U16 Aufgabe 5

- Besprechung Aufgabe 3
- Notation
- AVR-Timer
- Umgang mit mehreren Interruptquellen
- Arbeiten mit einer diskreten Zeitbasis

U16-1 Besprechung von Aufgabe 3 (Lauflicht)

- Alle LEDs einschalten, dann in gleicher Reihenfolge wieder ausschalten
- Richtung durch Taster *während des Ablaufs* umschaltbar

1 Lauflicht: Ansatz 1

- Abhängig von einem Richtungszustand vor- oder rückwärts laufen
- Jeweiliger Ablauf für jede Richtung in eine Funktion ausgelagert

```
char richtung=0;
while(1) {
    if(richtung == 0) { /* Lauflicht vorwärts */
        richtung = laufe_vorwaerts();
    } else { /* Lauflicht rückwärts */
        richtung = laufe_rueckwaerts();
    }
}
```

1 Lauflicht: Ansatz 1

- Absolute Werte in den Ports setzen

```
char laufe_vorwaerts() {
    /* einschalten */
    PORTB = 0x04; /* rote LED */
    wait();
    PORTB = 0x06; /* gelbe LED dazuschalten */
    wait();
    PORTB = 0x07; /* gruene LED dazuschalten */
    ...
    /* ausschalten */
    PORTB = 0x03; /* rote wieder ausschalten */
    wait();
    PORTB = 0x01; /* gelbe LED wieder ausschalten */
    wait();
    PORTB = 0x00; /* alle Ampel-LEDs jetzt aus */
    ...
    return wait(); /* Richtung zurückgeben */
}
```

- Vereinfachbar? Richtung erst änderbar, wenn alle LEDs aus!

1 Verbesserter Ansatz 1

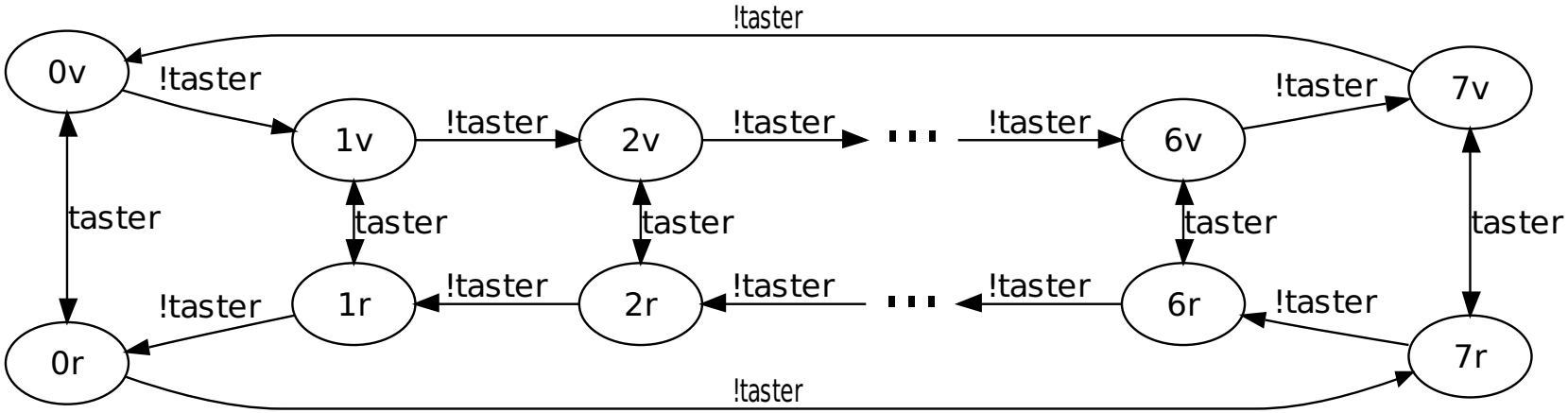
- Verwendung von *exclusive-or* (*xor*) für einzelne LEDs
  - ➤ Umschalten statt explizit ein- oder ausschalten

```
char laufe_vorwaerts() {
    /* einschalten oder ausschalten */
    PORTB ^= (1<<2); /* rote LED umschalten */
    wait();
    PORTB ^= (1<<1); /* gelbe LED umschalten */
    wait();
    PORTB ^= (1<<0); /* gruene LED umschalten */
    ...
    return wait();
}
```

- Richtung jetzt änderbar wenn alle LEDs an oder aus
- Richtung mitten im Ablauf ändern?
  - ➤ Mit diesem Ansatz nicht möglich

2 Ansatz 2: Zustandsautomat

- Ein Zustand pro LED und Richtung



- Tastendruck wechselt Richtung, aber nicht LED
- Verstreichen der Wartezeit ohne Tastendruck wechselt LED in aktueller Richtung

2 Zustandsautomat

- Direkte Umsetzung des Automaten

```
char zustand=0; /* 0-7 vorwärts, 8-15 rückwärts */

switch(zustand) {
    case 0: /* Rote LED, vorwärts */
        PORTB ^= (1<<2); zustand=wait()?8:1; break;
    ...
    case 7: /* LED an Port D, Pin 0 */
        PORTD ^= (1<<0); zustand=wait()?15:0; break;
    case 8: /* Rote LED, rückwärts */
        PORTB ^= (1<<2); zustand=wait()?15:0; break;
    case 9: /* Gelbe LED, rückwärts */
        PORTB ^= (1<<1); zustand=wait()?8:1; break;
    ...
}
```

- `wait()` in jedem Fall identisch
- Zustandsänderung berechenbar

2 Berechnung des Folgezustandes

```
char zustand=0; /* 0-7 vorwärts, 8-15 rückwärts */
```

```
char nr=wait();
if((nr==0&&zustand>=8) || (nr==1&&zustand<=7)) { /* Richtungsänderung */
    zustand = (zustand+8) % 16;
} else {
    if(zustand<=7) { zustand = (zustand+1)%8; } /* vorwärts */
    else { zustand = ((zustand-1) % 8)+8; } /* rückwärts */
}
switch(zustand) {
    case 0: /* Rote LED, vorwärts */
    case 8: /* Rote LED, rückwärts */
        PORTB ^= (1<<2); break;
    ...
```

- Vorwärts- und Rückwärtszustände im switch identisch
- Richtungszustand in eigener Variable halbiert Wertebereich von `zustand`
  - ➤ `zustand=0..7` beschreibt aktuelle LED-Position

2 Richtung in eigener Variable

- Richtung wird getrennt von der LED-Position betrachtet

```
char zustand=0; /* 0-7, aktuelle LED */
char richtung=0; /* 0=vorwärts, 1=rückwärts */

if(wait() != richtung) { /* Richtungsänderung */
    richtung ^= 1;
    zustand = (zustand+8) % 16;
} else {
    if(richtung==0) { zustand = (zustand+1)%8; } /* vorwärts */
    else { zustand = (zustand+8-1) % 8; } /* rückwärts */
}
switch(zustand) {
    case 0: /* Rote LED */
        PORTB ^= (1<<2); break;
    ...
```

- Eine Variante für eine gute Lösung

3 Pollen des Tasters in der Warteschleife

```
char wait() {
    static char richtung=0;

    unsigned int len;
    for(len=0; len<5000; len++) {
        if( (PIND & (1<<2)) == 0) { /* Taster gedrückt */


                richtung ^= 1; /* Richtung ändern */



        }
    }
    return richtung;
}
```

- Funktion kennt immer die aktuelle Richtung (wie?)
- Problem?

3 Probleme durch Polling

- Programm muss Flankenerkennung selbst implementieren
  - ➤ um Mehrfacherkennung desselben Tastendrucks zu verhindern
  - ➤ Loslassen des Tasters muss explizit erkannt werden
  - ➤ auch über die Dauer der Warteschleife hinweg
- Tastendrücke können leicht verloren gehen
  - ➤ wenn Tastendruck außerhalb der Warteschleife erfolgt

3 Pollen des Tasters in der Warteschleife

```
char wait() {
    static char richtung=0;
    static char vt=1; /* Vorheriger Tasterzustand*/
    unsigned int len;
    for(len=0; len<5000; len++) {
        if( (PIND & (1<<2)) == 0) { /* Taster gedrückt */
            if(vt==1) {
                vt=0;
                richtung ^= 1; /* Richtung ändern */
            }
        } else { /* Taster nicht gedrückt */
            vt=1;
        }
    }
    return richtung;
}
```

- Wartefunktion merkt sich vorherigen Zustand des Tasters
- Richtungsänderung nur bei fallender Flanke (des Tasters und von `vt`)

4 Alternative Lösungsmöglichkeit

- Geschickte Speicherung von Port-Adressen und Pin-Positionen in Arrays
- Zustandsvariablen analog zur ersten Variante

```
static volatile unsigned char *ports[] = {
    &PORTB,&PORTB,&PORTB,&PORTD,&PORTD,&PORTD,&PORTD,&PORTD
};
static unsigned char pins[] = {
    2,1,0,7,6,5,1,0
};
```

```
while(1) {
    if(wait() != richtung) { /* Richtungsänderung */
        ...
    }
    /* LED umschalten */
    *ports[zustand] ^= (1<<pins[zustand]);
}
```

- Arrays leicht um weitere LEDs erweiterbar

U16-2 Notation

- Einige Einstellungen werden mit mehreren Bits konfiguriert
  - ➤ logisch zusammengehörig, im Register aber nicht unbedingt benachbart
  - ➤ evtl. sogar in verschiedenen Registern!
  - ➤ die Bits müssen also *einzeln* manipuliert werden
- Um mehrere Bits zu referenzieren, verwenden wir in den Folien die Schreibweise `Makroname`*`Bitnummer:Bitnummer`*
  - ➤ Beispiel: `WGM13:0` entspricht den Bits `WGM13`, `WGM12`, `WGM11`, `WGM10`
- Um eine bestimmte Besetzung dieser Bits zu beschreiben, schreiben wir
  - ➤ `WGM13:0 = 0b0100` für `WGM13=0, WGM12=1, WGM11=0, WGM10=0`
  - ➤ entspricht also der Binärschreibweise unter der Annahme, dass die Bits zusammengehörig sind
- Keine dieser Schreibweisen ist im C-Code gültig: Hier müssen die Makronamen für die einzelnen Bitpositionen verwendet werden
  - ➤ `WGM13` ist die Position des `WGM13`-Bits im Register

U16-3 Timer-Einheiten der AVR-Mikrokontroller

- Timer: Zählregister `TCNTx`, das mit jedem Takt hochgezählt wird
  - 8-bit und 16-bit Timer
  - interner Takt mit oder ohne Prescaling oder externer Taktgeber
- Verschiedene Untereinheiten
  - Input Capture Unit (kann z.B. Rauschen filtern)
  - Output Compare Unit
- Verschiedene Betriebsmodi
- Timer des ATmega8
  - Timer0: 8-bit Timer
  - Timer1: 16-bit Timer mit Pulsbreiten-Modulator (PWM)
  - Timer2: 8-bit Timer mit Pulsbreiten-Modulator (PWM)
- Wir verwenden Timer1

1 Taktung des Timers

- Einstellbar durch Prescaler (`CS12:0`-Bits im `TCCR1B`-Register)

CS12	CS11	CS10	Taktung
0	0	0	Kein Takt (Timer gestoppt)
0	0	1	I/O-Takt (kein Prescaler)
0	1	0	I/O-Takt/8
0	1	1	I/O-Takt/64
1	0	0	I/O-Takt/256
1	0	1	I/O-Takt/1024
1	1	x	Externer Taktgeber

- Die durch Hardware realisierbaren Zeitspannen sind begrenzt
  - ➤ nach unten durch die maximale Taktfrequenz des Timers
  - ➤ nach oben durch Breite des Timerregisters und minimalen Prescaler
  - ➤ längere Zeitintervalle können durch die Anwendung realisiert werden

I/O-Takt: 1 MHz	Min.	Max.
8-bit	1us	ca. 261ms
16-bit	1us	ca. 67s

  - ➤ maximale Genauigkeit erreicht man durch höchst-mögliche Taktung

2 Timer-Register

- Konfiguration der Timereinheit *x=0,1,...* erfolgt über Register
- Timerkontrollregister
  - `TCCRx` (8-bit Timer)
  - `TCCRxA` und `TCCRxB` (16-bit Timer)
- Zählregister `TCNTx`
- Output Compare Register `OCRx`
  - ➤ manche Timer haben mehr als eine Output-Compare-Einheit
  - ➤ hier ist das Register für Einheit *i=A,B,...*: `OCRxi`
- Interrupt Mask Register `TIMSK`
- Interrupt Flag Register `TIFR`
- (Input Capture Register `ICRx`)

3 Output-Compare-Einheiten

- Vergleichen Zählerstand (`TCNT`*x*) bei jedem Inkrement mit einem Vergleichswert (`OCR`*xi*)
- Stimmen die Werte überein (Compare Match), wird das `OCF`*xi*-Flag im `TIFR`-Register gesetzt
  - ➤ Abfrage des Flags durch Polling
  - ➤ Interruptgenerierung durch Setzen des `OCIE`*xi*-Bits in `TIMSK`
  - ➤ Das avr-libc-Makro für den Interruptvektor lautet `TIMER`*x*`_COMP`*i*`_vect`
- Timer1 des ATmega8 hat zwei Output-Compare-Einheiten A und B

4 Timer-Betriebsmodi des Timer1 des ATmega8

- Normal Mode
  - ➤ Timer wird inkrementiert bis zum Überlauf
  - ➤ Interruptmöglichkeit bei Timerüberlauf
- Diverse PWM-Modi
- Clear Timer on Compare Match (CTC) Mode
  - ➤ Verwendet Output-Compare-Einheit A
  - ➤ Vergleichswert im *Output Compare Register 1A* `OCR1A`
  - ➤ Rücksetzung von `TCNT1` bei *Compare Match* (`TCNT1==OCR1A`)
  - ➤ Interruptmöglichkeit bei *Compare Match*
- Moduskonfiguration: `WGM13:2`-Bits (`TCCR1B`) und `WGM11:0`-Bits (`TCCR1A`)
  - ➤ Normal-Mode: `WGM13:0=0b0000`
- CTC-Mode: `WGM13:0=0b0100`

5 Konfigurationsbeispiel

- Mit Timer1 soll alle **t**=100ms ein Interrupt generiert werden
- Der I/O-Takt beträgt **f**=1 MHz
- Erreichbar mit CTC-Modus
- Wie bestimme ich einen geeigneten Prescaler **P** und Compare-Wert **C**?
  - ➤ **C = t * f * P**
  - ➤ Beginne mit möglichst großem **P** (1, 1/8, 1/64, ...)
  - ➤ Bis **C** in das `OCRxi`-Register passt (bei 16-bit Timer: **C**<65536)

P	Rechnung	C
1	**C**=0,1s * 1000000Hz * 1	100000
1/8	**C**=0,1s * 1000000Hz * 1/8	12500

- Nicht jede Zeitspanne kann genau konfiguriert werden, aber durch Nutzung des größt-möglichen Prescalers kann die Genauigkeit maximiert werden

5 Konfigurationsbeispiel

- Mit Timer1 soll alle **t**=100ms ein Interrupt generiert werden
- Auslösezeit: **t = C / P / f**

```
/* CTC-Modus, Prescaler P=1/8, Compare Wert C=12500 */
/* (12500 / (1/8)) / 1000000Hz = 0,1s = 100ms */
TCCR1B |= (1<<CS11); /* Prescaler 1/8: CS12:0 = 0b010 */
TCCR1B &= ~((1<<CS12) | (1<<CS10));

TCCR1B &= ~(1<<WGM13); /* CTC Mode: WGM13:0 = 0b0100 */
TCCR1B |= (1<<WGM12);
TCCR1A &= ~((1<<WGM11) | (1<<WGM10))

TCNT1 = 0; /* Zählerstand zurücksetzen */
OCR1A = 12500; /* Vergleichswert setzen */
TIMSK |= (1<<OCIE1A); /* OC-Interrupt 1A aktivieren */
```

```
ISR(TIMER1_COMPA_vect) { /* tue etwas */ }
```

U16-4 Umgang mit mehreren Interruptquellen

- Interrupts von Taster und Timer

```
cli();
while(...) {
    sleep_enable();
    sei();
    sleep_cpu();
    sleep_disable();
    cli();
}
sei();
```

- Welcher Interrupt (welches Ereignis) hat den sleep-Modus unterbrochen?
  - ➤ Bestimmung durch die Anwendung
  - ➤ Sleep-Bedingung muss in einer Schleife abgeprüft werden, da das Aufwachen evtl. durch ein Ereignis, welches nicht mit der Wartebedingung zusammenhängt, verursacht wurde

U16-5 Arbeiten mit einer diskreten Zeitbasis

- Fester Takt kann Anwendung mit diskreter Zeitbasis versorgen
  - ➤ Anwendung kann die aktuelle Zeit abfragen
  - ➤ Zeitbasis ist das Zählregister
  - ➤ für gröbere Zeitschritte eine Variable des Programms
- Takt ausreichend hoch für den Bedarf der Anwendung
  - ➤ die abfragebare Zeit ist nicht kontinuierlich
  - ➤ diskret in Stufen 1/Takt
- Schutz des Zählers vor Überlauf
  - ➤ relative Zeiten betrachten (vergangene Zeit *seit einem Ereignis*)
  - ➤ Zähler zurücksetzen, wenn ein neues Ereignis betrachtet wird
  - ➤ maximal benötigten Zählstand vorher bestimmen und Zähler entsprechend schützen (nicht möglich, wenn Zähler ein Zählregister ist)

1 Beispiel

- Anwendung benötigt Zeitschritte von 100ms
- Bei uns im Hardwarezähler nicht realisierbar:
  - ➤ bei Prescaler 1/1024 Schritte von etwa 1ms
  - ➤ Softwarezähler für gröbere Schritte

```
#define MAXTIME 255 /* max. Wert des 8-bit Zählers */
static volatile unsigned char time=0; /* Softwarezähler */

ISR(TIMER1_COMPA_vect) {
    if(time < MAXTIME) { time++; }
}

void wait(unsigned char len) {
    time=0; /* Zeit ab jetzt betrachten */
/* Schlafe, bis len/10 Sekunden verstrichen sind */
    ...
    while(time < len) { /* ...sleep... */ }
    ...
}
```