

# Teil 6 – C-Programmierung auf Mikrocontrollern

6.1. Datentypen mit definierter Länge

6.2. Bit-Operationen

6.3. Volatile

6.4. Ports und Register

```
// -----  
// Timer-Interrupt: Wenn der Sinus-Modus aktiv ist, regelt diese Interrupt-  
// Funktion die Ansteuerung des DA-Wandlers und ggf. des Trigger-Ausgangs.  
// -----  
ISR (TIMER0_COMPA_vect)  
{  
    PORTB = next_sin_value; // Nächsten Sinus-Stützpkt. an DA-Wandler senden  
    PORTD = next_dig_value; // Falls gewünscht, ggf. Triggersignal ausgeben  
  
    cbi(PORTD, DAC_ENABLE); // Chip-Enable-Signal für DA-Wandler generieren  
    asm volatile ("nop"); // (...Spannung kurz runter und wieder rauf...)  
    sbi(PORTD, DAC_ENABLE);  
  
    // Die beim nachfolgenden Timer-Interrupt auszugebenden Werte  
    // (next_sin_value, next_dig_value) werden hier schon vorbereitet,  
    // dann stehen sie beim nächsten Interrupt gleich zur Verfügung!  
    ++sin_index;  
    next_sin_value = pgm_read_byte_near(sin_table + sin_index);  
    if(!next_sin_value)  
    {  
        sin_index = 0;  
        next_sin_value = pgm_read_byte_near(sin_table + sin_index);  
        if(trigger_on) next_dig_value ^= (1<<DIG_SIGNAL);  
    }  
}
```

# Beispiel aus dem Praktikum

Demoprogramm zum Senden über die serielle Schnittstelle

```
//          +----- Startbit
//          | +----- Datenbit 0
//          | | +----- Datenbit 1
//          | | | +----- Datenbit 2
//          | | | | +----- Datenbit 3
//          | | | | | +----- Datenbit 4
//          | | | | | | +----- Datenbit 5
//          | | | | | | | +----- Datenbit 6
//          | | | | | | | | +----- Datenbit 7
//          | | | | | | | | | +----- Stoppbit
//          | | | | | | | | |
//          | | | | | | | | |
//          | | | | | | | | |
int bits_zum_senden[] = { 0, 0, 0, 0, 1, 1, 0, 1, 0, 1,
                        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, // kleine Pause...
                        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, // kleine Pause...
                        -1 }; // Ende der Übertragung - zurück zum Anfang
```



## 5.2.4.2.1 Sizes of integer types <limits.h>

expression that is an object of the corresponding type converted according to the integer promotions. Their implementation-defined values shall be equal or greater in magnitude (absolute value) to those shown, with the same sign.

— number of bits for smallest object that is not a bit-field (byte)

**CHAR\_BIT** 8

— minimum value for an object of type **int**

**INT\_MIN** -32767 //  $-(2^{15} - 1)$

— maximum value for an object of type **int**

**INT\_MAX** +32767 //  $2^{15} - 1$

— maximum value for an object of type **unsigned int**

**UINT\_MAX** 65535 //  $2^{16} - 1$

---

## 6.2.5 Types

There are five *standard signed integer types*, designated as **signed char**, **short int**, **int**, **long int**, and **long long int**. (These and other types may be designated in several additional ways as described in 6.7.2.) There may also be

For each of the signed integer types, there is a corresponding (but different) unsigned integer type (designated with the keyword **unsigned**) that uses the same amount of storage (including sign information) and has the same alignment requirements. The type

## Ausschnitt aus stdint.h

Integer-Typen für AVR-8-Bit-Mikrocontroller

```
[-] /*  
 * ISO/IEC 9899:1999 7.18 Integer types <stdint.h>  
 */  
  
typedef signed char int8_t;  
typedef unsigned char uint8_t;  
typedef signed int int16_t;  
typedef unsigned int uint16_t;  
typedef signed long int int32_t;  
typedef unsigned long int uint32_t;
```

## Beispiel aus dem Praktikum

LED am Ausgang PB5 ein- und ausschalten

```
int main(void)
{
    DDRB = 0b00100000; // PB5 als Ausgang konfigurieren

    while(1)
    {
        PORTB = 0b00100000; // ein
        _delay_ms(500);

        PORTB = 0b00000000; // aus
        _delay_ms(500);
    }
}
```

# C++ Operator Precedence

Die folgende Tabelle zeigt Ausführungspriorität und Assoziativität der C++-Operatoren. nach unten von vorrangiger zu nachrangiger Priorität hin geordnet.

| Priorität | Operator | Beschreibung                             |
|-----------|----------|--|
| <b>1</b>  | ::       | Bereichsauflösung                        |
|           | ++ --    | Suffix-/Postfix-Inkrement und -Dekrement |
|           | ()       | Funktionsaufruf                          |
|           | []       | Array-Indizes                            |
|           | + -      | Addition und Subtraktion                 |
| <b>7</b>  | << >>    | bitweise Rechts- und Linksverschiebung   |
| <b>8</b>  | < <=     | kleiner-als und kleiner-gleich           |
|           | > >=     | größer-als und größer-gleich             |
| <b>9</b>  | == !=    | gleich und ungleich                      |
| <b>10</b> | &        | bitweises AND                            |
| <b>11</b> | ^        | bitweises XOR (entweder-oder)            |
| <b>12</b> |          | bitweises OR (ein oder beide)            |
| <b>13</b> | &&       | logisches AND                            |
| <b>14</b> |          | logisches OR                             |

Quelle: [1]

## Beispiel aus dem Praktikum (a)

Timer 0 konfigurieren: ca. 20 Timer-Interrupts pro Sekunde

| Bit           | 7   | 6      | 5      | 4     | 3    | 2     | 1     | 0   |        |        |        |        |       |      |       |       |        |
|---------------|---|--------|--------|-------|------|-------|-------|-----|--------|--------|--------|--------|-------|------|-------|-------|--------|
|               | <table border="1"><tr><td>COM0A1</td><td>COM0A0</td><td>COM0B1</td><td>COM0B0</td><td>-</td><td>-</td><td>WGM01</td><td>WGM00</td></tr></table> |        |        |       |      |       |       |     | COM0A1 | COM0A0 | COM0B1 | COM0B0 | -     | -    | WGM01 | WGM00 | TCCR0A |
| COM0A1        | COM0A0  | COM0B1 | COM0B0 | -     | -    | WGM01 | WGM00 |     |        |        |        |        |       |      |       |       |        |
| Read/Write    | R/W   | R/W    | R/W    | R/W   | R    | R     | R/W   | R/W |        |        |        |        |       |      |       |       |        |
| Initial Value | 0   | 0      | 0      | 0     | 0    | 0     | 0     | 0   |        |        |        |        |       |      |       |       |        |
|               | <table border="1"><tr><td>FOC0A</td><td>FOC0B</td><td>-</td><td>-</td><td>WGM02</td><td>CS02</td><td>CS01</td><td>CS00</td></tr></table>        |        |        |       |      |       |       |     | FOC0A  | FOC0B  | -      | -      | WGM02 | CS02 | CS01  | CS00  | TCCR0B |
| FOC0A         | FOC0B   | -      | -      | WGM02 | CS02 | CS01  | CS00  |     |        |        |        |        |       |      |       |       |        |
| Read/Write    | W   | W      | R      | R     | R/W  | R/W   | R/W   | R/W |        |        |        |        |       |      |       |       |        |
| Initial Value | 0   | 0      | 0      | 0     | 0    | 0     | 0     | 0   |        |        |        |        |       |      |       |       |        |

```
// PB3 als Ausgang konfigurieren
```

```
DDRB = 0b00001000;
```

```
// Timer 0 konfigurieren
```

```
TCCR0A = 0b00000010; // CTC Modus --> WGM01-Bit setzen
```

```
TCCR0B = 0b00000101; // Prescaler: 1024 --> CS00- und CS02 setzen
```

```
OCR0A = 58; // Vergleichswert für Timer
```

```
// Compare Interrupt erlauben
```

```
TIMSK0 = 0b00000100; // OCIE0A-Bit setzen
```



## Beispiel aus dem Praktikum (b)

Timer 0 konfigurieren: ca. 20 Timer-Interrupts pro Sekunde

```
// PB3 als Ausgang konfigurieren
DDRB |= _BV(PB3);

// Timer 0 konfigurieren
TCCR0A |= _BV(WGM01);           // CTC Modus ("Clear Timer on Compare Match")
TCCR0B |= _BV(CS00) + _BV(CS02); // Prescaler: 1024
OCR0A = 58;                     // Vergleichswert für Timer

// Compare Interrupt erlauben
TIMSK0 |= _BV(OCIE0A);
```

## Beispiel

Variable "millis" wird im Interrupt und im Hauptprogramm genutzt

```
volatile int64_t millis = 0; // Millisekunden seit Programmstart
ISR(TIM0_COMPA_vect) // Timer Compare Match Interrupt
{
    ++millis;
}
int main(void)
{
    int seconds;
    ...
    while(1)
    {
        // Achtung: Timer-Interrupts kurz deaktivieren, damit "millis"
        // nicht während (!) der folgenden Berechnung verändert wird!
        cli(); seconds = millis / 1000; sei();
        if(seconds % 2 == 0)
            PORTB = 0b00001000;
        else
            ...
    }
}
```

# ATTiny13

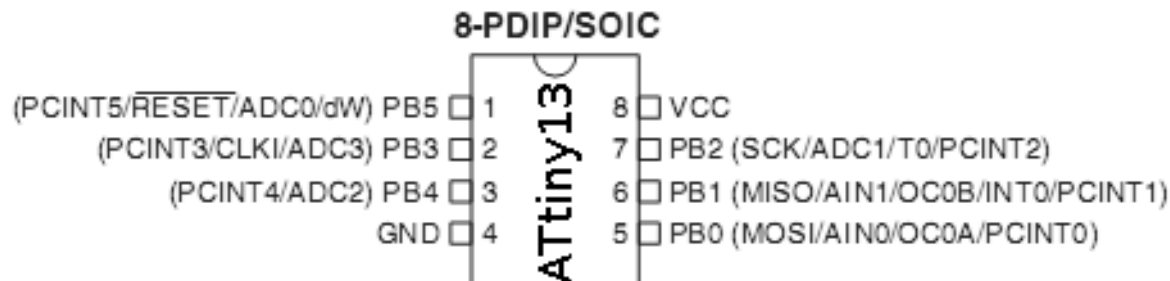
Übersicht über alle Register, siehe Datenblatt S. 157

## Register Summary

| Address | Name     | Bit 7 | Bit 6 | Bit 5  | Bit 4  | Bit 3  | Bit 2  | Bit 1  | Bit 0  |
|---------|----------|-------|-------|--------|--------|--------|--------|--------|--------|
| 0x3F    | SREG     | I     | T     | H      | S      | V      | N      | Z      | C      |
| 0x3E    | Reserved | -     | -     | -      | -      | -      | -      | -      | -      |
| SP[7:0] |          |       |       |        |        |        |        |        |        |
| 0x1A    | Reserved | -     |       |        |        |        |        |        |        |
| 0x19    | Reserved | -     |       |        |        |        |        |        |        |
| 0x18    | PORTB    | -     | -     | PORTB5 | PORTB4 | PORTB3 | PORTB2 | PORTB1 | PORTB0 |
| 0x17    | DDRB     | -     | -     | DDB5   | DDB4   | DDB3   | DDB2   | DDB1   | DDB0   |
| 0x16    | PINB     | -     | -     | PINB5  | PINB4  | PINB3  | PINB2  | PINB1  | PINB0  |
| 0x15    | PCMSK    | -     | -     | PCINT5 | PCINT4 | PCINT3 | PCINT2 | PCINT1 | PCINT0 |



Das Register PORTB liegt an Adresse 0x18



Quelle: [2]

## pgmspace.h

Statische Daten im Programm-Flash ablegen

```
const uint8_t sinus_100_tab[] PROGMEM =
{
    // Vorberechnete Sinustabelle mit 1000 Stützpunkten --> 100 Hz
    35,  35, 195, 195,  67,  67, 131, 131,   3,   3,
    93,  93, 157, 157,  29,  29, 237, 237, 109, 109,
    .....
    19,  19,  19, 227, 227,  99,  99, 163, 163, 163,   0
};
// Ende der Tabelle: -----^

uint8_t next_sin_value = 0; uint16_t sin_index = 0;

// Timer-Interrupt, 100 kHz
ISR (TIMER0_COMPA_vect)
{
    PORTB = next_sin_value; // Nächster Sinuswert zum DA-Wandler
    next_sin_value = pgm_read_byte_near(sin_table + ++sin_index);
    if(!next_sin_value)
    {
        sin_index = 0; // Tabellenende? --> Zurück zum Anfang...
        next_sin_value = pgm_read_byte_near(sin_table + sin_index);
    }
    . . . .
}
```

## Quellenverzeichnis

- [1] [de.cppreference.com](http://de.cppreference.com): Sprache, Operator-Vorrang (Stand: 10.05.2016)
- [2] [www.atmel.com](http://www.atmel.com): Datenblatt ATtiny13 (Stand: 12.04.2016)