Formelsammlung

LOKAL

1.5.2 TG Informationstechnik

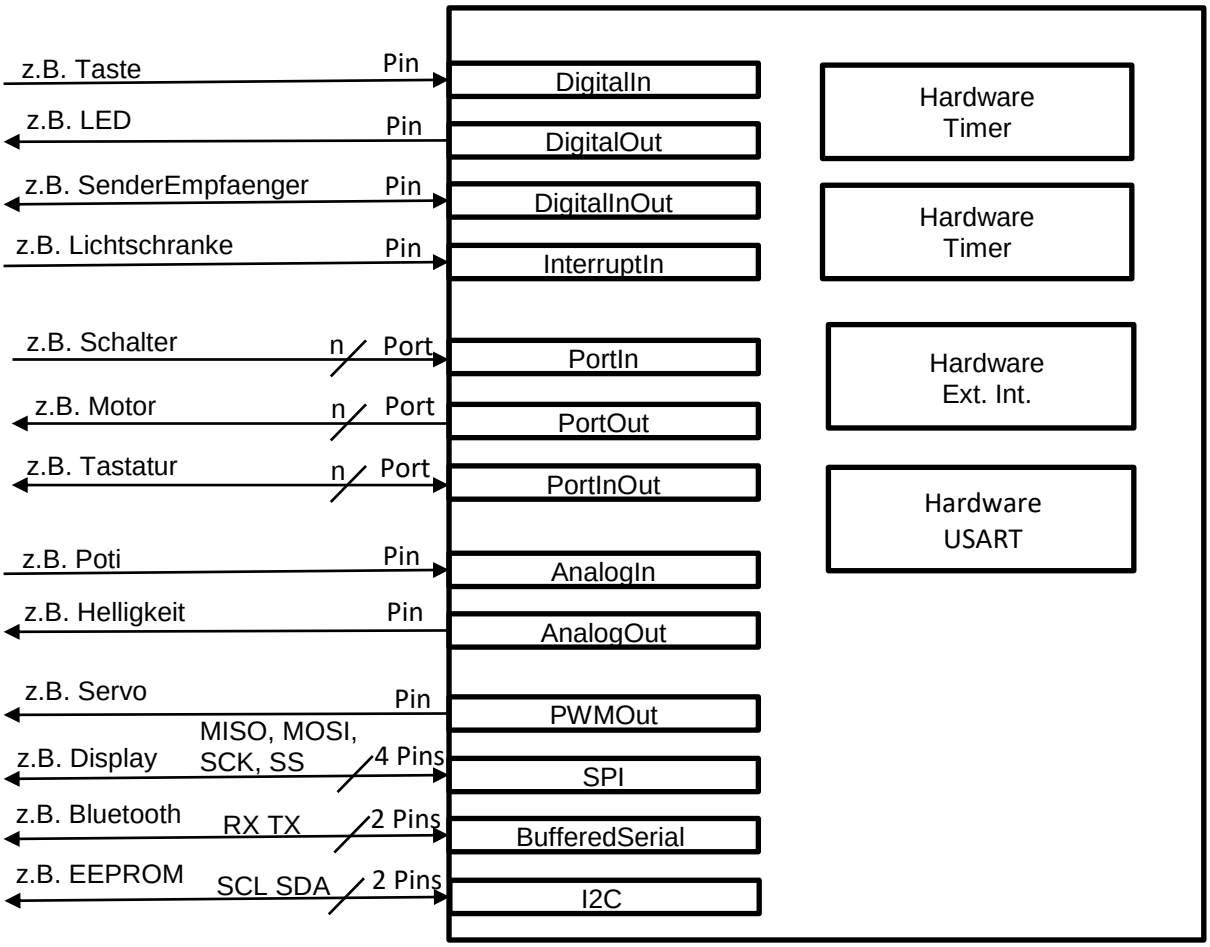
Version: V 4.28

Gültig ab Abitur 2024

Inhaltsverzeichnis:

1 Hardware - Mikrocontrollertechnik

1.1 Blockschaltbild „Prüfungscontroller“



1.2 Hochsprache C/CPP

1.2.1 Datentypen

Datentyp	Bits	Vorzeichen	Wertebereich
`unsigned char`	8	+	0 .. 255
`(signed) char`	8	- +	-128 ..127
`unsigned int`	32/16	+	0 .. 4294967295 bzw. 0 .. 65535
`int`	32/16	- +	-2147483648 .. 2147483647 bzw. 32768 .. 32767
`float`	32	- +	-3,4E38 .. 3,4E38

1.2.2 Zeiger und Referenzen

int x=127; //Wert
int *y; //Zeiger

*y=x; //der Zeiger weist auf eine Variable mit dem Wert von x
y=&x; //der Zeiger bekommt die Adresse der Variable x im Speicher
Beispiel:

	Adresse	RAM
x	0x20000000	127
y	0x20000004	0x20000000

printf("%d %x %d\r\n",x,(int)y,*y);
liefert folgende Ausgabe: 127 0x20000000 127

1.2.3 Operatoren

Mathematische Operatoren	
++	Inkrement
--	Dekrement
-	Vorzeichen
*	Multiplikation
/	Division
%	Modulo, Rest der Division
+	Plus
-	Minus

Priorität
Höchste
Niedrigste

Vergleichs- und logische Operatoren	
!	NOT
>	Größer
>=	Größer gleich
<	Kleiner
<=	Kleiner gleich
==	Gleich
!=	Ungleich
&&	AND
||	OR

Da ein Gleichheitszeichen in C ein Zuweisungsoperator ist, weist man z.B. mit `x = 10;` der Variablen x den Wert `10` zu.

Bitweise Operatoren	
&	UND
|	ODER
^	EXOR
~	Einerkomplement
<<	Nach links schieben
>>	Nach rechts schieben

Kurzschreibweisen	
+=	`x += 3; wie x = x + 3`
-=	`x -= 3; wie x = x - 3`
*=	`x *= 5; wie x = x * 5`
/=	`x /= 7; wie x = x / 7`

Beispiele	Ergebnisse
`x = 10;`	
`y = ++x;`	`y = 11`
`y = x++;`	`y = 10`
`y = 0x11;`	
`y = y<<1;`	`y = 0x22`
(Bitweise um 1 nach links schieben)	

1.2.4 Schleifen

1.2.4.1 FOR-Schleife (zählergesteuerte Schleife)

Schleife, mit einer genau berechenbaren Anzahl an Wiederholungen.

```
for (<zaehlvariable=startwert>;<bedingung>;<schrittweite>) {
   …
}
```

- `startwert` Anfangswert der Zählvariablen
- `bedingung` Schleife wird so lange durchlaufen, wie die Bedingung wahr ist
- `schrittweite` Anweisung zum Erhöhen oder Erniedrigen der Zählvariablen

Beispiel:

```
// DigitalOut ausgang(PC_0) 10x invertieren
for (int i=0; i<10; i++) {
   ausgang = !ausgang;
}
```

1.2.4.2 WHILE-Schleife (kopfgesteuerte Schleife)

Schleife, die wiederholt wird, so lange die Bedingung am Schleifenanfang erfüllt ist.

```
while (<bedingung>) {
   …
}
```

Solange die am Anfang stehende **Bedingung erfüllt ist,** wird die Schleife wiederholt. Die Prüfbedingung steht **vor den Anweisungen**, sie heißt deshalb **kopfgesteuerte Schleife.**

Wenn die am Schleifenanfang stehende **Bedingung nicht erfüllt ist,** dann wird die gesamte Schleife übersprungen.

Beispiel:

```
// Solange der Taster DigitalIn taster(PA_1) gedrückt ist, wird der
// Ausgang DigitalOut ausgang(PC_0) invertiert
while (taster==true) {
   ausgang = !ausgang;
}
```

1.2.4.3 Do-WHILE-Schleife (fußgesteuerte Schleife)

Schleife, die mindestens einmal durchlaufen wird, da erst am Ende der Schleife mit der Überprüfung der Bedingung entschieden wird, ob die Schleife wiederholt werden muss.

```
do {
   …
} while (<bedingung>);
```

Beispiel:

```
// Die Schleife wird maximal 100 mal und minimal 1 mal durchlaufen. Sie wird
früh-
// zeitig abgebrochen, wenn der Taster DigitalIn taster(PA_1) gedrückt (=1) wird.
x = 100;
do {
   x--;
} while ((x>0) && taster==0);
```

1.2.5 Programmverzweigungen

1.2.5.1 Einfache Verzweigung mit if

Bei der if-Anweisung werden die Anweisungen innerhalb des if-Blocks nur dann ausgeführt, falls die Bedingung wahr ist.

```
if (<bedingung>) {
   <anweisung1>;
   <anweisung2>;
   …
}
```

Beispiel:

```
// Wenn taster1 gedrückt ist, soll ausgang1 eins und ausgang2 null werden.
// Drückt man dagegen taster2, wird nur ausgang2 zu eins.
if (taster1==1) {
   ausgang1 = 1;      // Block mit mehreren Anweisungen wird ausgeführt,
   ausgang2 = 0;      // wenn die Bedingung hinter if wahr ist
}
if (taster2==1)
   ausgang2 = 1;      // Nur eine Anweisung, keine {} notwendig
```

1.2.5.2 Zweiseitige Verzweigung mit if

Bei der if/else-Anweisung kann zwischen **zwei Alternativen** entschieden werden. Ist die Bedingung wahr, so wird die erste Alternative (if-Block), ansonsten die zweite Alternative (else-Block) an Anweisungen ausgeführt.

```
if (<bedingung>) {
   <anweisung1>;
   <anweisung2>;
   …
} else {
   <anweisung3>;
   <anweisung4>;
   …
}
```

Beispiel:

```
// Wenn taster1 gedrückt ist, soll ausgang1 eins und ausgang2 null werden,
// andernfalls soll ausgang1 null und ausgang2 eins werden.
if (taster1==1) {
   ausgang1 = 1;      // Block mit mehreren Anweisungen wird ausgeführt,
   ausgang2 = 0;      // wenn die Bedingung hinter if wahr ist
} else {
   ausgang1 = 0;      // Block mit mehreren Anweisungen wird ausgeführt,
   ausgang2 = 1;      // wenn die Bedingung hinter if nicht wahr ist
}
```

1.2.5.3 Mehrere Verzweigungen mit if

```
if (<bedingung1>) {
   <anweisung1>;
   …
} else if (<bedingung2>) {
   <anweisung2>;
   …
} else {
   <anweisung3>;
   …
}
```

1.2.5.4 Fallunterscheidung mit switch

Mit der switch-Anweisung kann aus einer **Reihe von Alternativen** ausgewählt werden. Es ist zulässig, dass mehrere Möglichkeiten gültig sind und dieselbe Wirkung haben. Sie werden nacheinander aufgelistet. Passt keine der Möglichkeiten, dann wird die **default**-Einstellung ausgeführt.

Achtung! Auf keinen Fall **break** vergessen!!!

```
switch (<vergleichswert>) {
   case <wert1>:
      <anweisung1>;
      <anweisung2>;
      …
      break;
   case <wert2>:
      <anweisung3>;
      <anweisung4>;
      …
      break;
   …
   default:
      <anweisung5>;
      …
}
```

Beispiel:

```
// In der Variablen ergebnis ist ein Messergebnis oder eine Zahl gespeichert.
// Abhängig vom genauen Wert sollen nun bestimmte Reaktionen erfolgen.
switch (ergebnis) {
   case 0x00:
   case 0x10:
   case 0x20:
      ausgang1 = 1;
      break;
   case 0x30:
      ausgang1 = 0;
      break;
   case 0x40:
      ausgang1 = ~ausgang1;
      break;
   default:
      ausgang2 = 1;
      break;
}
```

Hinweis: **Switch-Variablen** müssen einen **einfachen Datentyp** verwenden. Hinter **case** müssen **Konstanten** stehen. Dies können mit #define am Anfang des Programms deklariert werden.

Beispiel:

```
# define rechts  0x10     // ohne Semikolon!!
# define links  0x20
# define rechtskurve 0b0100
# define linkskurve 0b1000

unsigned char richtung;
…

switch (richtung) {
   case rechts:
      motor = rechtskurve;
      break;
   case links:
      motor = linkskurve;
      break;
   default:
      motor = vorwaerts;
      break;
}
```

1.2.5.5 Operationen (Unterprogramme, Funktionen)

Deklaration von Operationen

Beispiele:

```
void addieren(void);                  // ohne Rückgabewert, ohne Parameter
void zeitms(int msec);                // ohne Rückgabewert, mit Parameter
float berechneQuadrat(float pQ);      // mit Rückgabewert, mit Parameter
```

Definition von Operationen

Beispiel:

```
int a, result;                        // globale Variablen
void addieren(void) {                 // Operationsname
   result = a + a;                    // Anweisung(en)
}
```

Operationen mit Übergabewert

Beispiel:

```
void zeitms(int msec) {               // Übergabewert msec
   int t1;                            // lokale Variable
   for (t1=msec;t1!=0;t1--)
      wait_us(1000);                  // Zeitschleife;
}
```

Operationen mit Rückgabewert

Beispiel:

```
float berechneQuadrat(float pQ=10) {  // Parameter mit Standardwert
   return pQ*pQ;                      // Rückgabewert
}
```

1.2.6 Beispiel eines C/CPP-Programms

Beispiel: main.cpp:

```
#include "mbed.h"                                   //includes
#include "platform/mbed_thread.h"

// Blinking rate in milliseconds                    //defines
#define BLINKING_RATE_MS 500

InterruptIn anforderung(PA_1);                      //Deklarationen
PortOut ampel(PortC,0xFF);

volatile bool angefordert=false;                    //globale Variablen

char phasen[10]= {                                  //array
    0b01001001, //0 rot - rot - rot
    0b01001101, //1 rot - rot - rotgelb
    0b01001010, //2 rot - rot - grün
    0b01001100, //3 rot - rot - gelb
    0b01001001, //4 rot - rot - rot
    0b01101001, //5 rot - rotgelb - rot
    0b01010001, //6 rot - grün - rot
    0b01100001, //7 rot - gleb - rot
    0b01001001, //8 rot - rot - rot
    0b10001001}; //9 grün - rot - rot

void anfordern(void){                               //Unterprogramme und ISR
   angefordert=true;
}

int main(){                                         //Hauptprogramm
    int z=0;                                        //lokale Variablen

    anforderung.rise(&anfordern);                   //Initialisierungen
    anforderung.mode(PullDown);

    while (true) {                                  //Endlosschleife
        ampel=phasen[z];                            //Arrayinhalt ausgeben
        thread_sleep_for(BLINKING_RATE_MS);         //warten
        if ((z==0||z==4||z==8)&&angefordert)        //Fußgängeranforderung auswerten
        {
            ampel=phasen[9];                        //Ausgabe
            thread_sleep_for(BLINKING_RATE_MS); //warten
            angefordert=false;
        }
        z++;
        if (z>8) z=0;
    }
}
```

1.3 On Chip Peripherie

1.3.1 Ports: Eingabe und Ausgabe

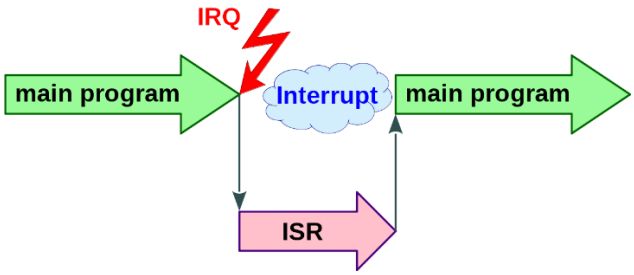
	Befehl	Beispiel
Parallele Ausgabe		
Deklaration	`PortOut name(Port,Maske);`	`PortOut ampel(PortC,0xFF);`
	• Mit Maske können die Bits ausgewählt werden, die ausgegeben werden sollen. • Mögliche Werte für Port: `PortA`, `PortB`, `PortC`	
Verwendung	`meinPortname = Wert;`	`ampel = 0b00000001;`
Einzelbitausgabe		
Deklaration	`DigitalOut name(Portpin);`	`DigitalOut roteLED(PC_0);`
	• Mögliche Werte für Portpin: `PA_0 .. PA_15, PB_0 .. PB_15, PC_0 .. PC_15`	
Verwendung	`name = wert;`	`roteLED=1;` `roteLED=0;`
Parallele Eingabe		
Deklaration	`PortIn name(Port,Maske);`	`PortIn schalter(PortB,0b11111111);`
	• Mit Maske können die Bits ausgewählt werden, die eingelesen werden sollen. • Mögliche Werte für Port: `PortA`, `PortB`, `PortC`	
Konfiguration	`name.mode(pull);`	`schalter.mode(PullDown);`
	• Mögliche Werte für `pull`: `PullDown`, `PullUp`	
Verwendung	`wert = name;`	`ampel = schalter;` `if ( schalter==25 )…`
Einzelbiteingabe		
Deklaration	`DigitalIn name(Portpin);`	`DigitalIn taste(PA_1);`
	Mögliche Werte für `Portpin`: `PA_0 .. PA_15`, `PB_0 .. PB_15`, `PC_0 .. PC_15`	
Konfiguration	`name.mode(pull);`	`taste.mode(PullDown);`
	Mögliche Werte für `pull`: `PullDown`, PullUp	
Verwendung	`wert = name;`	`bool x = taste;` `if (taste==0)…` (oder `false`)
Bidirektionale Ports		
Parallele Ein-, Ausgabe		
Deklaration	`PortInOut name(Port,Maske);`	`PortInOut se1Out(PortB,0xFF);`
	• Mit Maske können die Bits ausgewählt werden, die ausgegeben werden sollen. • Mögliche Werte für `Port`: `PortA`, `PortB`, `PortC`	

Konfiguration	`name.output(); //als Ausgang`
	oder
	`name.input(); //als Eingang`
		`se1InOut.output();`
		oder
		`se1InOut.input();`

	`name.mode(einstellung);`	`se1InOut.mode(OpenDrain);`
	Mögliche Werte für `einstellung`: `PullUp`, `PullDown`, `OpenDrain`	

Verwendung	`name = wert; //als Ausgang`
	`wert =name; //als Eingang`
		`se1InOut = 0b10101010;`
		`int x = se1InOut;`

1.3.2 Externer Interrupt



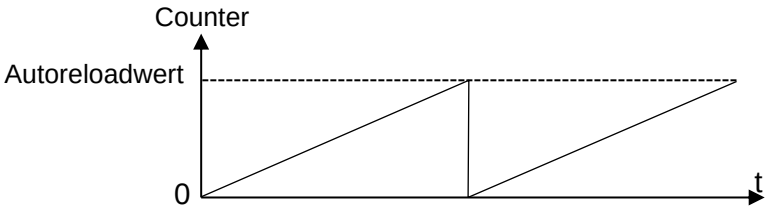
	Befehl	Beispiel
Externer Interrupt		
Deklaration	`InterruptIn name(Portpin);`	`InterruptIn taste(PA_1);`
	• Mögliche Werte für `Portpin`: `PA_0 .. PA_15, PB_0 .. PB_15, PC_0 .. PC_15`	

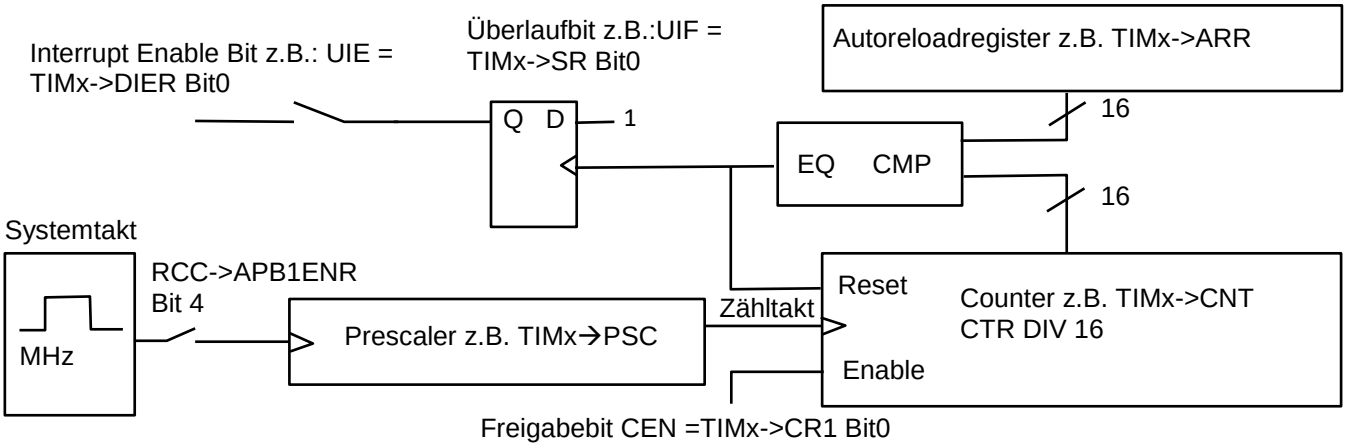
Konfiguration Flanke und ISR	`name.rise(&isr); //steigende Flanke`
	oder
	`name.fall(&isr); //fallende Flanke`
	(beides gleichzeitig möglich)
		`taste.rise(&isrTaste);`
		oder
		`taste.fall(&isrTaste);`

Freigabe	`name.enable_irq();`	`taste.enable_irq();`
Sperre	`name.disable_irq();`	`taste.enable_irq();`
Globale Freigabe	`__enable_irq();`	
Globale Sperre	`__disable_irq();`	

1.3.3 Timer

Prinzip:





Beispiel STM32L152

Befehl	C

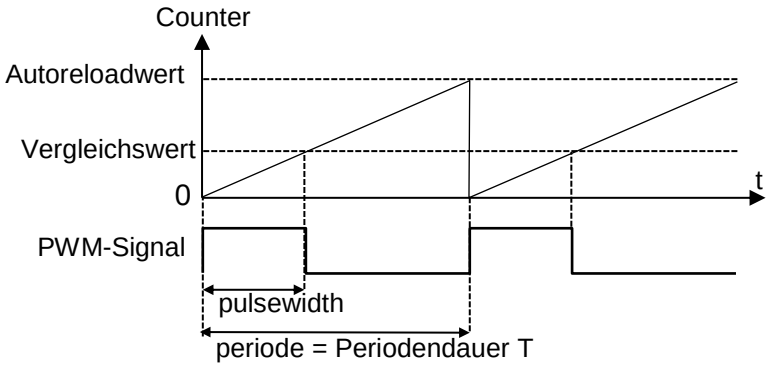
Timer mit Takt versorgen	`RCC->APB1ENR |= 0b10000; //TIM6`
	`RCC->APB1ENR |= 0b100000; //TIM7`

Timerstarten	`TIMx->CR1=1;  //setzt CEN auf 1`
Timer stoppen	`TIMx->CR1=0;  //setzt CEN auf 0`
Autoreloadregister mit *Wert* laden (soweit zählt der Timer bevor er wieder mit 0 beginnt)	`TIMx->ARR=Wert;`
Prescaler einstellen Wert=31 bedeutet Zählperiode 1µs Wert 31999 bedeutet Zählperiode 1ms	`TIMx->PSC=Wert;`
Zähler auf 0 setzen (auch andere Werte sind möglich)	`TIMx->CNT=0;`
Update Interrupt Flag (UIFx) zurücksetzen	`TIMx->SR=0;`
Update Interrupt Flag (UIFx) abfragen	`if (TIMx->SR==1){` `}`
Timerinterrupt Timerinterrupt: Zusätzliche Initialisierung	
ISR in die Vektortabelle eintragen	`NVIC_SetVector(TIMx_IRQn,(uint32_t)&isrTIMx);`
Interrupt freigeben NVIC	`HAL_NVIC_EnableIRQ(TIMx_IRQn);`
Interrupt freigeben Timer UIE=1	`TIMx->DIER=1;`
Globale Freigabe	`__enable_irq();`
Globale Sperre	`__disable_irq();`
Timerinterrupt: Interrupt Service Routine	
Als 1. Instruktion Überlaufflag rücksetzen Timer UIF=0	`TIMx->SR=0;`

Hinweise:
- Bei Timer TIM6 x=6, bei Timer TIM7 x=7

1.3.4 Puls-Weiten-Modulation (PWM)

Prinzip:

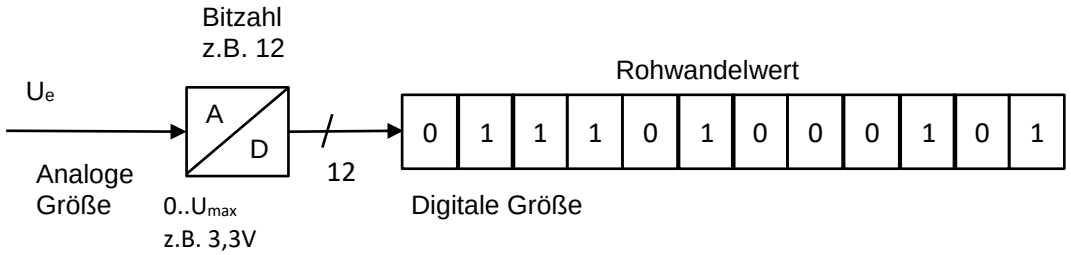


PWM	Befehl	Beispiel
Deklaration	`PwmOut name(Portpin);`	`PwmOut licht(PC_6);`
	• Mögliche Werte für `Portpin`: siehe lokale Formelsammlung	
Konfiguration Periode in Sek.	`name.period(float sekunden);`	`licht.period(5);`
Periode in ms	`name.period_ms(int ms);`	`licht.period_ms(100);`
Periode in µs	`name.period_us(int us);`	`licht.period_us(100);`
Verwendung Tastgrad	`name = Tastgrad;` mit `Tastgrad` von 0 .. 1;	`licht = 0.5;` Tastgrad von 50%

1.3.5 Analog – Digital – Wandlung

AD-Wandler	Befehl	Beispiel
Deklaration	`AnalogIn meinAnalogin(Portpin);`	`AnalogIn poti(PA_0);`
	• Mögliche Werte für `Portpin`: siehe lokale Formelsammlung	
Verwendung `float`	`float x = meinAnalogIn;` `x` nimmt Werte von 0..1 an	`float x;` `x = poti;`

Verwendung `unsigned short`	`unsigned short x;`
	`x = meinAnalogin.read_u16();`
	`x` nimmt Werte von 0..0xFFFF an, unabhängig von der Auflösung des AD-Wandlers
		`unsigned short x;`
		`x = poti.read_u16();`



Berechnungsformeln

Rohwandelwert $= \frac{U_e}{U_{max}} \cdot \left(2^{\text{Bitzahl}} - 1\right)$ z.B. $\frac{U_e}{3{,}3V} \cdot 4095$

Wandelwert als Kommazahl: $x = (Ue/Umax)$ z.B. $x = (Ue/3.3)$

Stufung (analoge Auflösung): Umax/4095 z.B. 3,3V/4095

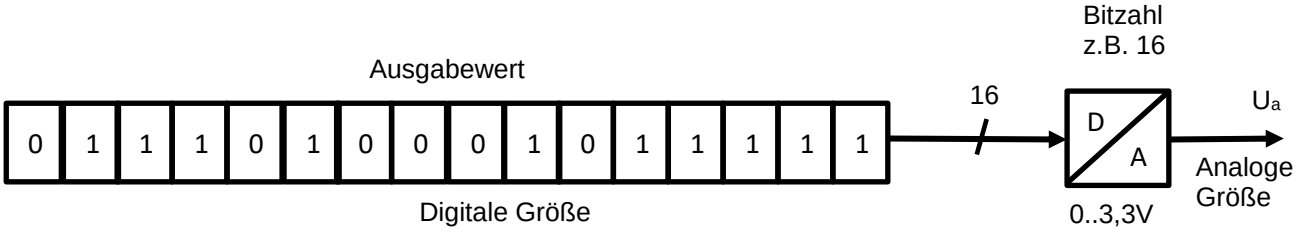
Wandelwert als Ganzzahl linksbündig: `unsigned short x = (Ue/Umax) * 65535`

1.3.6 Digital – Analog – Wandlung

	Befehl	Beispiel
Digital-Analog Wandler		
Deklaration	`AnalogOut meinAnalogOut(Portpin);`	`AnalogOut ausgang(PA_5);`
	• Mögliche Werte für `Portpin`: siehe lokale Formelsammlung	

Verwendung `float`	`float x;`
	`meinAnalogOut =x ;`
	mit x von 0..1
		`float x=0.5;`
		`ausgang = x;`

Verwendung `unsigned short`	`unsigned short x;`
	`meinAnalogOut.write_u16(x);`
	mit x von 0..0xFFFF, unabhängig von der Ausflösung des DA-Wandlers
		`unsigned short x = 0x8000;`
		`ausgang.write_u16(x);`



Berechnungsformeln

`float x:` $U_a = x \cdot 3{,}3V$

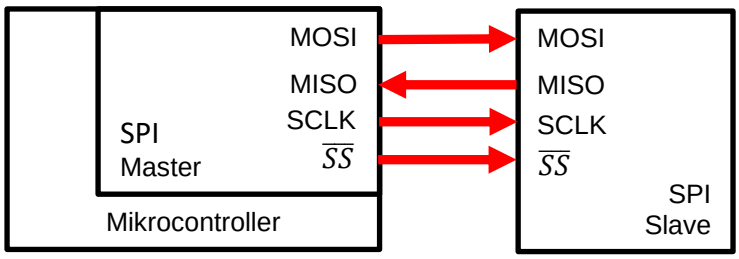
`unsigned short x:` $U_a = \frac{x \cdot 3{,}3V}{65535}$

1.4 Externe Kommunikationsmöglichkeiten

1.4.1 Serial Peripheral Interface (SPI)

Das **Serial Peripheral Interface** (**SPI**) dient der Kommunikation des Mikrocontrollers mit **Modulen** auf der Platine. Module sind

- Anzeigen,
- Speicher,
- LAN-Bausteine
- …



Signale

MOSI (Master Out Slave In): Sendeleitung

MISO (Master In Slave Out): Empfangsleitung

SCLK (Serial Clock): Taktleitung

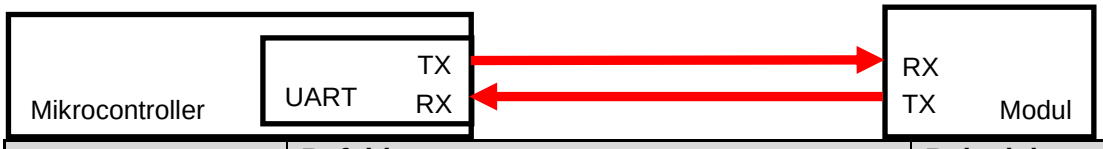
SS (Slave Select): Auswahl des Slaves (Lowaktiv)

	Befehl	**Beispiel**
Serial Peripheral Interface (SPI)		
Deklaration	`SPI name(MOSI,MISO,SCLK);` `DigitalOut SS(Portpin);`	`SPI LANmodul(PB_15,PB_14,PB_13);` `DigitalOut SS(PB_12);`
	• Mögliche Werte für `MOSI`, `MISO`, `SSCL`: siehe lokale Formelsammlung	

Initialisierung	`name.format(Bitzahl,0bXY);`
	`Bitzahl` 8 oder 16
	X: POL (Polarität des Takts)
	0=Highaktiv, Ruhepegel=0,
	1=Lowaktiv, Ruhepegel=1
	Y: Phase, Abtastflanke
	0=Abtastung bei 1. Flanke des Takts
	1=Abtastung bei 2. Flanke des Takts
	`name.frequency(`f`);`
	mit f = Frequenz in Hz
		`LANmodul.format(8,0b00);`
		`LANmodul.frequency(50000);`

Verwendung	`empfang = name.write(sendung);`
	`int sendung; // Zahlenwert der zum`
	`// Slave geschickt wird`
	`int empfang; // Zahlenwert, der vom`
	`// Slave zurück`
	`// geschickt wird`
		`empfang =`
		`LANmodul.write(sendung);`

1.4.2 Universal Asynchronous Receiver Transmitter (UART)



	Befehl	Beispiel
Universal Asynchronous Receiver Transmitter (UART)		
Deklaration	`BufferedSerial name(PinName tx, PinName rx,int baud);`	`BufferedSerial hc05(PB_10,PB_11,9600);`
	• Mögliche Werte für `tx`, `rx`: siehe lokale Formelsammlung	

Verwendung
Daten empfangen
	`int anz = name.read(char* daten, int length);`
	`char* daten`
	//char-Array zur Aufnahme der Daten
	`int length` //Maximale Bytezahl
	return: `int anz`
	Anzahl der empfangenen Datenbytes
		`char daten[16];`
		`int anz;`
		`anz = hc05.read(daten,16);`

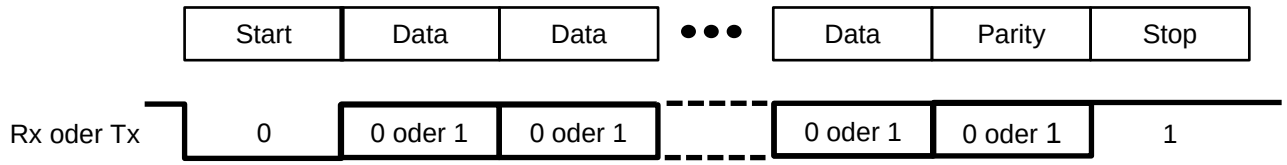
Daten senden	`int anz = name.write(char* daten, int length);`
	`char* daten`
	char-Array mit den Versendedaten
	`int length`
	maximale Bytezahl zum versenden
	`return int anz`
	Anzahl der versendeten Datenbytes
		`char daten[16];`
		`int anz;`
		`anz = hc05.write(daten,16);`

Verfügbar prüfen	`bool verfuegbar = name.readable();`	`bool verfuegbar = hc5.readable();`
Versendbar prüfen	`bool versendbar = name.writeable();`	`bool versendbar = hc5.writeable();`

Hinweis

- Mit `name.write((char*)&x, sizeof(x));` kann eine Variable x beliebigen Typs gesendet, oder mit `read`, auch empfangen werden.

Frame

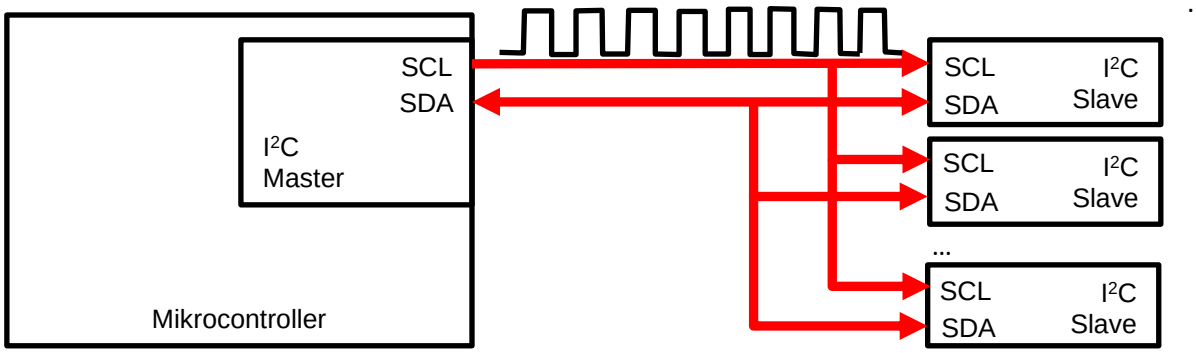


Eine UART-Übertragung beginnt immer mit einem Startbit (Low). Darauf folgen

- 5-8 **Datenbits** (Standard = 8)
- 0 oder 1 **Paritybit** (Standard = 0 none)
- 1 oder 2 **Stopbits** (Standard =1)

Falls ein Paritybit programmiert wurde, kann es gerade Parity (even) oder ungerade Parity (odd) anzeigen.

1.4.3 Inter-Integrated Circuit (I^2C) SCL (Serial Clock): Taktleitung SDA (Serial Data): Datenleitung



I^2C	Befehl	Beispiel
Deklaration	`I²C name(SDApin,SCLpin);`	`I²C lm75(PB_11,PB_10);`
	• Mögliche Werte für `SDApin`, `SCLpin`: siehe lokale Formelsammlung	
Initialisierung	`frequency(int hz)` `hz`: Taktfrequenz in Hertz Bei Bedarf kann die Taktfrequenz eingestellt werden.	`lm75.frequency(10000);`
Verwendung		

Daten empfangen	`int read(int adresse,char* daten,int laenge, bool weiter);`
	`int adresse`
	8-Bitadresse des Kommunikationspartners. Bei vielen Modulen ist eine 7-Bitadresse angegeben. Wenn das so ist, muss der Wert verdoppelt werden.
	8-Bitadresse = 7-Bitadresse * 2
	`char* daten`
	Ein char-Array, das die zu lesenden Daten aufnehmen soll
	`int laenge`
	Die Anzahl der zu lesenden Bytes
	`bool weiter`
	Bei 1 oder true endet die Übertragung nicht mit einer Stop-Bedingung. Die Übertragung wird nicht abgeschlossen.
		`lm75.read(0xA0,daten,8, false);`

Daten senden	`int write(int adresse,char* daten,int laenge, bool weiter);`
	`int adresse`
	8-Bitadresse des Kommunikationspartners. Bei vielen Modulen ist eine 7-Bitadresse angegeben. Wenn das so ist, muss der Wert verdoppelt werden.
	8-bitadresse = 7-Bitadresse * 2
	`char* daten`
	Ein char-Array, das die zu sendenden Daten enthält.
	`int laenge`
	Die Anzahl der zu sendenden Bytes
	`bool weiter`
	Bei 1 oder `true` endet die Übertragung nicht mit einer Stop-Bedingung. Die Übertragung wird nicht abgeschlossen. Das ist bei vielen Speichermodulen wichtig. Diesen muss zuerst mit einer `write`-Anweisung eine Speicheradresse gesendet werden, um dann von dieser Speicheradresse die Daten lesen zu können
		`lm75.write(0xA0, speicheradresse,2,true);`
		Mit
		`char speicheradresse[2] = {0x00,0x02};`
		`// speicheradresse 16Bit`
		`// 0x0002`

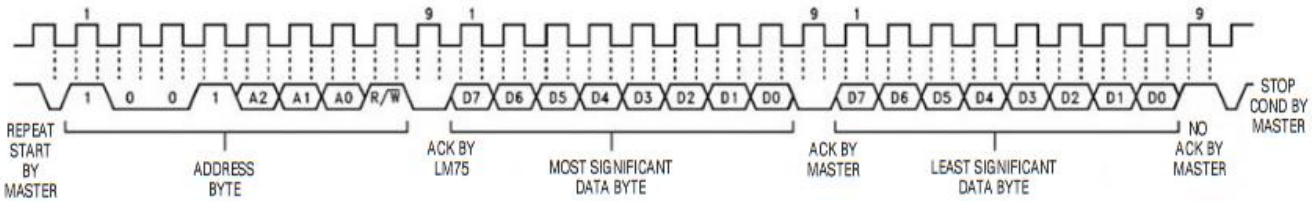
Hinweise

- Mit `lm75.write(0xA0,(char*)&x,sizeof(x));` kann eine Variable `x` **beliebigen** Typs gesendet, oder mit `read`, auch empfangen werden.
- Der Return-Wert ist 0 bei erfolgreicher Datenübertragung und 1 bei fehlerhafter Datenübertragung.

Beispielhaft aufgeführte I²C-Bausteine bzw. Auszug Datenblätter Quelle: www.alldatasheet.com

LM 75:

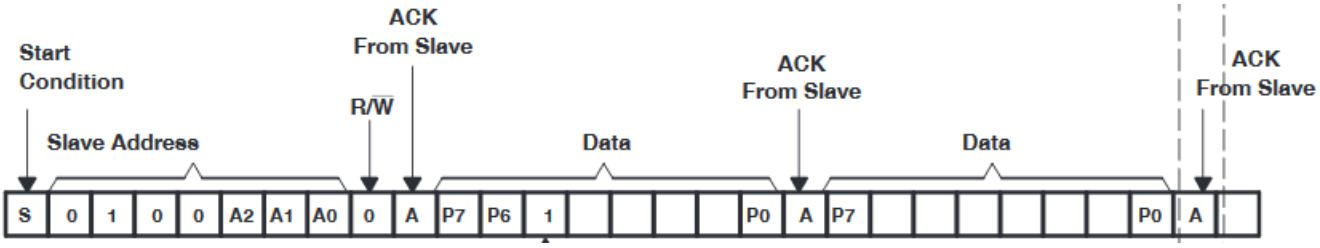
BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
1	0	0	1	A2	A1	A0	RD/$\overline{W}$



UPPER BYTE								LOWER BYTE							
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Sign bit 1= Negative 0 = Positive	MSB 64°C	32°C	16°C	8°C	4°C	2°C	1°C	LSB 0.5°C	X	X	X	X	X	X	X

X = *Don't care.*

PCF 8574



bzw. Aufgabenblatt (Datenblattauszug) als Beilage zum Abitur