwww«, Informatikwerkstatt, ... und Datenmaterial auf der Webseite durchblättern.

Die weiteren Programmieraufgaben richten sich nach den individuellen Vorkenntnissen.

Die nachfolgenden Aufgabenstellungen sind Vorschläge, die auch abgewandelt werden dürfen.

Funktionierende Programme sind dem Betreuer zur Kontrolle der erbrachten Leistungen vorzuführen. Richtwert: je Teilnehmer alle 2 Wochen mindestens eine Aufgabe angemessener Schwierigkeit.

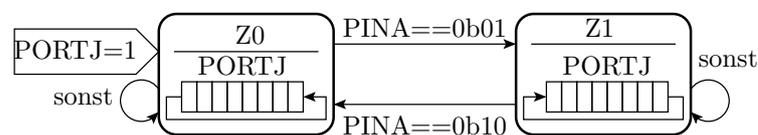
Aufgabe 1.1: Logik mit Schaltern und LEDs

- Schreiben Sie in Anlehnung an das Projekt »bit_io2« ein Programm, dass in der Endlosschleife bei jedem Durchlauf die Schalterwerte an Port A einliest und auf die LEDs an Port J folgende logische Ausdrücke ausgibt:
 - LED0 = SW1 & SW2 & SW3 & SW4
 - LED1 = (SW1 | SW2) & (SW3 & SW4)
 - LED2 = SW1 & (SW2 ^ SW3 ^ SW4)
 - LED3 bis LED7 selbst wählbare Ausdrücke.
- Zeichnen Sie sich eine Wertetabelle wie auf der nächsten Folie auf Papier und füllen Sie diese aus.
- Kontrollieren Sie für alle 16 möglichen Schaltereingaben anhand der ausgefüllten Wertetabelle, dass die richtigen Leuchtdioden leuchten.

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
LED 0	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
LED 1	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
LED 2	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
LED 3	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
LED 4	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
LED 5	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
LED 6	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
LED 7	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○

Hinweise:

- Entwickeln Sie das Programm LED-weise, d.h. zuerst nur für die Ausgabe auf eine LED. Dann Testen und Fehlerbeseitigung. Dann für die Ausgabe auf zwei LEDs etc.
- Nutzen Sie den Debugger, Schrittbetrieb, Unterbrechungspunkte, ...

Aufgabe 1.2: Laufficht-Programm

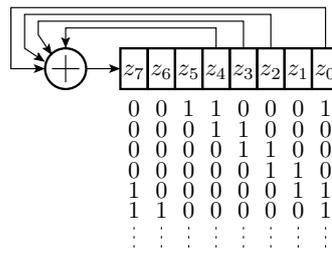
Programmieren Sie den Ablauf zur Erzeugung eines Lauffichts oben:

- Bei Programmstart ist PORTJ mit 1 zu initialisieren.
- Im Zustand Z0 soll auf den LEDs die »Eins« nach links und
- in Zustand Z1 nach rechts rotieren.

Schrittdauer mit Warteschleife auf ≈ 500 ms einstellen.

Hinweis: Eine Anregung für die Programmierung der Rotationen (Verschiebung im Kreis) finden Sie im Beispielprogramm für das rückgekoppelte Schieberegister.

Aufgabe 1.3: Pseudo-Zufallszahlengenerator



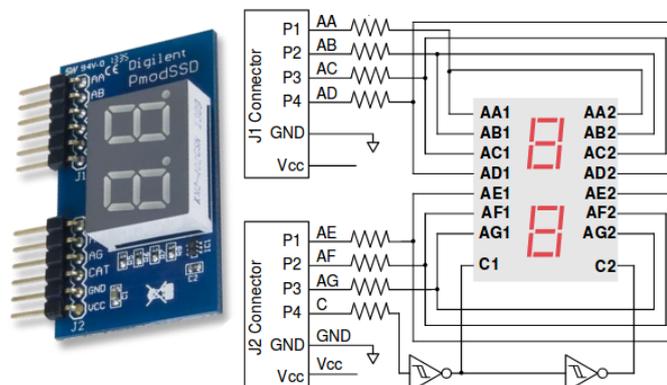
- Schreiben Sie ein Programm, dass ausgehend vom Startwert 0b00110001 mit der Übergangsfunktion des rückgekoppelten Schieberegisters oben mit einem Zyklustakt von ca. 1s zyklisch alle Zustände durchläuft und an die LEDs an Port J ausgibt.
- Übernehmen Sie die Tabelle auf der nächsten Folie auf Papier und füllen Sie sie aus.

Schritt	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
LED 0	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
LED 1	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
LED 2	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
LED 3	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
LED 4	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
LED 5	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
LED 6	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
LED 7	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
0x31																								

Hinweise:

- Zum notieren der Zustandsfolge bietet sich die Nutzung von Unterbrechungspunkten an.
- Der Automat arbeitet zyklisch. Nach Erreichen des Anfangszustands 0b00110001 (0x31) wiederholt sich die Zustandsfolge.

Aufgabe 1.4: 7-Segment-Decoder (Experten)



Stecken Sie ein »PmodSSD« (siehe Bild) an zwei benachbarte freie Ports und steuern Sie es so an, das die rechte Ziffer den Hex-Wert der Schaltereingabe an Port A anzeigt.

8 Zusatzmaterial

C-Programmierung

```
// Kommentar bis Zeilenende
/*
  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()
{
  ... // * Anweisungen, die nacheinander
  ... //   auszuführen sind.
}
```

Aufgabe 1.5: Programm vervollständigen

```
#include <avr/io.h>
uint8_t a;           //Variablenvereinbarung
int main(){
  DDRA =             ; //Port A Eingänge
  DDRJ =             ; //Port J Ausgänge
  uint8_t b;
  while(...){       //Endlosschleife
    a =             ; //Eingabewerte lesen
                   //a.0=(a.0&a.1)|(a.2&a.3)
                   ;
                   ; //Ausgabe an Led 0 ohne
  }                 //andere Led's am Port J
}                   //zu ändern
```

1. Was passiert, wenn die Include-Anweisung fehlt?
2. Vervollständigen Sie das Programm.

Lösung

1. Compiler meldet DDRB, PINB oder PORTB nicht definiert.
2. Vervollständigtes Programm:

```
#include <avr/io.h>
uint8_t a;           //Variablenvereinbarung
int main(){
  DDRA = 0           ; //Port A Eingänge
  DDRJ = ~0          ; //Port J Ausgänge
  uint8_t b;
  while( 1 ){       //Endlosschleife
```

```

a = PINA          ; //Eingabewerte lesen
a = (a & (a>>1))| //a.0=(a.0&a.1)|(a.2&a.3)
  ((a>>2)&(a>>3));
PORTJ = (PORTJ&(~1))|(a&1); //Ausgabe an Led 0
} //ohne andere Led's am
} //Port J zu ändern

```

Aufgabe 1.6: Programm vervollständigen

```

#include <avr/io.h>
int main(){
  DDRA =          // Init. als Eingänge
  DDRJ =          // Init. als Ausgänge
  ...    a;       // Vereinbarung 8-Bit-Variable
  while(...){    // Endlosschleife
  ...          // Lesen der Eingabe in a
              // EXOR des gelesenen mit dem nach
              // recht verschobenen gelesenen
              // Wert
              // löschen der Bits 1 bis 7
              // Ausgabe Bit 0 auf PJ.4 (LED5)
  }
}

```

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

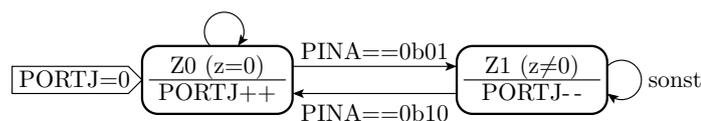
Lösung

```

#include <avr/io.h>
int main(){
  DDRA = 0x00;    // Init. als Eingänge
  DDRJ = 0xFF;    // Init. als Ausgänge
  uint8_t a;      // Variablenvereinbarungen
  while(1){      //
    a = PINA;     // Lesen der Eingabe in a
    a = (a>>1)^a; // EXOR des gelesenen mit dem nach
                // rechtverschobenen gelesenen Wert
    a = a & 1;    // löschen der Bits 1 bis 7
    PORTJ = a<<4; // Ausgabe Bit 0 auf PJ.4 (LED5)
  }
}

```

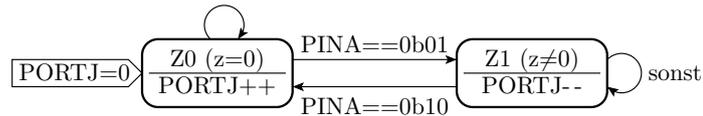
Aufgabe 1.7: Vor/Rückwärtszähler (Reeng.)



Ergänzen Sie im Programmrahmen auf der nächsten Folie:

- Einstellung der Anschlüsse PA.0 und PA.1 als Eingänge und der von Port J als Ausgänge.
- Schrittfunktion:

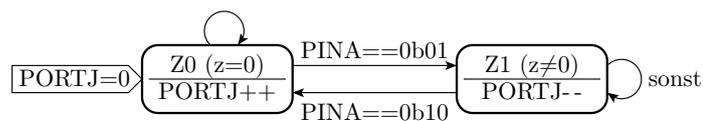
- Anfangszustand nach Programmstart: $z=0$ und $\text{PORTJ}=0$.
 - Wenn Eingabe $\text{PA}=0b01$: Wechsel nach $z=1$.
 - Wenn Eingabe $\text{PA}=0b10$: Wechsel nach $z=0$.
 - Sonst Zustand unverändert.
 - In $Z0$ wird Port J hoch- und in $Z1$ runtergezählt.
- Schrittdauer $\approx 2\text{ s}$ (Warteschleife).



```
#include <avr/io.h>

...     z=0;           // Typ? WB: 0 bis 1
...     Ct;           // Typ? Zähler 0 .. 400000

int main(){
  DDRA  = ...        ; // PA0 und PA1 Eingänge
  DDRJ  = ...        ; // Port J Ausgänge
  PORTJ = ...        ; // Anfangswert 0
  ...    {           // Endlosschleife
  ...    // Übergangsfunktion siehe
  ...    // nächste Folie
  }      // Welcher Block endet hier?
}
```



Übergangsfunktion:

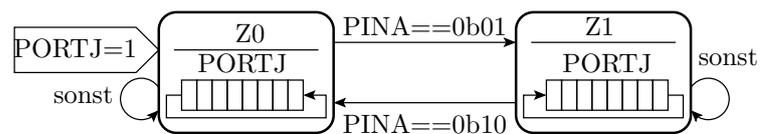
```
while(1){ // Endlosschleife
  if (... ){ // Wenn Zustand Z0
    ... ; // Port J hochzählen
    if (... ) ; // PINA==0b01, Zustand=Z1
  }
  else { // sonst
    ... ; // Port J runterzählen
    if (... ) ; // PINA==0b01, Zustand=Z1
  }
  for (... ); // Warteschl. 2s
}
}
```

Lösung:

```
#include <avr/io.h>
uint8_t z=0
int main(){
  DDRA = ~0x03;           // PA0 und PA1 Eingänge
  DDRJ = 0xFF;           // Port J Ausgänge
  PORTJ = 0;              // Anfangsausgabewert
  while(1){               // Endlosschleife

    if (z == 0){          // Wenn Zustand Z0
      PORTJ++;            // Port J hochzählen
      if (PINA&3==1) z=1; // wenn PINA==0b01, Zustand=Z1
    } else {               // sonst
      PORTJ--;            // Port J runterzählen
      if (PINA&3==2) z=0; // wenn PINA==0b10, Zustand=Z1
    }
  }
  for (u32_t ct=0;ct<400000;ct++); // Warteschleife
} // Welcher Block endet hier?
} // Welcher Block endet hier?
```

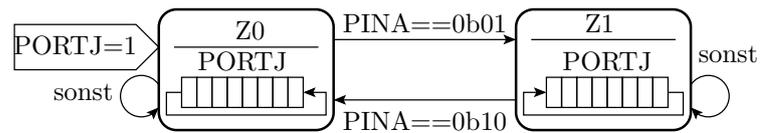
Aufgabe 1.8: Laufflicht



```
#include <avr/io.h>
... //8-Bit Variable a
int main(){
  ... //Port A (Schalter): Eing.
  ... //Port J (Led): Ausgänge
  while(1){ //Endlosschleife
    for ( ... ); //Warteschleife
    if ( ... ) //wenn SW1=1
      ... //Rotation a links
    else //sonst
      ... //Rotation a rechts
    ... //Ausgabe a an die Led
  }
}
```

Aufgaben hierzu:

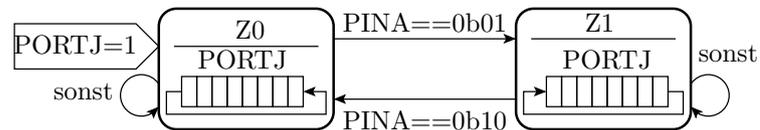
1. Testen Sie das Beispielprojekt »bit_io3« (nächste Folie).
2. Unterschiede zwischen der Ziel- und der Ist-Funktion?
3. Korrigieren Sie das Programm.



Aufgaben hierzu:

1. Testen Sie das Beispielprojekt »bit_io3« (nächste Folie).
2. Unterschiede zwischen der Ziel- und der Ist-Funktion?
3. Korrigieren Sie das Programm.

Lösung



```
#include <avr/io.h>
uint8_t a=1; //8-Bit Ausgabewert
int main(){
  DDRA = 0; //Schalter-Port: Eingänge
  DDRJ = 0xFF; //LEDs-Port: Ausgänge
  while(1){ //Endlosschleife
    for (uint32_t 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
  }
}
```