Informatikwerkstatt, Foliensatz 8
Timer

G. Kemnitz

Institut für Informatik, TU Clausthal (IW-F8)

22. November 2021

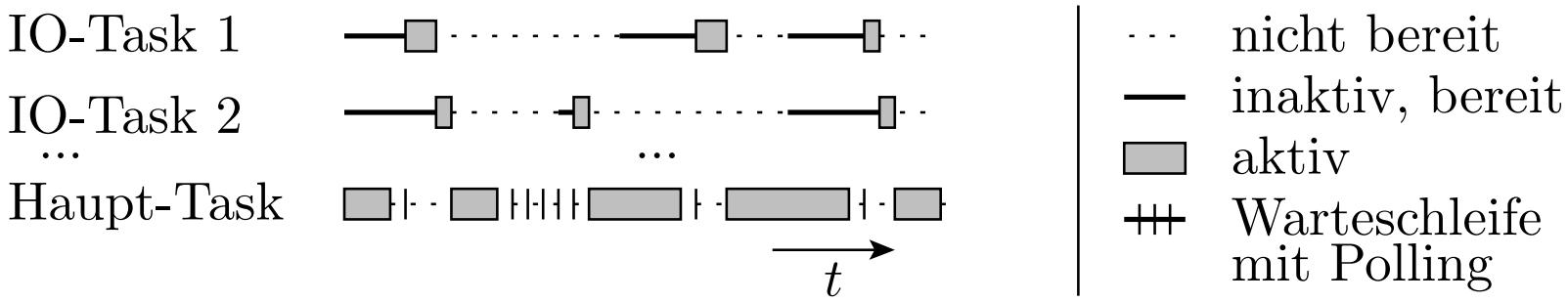
Inhalt:

Interaktive Übungen:

1. Normalmodus (F8-test_timer/test_timer).
2. CTC-Modus (F8-test_timer/test_timer).
3. PWM (F8-test_timer/test_timer).

Wiederholung

Geplantes Task-Scheduling



- Wenn der Haupt-Task keine Arbeit hat, fragt er reihum die EA-Tasks ab, ob sie bereit sind. Wenn einer bereit ist, Abarbeitung bis zum Start der nächsten Ein- oder Ausgabe.
- Falls kein Task bereit ist, wiederholt der Haupt-Task die Abfrage zyklisch.
- Nach Abarbeitung aller bereiten EA-Tasks hat der Haupt-Task möglicherweise wieder Daten für seine Fortsetzung.
- Wie kann man in einem solchen »nebenläufigen« Ablauf echte Zeiten messen, einstellen, eine Systemuhr programmieren, ...? (Für ein Fahrzeugsteuergerät unentbehrlich.)

Timer

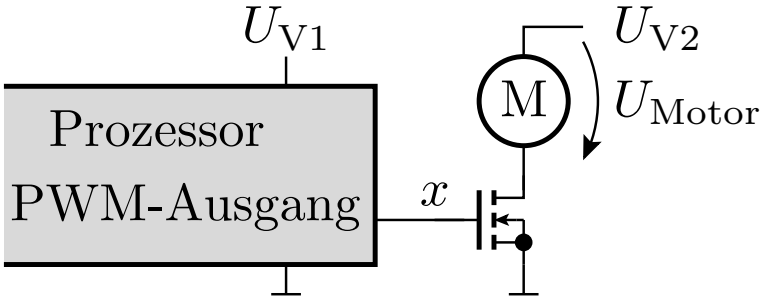
Timer

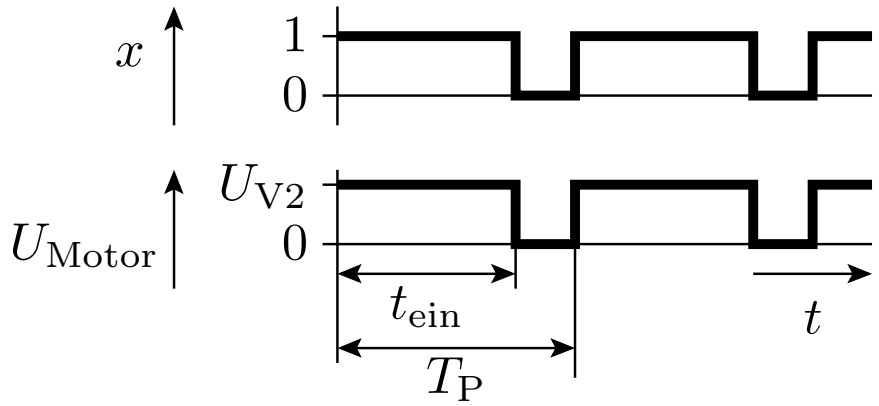
Ein Timer ist eine Hardware-Einheit aus Zähl-, Vergleichs-, Konfigurationsregistern, ... zur

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

PWM-Signale dienen

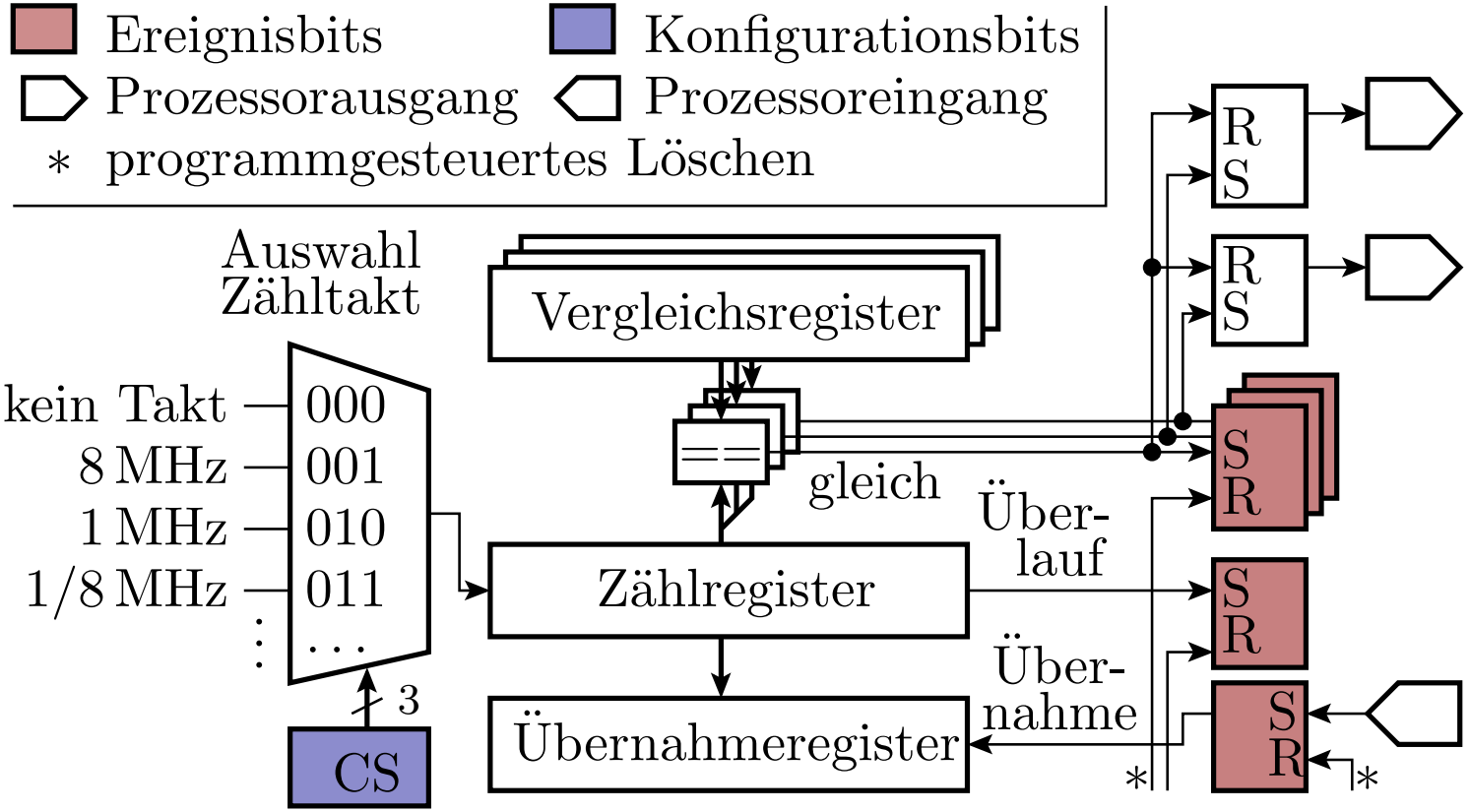
- zur Informationsübertragung z.B. an Modellbauservos und
- zur stufenlosen Leistungssteuerung, z.B. unserer Motoren.





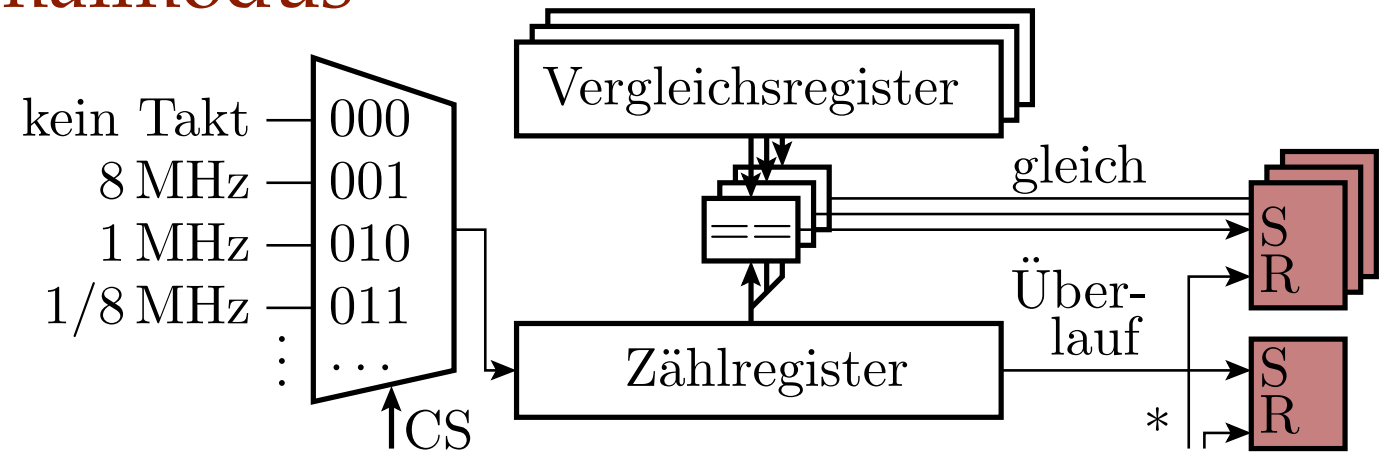
Funktionsweise

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 löschbar.

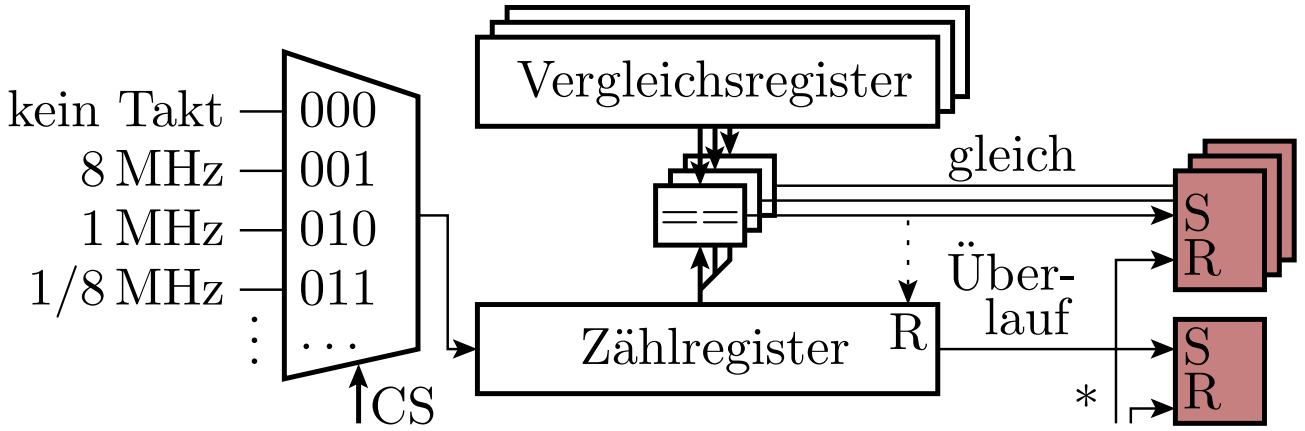
Normalmodus

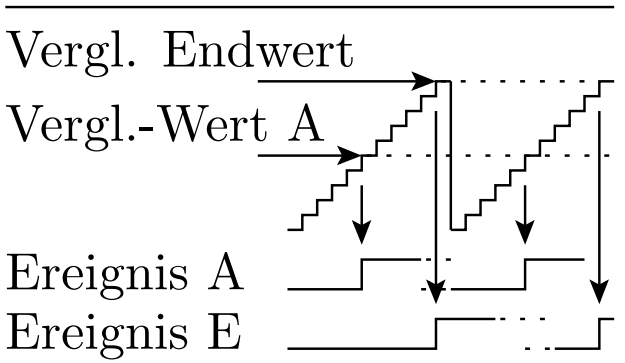


- Zählregister zählt zyklisch bis zum Überlauf.
- Beim Überlauf wird ein Überlaufbit und bei Gleichheit mit einem Vergleichsregister ein Vergleichsereignisbit gesetzt.
- Beispiel Wartefunktion:

```
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

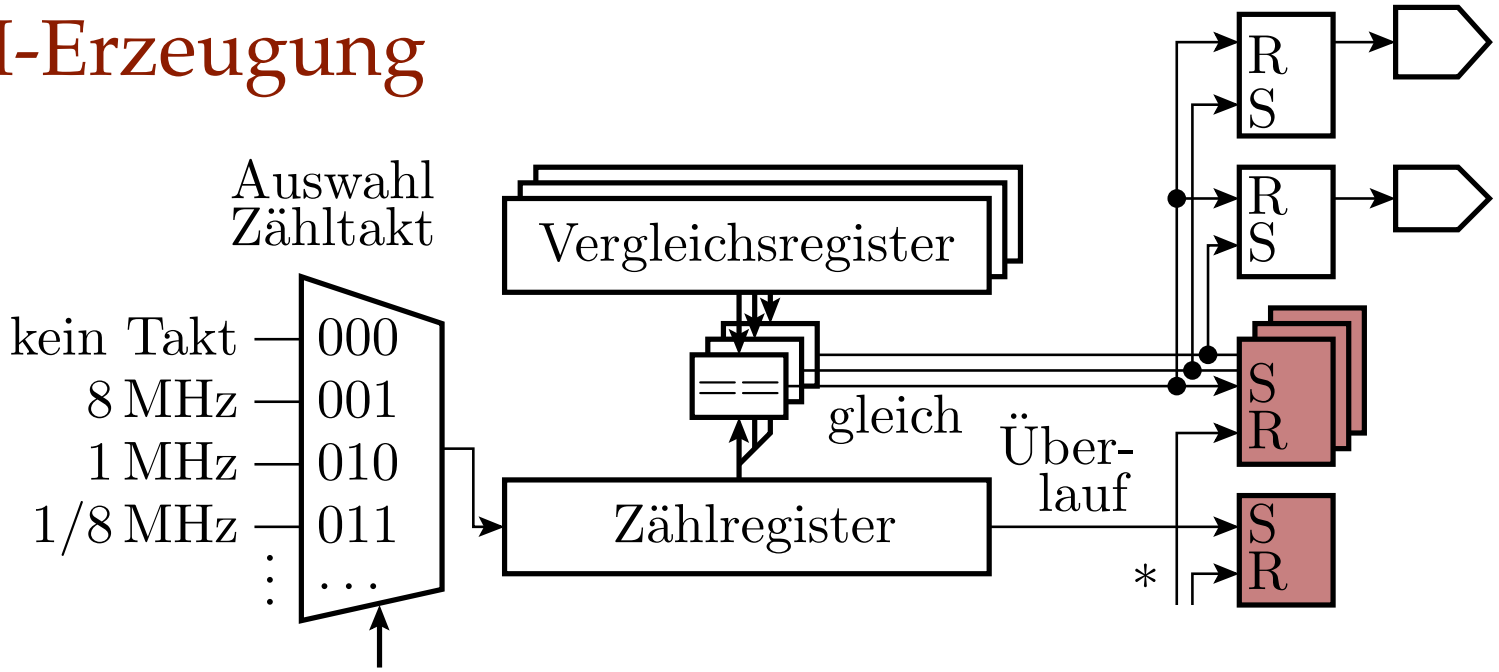




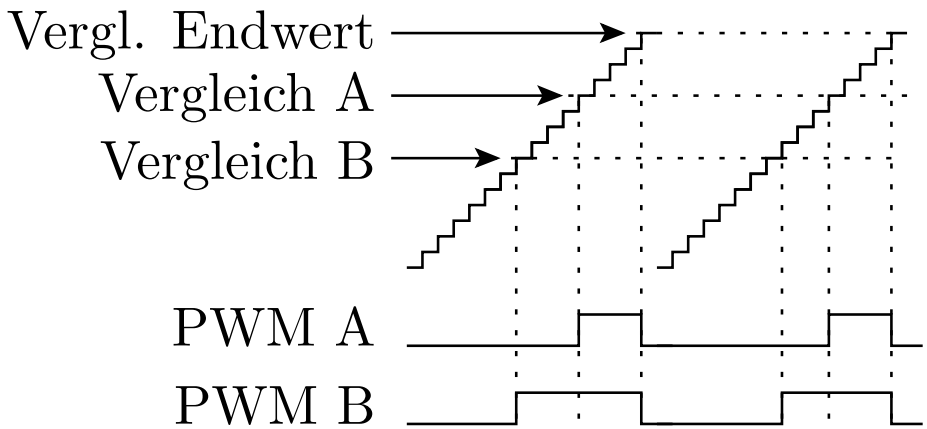
- Zähler wird bei Gleichheit mit Vergleichsregister rückgesetzt.
- Auslösung zyklischer Ereignisse, z.B. Uhrenprozess:

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

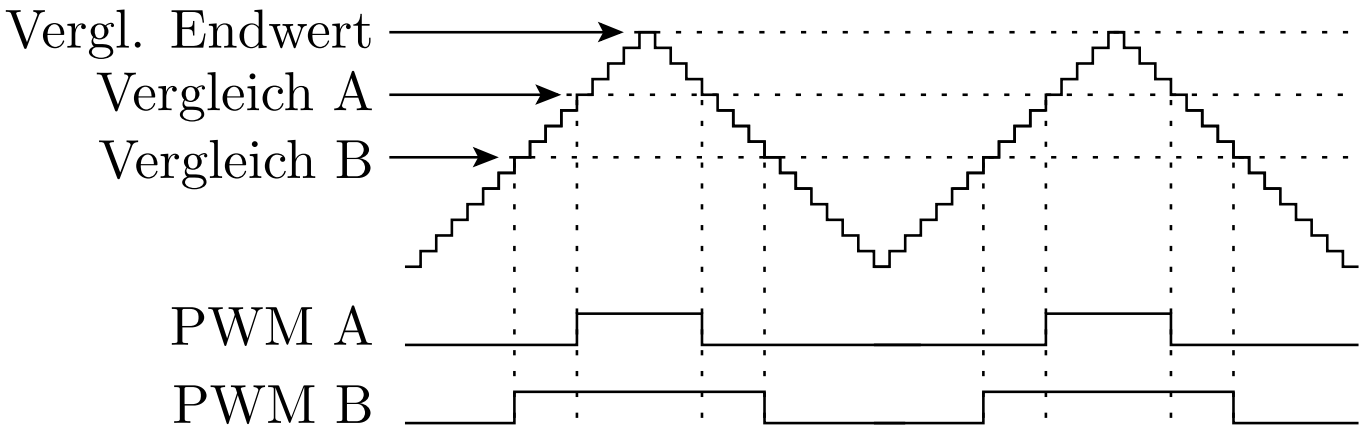
PWM-Erzeugung



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



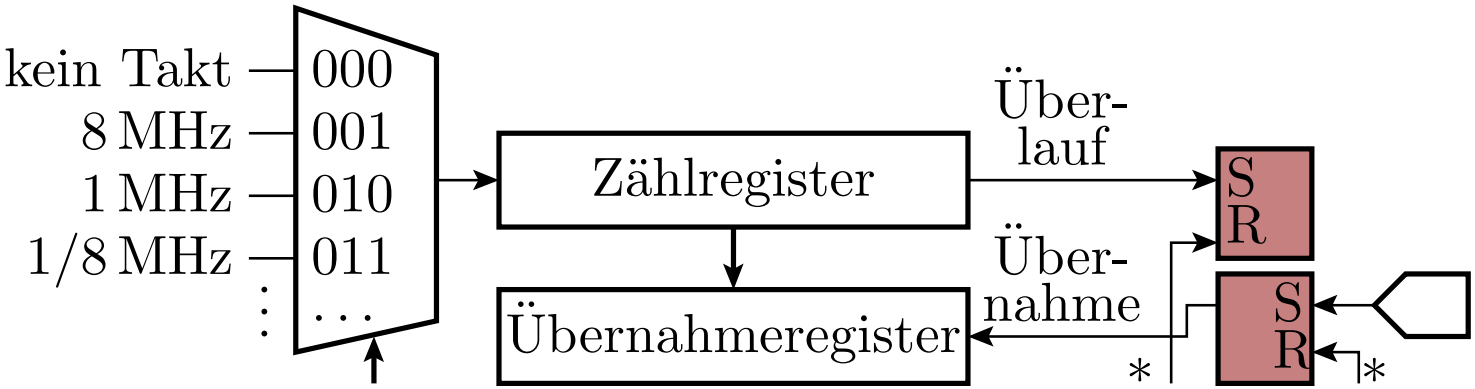
Symmetrische PWM[1]



- 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.

[1] Im Datenblatt unseres Prozessors ist das die phasenrichtige und die vorhergehende normale PWM die schnelle (Fast-) PWM.

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.

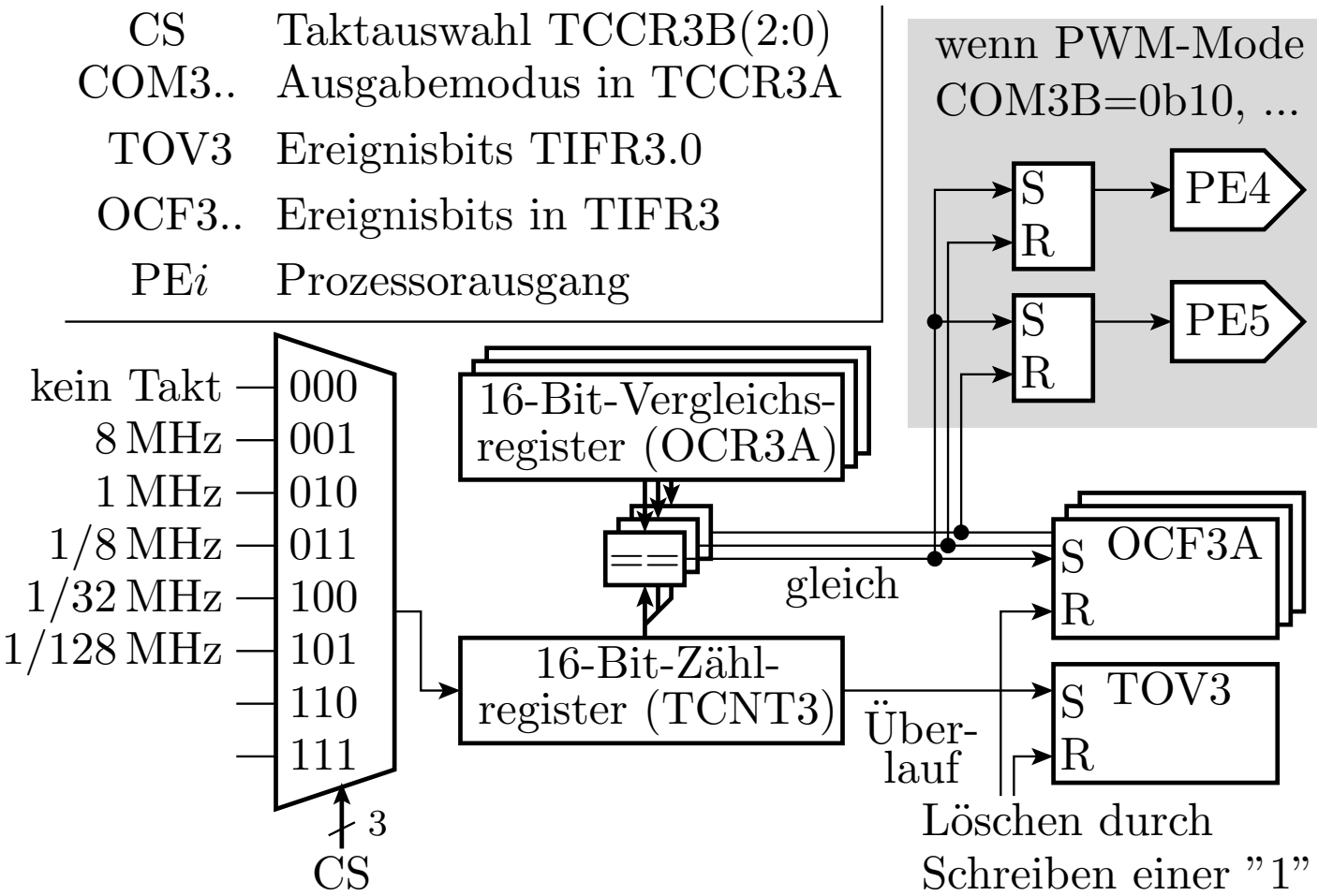
Nutzung der Timer in den Beispielprojekten:

- Timer 0: Treiber »wegmess« Abtastintervall.
- Timer 1: Treiber »comir_tmr« Programmuhr und Wartezeitzähler.
- 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.

Timer 3

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



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

Betriebsarten (Auswahl)

Bit	7	6	5	4	3	2	1	0
TCCR3A							WGM1	WGM0
TCCR3B				WGM3	WGM2	CS2	CS1	CS0

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

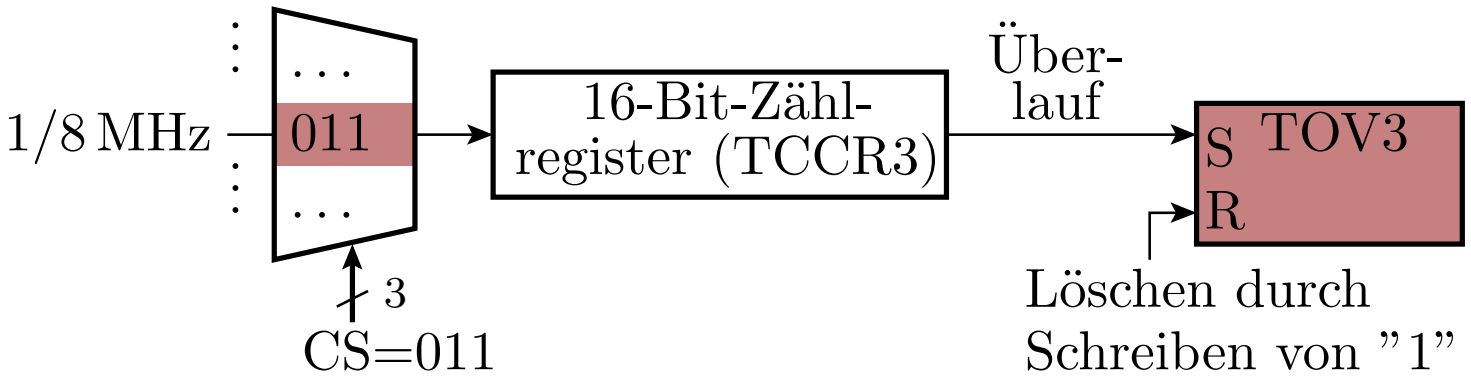
$^{(*1)}$ symmetrische oder phasenausgerichtete PWM.

$^{(*2)}$ Schnelle oder normale PWM.

Experimente

Normalmodus, LED mit Timer hochzählen

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



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

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

```
#include <avr/io.h>
int main(void){
  TCCR3A = 0;          // WGM3[1:0] = 0
  TCCR3B = 0b011;      // WGM3[3:2] = 0, CS3=0b011
  DDRJ = 0xFF;
  while(1){
    if (TIFR3 & (1<<TOV3)){ // Warte auf Überlauf
      PORTJ++;              // Erhöhe Led-Ausgabe
      TIFR3 = (1<<TOV3);    // Lösche Überlaufbit
    }
  }
}
```

- Projekt F8-test_time\test_timer öffnen.
- Alle außer erste Main-Funktion auskommentiert lassen.
- Übersetzen. Start im Debugger . Continue .
- LED-Zählfrequenz kontrollieren.
- Anhalten . Unterbrechungspunkt wie im Bild setzen.
- Continue bis .

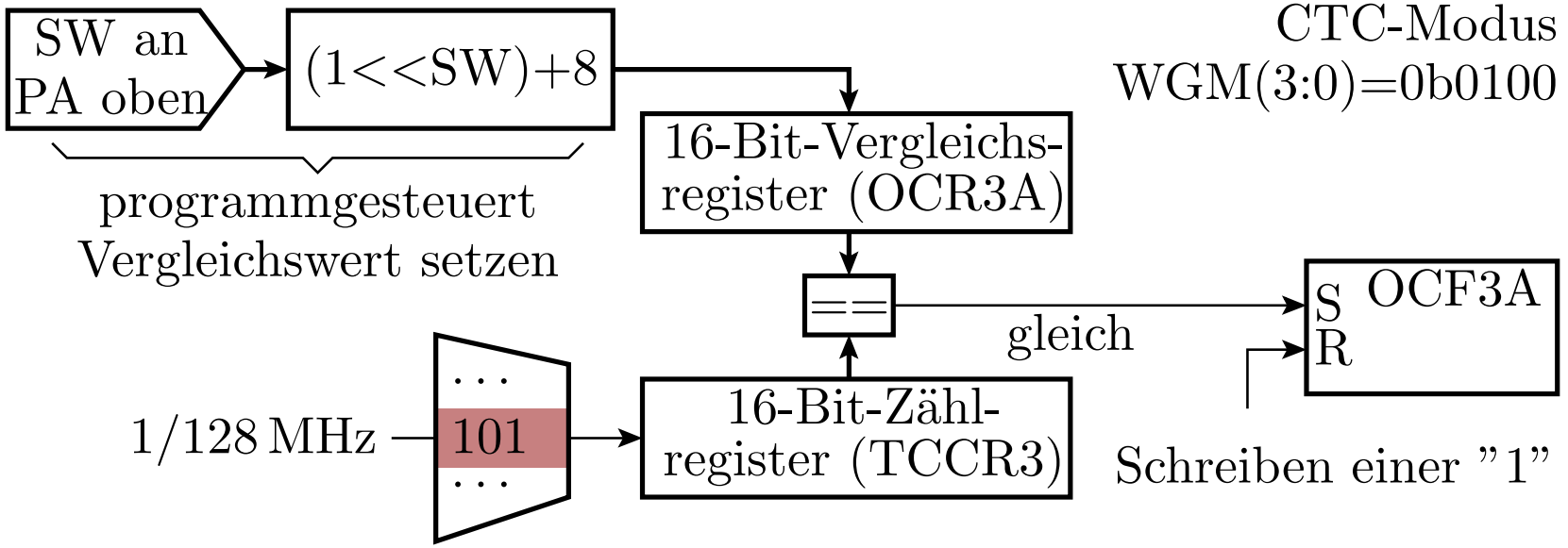
IO-View am Unterbrechungspunkt

TIMER_COUNTER_3

Name	Address	Value	Bits
TIFR3	0x38	0x0F	
OCF3A		0x01	
TOV3		0x01	
TIMSK3	0x71	0x00	
TCCR3A	0x90	0x00	
TCCR3B	0x91	0x03	
WGM3		0x00	
CS3		0x03	
TCNT3	0x94	0x0000	
OCR3A	0x98	0x0000	

»TOV3« gesetzt. Zähler »TCNT3« null, warum? »OCF3A« ist auch gesetzt, da »OCR3A==0« in jedem Zählzyklus erreicht wird und »OCF3A« nie gelöscht wird.

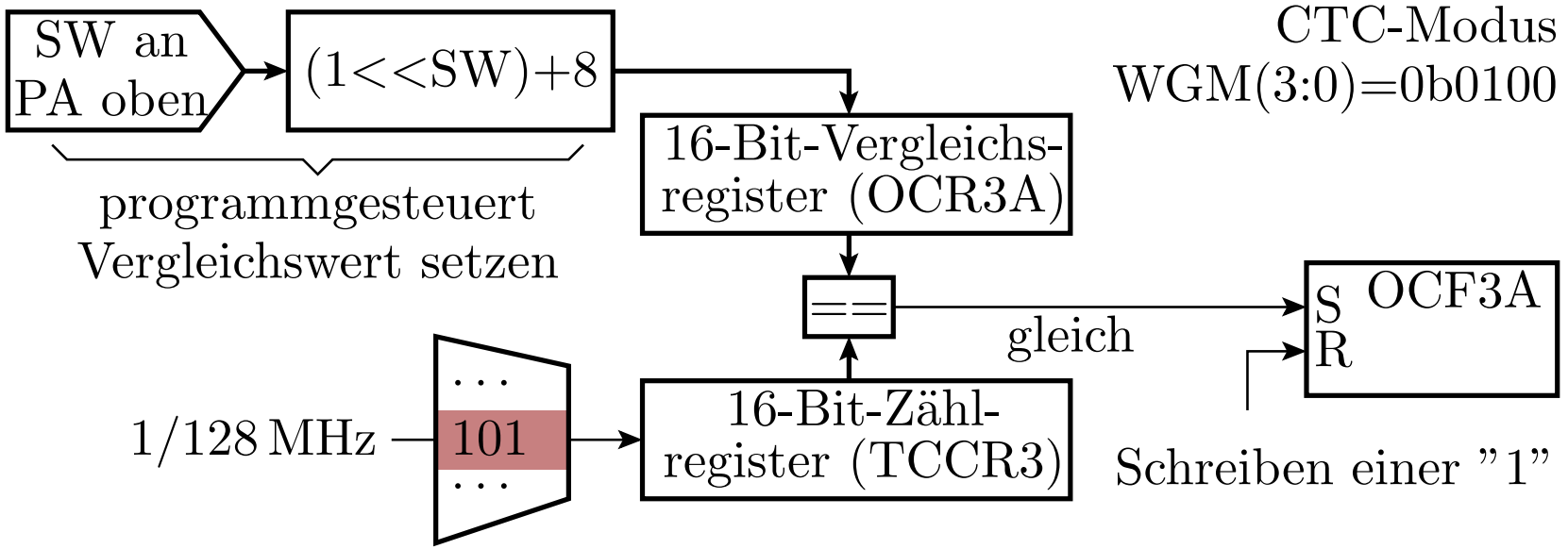
CTC-Modus, umschaltbare Zähltaktperiode



LED-Zähltakt:

$$f_{\mathrm{LED}} = \frac{\frac{1}{128}\,\mathrm{MHz}}{\mathrm{OCR3A}} \text{ mit OCR3A=8+}(1 << sw)$$

sw	0000	0010	0100	1000	1001	1010	1011	1100
OCR3A	1+8	4+8	16+8	264	520	1034	2056+	4104
$\frac{f_{\mathrm{LED}}}{\mathrm{Hz}}$	868	651	326	30	15	7,6	3,8	1,9



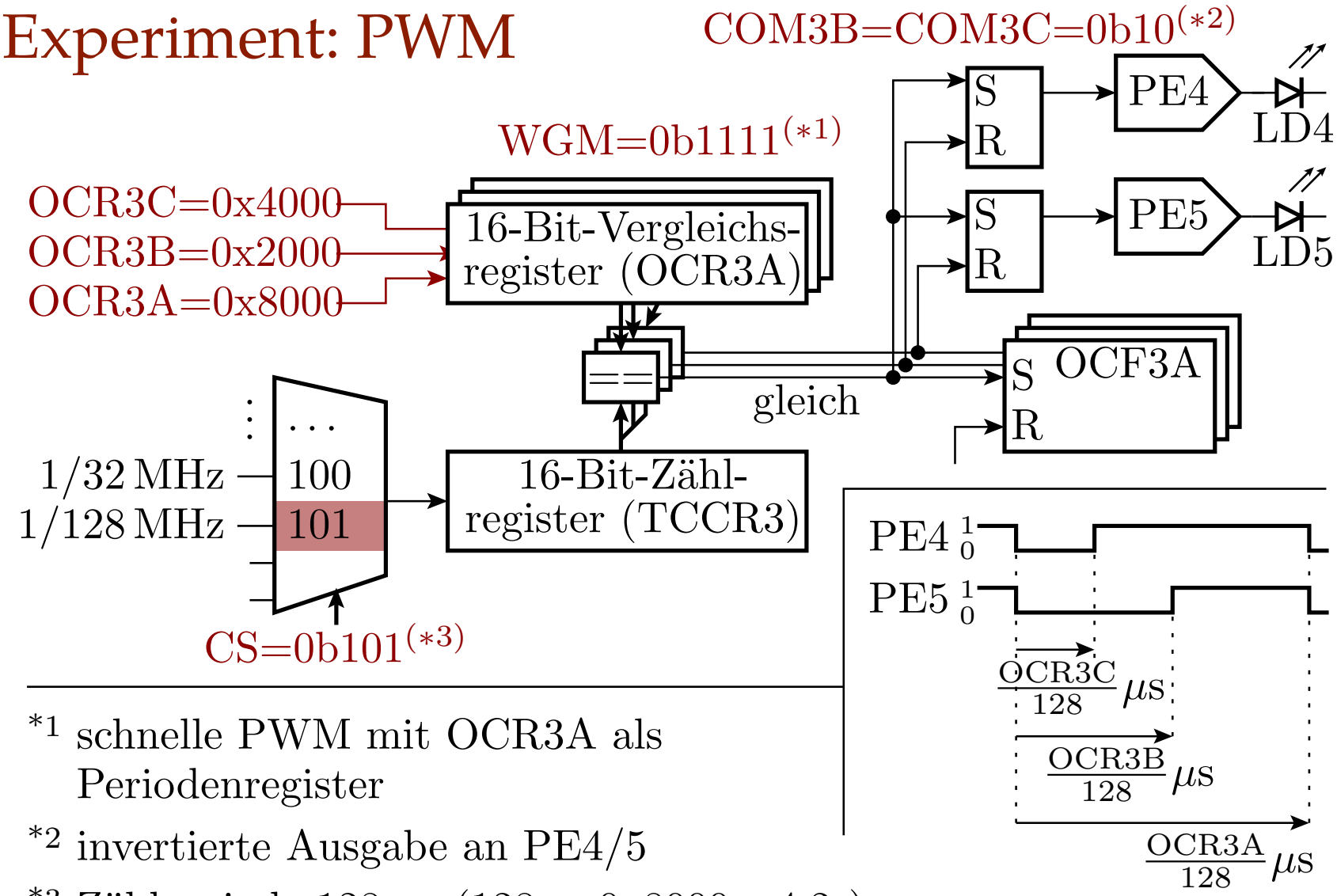
Testprogramm:

- 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 ≈2 Hz.
- Schalterwert erhöhen/verringern und Frequenz kontrollieren.

Experiment: PWM



*1 schnelle PWM mit OCR3A als Periodenregister

*2 invertierte Ausgabe an PE4/5

*3 Zählperiode 128 μs (128 μs·0x8000≈ 4,2s)

- LD4, LD5: LEDs auf PMOD8LD an JE

Testprogramm:

- Timer initialisieren.
- Endlosschleife, die nichts tun muss.

- LED-Modul »PMOD8LD« an JE[2].
- Im Projekt F8-test_timer\test_timer alle außer dritte Main-Funktion auskommentieren.
- Übersetzen. Start im Debugger. Continue.
- Kontrolle:
  - Blinkperiode: $\frac{\mathrm{0x8000}}{128}\,\mu\mathrm{s} \approx 2{,}56\,\mathrm{s}$
  - Ausschaltzeit LED4 25%: $\frac{\mathrm{0x2000}}{128}\,\mu\mathrm{s} \approx 0{,}64\,\mathrm{s}$
  - Ausschaltzeit LED5 50%: $\frac{\mathrm{0x4000}}{128}\,\mu\mathrm{s} \approx 1{,}28\,\mathrm{s}$
- Anhalten. Unterbrechungspunkt siehe nächste Folie setzen. Continue bis und Kontrolle der SFR-Werte.
- Ausprobieren mit anderen Haltepunkten, Pulsbreiten, ...

[2] Ausgabe PE4 an LD4 und PE5 an LD5.

```
#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);
  }
}
```

- Werte der Timer-Register am Haltepunkt:

TIMER_COUNTER_3

Name	Address	Value	Bits
TIFR3	0x38	0x03	
OCF3C		0x00	
OCF3B		0x00	
OCF3A		0x01	
TCCR3A	0x90	0x2B	
COM3A		0x00	
COM3B		0x02	
COM3C		0x02	
TCCR3B	0x91	0x1D	
TCNT3	0x94	0x0000	
OCR3A	0x98	0x8000	
OCR3B	0x9A	0x2000	
OCR3C	0x9C	0x4000	

- Verringern Sie den CS-Wert im Debugger am Unterbrechungspunkt auf CS=0b100.
  Unterbrechungspunkt löschen und Continue  .
  Wie ändert sich die Pulsperiode und die relative Pulsbreite?
- Schlagen Sie im Prozessordatenblatt nach, was mit COM3B und COM3C eingestellt wird. Programmänderung, so dass die LED-Ausgaben an PE4 und PE5 gegenüber dem Vorgabeprogramm invertiert werden.
- Die »OCR...« Werte lassen sich nicht im Debugger ändern, bzw. beim nächsten Debugger-Stopp steht wieder der alte Wert in den Registern. Workaround: Wertezuweisung aus einer Variablen in der Hauptschleife und Änderung der Variablenwerte im Debugger.
- Eine PWM mit einer Taktperiode im Millisekundenbereich wird später zur Steuerung der Motorgeschwindigkeit genutzt.

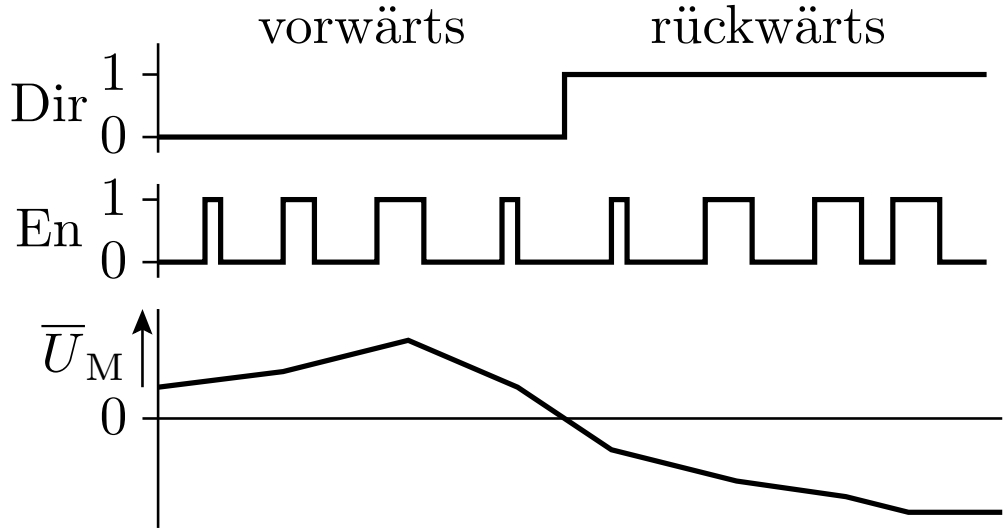
Drehzahlsteuerung

Prinzip und Motortest

Drehzahlsteuerung durch Pulsweitenmodulation

Pulsweitenmodulation (PWM) schaltet die Motoren schnell ein und aus. Drehzahlsteuerung über die relative Einschaltzeit.

En	1	0	0	1
Dir	0	0	1	1
	+ M −	M	− M +	

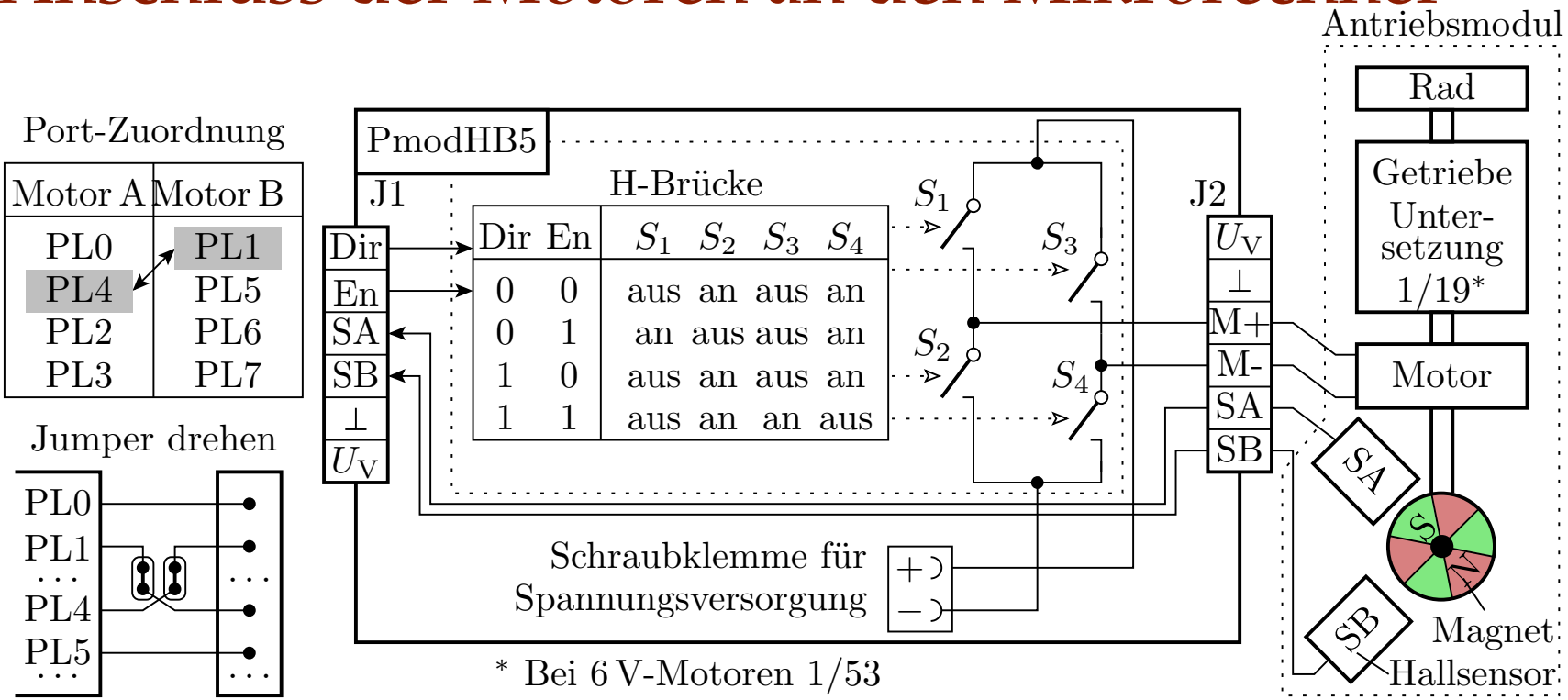


An den Antriebsbaugruppen erfolgt die Einstellung

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

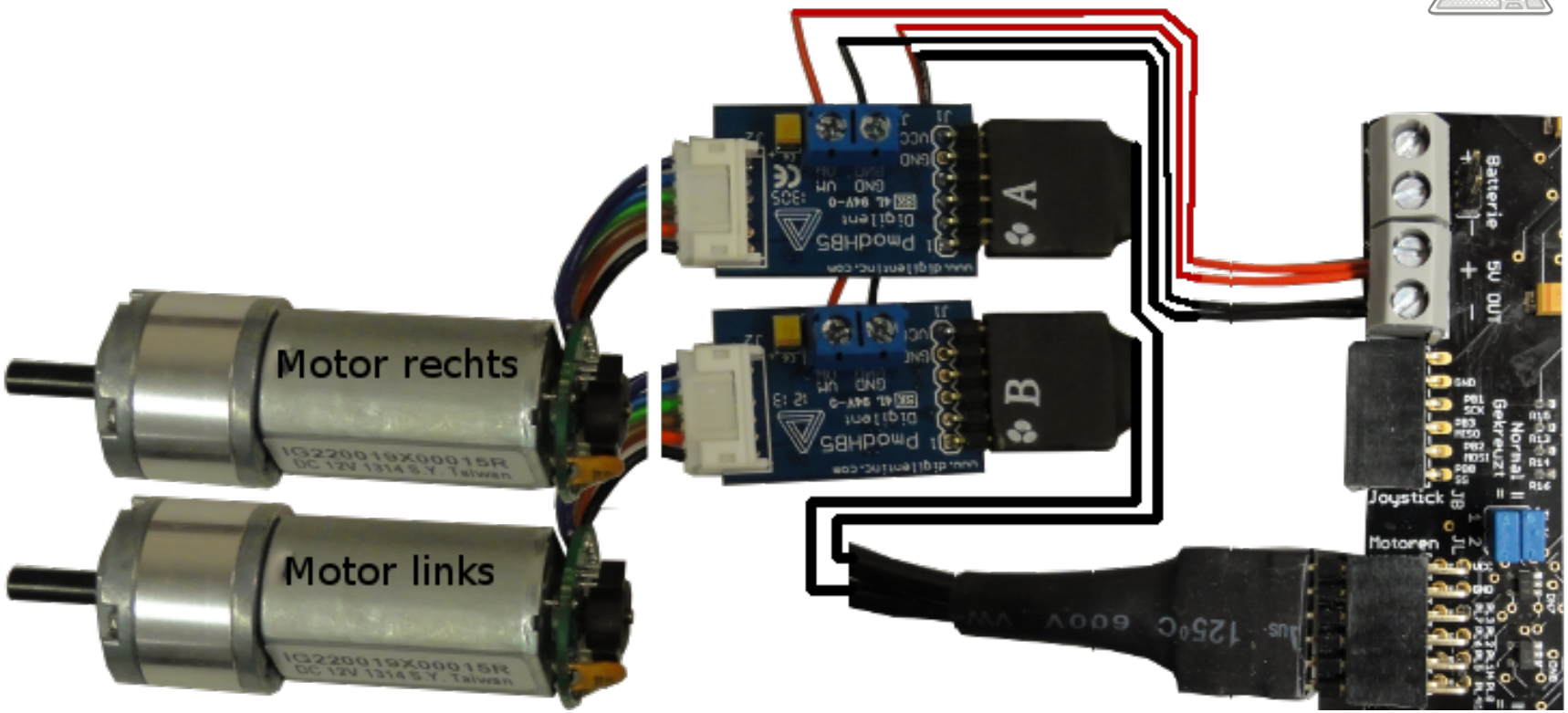
Achtung: Der Wert von Dir darf nur bei EN=0 geändert werden!

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 Hallsensorsignale an den Mikrorechner.

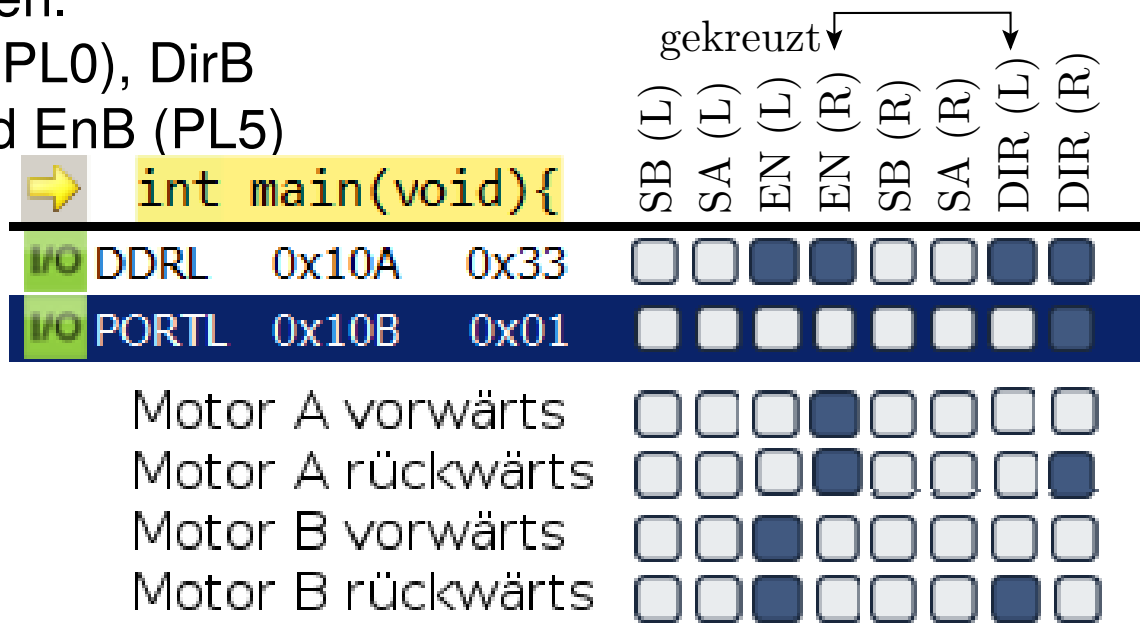
Praktischer Aufbau



- 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!

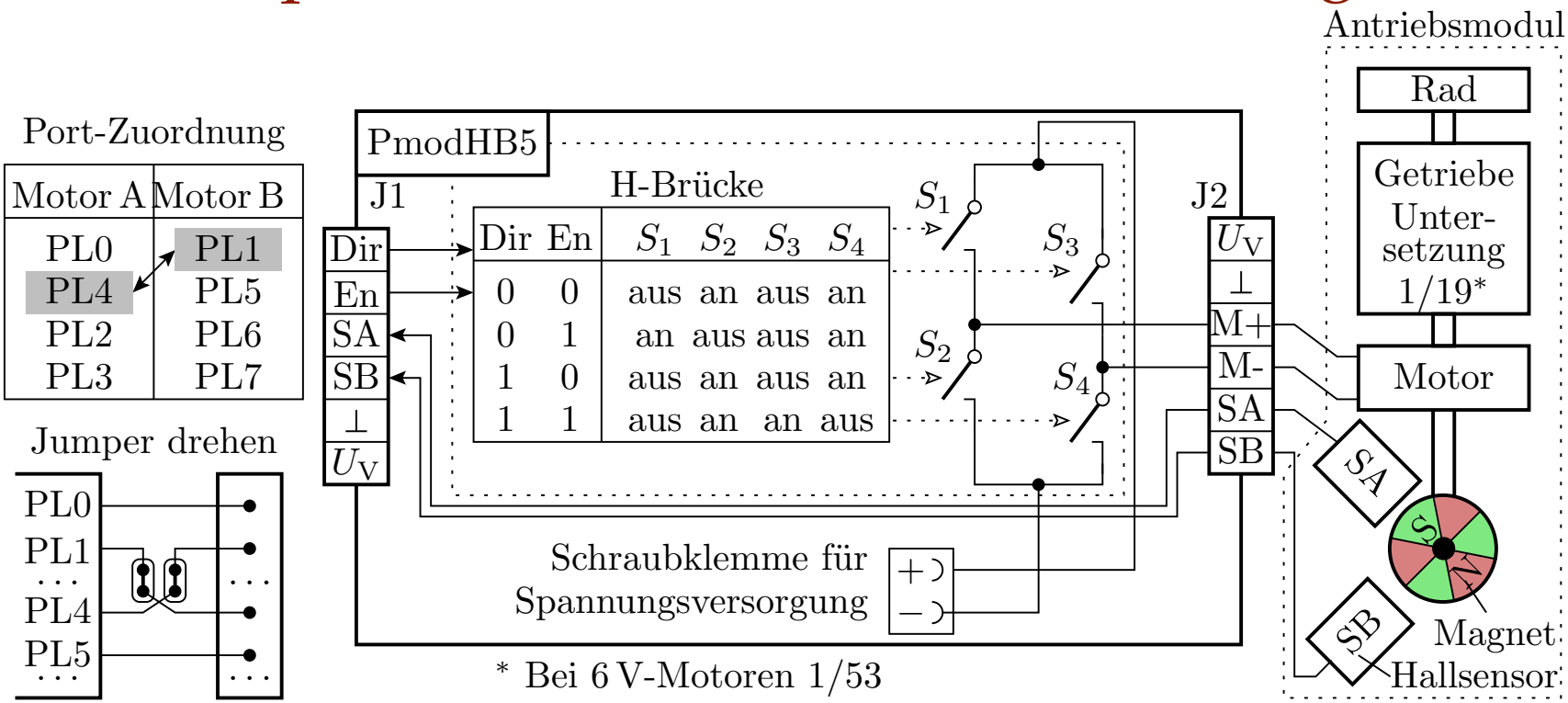


- Motoren vor- und rückwärts drehen lassen.
- Kontrolle der Sensorausgaben mit Multimeter[3].

[3] Die Anzeige von »PINL« wird nur bei Programm-Start-Stop aktualisiert.

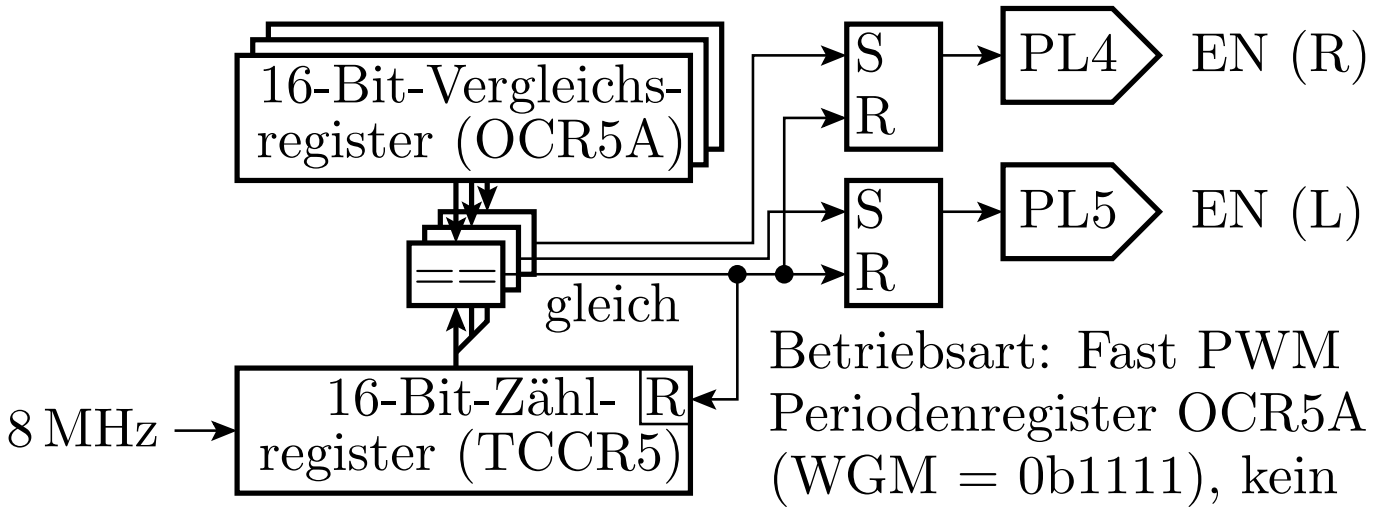
Treiber »pwm«

Treiber »pwm« für die Drehzahlsteuerung

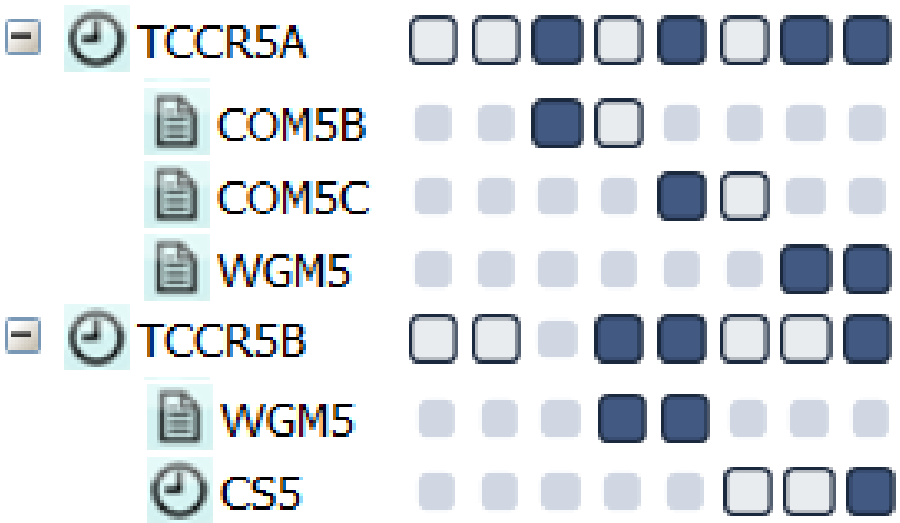


- Der Treiber erwartet die dargestellte Hardware und erzeugt die Dir- und En-Signale für beide Motoren.
- Die gepulsten En-Signale generiert Timer 5 im PWM-Modus an PL4 und PL5.

Timer-Einstellung für die Enable-Signale



Betriebsart: Fast PWM
Periodenregister OCR5A
(WGM = 0b1111), kein
Taktvorteiler (CS=0b001)
PWM-Ausgabe ”Clear on
Compare” (COM5B/C=0b10)



Funktionen des Treibers

- Keine privaten Daten.
- Initialisierungsfunktion.
- Keine Schrittfunktion.
- Jeweils eine Funktion für Stopp und Start beider Motoren.
- Jeweils eine Funktion zur Einstellung der Pulsbreite.

Initialisierungsfunktion:

```
void pwm_init(){
 DDRL  =0b00110011;  //EN und DIR als Ausgänge
 pwm_stop();         //Zähltakt und PWM aus ...
 TCCR5C = 0b00000000;//Zählregister löschen
 OCR5A = 0x2000;     //Periodenregister (ca. 1ms)
 OCR5B = 0;          //Motor R: Pulsbreite 0
 OCR5C = 0;          //Motor L: Pulsbreite 0
}
```

Stoppfunktion für beide Motoren: Zähltakt und PWM-Ausgabe aus.

```
void pwm_stop(){
  TCCR5A = 0;    //PWM ausschalten
  TCCR5B = 0;    //Zähltakt aus
  PORTL  = 0;    //Enable (Motoren) ausschalten
}
```

Startfunktion für beide Motoren: Zähltakt und PWM-Ausgabe ein.

```
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.

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[4]:

```
#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
```

[4] 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[5].

```
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
 }
}
```

[5]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.

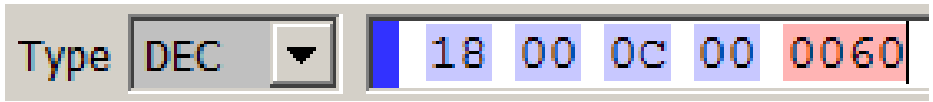
Treiber »pwm« ausprobieren

- Hardware-Aufbau siehe Folie 34.
- 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.

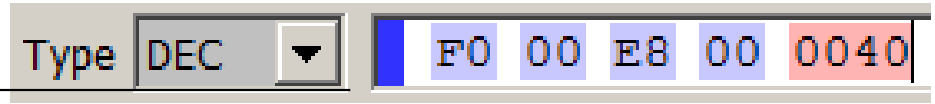
Testbeispiele:

- Motoren R und L mit 50% für 3 s vorwärts[6]:

- Motor R mit 75% und Motor L mit 37,5% für 6 s vorwärts:



- Motor R mit 50% und Motor L 75% rückwärts für 4 s.



[6] In den Bildern blaue Eingaben »HEX« und rote Eingaben »DEC«.

Erstellung weiterer Testbeispiele

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

$$\eta_{\mathrm{R}} = \begin{cases} 1 & B_0 \geq 0\mathrm{x}20 \\ \frac{|16 \cdot B_0 + B_1|}{0\mathrm{x}2000} & B_0 < 0\mathrm{x}20 \end{cases}$$

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

$$\eta_{\mathrm{L}} = \begin{cases} 1 & B_2 \geq 0\mathrm{x}20 \\ \frac{|16 \cdot B_2 + B_3|}{0\mathrm{x}2000} & B_2 < 0\mathrm{x}20 \end{cases}$$

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

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

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 ud 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 1 s 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 25 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).