Informatikwerkstatt, Foliensatz 4
Joystick und Motorsteuerung

G. Kemnitz

Institut für Informatik, Technische Universität Clausthal

21. November 2013

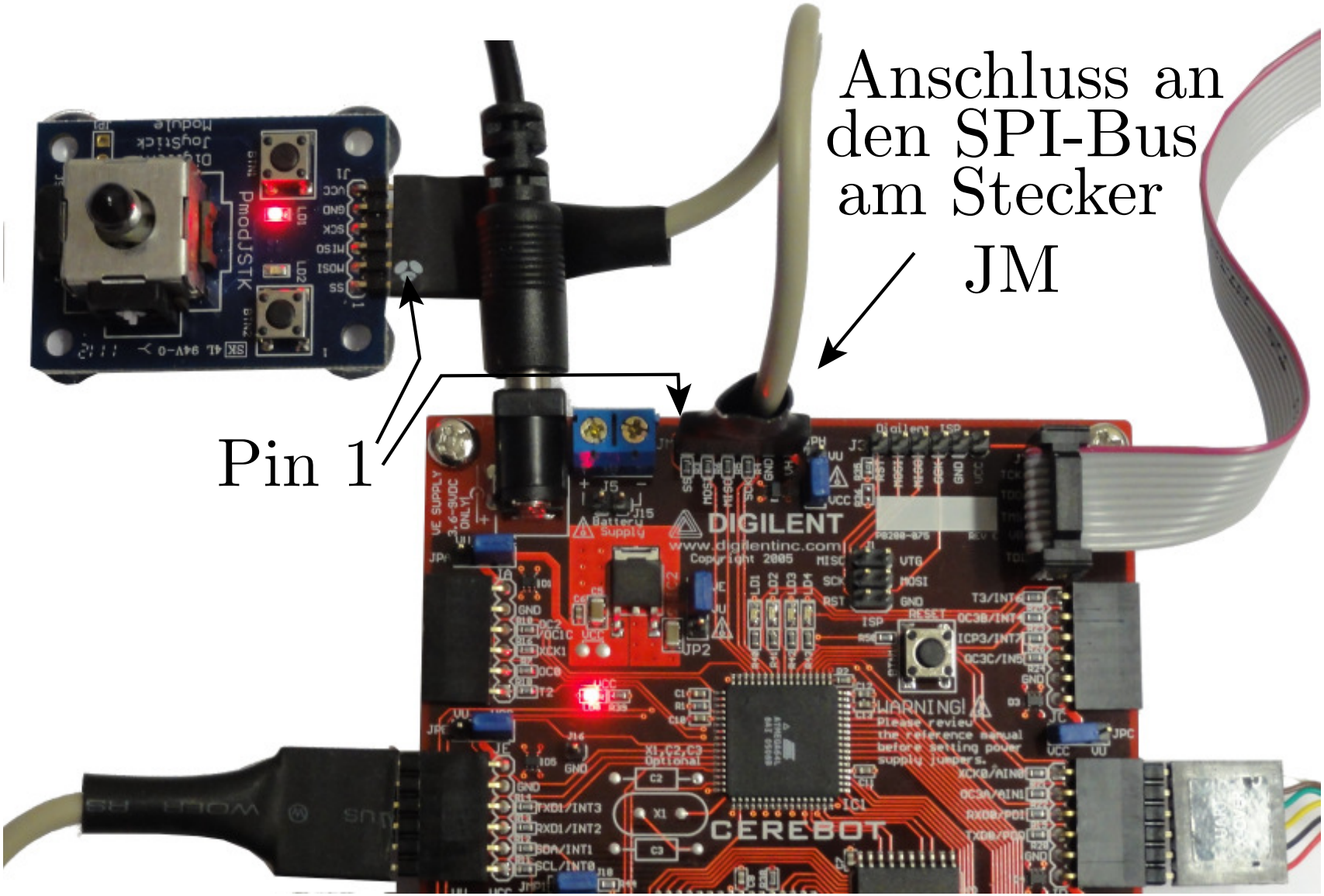
Inhalt des Foliensatzes

Joystick

Motoren, Wegmessung

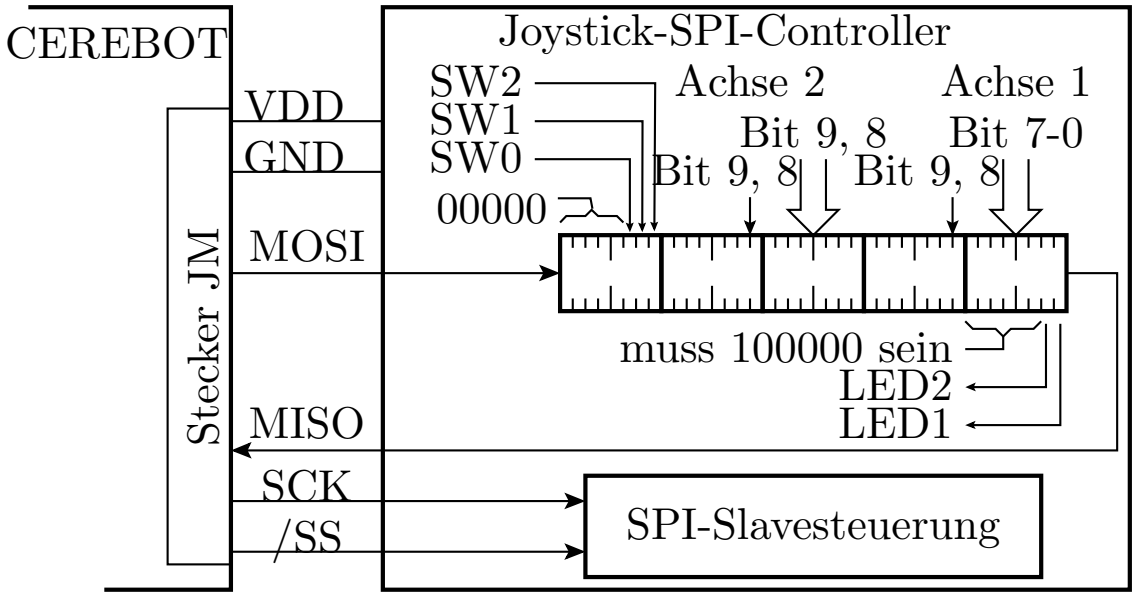
Joystick

Anschluss des Joysticks



SPI-Funktionalität im Joystick

SPI – Serial Peripherial Bus

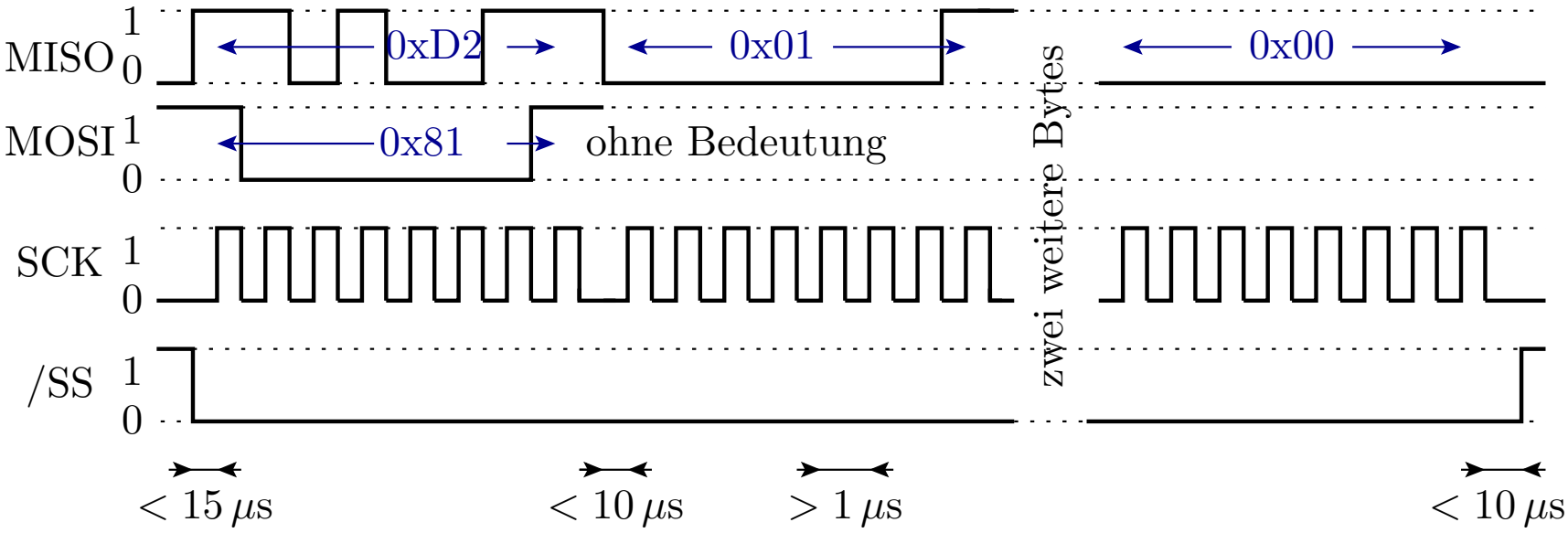


Slave-Steuerung

Aktion	/SS	SCK
Übernahme	⌐	
Schieben	0	⌐
Ausgabe	⌐	
keine	sonst	

MOSI Master Out Slave In
MISO Master In Slave Out
SCK Schiebetakt
/SS Slave-Auswahl

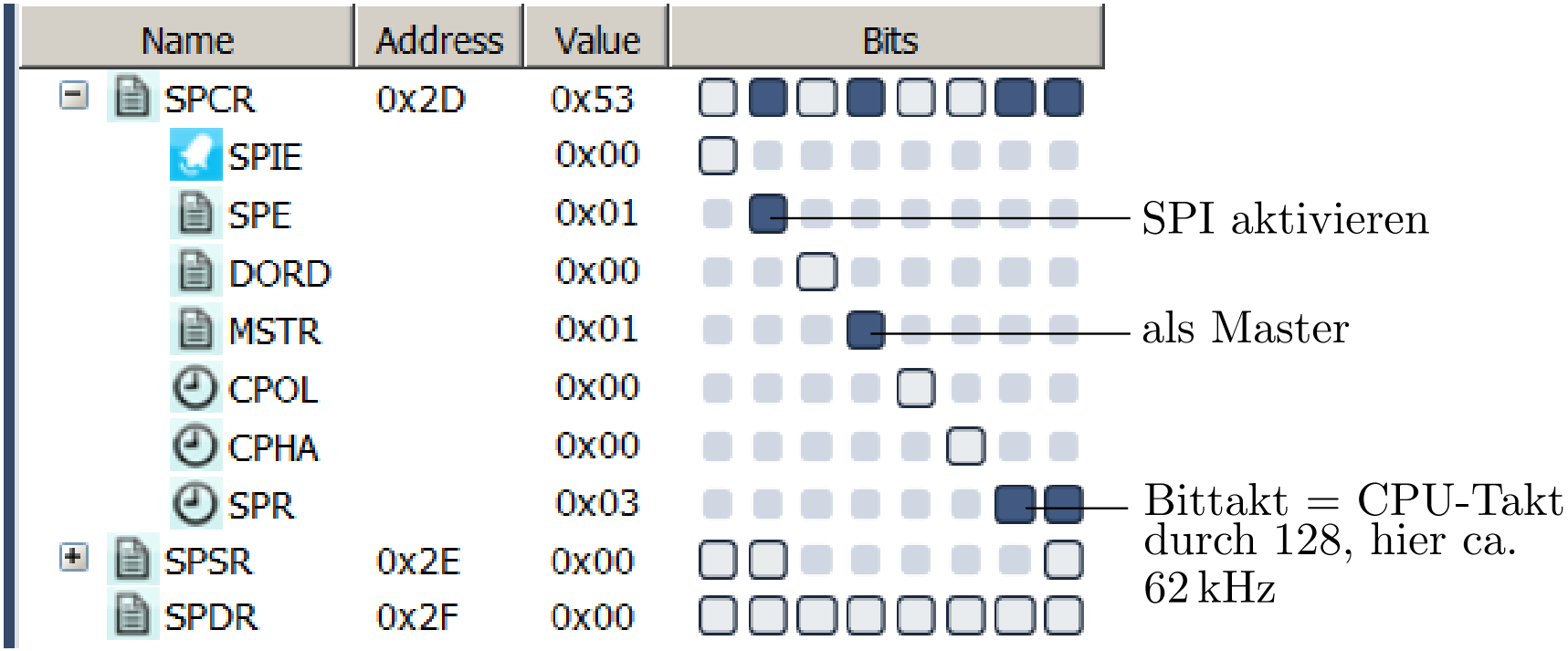
- Der SPI-Slave des Joysticks ist ein 5x8 Bit Schieberegister, das vom Joystick Daten übernimmt, geschoben wird und Daten übergibt. Steuerung über SCK und /SS.
- Der Master im Mikrorechner muss /SS aktivieren, 5 Bytes schicken - warten - übernehmen und /SS deaktivieren.



- Die Zeiten im Bild sind einzuhalten, sonst Übertragungsfehler. Eine sofortige serielle Weitergabe nach Byteempfang verboten, da Sendedauer bei 9600 Baud ca. 1 ms $\gg$ 10 μs max. zulässige Pause zwischen zwei SPI-Übertragungen.
- Gesendet wird ein Byte 0b100000$l_2 l_1$ (l_i – LED-Ausgabewert) und 4×beliebige Bytes.
- Empfangen werden zwei Bytes mit dem x– zwei Bytes mit dem y–Wert und eines mit drei Tasterwerten.

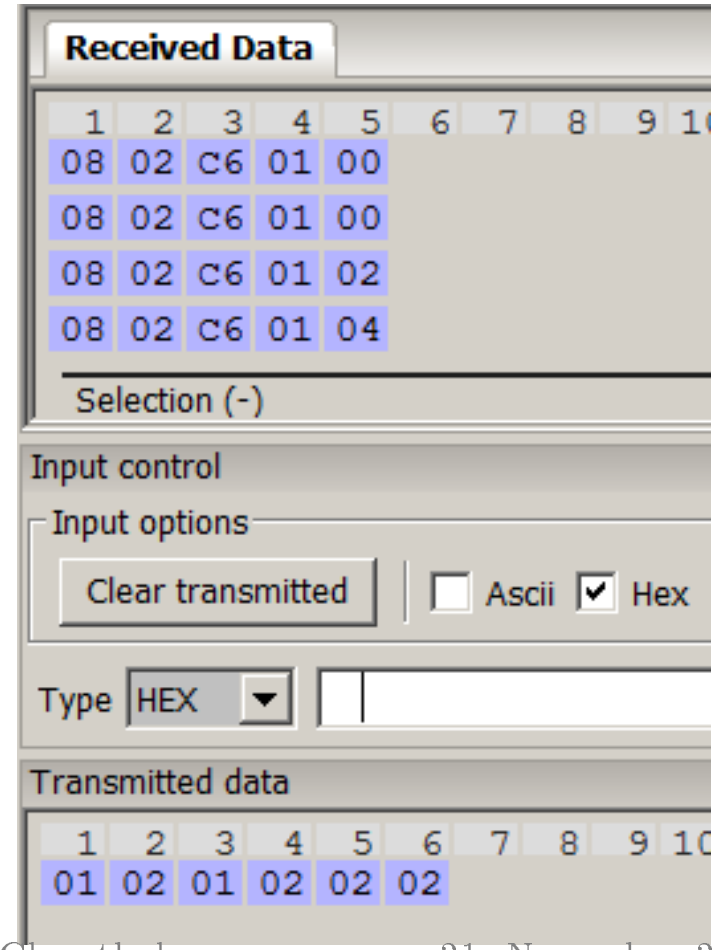
Initialisierung des SPI-Busses im Mikroprozessor

- SPI-Bus aktivieren, auf Master setzen und Frequenz von SCK über Teilerwert festlegen.



Aufgabe 4.1: Testbeispiel für Joystick vorbereiten

- Neues Projekt »Joystick« mit den Dateien TestJoystick.c, Joystick.c/h, comPC.c/h von der Web-Seite anlegen.
- Spannung aus. Joystick wie auf Folie 4 an JM stecken. Programm übersetzen und starten.
- HTerm starten mit 9600 Baud 8n1, nur Hex-Anzeige und Newline every 5 characters.



Das Programm wartet auf Eingabe und gibt die Bytes B_1 bis B_5 aus:

- x-Wert: $B_2 \cdot 2^8 + B_1$
- y-Wert: $B_4 \cdot 2^8 + B_3$
- B_5 mit den Tasterwerten in Bit 0 bis 2.

Das Testprogramm für den Joystick

```
 9  #include <avr/io.h>
10  #include "comPC.h"
11  #include "Joystick.h"
12  uint8_t dat[5];                 // Datenpuffer
13
14  int main(void){
15      initUSART0();               // USART0 initialisieren
16      initSPI();                  // SPI-Initialisierung
17      uint8_t din, i;             // Daten und Zähler
18      while(1) {
19        din = getByte();
20        comSPI(dat, din);
21        for (i=0; i<5; i++){// Wiederhole für i=0 bis 4
22          sendByte(dat[i]);   // versende ein Byte
23        }
24    }
25  }
```

- Die Funktionen initUSART0(), getByte(), sendByte() sind in comPC.h definiert. Funktionsweise siehe Foliensatz 3.
- Initialisierung des SPI-Busses:

```
 9  // Initialisierung SPI-Bus
10  // Stecker  Belegung  Bedeutung
11  // JM-1     /SS  PB0  Slave-Auswahl       (Ausgang)
12  // JM-2     MOSI PB2  Master-Out-Slave-In (Ausgang)
13  // JM-3     MISO PB3  Master-In-Slave-Out (Eingang)
14  // JM-4     SCK  PB1  Schiebetakt         (Ausgang)
15  void initSPI(){
16    DDRB = 0xf7;  // /SS, MOSI, SCK: Ausgänge
17    PORTB |= 0x1; // /SS=1 (Slave deaktiviert)
18    // SPI als Master mit f_SCK=f_CPU/128 einschalten
19    SPCR = (1<<SPE)|(1<<MSTR)|(0b11<<SPR0);
20  }
```

(SPE – SPI Enable; MSTR – Master; SPR – Einstellung des Teilerfaktors)

```
22  // Datenaustausch mit dem Joystick
23  // ptr:    Zeiger auf einen 5Byte großen Puffer
24  // LedDat: einzuschaltende Joystick-Leds, Wert 0 bis 3
25  void comSPI(uint8_t *buffer, uint8_t LedDat){
26    PORTB &=~1;              // Slave aktivieren
27                             // 1. Byte senden
28    SPDR = 0b10000000|(LedDat & 0b11);
29    uint8_t i;
30    for (i=0;i<5;i++){
31                             // Warte auf SPI-Übertragung
32      while (!(SPSR & (1<<SPIF))){}
33      buffer[i] = SPDR;
34      if (i<4) SPDR = 0;
35    }
36    PORTB |= 1;              // Slave deaktivieren
37  }
```

- Das SPIF-Bit wird gelöscht, wenn zuerst SPSR gelesen und dann ein Zugriff auf SPDR erfolgt.

Aufgabe 4.2: Joystick-Programm

Schreiben Sie ein Programm, das auf HTerm als Text die Eingabe einer Zahl von 0 bis drei auffordert, die LEDs auf dem Joystick entsprechend dieses Wertes setzt und folgenden Ausgabetext erzeugt:

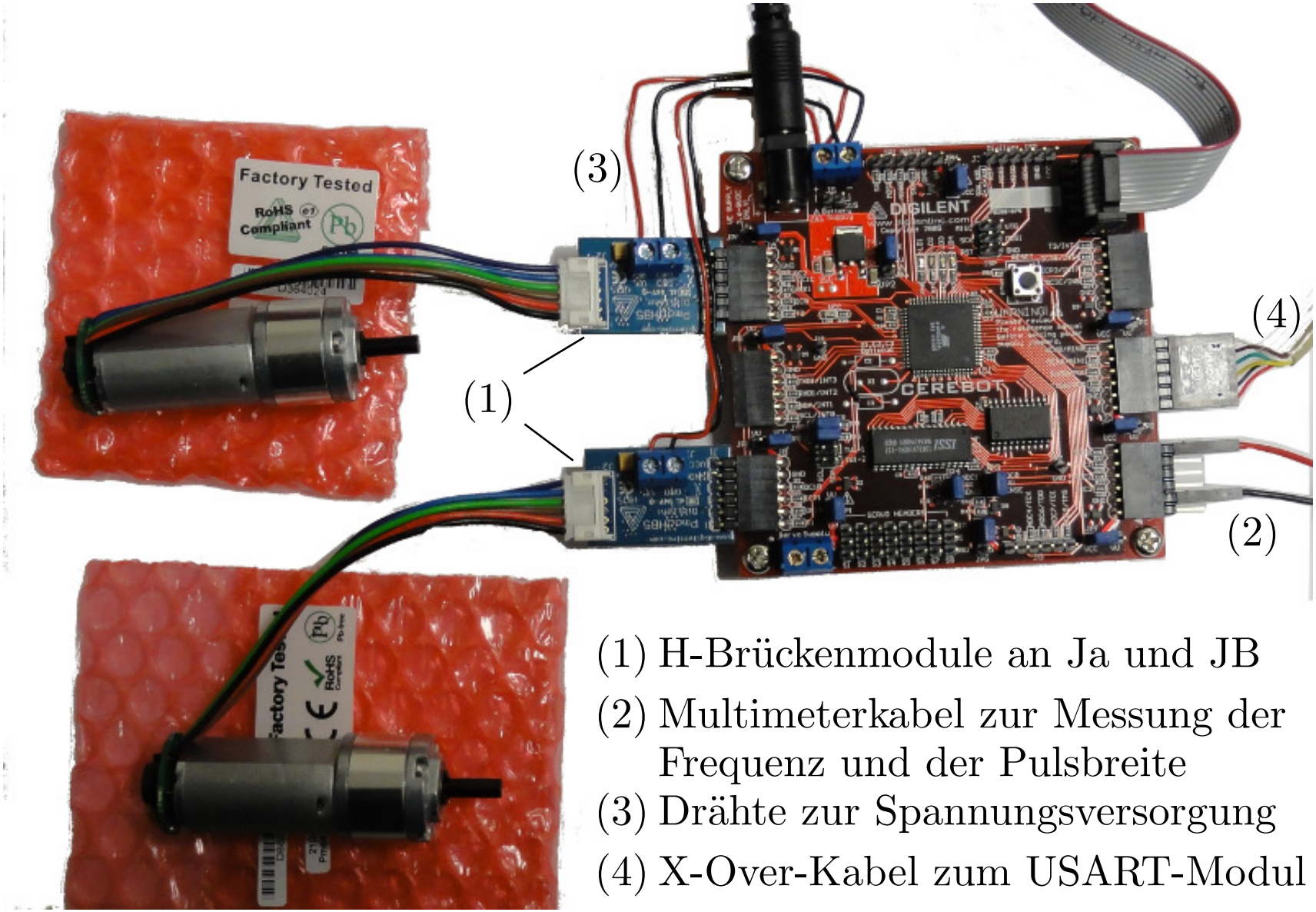
```
x=<gelesener Wert>, y=<gelesener Wert>, BTN1=1/0,
BTN2=1/0, Hebeltaste=1/0
```

Ändern Sie das Programm so, dass die Ausgabe auf dem LD-Display erfolgt.

Aufgabe 4.3: Joystick-LCD-Ausgabe

Schreiben Sie ein Programm, dass die Joystickwerte fortlaufend auf dem LC-Display anzeigt.

Motoren, Wegmessung



(1) H-Brückenmodule an Ja und JB

(2) Multimeterkabel zur Messung der Frequenz und der Pulsbreite

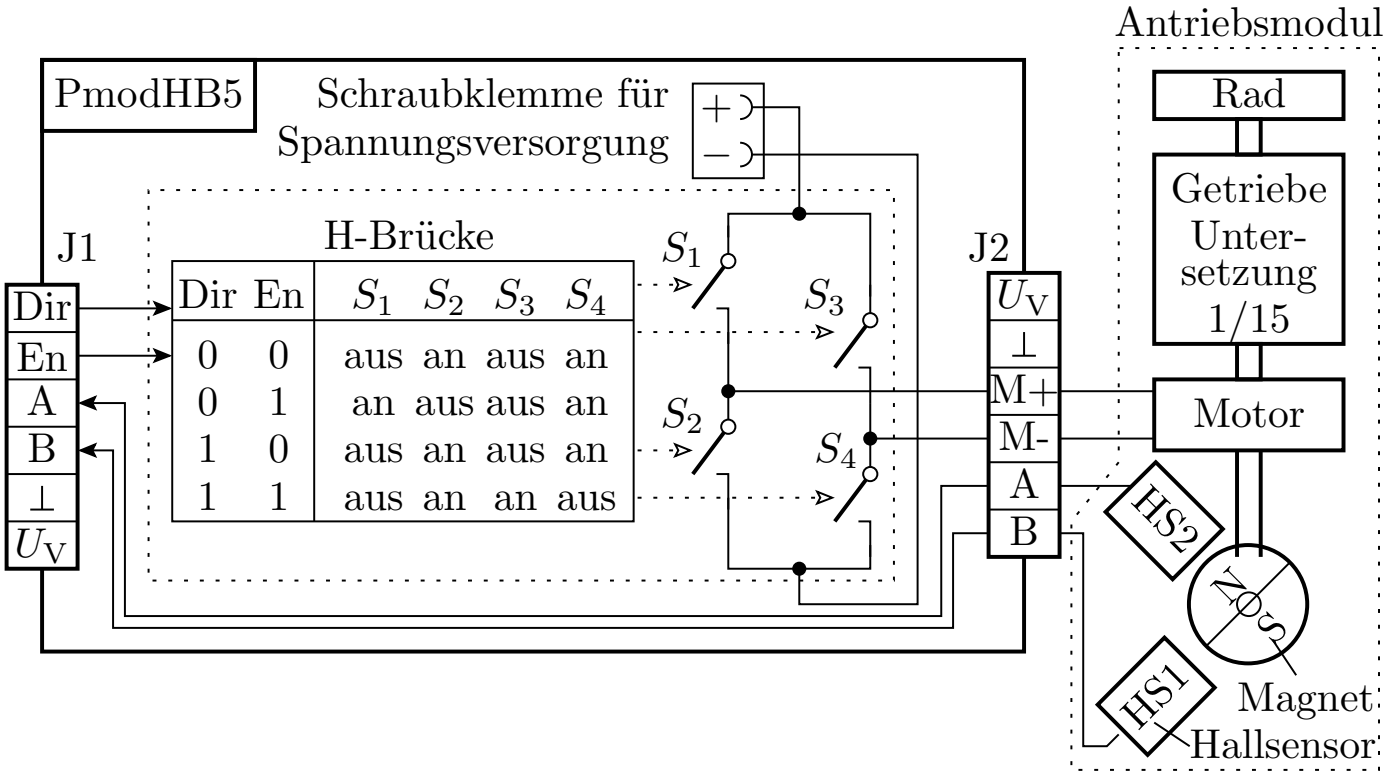
(3) Drähte zur Spannungsversorgung

(4) X-Over-Kabel zum USART-Modul

Die Antriebs- und HB5-Module

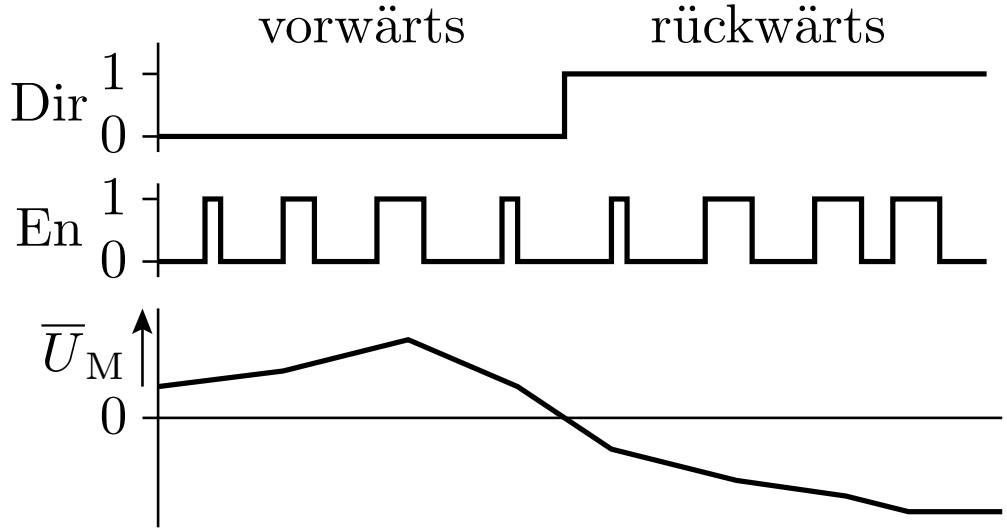
Port-Zuordnung Mikrocontroller

Motor an JB	Motor an JA
PD6	PD7
PB5	PB4
PD4	PD5
PB6	PB7



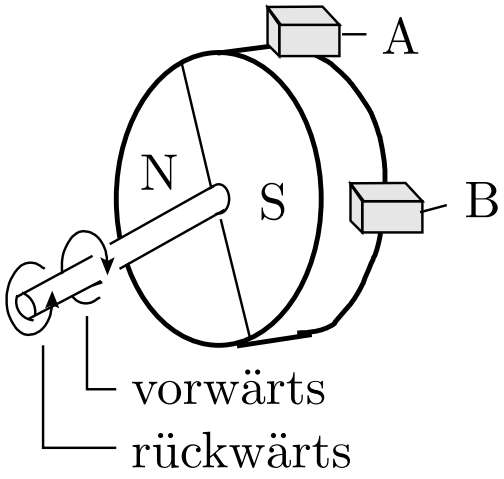
Motoransteuerung

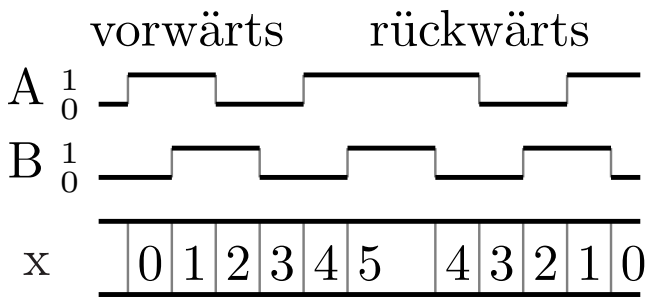
En	1	0	0	1
Dir	0	0	1	1
	+ M −	M	− M +	

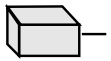


- Einstellung der Drehrichtung über das Richtungsbit Dir.
- Einstellung der Drehgeschwindigkeit über den Anteil der Zeit, in der das En-Signal eins ist (relative Pulsbreite). Schaltabstand im unteren bis mittleren Millisekundenbereich. Ausprobieren!

Positionsmessung





	Hallsensoren
x	Winkel in Viertelkreisschritten
A*, B*	Abtastwerte Zeitschritt zuvor

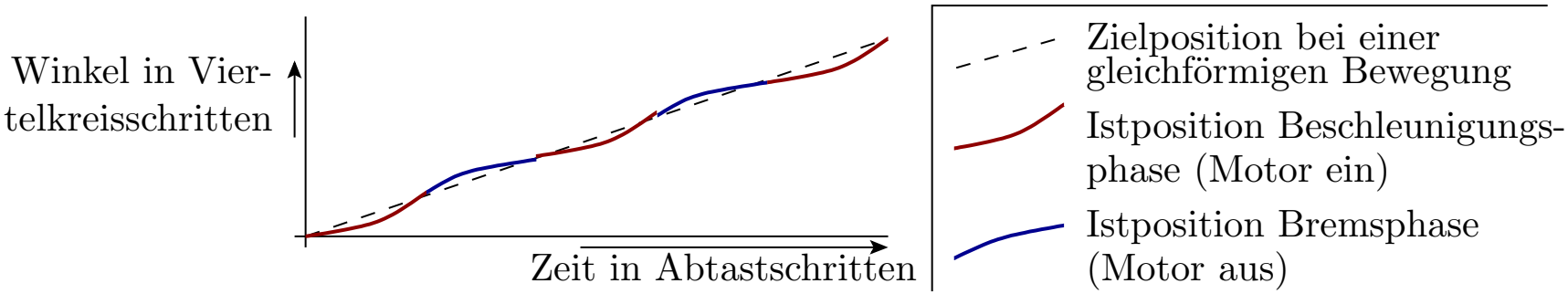
A*	B*	A	B	x
0	0	0	0	—
0	0	0	1	−1
0	0	1	0	+1
0	1	0	1	—
0	1	0	0	+1
0	1	1	1	−1
1	0	1	0	—
1	0	1	1	+1
1	0	0	0	−1
1	1	1	1	—
1	1	1	0	−1
1	1	0	1	+1

- Abtastung mindestens einmal zwischen zwei Signalwechseln.
- Auf den Signalen A und B treten bis zu 600 Signalwechseln pro s auf. Der Mikrorechner muss die Sensorwerte 800 bis 1000 mal pro s einlesen und auswerten.
- Es bietet sich die Byteversendezeit als Wartezeit an.

Positionsreglung

Für die Steuerung des Fahrzeugs ist es wünschenswert, dass der Mikrorechner aus der Vorgabe der Zielposition und Zielgeschwindigkeit die Ein- und Ausschaltzeitpunkte selbst bestimmt. Einfachste Lösung ist ein Zweipunktregler:

- Wenn Vorwärtsbewegung
  - wenn Ziel- größer Istposition, Motor vorwärts einschalten,
  - sonst Motor aus.
- Sonst wenn Rückwärtsbewegung
  - wenn Ziel- größer Istposition Motor rückwärts einschalten,
  - sonst Motor aus.



Konzeption des Testprogramms

Schleife, die 960 mal je Sekunde (Sendezeit für ein Byte)

- auf eine Steuereingabe wartet und
- wenn die Motorsteuerung eingeschaltet ist, für jeden Motor
  - eine Wegmesschritt durchführt,
  - entscheidet, ob Motor ein- oder aus sein soll und
  - die Ergebnisse zur Kontrolle an den PC schickt.

Bei 960 Schleifendurchläufen pro Sekunde kann die Eingabe und die Ausgabe nur je ein Byte umfassen.

Steuerbyte	
`1000 0000`	Antrieb ein
`1000 0001`	Antrieb aus
`aaaa aaa0`	Sollgeschwindigkeit* Motor A
`bbbb bbb0`	Sollgeschwindigkeit* Motor B

* 7Bit-Zweierkomplement
Wertebereich −63 bis +63

Kontrollbyte		
Bit 7	Motor A	-1
Bit 6		+1
Bit 5		Dir
Bit 4		En
Bit 3	Motor B	-1
Bit 2		+1
Bit 1		Dir
Bit 0		En

Aufgabe 4.4: Testbeispiel für Motorsteuerung vorbereiten

- Neues Projekt »TestMotor« mit den Dateien TestMotor.c, Motorsteuerung.c/h und comPC.c/h von der Web-Seite anlegen.
- Spannungungsversorgung abziehen. Beide H-Brücken, Motormodule, serielle Schnittstelle und Multimeter wie auf Folie 15 zusammenstecken. Drähte für die Motorspannungsversorgung abisolieren und festschrauben. Spannungsversorgung anschließen.
- Übersetzen und im Debugger starten.

Test 1: Motor von Hand drehen. Die auf LD1 bis LD4 durchgereichten Sensorsignale kontrollieren. Welche LEDs sind Motor A und welche sind Motor B zugeordnet?

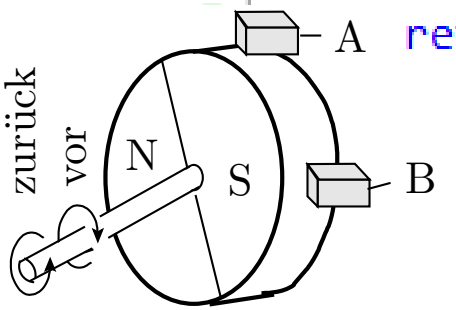
Test 2: HTerm starten.

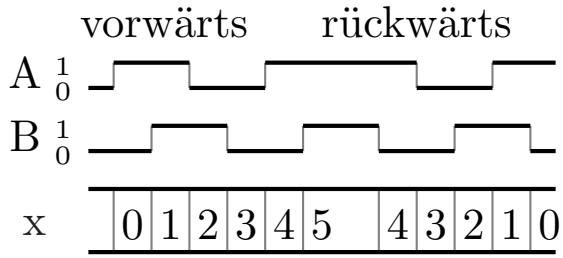
- Eingabe von 0x80. Dauerempfang von ca. 960 Byte pro Sekunde. An PF1 sollte die Hauptschleifenfrequenz von ca. 960 Hz zu messen sein.
- Eingabe von Sollgeschwindigkeiten:
  - gerade Zahl von 0x2...0x7E: Vorwärtsbewegung Motor A
  - gerade Zahl von 0x82...0xFE: Rückwärtsbewegung Motor A
  - ungerade Zahl von 0x3...0x7F: Vorwärtsbewegung Motor B
  - ungerade Zahl von 0x83...0xFF: Rückwärtsbewegung Motor B
  - 0: Motor A hält an
  - 1: Motor B hält an
- Steuerung stoppen durch Eingabe von 0x81. Zeichenempfang hört auf.

Private globale Daten der Motorsteuerung

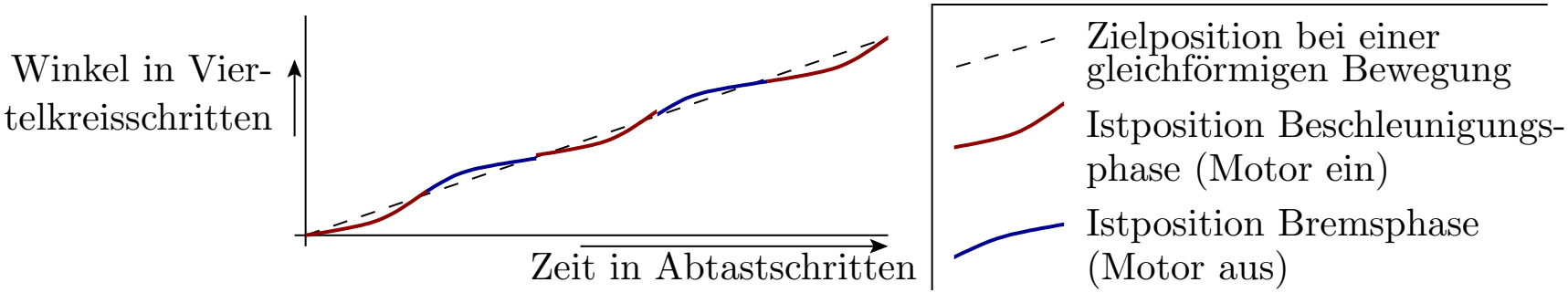
```
11  // Datenstruktur für die Messung und Regelung der
12  // der Motorposition
13  struct MCTRL_t { //
14    uint8_t state;  // aktuelle und vorherige Sensorwerte
15    int32_t CtIst;  // Wegzähler Istwert in 2^-6 Schritten
16    int32_t CtSoll; // Wegzähler Sollwert in 2^-6 Schritten
17    int8_t vSoll;   // Sollgeschwindigkeit in 2^-6
18                    // Schritten je Schleifenzeit
19    uint8_t err;    // Zähler für Unterabtastfehler
20  };
21
22  void initMCTRL(struct MCTRL_t *m, uint8_t x)...
28
29  // private globale Daten der Motorsteuerung
30  struct MCTRL_t MCTRL_MA, MCTRL_MB;
31  uint16_t MCTRL_ts;    // Zähler für Zeitschritte
32  uint8_t  MCTRL_an;    // wenn Steuerung an 1, sonst 0
```

```
45  // Umrechnung der Geberwerte in Wege-Increments
48  uint8_t QuadEnc(struct MCTRL_t *m, uint8_t x){
49    m->state = ((m->state<<2)&0b1100)|(x&0b11);
50    switch (m->state){
51      case 0b0010:
52      case 0b0100:
53      case 0b1011:
54      case 0b1101:
55       m->CtIst+=(1<<6);
56       return 0b10;
57      case 0b0001:
58      case 0b0111:
59      case 0b1000:
60      case 0b1110:
61       m->CtIst-=(1<<6);
         return 0b01;
        case 0b0011:
        case 0b0110:
        case 0b1100:
        case 0b1001:
         m->err++;
         return 0b11;
    }
    return 0;
  }
```





A*	B*	B	A	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




```
63  // Motorregelung: Berechnet aus Motordaten Steuerbits
64  // Rückgabebit 0: En (Motorfreigabe)
65  // Rückgabebit 1: Dir (Drehrichtung)
66  uint8_t Motorregelung(struct MCTRL_t *m){
67    m->CtSoll += m->vSoll;// neue Sollposition
68      // Sollbewegung vorwärts und Soll- größer Istwert
69    if ((m->vSoll>0)&& (m->CtSoll>m->CtIst)){
70        return 0b01;       // Motor vorwä. ein (Dir=0, En=1)
71    } // Sollbewegung rückwärts und Soll- kleiner Istwert
72    else if ((m->vSoll<0) && (m->CtSoll<m->CtIst)){
73      return 0b11;         // Motor rückw. ein (Dir=1, En=1)
74    }
75    return 0;              // Motor aus (Dir=0, En=0)
76  }
```

```
79  // Kopieren eines Bits aus einem in ein anderes Byte
80  // d, nd: Byte und Bitnummer des Kopierziels
81  // s, ns: Byte und Bitnummer der Kopierquelle
82  uint8_t cpybit(uint8_t d, uint8_t nd,
83                 uint8_t s, uint8_t ns)...
87
88  // Schrittfunktion der Gesamtsteuerung für beide Motoren
89  void Antriebsschritt(){
90    uint8_t wa, wb, qa, qb, msa, msb, sbyte;
91    MCTRL_ts++;          // Zeit weiterschalten
92    // für beide Motoren neue Sensorwerte verarbeiten
93    wa = ((PIND>>4)&0b10)|((PINB>>7)&0b01); //Sensoren A
94    wb = ((PIND>>3)&0b10)|((PINB>>6)&0b01); //Sensoren B
95    qa = QuadEnc(&MCTRL_MA, wa);  //Sensorwerte A => Inc.
96    qb = QuadEnc(&MCTRL_MB, wb);  //Sensorwerte B => Inc.
```

```
98    // für beide Motoren Reglerfunktion ausführen
99    msa = Motorregelung(&MCTRL_MA);    // Berechnung (DirA,EnA)
100   msb = Motorregelung(&MCTRL_MB);    // Berechnung (DirB,EnB)
101   if ((PORTB&(1<<4))!= (msa&1)){     // wenn sich ENA ändert
102     PORTF = cpybit(PORTF, 1, msa, 0); // EnA => PF1
103     PORTB = cpybit(PORTB, 4, msa, 0); // EnA => PB4
104   }
105   if ((PORTB&(1<<5))!= (msb&1)){     // wenn sich ENB ändert
106     PORTF = cpybit(PORTF, 2, msb, 0); // EnB => PF2
107     PORTB = cpybit(PORTB, 5, msb, 0); // EnB => PB5
108   }
109   PORTD = cpybit(PORTD, 7, msa, 1);    // DirA => PD7
110   PORTD = cpybit(PORTD, 6, msb, 1);    // DirB => PD6
111
112   // Alle Daten in einem Sendebyte zusammenfassen
113   sbyte = (qa<<6) | (msa<<4) | (qb<<2) | msb;
114
115   PORTF |= 1;      // PF0 wird zur Messung der Schleifen-
116   sendByte(sbyte);// frequenz und der anteiligen Wartezeit
117   PORTF &= ~1;     // während des Sendens gesetzt.
118 }
```

Zusammengefasst besteht ein Antriebsschritt darin, für beide Motoren:

- Sensorwerte lesen [93-94]
- damit Weg-Increments bestimmen und Istposition aktualisieren [95-96]
- Sollposition und Motorsteuerwerte (En, Dir) berechnen [99-100]
- Steuerwerte an Motor und Kontroll-ausgänge weitergeben [101-110]
- Die Increment und Motorsteuerwerte in einem Byte versenden [112-117]

Aufbau des Kontrollbytes

Bit 7	Motor A	-1
Bit 6		+1
Bit 5		Dir
Bit 4		En
Bit 3	Motor B	-1
Bit 2		+1
Bit 1		Dir
Bit 0		En

Die folgende Funktion empfängt und verarbeitet die Steuerbytes:

Aufbau eines Steuerbytes

`1000 0000`	Antrieb ein
`1000 0001`	Antrieb aus
`aaaa aaa0`	Sollgeschwindig-keit* Motor A
`bbbb bbb0`	Sollgeschwindig-keit* Motor B

* 7Bit-Zweierkomplement
Wertebereich -63 bis $+63$

```
125 void Kommandointerpreter(){
126   int8_t dat;
127   if (getByteNB(&dat)){
128     if (dat==-0x80){          // Antrieb einschalten
129       MCTRL_an   = 1;         // Zustand setzen
130       MCTRL_ts   = 0;         // Zähler initialisieren
131       uint8_t wa = ((PIND>>4)&0b10)|((PINB>>7)&0b01);
132       uint8_t wb = ((PIND>>3)&0b10)|((PINB>>6)&0b01);
133       initMCTRL(&MCTRL_MA, wa);
134     } initMCTRL(&MCTRL_MB, wb);
136     else if (dat==-0x7F){   // Antrieb ausschalten
137     } initMotorsteuerung(); // und Motoren aus
139     else if ((dat&1)==0){ // gerade:Sollgeschw. A
140     } MCTRL_MA.vSoll=dat/2;
142     else{                       // ungerade: Sollgeschw. B
143     } MCTRL_MB.vSoll=dat/2;
  } }
```

```
19  int main(void){
20    // Ausgänge: PD6..7, PB4..5, PE4..7
21    DDRB = 0b00110000; // EnA, EnB
22    DDRD = 0b11000000; // DirA, DirB
23    DDRE = 0b11110000; // LD4 bis LD1
24    DDRF = 0b111;      // PF0 bis PF2
25    initMotorsteuerung();
26    initUSART0();
28    while(1) {
29      // Ausgabe der Sensorwerte an LD1 bis LD4
30          PORTE=cpybit(PORTE, 4, PIND, 5);
31          PORTE=cpybit(PORTE, 5, PINB, 7);
32          PORTE=cpybit(PORTE, 6, PIND, 4);
33          PORTE=cpybit(PORTE, 7, PINB, 6);
34      Kommandointerpreter();
35      if (MotorenAn()){
36        Antriebsschritt();
        }
    }
}
```

Datenauswertung mit Matlab

Der Mikrocontroller liefert 960 Bytes pro Sekunde. Wie können wir das Verhalten der Regelung kontrollieren? Dafür eignet sich ein Matlabprogramm mit einem Programmablauf siehe rechts.

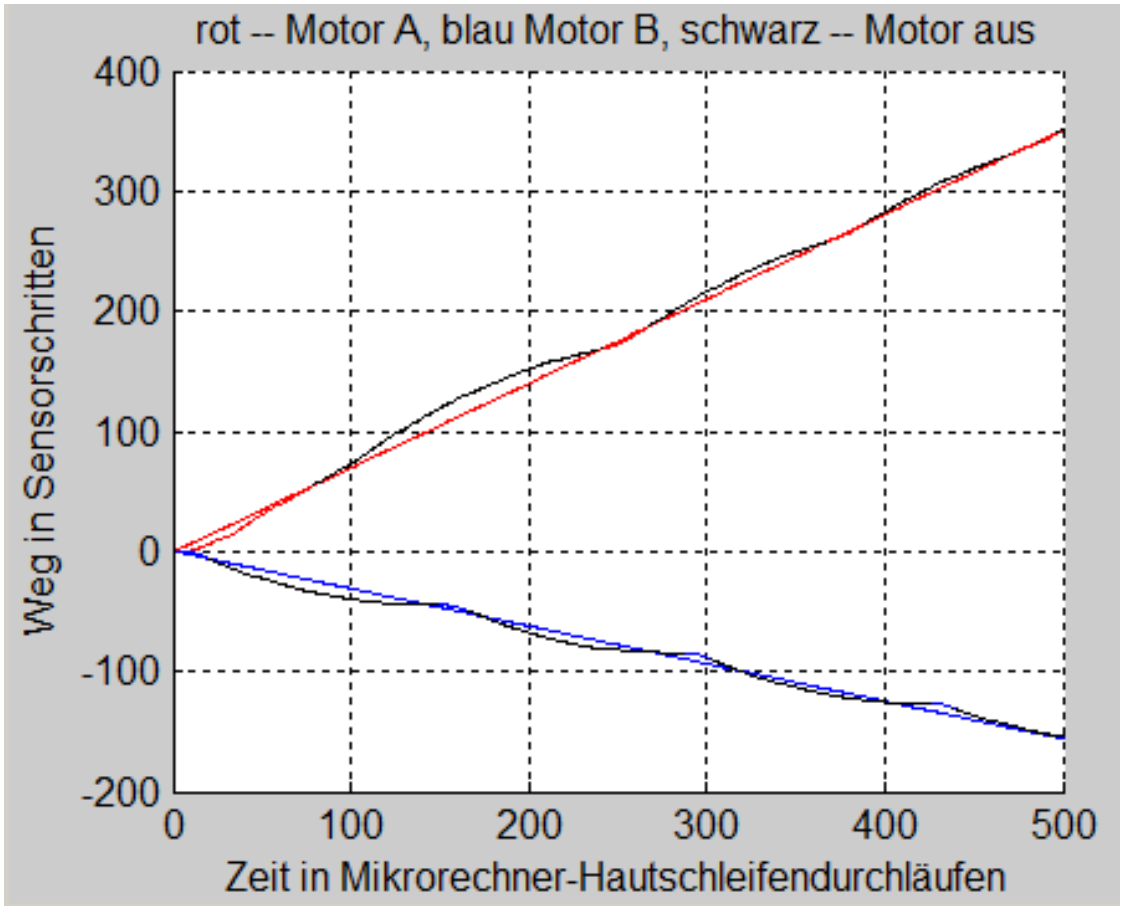
- serielle Verbindung öffnen
- Startkommmado 0x80 senden
- Motorgeschwindigkeiten senden
- wiederhole für eine gegebene Anzahl
  - Byte empfangen
  - in seine einzelnen Bits zerlegen
  - Soll- und Istpositionen berechnen
  - in Tabellendarstellung bringen
  - in eine Graphik einzeichnen

- HTerm Verbindung schließen.
- Matlab öffnen.
- Das m-Skript TestMotor.m von der Webseite laden, öffnen und starten.

Tabellarische Ausgabe

Zeit	Motor A						Motor B					
	En	Dir	+	-	ist	soll	En	Dir	+	-	ist	soll
2	1	0	0	0	0	0.70	1	1	0	0	0	-0.31
3	1	0	0	0	0	1.41	1	1	0	0	0	-0.63
4	1	0	0	0	0	2.11	1	1	0	0	0	-0.94
5	1	0	0	0	0	2.81	1	1	0	0	0	-1.25
6	1	0	0	0	0	3.52	1	1	0	0	0	-1.56
7	1	0	1	0	1	4.22	1	1	0	0	0	-1.88
8	1	0	0	0	1	4.92	1	1	0	0	0	-2.19
9	1	0	0	0	1	5.63	1	1	0	0	0	-2.50
10	1	0	0	0	1	6.33	1	1	0	1	-1	-2.81
11	1	0	1	0	2	7.03	1	1	0	0	-1	-3.13
12	1	0	0	0	2	7.73	1	1	0	0	-1	-3.44

Graphische Ausgabe



Erläuterung des Matlabprogramms

```
clear;              % Daten löschen
clf;                % Plot löschen

% Festlegunge der Geschwindigkeiten und der Bewegungsdauer
Testzeit= 500;   % Zeit in Schleifenperioden (sp)
vA =  45;        % Geschwindigkeit in Sensorschritten pro sp
vB = -20;        % mal 64, WB: -63 bis 63, ganzzahlig

s = serial('COM9','BAUD',9600);
set(s,'Timeout',1000); % Timeout 1000s (kein sofortiger Abbruch
fopen(s);              % wenn der Prozessor im Debugger hält)
                       % SIO bei Absturz freigeben /Alternative
                       % Neustart von Matlab nach jedem Fehler
closeFID = onCleanup(@() fclose(s));
fwrite(s,[128]);       % Startkommando
```

```
%Geschwindigkeiten als 7Bit-Zweierkoplement * 2 (+1)
if vA<0 v = 128+vA; else v = vA; end;
fwrite(s, v*2);          % A erwartet geraden Wert
if vB<0 v = 128+vB; else v = vB; end;
fwrite(s, v*2+1);        % B erwartet ungeraden Wert


istA(1) = 0;             % Vektoren, zur Speicherung der Ist- und
sollA(1)= 0;             % Sollwerte der Motorpositionen in
istB(1) = 0;             % Anhängigkeit von der Zeit (in Mikro-
sollB(1)= 0;             % rechnerschleifendurchläufen)

fprintf('\n-----------------------------------------------|\n');
fprintf('Zeit|       Motor A              |          Motor B        |\n');
fprintf('    | En Dir + - |  ist  soll | En Dir + - |  ist  soll |\n');
hold on;    % graphische Ausgabe weiterzeichnen
```

```
for t =2:Testzeit
    dat=fread(s,1);     % ein Datenbyte vom Mikrorechner lesen
    if length(dat)<1, return, end % Abbruch, falls keine Daten

    % Aufspalten des Empfangsbytes in seine einzelnen Bits
    pA  = bitget(dat, 8);
    nA  = bitget(dat, 7);
    DirA= bitget(dat, 6);
    EnA = bitget(dat, 5);
    pB  = bitget(dat, 4);
    nB  = bitget(dat, 3);
    DirB= bitget(dat, 2);
    EnB = bitget(dat, 1);
    istA(t) = istA(t-1) + pA - nA;
    sollA(t)= sollA(t-1)+ single(vA)/64;
    istB(t) = istB(t-1) + pB - nB;
    sollB(t)= sollB(t-1)+ single(vB)/64;
```

```
    % tabellarische Ausgabe
    fprintf('%4i|  %i  %i  %i %i |',t, EnA, DirA, pA, nA);
    fprintf('%4i %6.2f | ', istA(t), sollA(t));
    fprintf(' %i  %i  %i %i |', EnB, DirB, pB, nB);
    fprintf('%4i %6.2f |\n', istB(t), sollB(t));

    % graphische Ausgabe
    plot([t],[sollA(t)], 'r', [t],[sollB(t)], 'b');
    if EnA plot([t],[istA(t)], 'r');
    else   plot([t],[istA(t)], 'k'), end;
    if EnB plot([t],[istB(t)], 'b');
    else   plot([t],[istB(t)], 'k'), end;
end;
```

```
hold off;          % Weiterzeichnen der Graphik abschließen
xlabel('Zeit in Mikrorechner-Hautschleifendurchläufen')
ylabel('Weg in Sensorschritten');
title('rot -- Motor A, blau Motor B, schwarz -- Motor aus');
grid on;

fwrite(s,[129]); % Motoren ausschalten
fclose(s);       % serielle Schnittstelle schließen
```

*Aufgabe 4.5: Aufbau des Fahrzeugs

- Bauen Sie das Fahrzeug mit Motorbaugruppen, Mikrorechner, HBrücken und Blue-Tooth-Modul zusammen. Aufbauhinweise siehe Webseite »Montage.pdf«.
- Programmieren Sie das Beispielprogramm in den Mikrorechner und testen Sie eine Geradeausfahrt mit Steuerung über Matlab und Bluetooth.
- Entwickeln Sie das Matlab-Programm so weiter, dass das Fahrzeug eine vorgegebene Bahn abfährt.

Hinweis: Spalten Sie aus dem vorgegeben Matlab-Programm eine Funktion ab, die eine Bewegung mit den Aufrufparametern

- Zeit in Schleifenschritten und
- je Motor Geschwindigkeit in Schleifenschritten 64-stel Schleifenschritten.

ausführt (Funktionsdefinition in Matlab siehe »google matlab function«). Ergänzen Sie ein Hauptprogramm, das die serielle Schnittstelle öffnet und die Bewegungsfunktion mehrfach aufgeruft.

*Aufgabe 4.6: Motorregelung verbessern

Die Zweipunktregelung läuft sehr ruckhaft. In der Matlab-Graphik auf Folie 33 erkennt man, dass der Motor etwa 5 bis 10 Zeitschritte maximal beschleunigt und danach 50 bis 100 Zeitschritte ausläuft. Wie könnte man das verbessern?

Hinweise finden Sie mit google unter dem Stichwort »empirischer Reglerentwurf«.