



Informatikwerkstatt, Foliensatz 1

Einführung bis Bitverarbeitung

G. Kemnitz

Institut für Informatik, TU Clausthal (IW1)
26. Oktober 2016



Inhalt:

Organisation

Entwicklungsumgebung

Das erste Programm

Bitverarbeitung

Fallunterscheidung

Auswahlweisung

Automaten und

Warteschleifen

Aufgaben

Zusatzteil

Interaktive Übungen:

- 1 Das erste Programm (bit_io1).
- 2 Beispielprogramm mit Bitverarbeitung (bit_io2).
- 3 Beispielprogramm "Lauflicht" (bit_io3).



Organisation



Organisation und Gruppeneinteilung

- In den ersten Veranstaltungen wird es einen Teile »Theorie und Experimente unter Anleitung für alle« und einen Teil nur für Studierende ohne Programmierkenntnisse geben, während dem die anderen selbstständig arbeiten.
- Sitzordnung: Ohne Vorkenntnisse vorn, die anderen hinten. Gruppe Labor ggf. auf Zusatzplätzen.
- 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. Account-Ausgabe. Einloggen unter Windows.
Abholen der Boxen mit der Hardware.



Arbeitsprogramm

- Kennenlernen der Entwicklungsumgebung (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 definierender Zielfunktion.

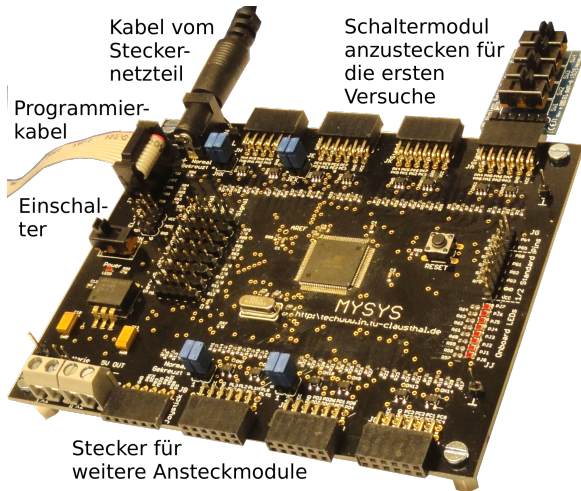
Zentrale Angebote für alle Gruppen: Praxisvorträge, ...



Entwicklungsumgebung

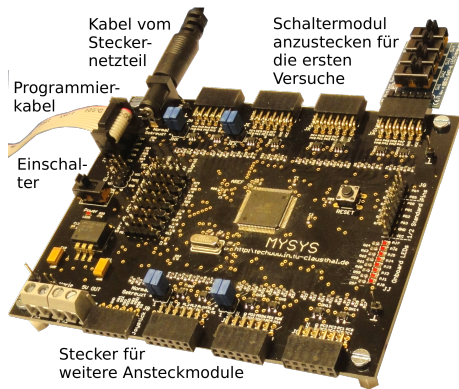


Das Versuchsboard



Inbetriebnahme der Baugruppe

- Anstecken des Programmieradapters.
- Anstecken des Netzteils (Achtung, nur 5V-Netzteile verwenden).
- Anstecken des Schaltermoduls an JA (Port A¹).
- Einschalten, erst wenn die Hardware fertig zusammengesteckt ist.




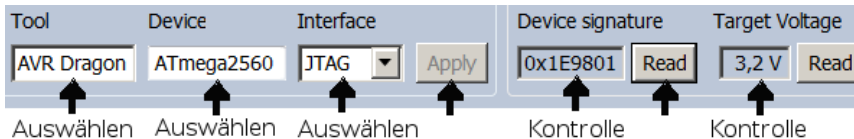
¹Oben angesteckt: SW1⇒JA.0, SW2⇒JA.1, SW3⇒JA.2, SW4⇒JA.3.
Unten angesteckt: SW1⇒JA.4, SW2⇒JA.5, SW3⇒JA.6, SW4⇒JA.7.



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





2. Entwicklungsumgebung

Kontrolle der Sicherungsbits (Fuses)



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

| | Fuse Name | Value |
|--------------------|---|---|
| Interface settings | | |
| Tool information | <input checked="" type="checkbox"/> BODLEVEL | DISABLED ▾ |
| | <input checked="" type="checkbox"/> OCDEN | <input type="checkbox"/> |
| Device information | <input checked="" type="checkbox"/> JTAGEN | <input checked="" type="checkbox"/> JTAG-Programmierung ein |
| Memories | <input checked="" type="checkbox"/> SPIEN | <input checked="" type="checkbox"/> SPI-Programmierung ein |
| Fuses | <input checked="" type="checkbox"/> WDTON | <input type="checkbox"/> Watchdog aus |
| | <input checked="" type="checkbox"/> EESAVE | <input type="checkbox"/> |
| Lock bits | <input checked="" type="checkbox"/> BOOTSZ | 4096W_1F000 ▾ |
| | <input checked="" type="checkbox"/> BOOTRST | <input type="checkbox"/> |
| Production file | <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 ▾ |





2. Entwicklungsumgebung

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



| Interface settings | Datasheet Information <table border="1"><thead><tr><th colspan="2">ATmega2560</th></tr></thead><tbody><tr><td>CPU</td><td>AVR8</td></tr><tr><td>Flash size</td><td>256 Kbytes (Befehlsspeichergröße)</td></tr><tr><td>EEPROM size</td><td>4 Kbytes</td></tr><tr><td>SRAM size</td><td>8 Kbytes (Datenspeichergröße)</td></tr><tr><td>VCC range</td><td>1,8 - 5,5 V (Versorgungsspannung)</td></tr><tr><td>Maximum speed</td><td>N/A</td></tr></tbody></table> | 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 |
|---------------------------|--|-----------------------------------|--|-----|------|------------|-----------------------------------|-------------|----------|-----------|-------------------------------|-----------|-----------------------------------|---------------|-----|
| 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 | | | | | | | | | | | | | |
| Tool information | | | | | | | | | | | | | | | |
| Device information | | | | | | | | | | | | | | | |
| Memories | | | | | | | | | | | | | | | |
| Fuses | | | | | | | | | | | | | | | |
| Lock bits | | | | | | | | | | | | | | | |
| Production file | | | | | | | | | | | | | | | |

 [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

```
#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

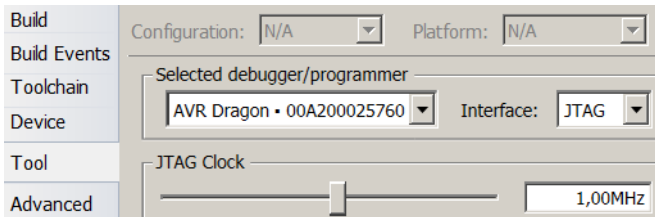
Project Name: »bit_io1«. Location: H:\Informatikwerkstatt.

Prozessortyp: Atmega2560.

- Programmier-Tool / Schnittstelle auswählen:

Project > bit_io1 Properties (Alt + F7) >

Tool > AVR Dragon ..., JTAG





3. Das erste Programm

- Unter Toolchain die Optimierung für den Übersetzer ausschalten² (»-O1« durch »-O0« ersetzen)



The screenshot shows the AVR Studio IDE interface. On the left, a sidebar contains a tree view with the following items: Build, Build Events, Toolchain*, Device, Tool, and Advanced. The 'Toolchain*' item is expanded to show 'AVR/GNU C Compiler'. This item is further expanded to show 'General', 'Preprocessor', 'Symbols', 'Directories', and 'Optimization'. The 'Optimization' item is selected, and its settings are displayed in the main window. The settings include: 'AVR/GNU C Compiler' followed by 'Optimization' (highlighted in yellow), 'Optimization Level:' set to 'None (-O0)', 'Other optimization flags:' (empty text box), and three checked checkboxes: 'Prepare functions for garbage collection', 'Prepare data for garbage collection', and 'Pack Structure members together'.

- Zeilennummern einschalten:

Tools > Options > Text Editor > All languages
> Line numbers✓

- Einstellungen Speichern (Strg + S)

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.




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)

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



Debugger-Ansicht



```
bit_io1 bit_io1.c + X
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

JTAG

IO PORTA

IO PORTB

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

Debug > Windows > Watch > Watch1

| <pre> 16 while(1) { 17 a = PINA; 18 PORTJ = a; 19 } 20 } </pre> | <p>Watch 1</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Value</th> <th>Type</th> </tr> </thead> <tbody> <tr> <td> a</td> <td>3</td> <td>uint8_t[data]@0x21fa ([R28]+1)</td> </tr> </tbody> </table> | Name | Value | Type | a | 3 | uint8_t[data]@0x21fa ([R28]+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 (●) setzen⁴.
- 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.

⁴Mouse-Click auf den grauen Rand vor der Anweisung



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).



4. Bitverarbeitung

■ 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 |



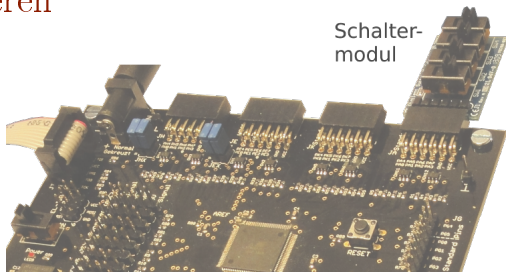
Programmieraufgabe: $LD0 = SW1 \wedge SW2$

| 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.



Fallunterscheidung

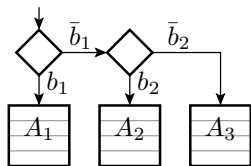


Binäre Fallunterscheidungen mit »if« und »else«

```

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 | $\neq 0$ |



5. Fallunterscheidung

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.



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;
}

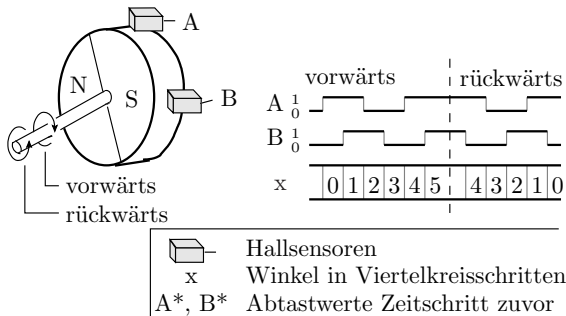
```

| Auswahlausdruck | | | |
|-----------------|-------|-----|--------------------|
| w_1 | w_2 | ... | sonst |
| A_1 | A_2 | ... | A_{sonst} |

| | | | | | |
|-----|---|---|---|---|-------|
| SW1 | 0 | 1 | 0 | | |
| SW2 | 0 | 0 | 1 | • | sonst |
| SW3 | 0 | 0 | 0 | • | |
| SW4 | 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.

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.



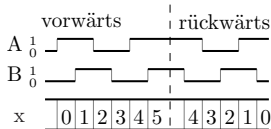
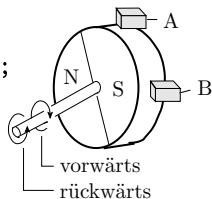
6. Auswahlanweisung

- 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 |



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 30 aus den aktuellen und vorherigen Hallsensorwerten und dem Ist-Drehwinkel.



7. Automaten und Warteschleifen

- 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 die Dauer der Berechnungsschritte mit einer Warteschleife in den Sekundenbereich zu verlängern.
- Eine Warteschleife ist eine Zählschleife, die nichts tut, als Zeit zu verbrauchen, z.B.:

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

Beispielaufgabe Laufflicht



- bei (SW1==1) soll ein Leuchtpunkt auf den LEDs an Port J nach rechts und sonst nach links rotieren.
- Dauer je Ausgabe- (Berechnungs-) Schritt ≈ 250 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 »F1-bit_io3« aus dem Archiv auf der Web-Seite.



Aufgaben



Aufgabe 1.1: Logische Ausdrücke



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



8. Aufgaben

| | | | | | | | | | | | | | | | | |
|-------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |

Hinweis: 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.



Aufgabe 1.2: Automat



- 1 Definieren Sie eine Übergangs- und eine Ausgabe-funktion für einen Automaten. Eingabe: SW1 und SW2. Zustand: Variable »z« mit den zulässigen Werten 1 bis 6. Beispiel einer Funktionsdefinition:

```
wenn z=1:  
  wenn die Eingabe SW=0b01:  
    Folgezustand z=2; Ausgabe 0b10011011;  
  sonst, wenn die Eingabe SW=0b10,  
    Folgezustand z=3; Ausgabe 0b00001111;  
  sonst:  
    Folgezustand bleibt; Ausgabe 0b00000000;  
sonst wenn z=2:  
  ...
```




8. Aufgaben

- 3 Programmieren Sie diese Funktion in eine Endlosschleife mit einer zusätzlichen Wartefunktion, bis Schalter WS2 ein und wieder ausgeschaltet ist für jeden Schleifendurchlauf.
- 4 Zeichnen Sie eine Tabelle für die Ausgabe- und die Übergangsfunktion wie auf der Folie auf Papier und füllen Sie diese aus.
- 5 Kontrollieren Sie für eine Beispielfolge von Zuständen und Eingaben, dass in jedem Schritt die richtigen LEDs leuchten.

| | | | | | | |
|------------|---------|---------|---------|---------|---------|---------|
| SW1 | | | | | | |
| SW2 | | | | | | |
| Zustand | | | | | | |
| Folgezust. | | | | | | |
| LED 0 | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ |
| LED 1 | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ |
| LED 2 | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ |
| LED 3 | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ |
| LED 4 | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ |
| LED 5 | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ |
| LED 6 | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ |
| LED 7 | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ |



Aufgabe 1.3: Pseudo-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;
}
```



8. Aufgaben

- 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 | | | | | |
| LED 0 | ● ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ |
| LED 1 | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ |
| LED 2 | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ |
| LED 3 | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ |
| LED 4 | ● ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ |
| LED 5 | ● ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ |
| LED 6 | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ |
| LED 7 | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ | ○ ○ ○ ○ |



Zusatzteil



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.
}
```



Zusatzaufgabe 1.1:



```
#include <avr/io.h>
uint8_t a;           //Variablenvereinbarung
int main(){
    DDRB = 0b01010101; //Port-Initialisierung
    uint8_t a;
    while(1){        //Endlosschleife
        a =           //Einlesen Eingabewerte
           //Ausgabe
    }
}
```

- 1 Was passiert, wenn die Include-Anweisung fehlt?
- 2 Welche Pins von Port B sind Ein- und welche Ausgänge?
- 3 Ergänzen 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;           //Variablenvereinbarung
int main(){
    DDRB = 0b01010101; //Port-Initialisierung
    uint8_t a;
    while(1){        //Endlosschleife
        a = PINB;    //Einlesen Eingabewerte
        PORTB = (~a)>>1; //Ausgabe
    }
}
```

- 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 Übergangereignis alle 2 s. Programmierung mit Warteschleife.



9. Zusatzteil

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

...          Ct=0;          // Zähler von 0 bis 400000

int main(){
  DDRA  = ...          ; // PA0 und PA1 Eingänge
  DDRJ  = ...          ; // Port J Ausgänge
  PORTJ = ...          ; // Anfangsausgabewert
  while(1){           // Endlosschleife
    if (...           ) // Wenn PA1=0 und PA0=1
      ...             // Port J hochzählen
    else if (...      ) // sonst wenn PA1=1
      ...             // Port J abwärts zählen
    for (              ); // Warteschl. 2s
  }
}
```



Lösung

```
#include <avr/io.h>
uint32_t Ct=0;           // Zähler von 0 bis 400000
int main(){
  DDRA = ~0x03;         // PA0 und PA1 Eingänge
  DDRJ = 0xFF;         // Port J Ausgänge
  PORTJ = 0;           // Anfangsausgabewert
  while(1){           // Endlosschleife
    if ((PINA&0x3)==1) // Wenn PA1=0 und PA0=1
      PORTJ++;        // Port J hochzählen
    else if ((PINA&0x2)==2) // sonst wenn PA1=1
      PORTJ--;        // Port J abwärts zählen
    for (ct=0;ct<400000;ct++); // Warteschl. 2s
  }
}
```

- Projektanlegen, eingeben, übersetzen und ausprobieren.