



## 2 Timer

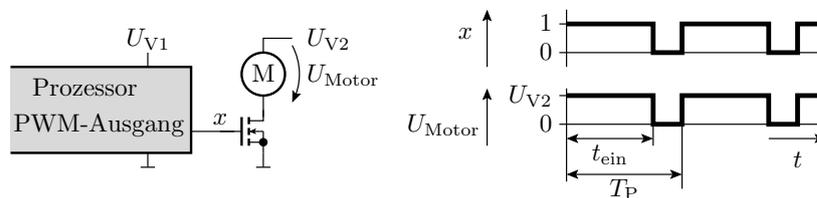
### Timer

Ein Timer ist eine Hardware-Einheit aus Zähl-, Vergleichs-, Konfigurationsregistern, ... zur

- Erzeugung von Wartezeiten,
- zeitgesteuerten Ereignisabarbeitung,
- Erzeugung pulswidenmodulierter (PWM-) Signale und
- Pulsweitenmessung.

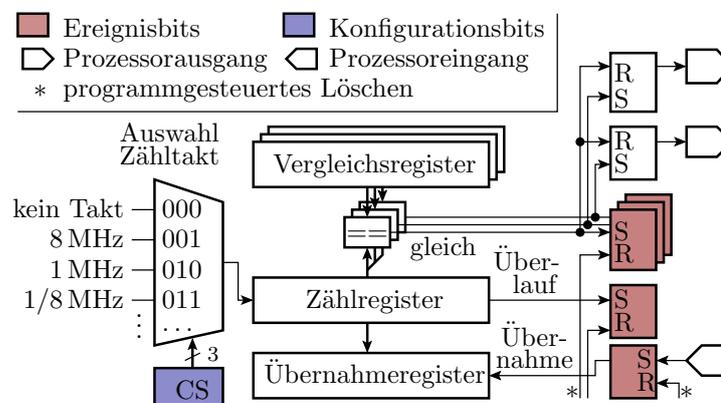
PWM-Signale dienen

- zur Informationsübertragung z.B. an Modellbauservos und
- zur stufenlosen Leistungssteuerung, z.B. unserer Motoren.



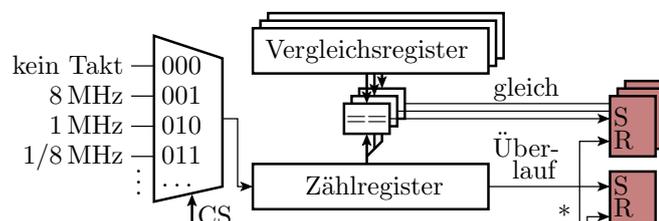
### 2.1 Funktionsweise

#### Aufbau und Funktionsweise eines Timers



- Kern eines Timers ist ein Zählregister mit einem vom Programm zuschaltbaren programmierbaren Takt.
- Die Ereignisbits (Überlauf, Gleichheit, externe Flanke) sind vom Programm les- und löschar.

#### Normalmodus

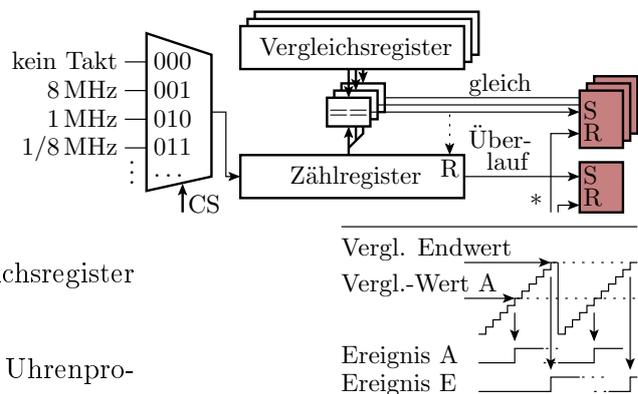


- Zählregister zählt zyklisch bis zum Überlauf.
- Beim Überlauf wird ein Überlaufbit und bei Gleichheit mit einem Vergleichsregister ein Vergleichsereignisbit gesetzt.
- Beispiel Wartefunktion:

```
void wait(uint32_t tw){
    <berechne und setze Takt und Vergleichswert>
    <Lösche Zähler und Vergleichsereignisbit>
    <warte bis Vergleichsereignisbit==1>
    <schalte Zähltakt aus> }

```

**CTC- (Clear Timer on Compare Match) Modus**

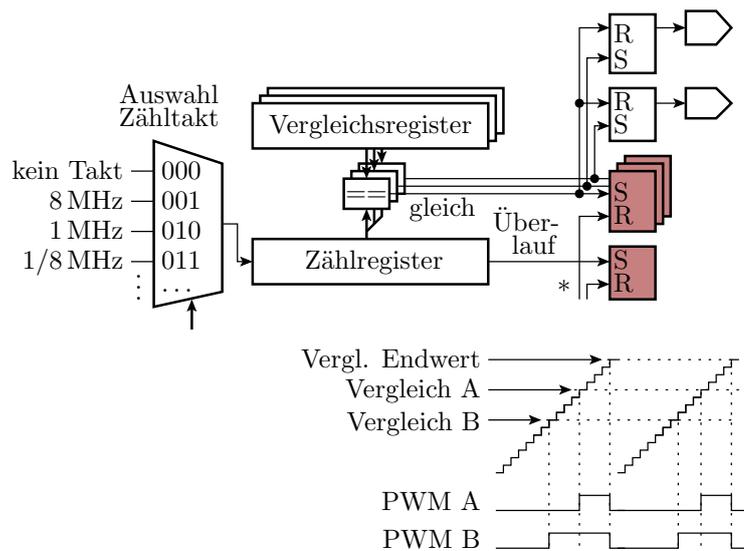


- Zähler wird bei Gleichheit mit Vergleichsregister rückgesetzt.
- Auslösung zyklischer Ereignisse, z.B. Uhrenprozess:

```
void Schrittfunktion Uhr(){
    if (<Vergleichs-Rücksetz-Ereignis>)
        <lösche Ereignisbit, schalte Uhr weiter>
}

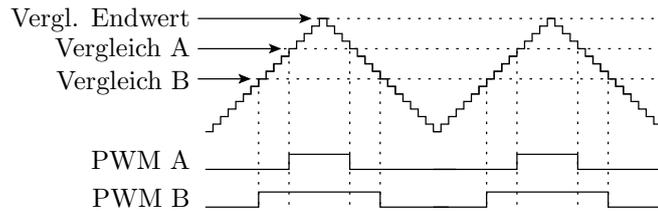
```

**PWM-Erzeugung**



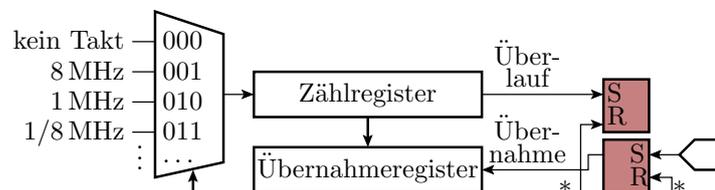
- Vergleichsereignis setzt Zählerrücksetzereignis (Überlauf oder CTC) löscht Ausgabe.
- Pulsenergieung z.B. zur Motoransteuerung ohne Schrittfunktion.

## Symmetrische PWM<sup>1</sup>



- Endvergleichswert schaltet die Zählrichtung um.
- Bei Gleichheit und Hochzählen wird die Ausgabe ein- und bei Gleichheit und Abwärtszählen ausgeschaltet.
- Bei dieser und der vorherigen PWM kann auch eine invertierte Ausgabe programmiert werden, so dass der Vergleichswert statt der Ausschalt-, die Einschaltzeit festlegt.

## Pulsweitenmessung



- Externes Ereignis (Schaltflanke) bewirkt Übernahme des Zählwerts in das Übernahmeregister.
- Programmgesteuerte Differenzbildung der Übernahmewerte zwischen den Übernahmeereignissen.

Der Zeitmessmodus von Timern wird in dieser Veranstaltung nicht genutzt.

## Timer des ATmega2560

- Zwei 8-Bit Timer (0 und 2).
- Vier 16-Bit-Timer (1, 3, 4 und 5).

Die Bit-Anzahl beschreibt die Größe der Zähl- und Vergleichsregister.

Nutzung der Timer in den Beispielprojekten:

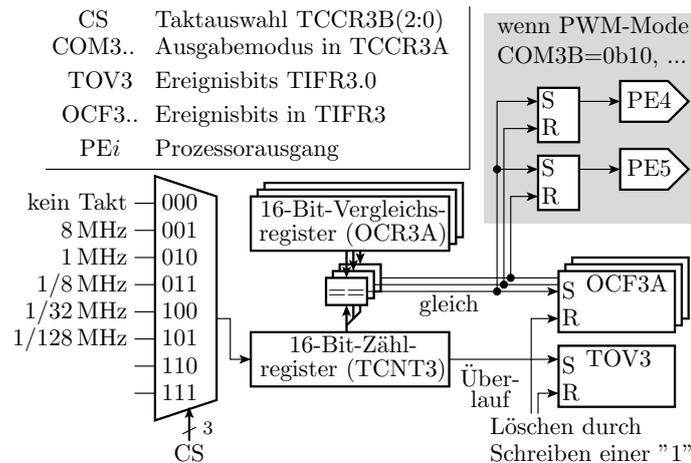
- Timer 0: Treiber »wegmess« Abtastintervall.
- Timer 1: Treiber »comir\_tmr« Programmuhr und Wartezeitähler.
- Timer 3:
  - Timer- und Interrupt-Experimente.
  - Treiber »comir\_PC« Empfangs-Timeout.
- Timer 5: Treiber »pwm« Motor-PWM.

Die ungenutzten Timer 2 und 4 sind noch frei für andere Aufgaben, z.B. als Timeout-Zähler für den Bluetooth-Empfang.

<sup>1</sup>Im Datenblatt unseres Prozessors ist das die phasenrichtige und die vorhergehende normale PWM die schnelle (Fast-) PWM.

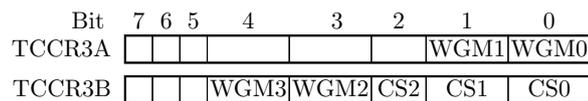
## 2.2 Timer 3

### Timer 3: 16-Bit, Normal-, CTC-, PWM-Mode



- Modusauswahl: WGM(3:0) in TCCR3A und TCCR3B.

### Betriebsarten (Auswahl)



WGM	Betriebsart	max. Zählwert
0b0000	normal	0xFFFF
0b0100	CTC	OCR3A
0b0001	sym. PWM <sup>(*1)</sup> , 8 Bit	0x00FF
0b0011	sym. PWM <sup>(*1)</sup> , 10 Bit	0x03FF
0b1011	sym. PWM <sup>(*1)</sup> , OCR	OCR3A
0b0101	fast PWM <sup>(*2)</sup> 8 Bit	0x00FF
0b0111	fast PWM <sup>(*2)</sup> 10 Bit	0x03FF
0b1111	fast PWM <sup>(*2)</sup> , OCR	OCR3A

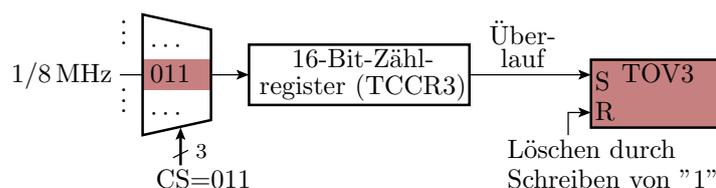
(\*1) symmetrische oder phasenausgerichtete PWM.

(\*2) Schnelle oder normale PWM.

## 2.3 Experimente

### Normalmodus, LED mit Timer hochzählen

- Timer im Normalmodus (WGM(3:0)=0) und CS=011:



- Bei jedem Überlauf des Zählregisters nach  $2^{16}$  Zählritten, Überlaufereignisbit löschen und LED-Ausgabe weiterzählen. LED-Zählfrequenz:

$$f_{LED} = \frac{\frac{1}{8} \text{ MHz}}{2^{16}} = 1,9 \text{ Hz}$$



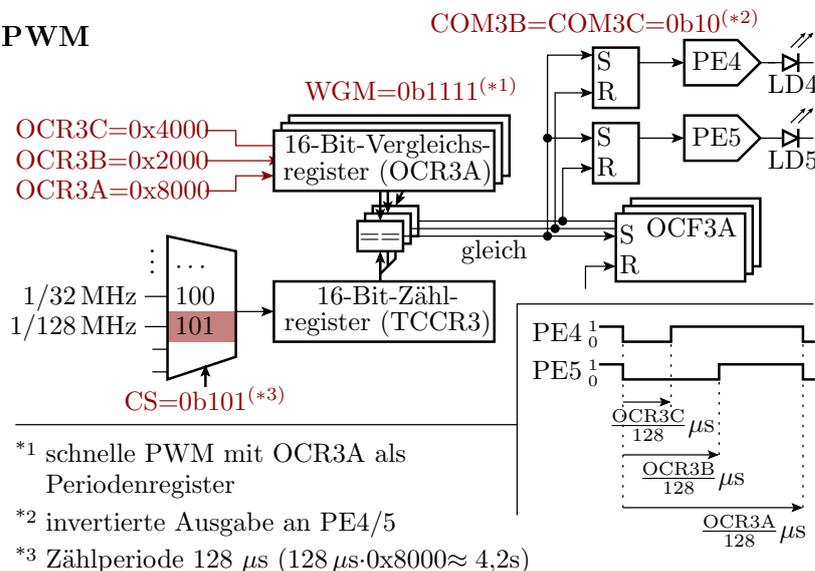
Testprogramm:

- Timer und LED-Ausgabe initialisieren.
- Wiederhole immer
  - Warte, bis Vergleichsbit »OCF3A« gesetzt.
  - LED-Ausgabe weiterzählern.
  - Vergleichsbit »OCF3A« löschen.
  - neuen Vergleichswert aus der Schaltereingabe bestimmen und in OCR3A schreiben.

```
#include <avr/io.h>
int main(void){ // Schaltermodul an JA oben
  TCCR3A = 0; // WGM3[1:0] = 0
  TCCR3B = 0b1101; // WGM3[3:2] = 1, CS3=0b101
  DDRJ = 0xFF;
  OCR3A = (1<<(PINA&0xF))+8; // Vergleichswert
  while(1){
    if (TIFR3 & (1<<OCF3A)){// Warte auf Gleichheit
      PORTJ++; // Erhöhe Led-Ausgabe
      TIFR3 = (1<<OCF3A); // Lösche Vergleichsbit
      OCR3A = (1<<(PINA&0xF))+8; // neuer Vgl.-Wert
    }
  }
}
```

- Im Projekt F8-test\_time\test\_timer alle außer zweite Main-Funktion auskommentieren.
- Schaltermodul an Port A oben anstecken. SW(4:1)=1100.
- Übersetzen. Start ( ,  ). Kontrolle Zähltakt  $\approx 2$  Hz.
- Schalterwert erhöhen/verringern und Frequenz kontrollieren.

### Experiment: PWM



- LD4, LD5: LEDs auf PMOD8LD an JE

Testprogramm:

- Timer initialisieren.
  - Endlosschleife, die nichts tun muss.
- 
- LED-Modul »PMOD8LD« an JE<sup>2</sup>.
  - Im Projekt F8-test\_timer\test\_timer alle außer dritte Main-Funktion auskommentieren.
  - Übersetzen. Start im Debugger . Continue .
  - Kontrolle:
    - Blinkperiode:  $\frac{0x8000}{128} \mu s \approx 2,56 \text{ s}$
    - Ausschaltzeit LED4 25%:  $\frac{0x2000}{128} \mu s \approx 0,64 \text{ s}$
    - Ausschaltzeit LED5 50%:  $\frac{0x4000}{128} \mu s \approx 1,28 \text{ s}$
  - Anhalten . Unterbrechungspunkt siehe nächste Folie setzen. Continue  bis  und Kontrolle der SFR-Werte.
  - Ausprobieren mit anderen Haltepunkten, Pulsbreiten, ...

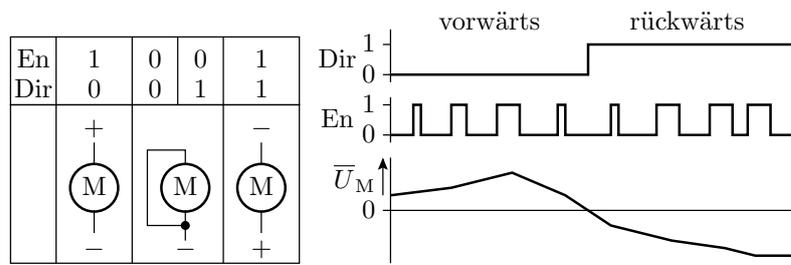
```

#include <avr/io.h>
int main(void){
    // Aktivierung der PWM-Ausgänge
    TCCR3A = 0b10<<COM3B0 | 0b10<<COM3C0 | 0b11;
    TCCR3B = 0b11101; // WGM3[3:2] = 0b11, CS3=0b101
    OCR3A = 0x8000; // Zählperiode
    OCR3B = 0x2000; // PE4 ein nach 25% Periode
    OCR3C = 0x4000; // PE5 ein nach 50% Periode
    DDRE = 0xFF;
    while(1){
        if (TIFR3 & (1<<OCF3A))
            TIFR3 = (1<<OCF3A);
        if (TIFR3 & (1<<OCF3B))
            TIFR3 = (1<<OCF3B);
        if (TIFR3 & (1<<OCF3C))
            TIFR3 = (1<<OCF3C);
    }
}

```

<sup>2</sup>Ausgabe PE4 an LD4 und PE5 an LD5.



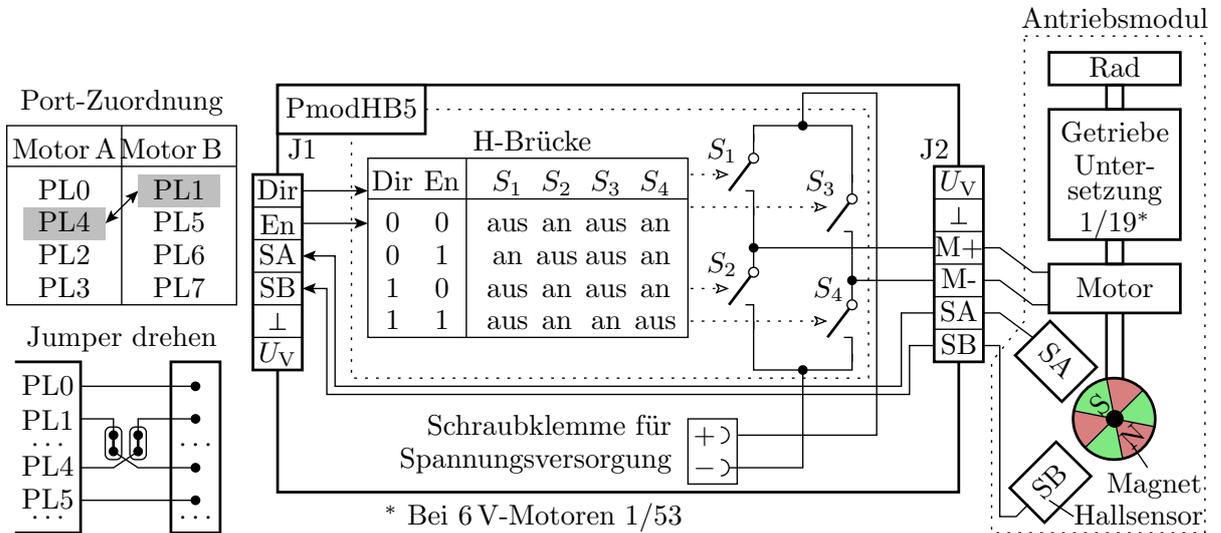


An den Antriebsbaugruppen erfolgt die Einstellung

- der Drehrichtung über ein Richtungsbit Dir und
- der relativen Pulsbreite mit dem En- (Enable-) Signal.

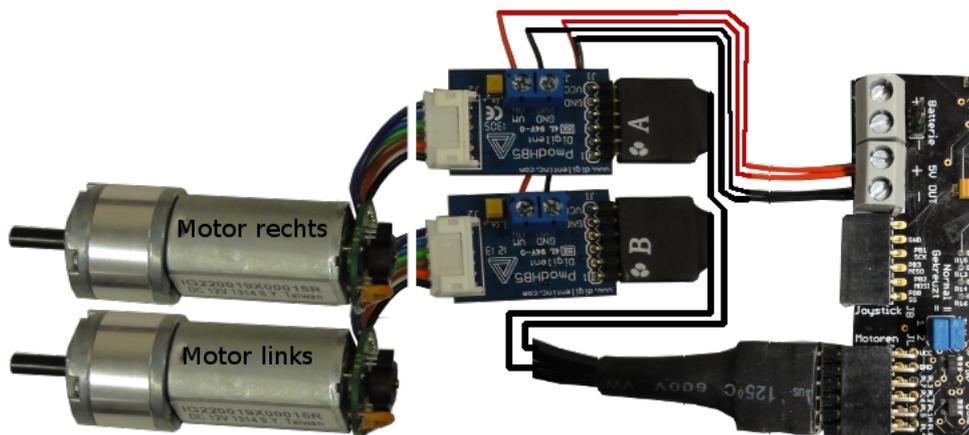
Achtung: Der Wert von Dir darf nur bei EN=0 geändert werden!

**Anschluss der Motoren an den Mikrorechner**



- Antriebsmodule: Motor, Untersetzungsgetriebe, rotierender Magnet + Hallsensoren zum Zählen der Winkelschritte.
- PmodHB5: H-Brücke, angesteuert über Dir und En. Rückgabe der Hallensorsignale an den Mikrorechner.

**Praktischer Aufbau**



- 2×H-Brücke PmodHB5 über Y-Kabel an JL,
- Motoren an die H-Brücken stecken,
- JLX »gekreuzt (=)« (Pin-Tausch PL0 und PL4),
- Spannungsversorgungsdrähte zuschneiden und anschrauben.

### Motoren ausprobieren

- Beliebiges Projekt im Debugger starten . Anhalten.
- I/O > Port L aufklappen.
- Zum Motortest DirA (PL0), DirB (PL1), EnA (PL4) und EnB (PL5) auf Ausgang und Ausgabewerte setzen.  
**DIR nur bei EN=0 ändern!**

The screenshot shows the I/O registers for Port L:

DDRL	0x10A	0x33	<input type="checkbox"/>							
PORTL	0x10B	0x01	<input type="checkbox"/>							

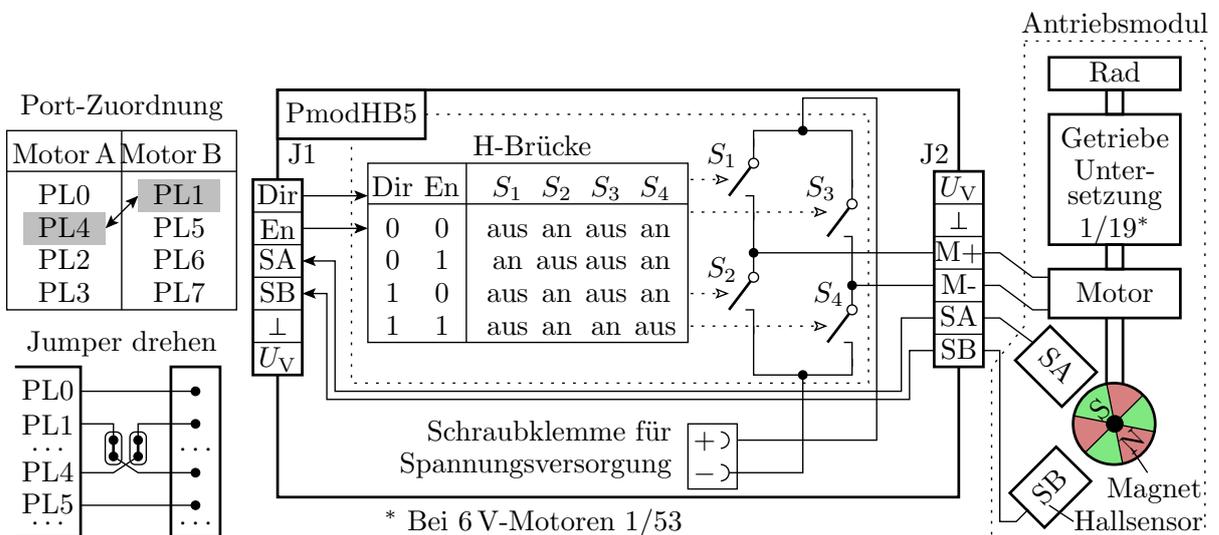
Control matrix for Motor A and Motor B:

Motor A vorwärts	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Motor A rückwärts	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Motor B vorwärts	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Motor B rückwärts	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

- Motoren vor- und rückwärts drehen lassen.
- Kontrolle der Sensorausgaben mit Multimeter<sup>3</sup>.

### 3.2 Treiber »pwm«

#### Treiber »pwm« für die Drehzahlsteuerung



- Der Treiber erwartet die dargestellte Hardware und erzeugt die Dir- und En-Signale für beide Motoren.
- Die gepulsten En-Signale generiert Timer 5 im PWM-Modus an PL4 und PL5.

<sup>3</sup>Die Anzeige von »PINL« wird nur bei Programm-Start-Stop aktualisiert.



```
void pwm_start(){
    //COM5B/C=0b10 (PWM-Ausgänge ein)
    TCCR5A = 0b00101011;
    //WGM=0b1111 CS=0b001 (Takt ein)
    TCCR5B = 0b00011001;
}
```

Übergabe der Pulsbreite für den rechten Motor:

```
void pwm_set_R(int16_t pwm){
    if (pwm>=0){
        OCR5B =pwm;
    } PORTL |=1;          //DIR-Bit (PL0) setzen
    else{
        OCR5B = -pwm;
    } PORTL &= ~1;      //DIR-Bit (PL0) löschen
}
```

- Der Geschwindigkeitswert ist 16-Bit vorzeichenbehaftet.
- Bei Betragswerten größer Periodenwert bleibt das Freigabesignal dauerhaft an.
- In der Funktion für den linken Motor

```
void pwm_set_L(int16_t pwm);
```

ist »OCR5B« durch »OCR5C« und »PL0« durch »PL1« zu ersetzen.

### 3.3 Treibertest

#### Das Testprogramm

Das Testbeispiel nutzt außer »pwm.h«:

```
#include "comir_pc.h" //PC-Eingabe
#include "comir_tmr.h" //Bewegungsdauer
```

In »comir\_pc.h« sind die Puffergrößen geändert auf<sup>4</sup>:

```
#define COM_PC_RMSG_LEN 6 //Empfang 6 Byte
#define COM_PC_SMSG_LEN 0 //keine Sendenachricht
```

Das Hauptprogramm:

```
uint8_t msg[COM_PC_RMSG_LEN];
int main(void){
    int16_t pwm; uint16_t time;
    com_pc_init(); // Init. PC-Kommunikation
    pwm_init(); // Init. Motor-Treiber
    tmr_init(); // Init. Timer-Treiber
```

---

<sup>4</sup>Den Treiber »comir\_pc.c« behandeln wir noch, und zwar nach den Interrupts.

- In der Enlosschleife wird auf eine 6-Byte-Nachricht gewartet.
- Wenn sie eintrifft, werden die PWM-Werte gesetzt, der Timer und die Bewegung gestartet.
- Nach der Wartezeit wird der Motor ausgeschaltet<sup>5</sup>.

```

sei(); //Interrupts global ein
while(1){
  if (com_pc_get(msg)){ //wenn neue Nachricht
    pwm = msg[0]<<8 | msg[1];
    pwm_set_R(pwm); //PWM-Wert für Motor R
    pwm = msg[2]<<8 | msg[3];
    pwm_set_L(pwm); //PWM-Wert für Motor L
    time = msg[4]<<8 | msg[5];
    tmr_start(time, 0); //Timer Kanal 0 starten
    pwm_start(); //PWM (Motoren) starten
  }
  if (!tmr_restzeit(0)) //wenn Timer abgelaufen
    pwm_stop(); //PWM und Motoren aus
}

```

### Treiber »pwm« ausprobieren

- Hardware-Aufbau siehe Seite 10.
- PmodUSBUSART an JH oben und USB-Verbindung zum PC.
- JHX und JLX auf »gekreuzt (=)« .
- Projekt »F11-test\_pwm\test\_pwm« übersetzen und starten.
- HTerm starten. 8N1 9600 Baud. Com Auswahl. Connect.

### Erstellung weiterer Testbeispiele

- Die Motoren werden mit 6-Byte-Nachrichten  $B_0B_1 \dots B_5$  ( $B_i$  – Byte  $i$ ) angesteuert.
- Byte  $B_0$  und  $B_1$  definieren die relative Pulsbreite Motor R:

$$\eta_R = \begin{cases} 1 & B_0 \geq 0x20 \\ \frac{|16 \cdot B_0 + B_1|}{0x2000} & B_0 < 0x20 \end{cases}$$

- Byte  $B_2$  und  $B_3$  definieren die relative Pulsbreite Motor L:

$$\eta_L = \begin{cases} 1 & B_2 \geq 0x20 \\ \frac{|16 \cdot B_2 + B_3|}{0x2000} & B_2 < 0x20 \end{cases}$$

- Byte  $B_4$  und  $B_5$ , auch zusammen als Dezimalzahl eingebbar, definieren die Bewegungsdauer:

$$t = \frac{16 \cdot B_3 + B_4}{10} \text{ s}$$

---

<sup>5</sup>Die Funktion »sei()« und andere interrupt-bezogene Features können ausprobiert werden. Grob umrissen geht es darum, dass die Schrittfunktionen nicht mehr zyklisch vom Programm, sondern von der Hardware bei Ereigniseintritt aufgerufen werden.

## 4 Aufgaben

### Aufgabe 8.1: Abarbeiten der Experimente

1. Normalmodus, LED mit Timer hochzählen.
2. CTC-Modus, umschaltbare Zähltaktperiode.
3. Experiment PWM, Pulsbreite mit LEDs visualisieren.
4. Anschluss und Ausprobieren der Motoren.
5. Treiber »pwm.c« ausprobieren. (Besser noch eine Woche warten.)

### Aufgabe 8.2: Warteschleife mit Timer

1. Ersetzen Sie im Projekt »bit\_io3\_mod«, Foliensatz 2 in »Warte\_1s()« in »myfunc.c« die Wartezählschleife durch eine Wartefunktion mit Timer 3 (Normalmodus).
2. Testen Sie bei dem Originalprogramm, wie stark die Wartezeit bei Übersetzung mit »-O0«, »-O1« und »-O2« vom Sollwert 1s abweicht.
3. Wiederholen Sie die Tests mit dem modifizierten Programm.

Hinweise:

- Festlegen eines geeigneten Vorteiler- und Timer-Startwerts.
- Programmstruktur der Wartefunktion:

```
void Warte_1s(){
    <Timer initialisieren und starten>
    while (!<Timerüberlauf>);
    <Timer anhalten>
}
```

### Aufgabe 8.3: PWM-Helligkeitssteuerung

Ändern Sie im Experiment PWM ab Folie 7 die Einstellungen von Timer 3 so, dass mit einer Periode von 1 ms

- am Ausgang PE4 eine PWM-Signal mit 10% Einschaltzeit und
- am Ausgang PE5 eine PWM-Signal mit 75% Einschaltzeit

ausgegeben wird. Kontrollieren Sie die PWM-Signale

1. mit einem LED-Modul an JE (kein flimmern, 10% bzw. 75% Helligkeit) und
2. mit dem Logikanalysator (Anstecken der LA-Anschlüsse für Masse, PE4 und PE5 über Doppelstecker an JE, XML-File anpassen, ..., Signalverläufe kontrollieren).