

Formelsammlung

Inhalt

1. Ports: Eingabe und Ausgabe mit MBED	3
Ausgabe.....	3
Eingabe.....	4
Bidirektionale Ports	5
2. Externer Interrupt	7
3. Basic Timer TIM6 und TIM7	8
Liste der Timerbefehle	8
4. Timerinterrupt.....	9
4.1 Timerinterrupt: Initialisierung	9
4.2. Timerinterrupt: Interrupt Service Routine.....	9
4.3. Ticker	10
4.4. Timeout	11
5. Ausgabe auf LCD-Display	12
6. Terminal	13
7. Zustandsdiagramm.....	14
7.1. Zustandsdefinition in C/CPP	14
7.2. Zustandsvariable C/CPP (MBED).....	14
7.3. Der Start-Pseudozustand.....	15
7.4. Aktivitäten	15
7.5. Zustandsübergang mit Wächterbedingung	15
7.6. Zustandsübergang mit Ereignis und Wächterbedingung	16
7.7. Internes Ereignis und Selbsttransition	17
7.8. Varianten von Transitionen	18
8. Blockdiagramm.....	19
9. Serial Peripheral Interface (SPI)	20
9.1 Signale	20
9.2 Mögliche Anschlüsse:	20
9.3 Deklaration	20
9.4 Initialisierung.....	20
9.5 Verwendung	20
10. Inter-Integrated Circuit (I2C)	21
10.1 Signale	21
10.2 Mögliche Anschlüsse	21

10.3 Deklaration	21
10.4 Initialisierung.....	21
10.5 Verwendung	22
11. Universal Asynchronous Receiver Transmitter (UART).....	23
11.1 UART Instruktionen	23
12. PWM.....	24
13. Analog – Digital – Wandlung	26

1. Ports: Eingabe und Ausgabe mit MBED

Ausgabe

a. Parallele Ausgabe:

- Ausgabeport initialisieren:

PortOut *meinPortname*(Port,Maske);

Beispiel:

int main()

{ //8-Bit Port mit dem Namen *ampel* an GPIOC

PortOut *ampel*(PortC,0xFF);

ampel=0b00000001; //PC_0 =1, PC_1 .. PC_7 =0

while (true) { //Endlosschleife

ampel=*ampel*<<1; //Lauflicht, die 1 wird um eine Stelle nach
//links geschoben

if (*ampel*==0) *ampel*=1; //Bei 0: von vorne

wait_ms(500); //500ms warten

}}

Mögliche Werte für Port: PortA (für GPIOA), PortB (für GPIOB) PortC (für GPIOC). Mit *Maske* können die Bits ausgewählt werden, die ausgegeben werden sollen. Beispiel: 0xFF bzw. 0b11111111 bedeutet, dass nur die Bits 0 .. 7 ausgegeben werden.

- Ausgabeport verwenden:

Der Port, im Beispiel *ampel*, kann fast wie eine beliebige int-Variable verwendet werden. (nicht ++ oder --)

b. Einzelne Bits ausgeben:

- Ausgabebit initialisieren:

DigitalOut *meinAnschlussname*(Portbezeichnung);

Mögliche Portbezeichnungen sind:

PA_0 .. PA_15, PB_0 .. PB_15, PC_0 .. PC_15

Für den Anschlussnamen sind beliebige Bezeichnungen möglich.

Sonderzeichen, Leerzeichen und Umlaute sollen aber vermieden werden

Beispiel: DigitalOut roteLED(PC_0); //Die rote LED ist an GPIOC Bit 0
angeschlossen.

Bei Bedarf können mit roteLED.mode(*PinMode*); weitere Einstellungen
vorgenommen werden: PullUp, PullDown, PullNone, OpenDrain

- Ausgabebit verwenden: Beliebige Zuweisungen sind möglich.

Beispiele: roteLED=1;
 roteLED=0;
 int x=1;
 roteLED=x;
 roteLED=grueneLED;
 roteLED=!roteLED;
 roteLED=true;
 roteLED=false;

Eingabe

a. Parallele Eingabe

- Eingabeport initialisieren:

```
PortIn meinPortname(Port,Maske);
```

Beispiel:

```
int main()
```

```
{ PortOut ampel(PortC,0xFF);
```

```
    PortIn schalterchen(PortB,0b11111111);
```

```
    schalterchen.mode(PullDown);
```

```
    ampel=1;
```

```
    while (true) {
```

```
        ampel=schalterchen;
```

```
    }
```

```
}
```

Mögliche Werte für Port: PortA (für GPIOA), PortB (für GPIOB) PortC (für GPIOC). Mit *Maske* können die Bits ausgewählt werden, die eingegeben werden sollen. Beispiel: 0xFF bzw. 0b11111111 bedeutet, dass nur die Bits 0 .. 7 eingegeben werden. In diesem Fall machen die Schalterchen eine Verbindung nach „1“. Offenen Schalter sollen „0“ sein. Deshalb:
schalterchen.mode(PullDown);

- Eingabeport verwenden:

Der Port, im Beispiel *schalterchen*, kann wie eine beliebige int-Variable verwendet werden.

c. Einzelne Bits einlesen:

- Eingabebit initialisieren:

```
DigitalIn meinAnschlussname(Portbezeichnung);
```

Mögliche Portbezeichnungen sind:

```
PA_0 .. PA_15, PB_0 .. PB_15, PC_0 .. PC_15
```

Für den Anschlussnamen sind beliebige Bezeichnungen möglich.

Sonderzeichen, Leerzeichen und Umlaute sollen aber vermieden werden

Beispiel: DigitalIn Taste(PA_1); //Die Taste ist an GPIOA Bit 1 angeschlossen.

Bei Bedarf können mit Taste.mode(*PinMode*); weitere Einstellungen vorgenommen werden: PullUp, PullDown, PullNone.

- Eingabebit verwenden: Wie eine Variable.

Beispiel:

```
int main()
```

```
{ PortOut ampel(PortC,0xFF);
```

```
    DigitalOut led(PA_5);
```

```
    DigitalIn Taste(PA_1);
```

```
    Taste.mode(PullDown);
```

```
    while (true) {
```

```
        led=Taste;
```

```
        if (Taste==true) ampel=0b10101010;    //auch möglich: Taste==1
```

```
        else ampel=0b01010101;
```

```
    }
```

```
}
```

Bidirektionale Ports

a. Parallel

```
int main()
{
    //PB als Bidirektionaler Port SE1 Sender Empfänger 1
    PortInOut se1Out(PortB); //Ausgang se1Out
    se1InOut.output();
    se1InOut.mode(OpenDrain);

    //Sendetaste
    DigitalIn eingabeTaste(PA_1);
    eingabeTaste.mode(PullDown);

    //Testausgabe
    PortOut testSE1(PortC);

    while (true) {
        if (eingabeTaste==1) se1InOut=0b10101010; //senden bei Taste
        else se1InOut=0b11111111;
        testSE1=se1In; //empfangen
    }
}
```

Der Port wird mit PortInOut, output() und mode(OpenDrain) gleichzeitig als Ausgang und Eingang konfiguriert.

Unter OpenDrain versteht man einen Ausgang der lediglich nach 0 schalten kann.

b. Einzelne Bit-Ports

Beispiel: PB_6 wird als bidirektionaler Port konfiguriert

```
int main()
```

```
{
```

```
    //PB_6 als Bidirektionaler Port SE1 Sender Empfänger 1
```

```
    DigitalInOut se1Out(PB_6);    //Ausgang: se1Out
```

```
    se1Out.output();              //= Ausgang
```

```
    se1Out.mode(OpenDrain);       //mit OpenDrain
```

```
    //Sendetaste
```

```
    DigitalIn eingabeTaste(PA_1);
```

```
    eingabeTaste.mode(PullDown);
```

```
    //Testausgabe
```

```
    DigitalOut testSE1(PC_0);
```

```
    while (true) {
```

```
        se1Out=!eingabeTaste; //senden
```

```
        testSE1=se1In;         //empfangen
```

```
    }
```

```
}
```

Der Port wird mit DigitalInOut, output() und mode(OpenDrain) gleichzeitig als Ausgang und Eingang konfiguriert.

Unter OpenDrain versteht man einen Ausgang der lediglich nach 0 schalten kann.

2. Externer Interrupt

Anweisung	Bedeutung
<code>InterruptIn name(Port);</code>	Ein Port wird als Interruptquelle mit Namen <i>name</i> festgelegt Beispiel: <code>InterruptIn taste(PA_1);</code>
<code>name.mode(PullDown);</code>	Der Interrupteingang bekommt einen PullDown
<code>name.rise(&isr)</code>	Bei einer steigenden Flanke am Port wird das Unterprogramm mit dem Namen <code>isr</code> (Interrupt Service Routine) automatisch aufgerufen
<code>name.fall(&isr)</code>	Bei einer fallenden Flanke am Port wird das Unterprogramm mit dem Namen <code>isr</code> (Interrupt Service Routine) automatisch aufgerufen
<code>name.enable_irq()</code>	Freigabe des Interrupts. Nach dieser Anweisung reagiert der Mikrocontroller auf die fallende oder steigende Flanke indem er automatisch die ISR aufruft
<code>Name.disable_irq()</code>	Sperre des Interrupts. Der Mikrocontroller reagiert nicht mehr auf Ereignisse am Port.
<code>__enable_irq()</code>	Globale Interruptfreigabe
<code>__disable_irq()</code>	Globale Interruptsperre

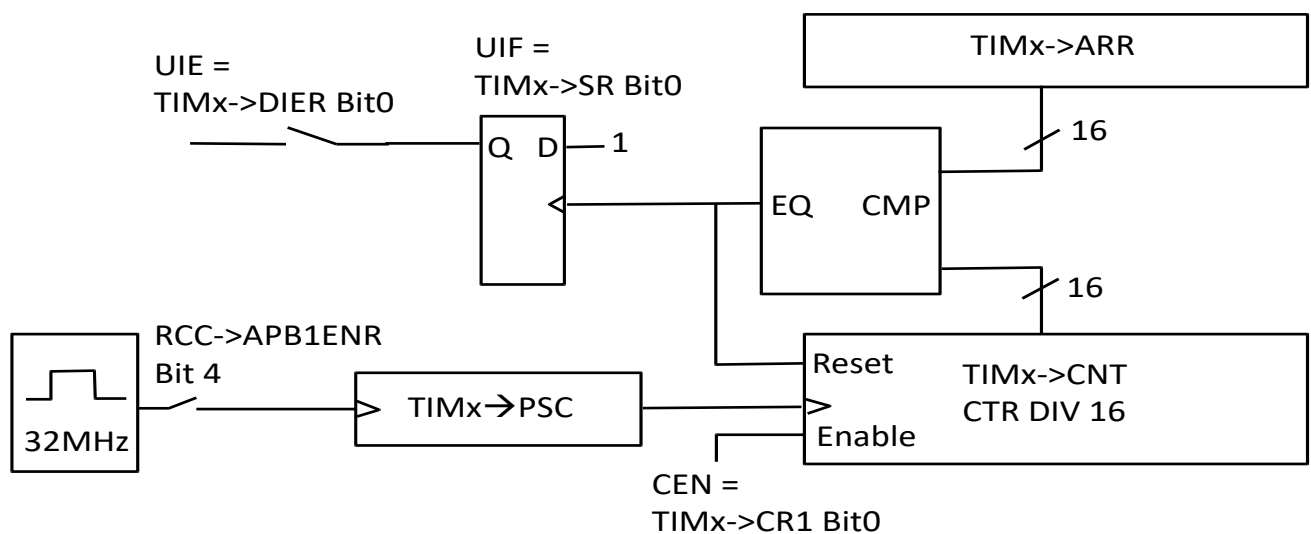
3. Basic Timer TIM6 und TIM7

Liste der Timerbefehle

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;
Timerinterrupt freigeben (UIEx=1)	TIMx->DIER=1;
Update Interrupt Flag (UIFx) abfragen	if (TIMx->SR==1) { }

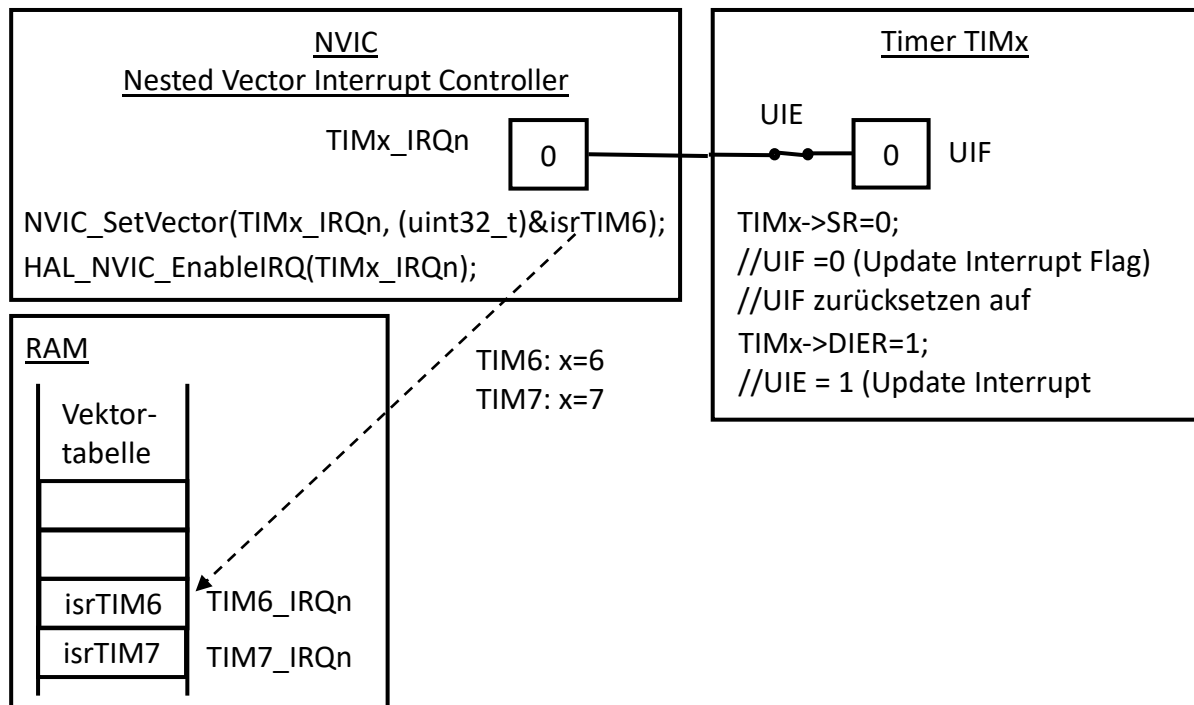
Hinweise:

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



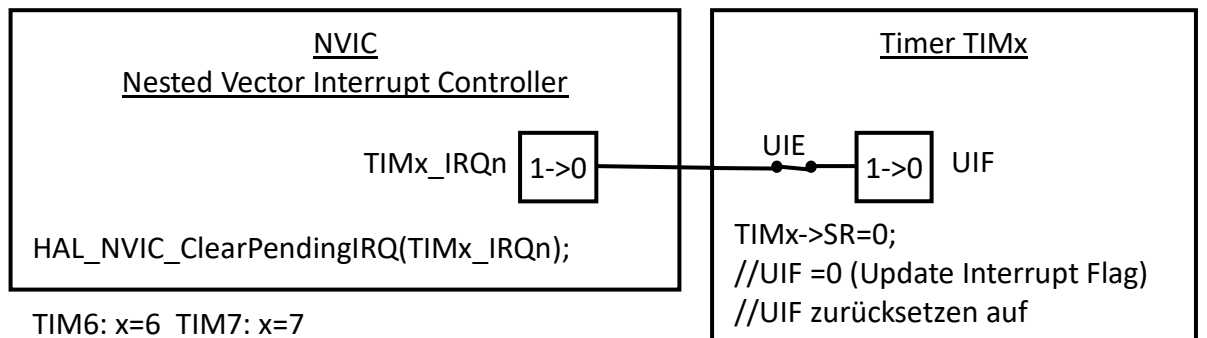
4. Timerinterrupt

4.1 Timerinterrupt: Initialisierung



Befehl	C
ISR in die Vektortabelle eintragen	<code>NVIC_SetVector(TIMx_IRQn, (uint32_t)&isrTIM6);</code>
Interrupt freigeben NVIC	<code>HAL_NVIC_EnableIRQ(TIMx_IRQn);</code>
Timerüberlaufflag UIF=0 zurücksetzen	<code>TIMx->SR=0;</code>
Interrupt freigeben Timer UIE=1	<code>TIMx->DIER=1;</code>

4.2. Timerinterrupt: Interrupt Service Routine



Befehl	C
Pendingbit rücksetzen NVIC	<code>HAL_NVIC_ClearPendingIRQ(TIMx_IRQn);</code>
Überlaufflag rücksetzen Timer UIF=0	<code>TIMx->SR=0;</code>

4.3. Ticker

[Die MBED-Bibliothek enthält eine einfache Implementierung des Timerinterrupts: Den Ticker](#)

```
void attach (&func, float t)
```

Attach a function to be called by the [Ticker](#), specifying the interval in seconds. [More...](#)

```
void attach\_us (&func, int t)
```

Attach a function to be called by the [Ticker](#), specifying the interval in microseconds. [More...](#)

```
void detach ()
```

Detach the function. [More...](#)

Deklaration:

```
Ticker meinTicker;
```

Callback-Operation (ISR):

```
void meineTickerOp()
```

```
{  
}
```

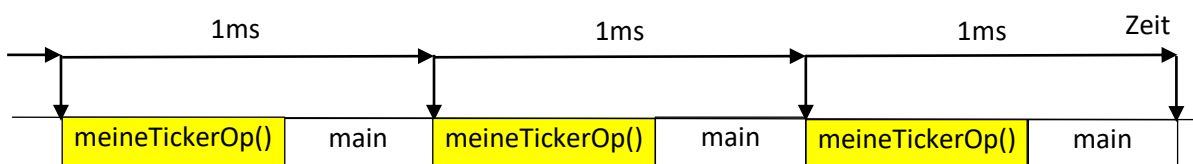
Initialisierung:

```
meinTicker.attach(&meineTickerOp,0.001); //meineTickerOp wird alle 1ms aufgerufen
```

oder

```
meinTicker.attach_us(&meineTickerOp,1000); //meineTickerOp wird alle 1ms aufgerufen
```

Zeitablauf:



Funktion:

Das Hauptprogramm wird zyklisch, hier alle 1ms, unterbrochen und die Callback-Operation (die ISR) wird ausgeführt.

4.4. Timeout

[Die MBED-Bibliothek enthält eine einfache Implementierung des Timerinterrupts: Den Timeout](#)

```
void attach (&func, float t)
```

Attach a function to be called by the [Timeout](#), specifying the interval in seconds.

```
void attach\_us (&func, int t)
```

Attach a function to be called by the [Timeout](#), specifying the interval in microseconds.

```
void detach ()
```

Detach the function.

Deklaration:

```
Timeout meinTimeout;
```

Callback-Operation (ISR):

```
void meineTimeoutOp()
```

```
{
```

```
}
```

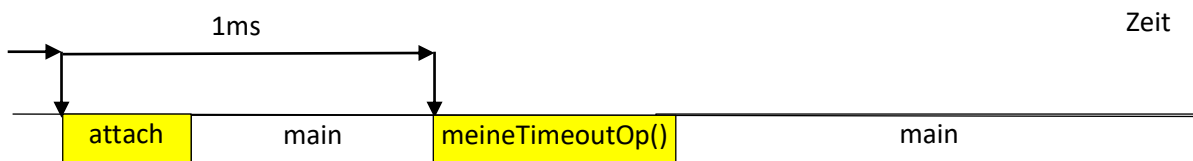
Initialisierung:

```
meinTimeout.attach(&meineTimeoutOp,0.001); //meineTimeoutOp wird nach 1ms aufgerufen
```

oder

```
meinTimeout.attach_us(&meineTimeoutOp,1000); //meineTimeoutOp wird nach 1ms aufgerufen
```

Zeitablauf:



Funktion:

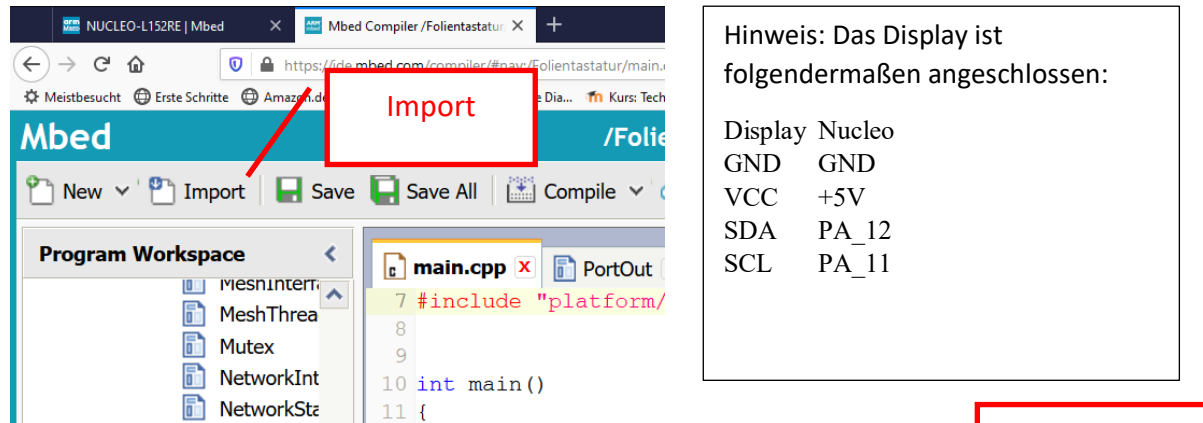
Das Hauptprogramm wird einmalig, hier nach 1ms, unterbrochen und die Callback-Operation (die ISR) wird ausgeführt.

5. Ausgabe auf LCD-Display

Hinweise zur Programmierung:

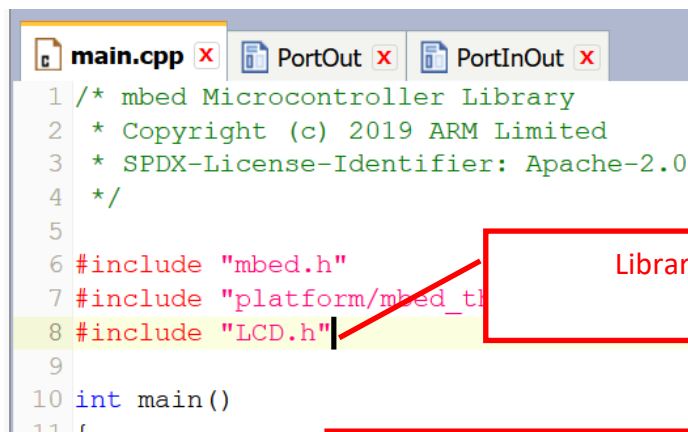
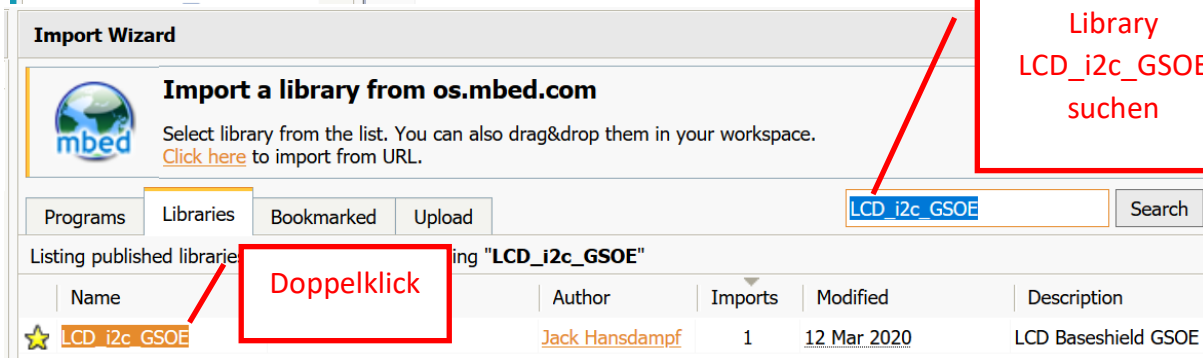
Da die Ausgabe auf das LCD-Display erfolgen soll benötigen Sie folgende Library:

Ergänzen Sie in der main.cpp an den entsprechenden Stellen:



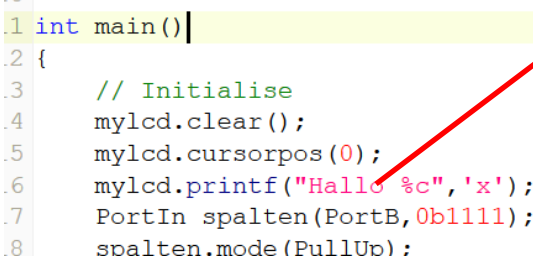
Hinweis: Das Display ist folgendermaßen angeschlossen:

Display Nucleo
GND GND
VCC +5V
SDA PA_12
SCL PA_11



Library mit #include „LCD.h“ einbinden

LCD-Display-Objekt mit Namen z.B. mylcd deklarieren und erzeugen



Befehle:

- mylcd.clear(); //löscht das Display
- mylcd.cursorpos(wert); //plaziert den Cursor 0..15: 1.Zeile, 64..79 2.Zeile (64=0x40)
- mylcd.printf(Formatstring, Werte);
//Ausgabe. Doku: Internet printf

6. Terminal

Bis MBED-OS < 6.0

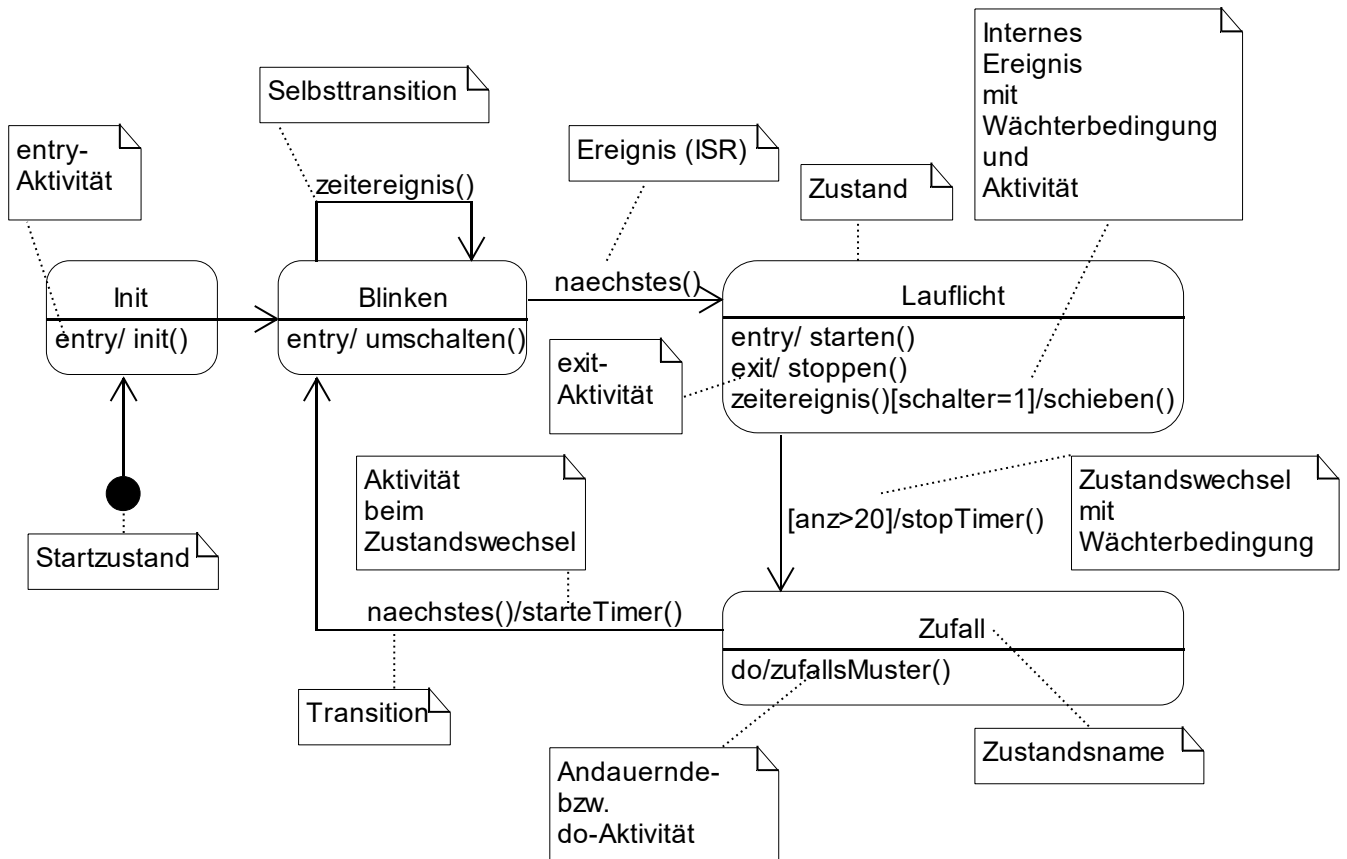
```
RawSerial pc(USBTX, USBRX);
```

```
pc.printf("Anzahl= %d\r\n",anzahl);
```

Ab MBED-OS 6.0:

```
printf("Anzahl= %d\r\n",anzahl);
```

7. Zustandsdiagramm



7.1. Zustandsdefinition in C/CPP

Zuständen sollten, aus Gründen der Übersichtlichkeit, Namen gegeben werden. Dadurch wird der Zusammenhang von Zustandsdiagramm und Programm verdeutlicht.

```
#define Zustandsbezeichnung Zustandsnummer
```

Beispiel:

```
#define A 0
```

```
#define B 1
```

7.2. Zustandsvariable C/CPP (MBED)

Ein Zustand kann durch eine Zustandsvariable gekennzeichnet werden:

```
Int zustand; //eine Zustandsvariable vom Typ int mit Namen zustand
```

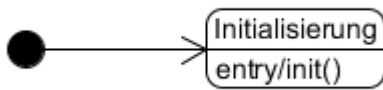
Oder:

```
PortOut zustand(PortC,0xFF);
```

Die Zustandsvariable kann eine Ganzzahlvariable oder ein Ausgangsport des Mikrocontrollers sein. Im zweiten Fall bewirkt ein Zustandswechsel gleichzeitig, dass die Ausgänge entsprechend dem neuen Zustand angepasst werden.

7.3. Der Start-Pseudozustand

So beginnen die meisten Zustandsdiagramme:



Der ausgefüllte Kreis symbolisiert den Startpunkt des Zustandsdiagramms. Oft ist er mit dem Start des Mikrocontrollerprogramms gleich zu setzen. Die Transition vom Startpunkt zum ersten Zustand ist immer unbeschriftet.

7.4. Aktivitäten

Aktivitäten sind Operationen oder Anweisungen die an bestimmten Stellen des Zustandsdiagramms ausgeführt werden



Aktivität	Wo und wann ausgeführt	Beispiel
Entry-Aktivität	Beim Eintritt in einen Zustand	op1()
Do-Aktivität	Andauernd solange der Zustand anhält	op2()
Exit-Aktivität	Beim Verlassen des Zustands	op3()
Aktivität an der Transition	Beim Zustandswechsel	op4()
Aktivität am internen Ereignis	Wenn das interne Ereignis eintritt und gegebenenfalls eine Wächterbedingung erfüllt ist	op5()

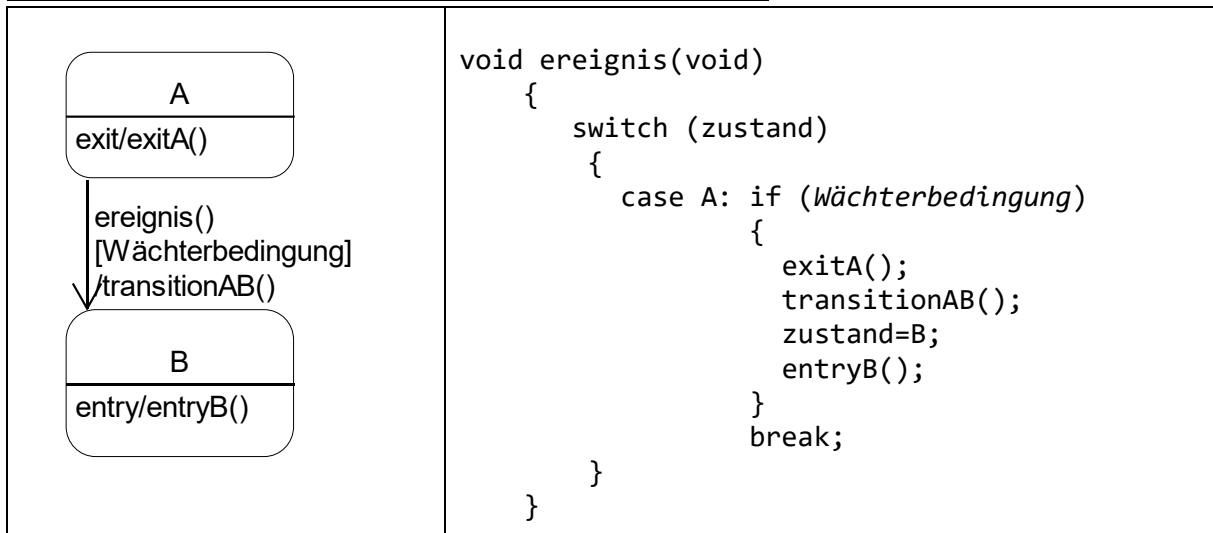
7.5. Zustandsübergang mit Wächterbedingung

<pre> graph TD A[A exit/exitA()] -- "[Wächterbedingung] /transitionAB()" --> B[B entry/entryB()] </pre>	<pre> int main() { while(true) { switch (zustand) { case A: if (Wächterbedingung) { exitA(); transitionAB(); zustand=B; entryB(); } break; } } } </pre>
--	---

In der Endlosschleife wird zuerst der Zustand geprüft. Falls sich der Mikrocontroller im Zustand A befindet, wird die Wächterbedingung an der Transition überprüft. Falls die Wächterbedingung erfüllt ist, erfolgt der Zustandswechsel. Es werden dann in folgender Reihenfolge die Aktivitäten ausgeführt:

1. Exit-Aktivität von Zustand A: exitA()
2. Aktivität an der Transition: transitionAB()
3. Zustandswechsel: zustand=B
4. Entry-Aktivität von Zustand B: entryB()

7.6. Zustandsübergang mit Ereignis und Wächterbedingung




Beim Ereignis handelt es sich um eine Interrupt Service Routine (ISR).

In der ISR wird zuerst der Zustand geprüft. Falls sich der Mikrocontroller im Zustand A befindet, wird die Wächterbedingung an der Transition überprüft. Falls die Wächterbedingung erfüllt ist, erfolgt der Zustandswechsel. Es werden dann in folgender Reihenfolge die Aktivitäten ausgeführt:

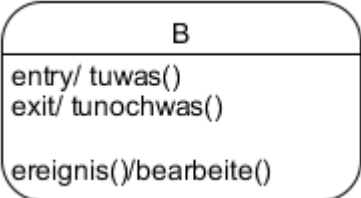
1. Exit-Aktivität von Zustand A: exitA()
2. Aktivität an der Transition: transitionAB()
3. Zustandswechsel: zustand=B
4. Entry-Aktivität von Zustand B: entryB()

7.7. Internes Ereignis und Selbsttransition

Selbsttransition	
 <pre> stateDiagram-v2 state A { entry tuwas() exit tunochwas() } A --> A : ereignis() </pre>	<pre> void ereignis(void) { switch (zustand) { case A: tunochwas(); zustand=A; tuwas(); } break; } } </pre>

Beim Ereignis handelt es sich um eine Interrupt Service Routine (ISR).

In der ISR wird zuerst der Zustand geprüft. Falls sich der Mikrocontroller im Zustand A befindet, wird die Wächterbedingung an der Transition überprüft (falls vorhanden). Falls die Wächterbedingung erfüllt ist, erfolgt der Zustandswechsel wieder nach A. Es werden dann die exit-, Transitions- und entry-Aktivitäten ausgeführt.

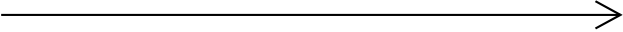
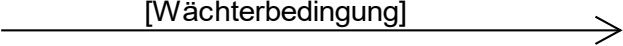
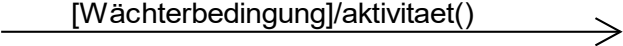
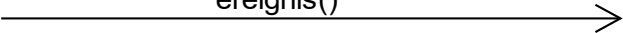
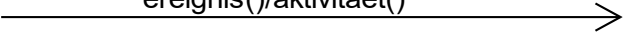
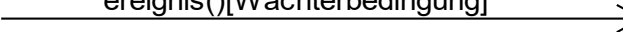
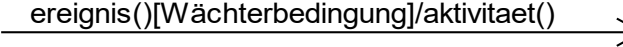
Internes Ereignis	
 <pre> stateDiagram-v2 state B { entry tuwas() exit tunochwas() ereignis()/bearbeite() } </pre>	<pre> void ereignis(void) { switch (zustand) { case A: bearbeite(); break; } } </pre>

Beim Ereignis handelt es sich um eine Interrupt Service Routine (ISR).

In der ISR wird zuerst der Zustand geprüft. Falls sich der Mikrocontroller im Zustand A befindet, wird die Wächterbedingung an der Transition überprüft (falls vorhanden). Falls die Wächterbedingung erfüllt ist, wird der Code, der zu diesem Ereignis in diesem Zustand gehört ausgeführt.

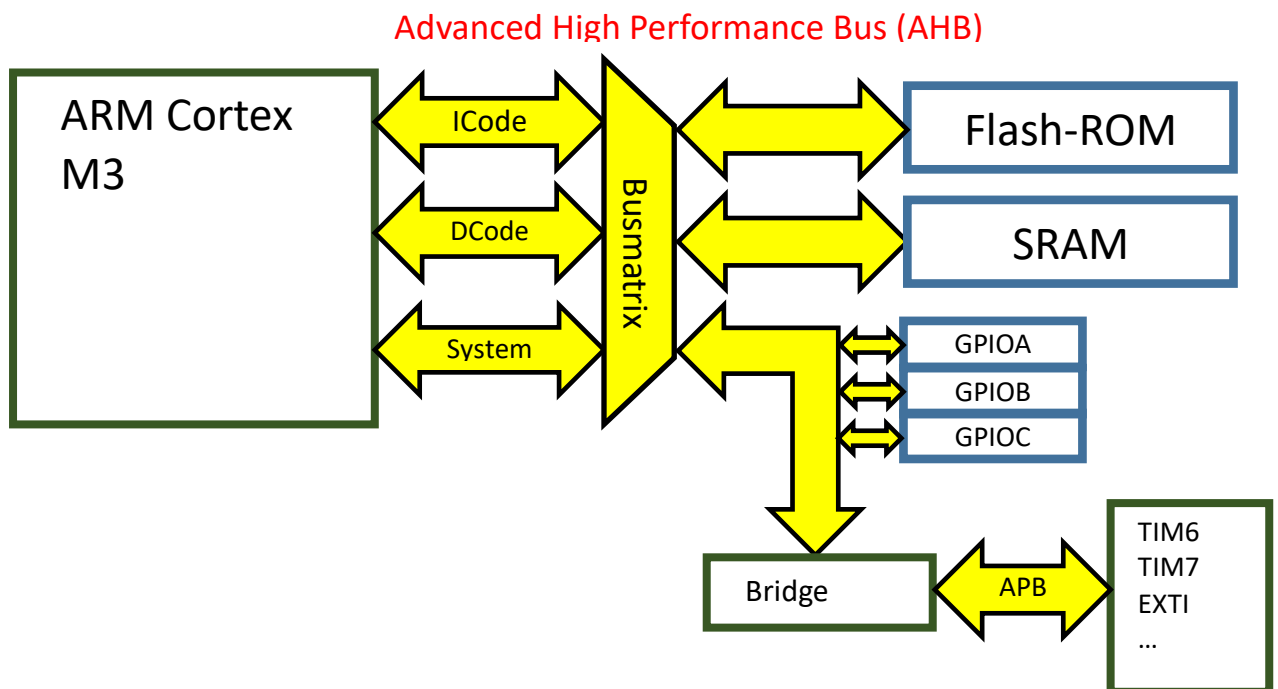
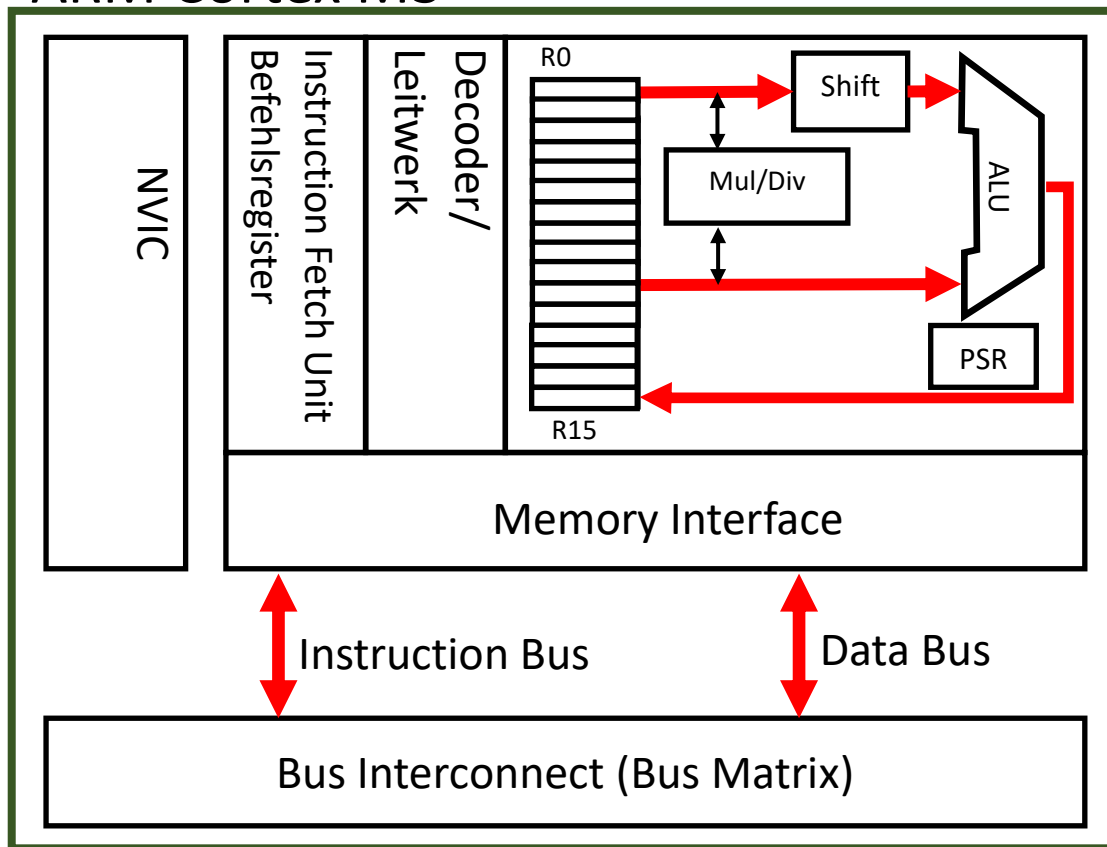
7.8. Varianten von Transitionen

Transitionen bezeichnen Zustandsübergänge und werden als Pfeil, mit offener Spitze vom Ausgangszustand zum Zielzustand, gezeichnet.

	a) Transition ohne Beschriftung bewirkt unmittelbaren Zustandswechsel nachdem die entry-Aktivität beendet wurde.
	b) Transition mit Wächterbedingung. Sobald die Wächterbedingung erfüllt wird, erfolgt der Zustandswechsel
	c) Transition mit Wächterbedingung und Aktivität. Sobald die Wächterbedingung erfüllt ist erfolgt der Zustandswechsel- Beim Zustandswechsel wird die Aktivität ausgeführt
	d) Transition mit Ereignis. Beim Ereignis handelt es sich um eine ISR. Beim Eintreten des Ereignisses erfolgt der Zustandswechsel
	e) Wie d), zusätzlich wird beim Zustandswechsel noch die Aktivität ausgeführt
	f) Wie d) zusätzlich wird die Wächterbedingung geprüft
	g) Wie f) zusätzlich wird die Aktivität ausgeführt

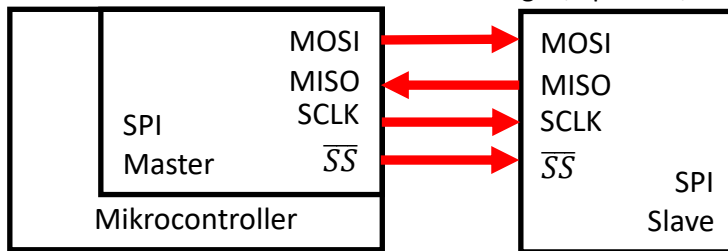
8. Blockdiagramm

ARM Cortex M3



9. Serial Peripheral Interface (SPI)

Das Serial Peripheral Interface (SPI) dient der Kommunikation des Mikrocontrollers mit „Modulen“ auf der Platine. Module können sein: Anzeigen, Speicher, LAN-Baustein usw..



9.1 Signale

MOSI: Master Out Slave In: Sendeleitung

MISO: Master In Slave Out: Empfangsleitung

SCLK: Serial Clock: Taktleitung

SS: Slave Select: Auswahl des Slaves (Lowaktiv)

9.2 Mögliche Anschlüsse:

	MOSI	MISO	SCLK
SPI2	PB_15	PB_14	PB_13
SPI1	PB_5 (D4)	PB_4 (D5)	PB_3 (D3)
SPI1	PA_7 (D11)	PA_6 (D12)	PA_5 (D13)

9.3 Deklaration

```
SPI name(MOSI,MISO,SCLK);
```

```
DigitalOut SS(Port);
```

9.4 Initialisierung

```
name.format(Bitzahl,0bXY);
```

mit Bitzahl = 8 oder 16

mit X: POL = Polarität des Takts 0=Highaktiv, Ruhepegel=0, 1=Lowaktiv, Ruhepegel=1

mit Y: PHA = Phase, Abtastflanke 0=Abtastung bei 1. Flanke des Takts, 1=Abtastung bei 2. Flanke

```
name.frequency(f); mit f = Frequenz in Hz
```

9.5 Verwendung

```
empfang = name.write(sendung);
```

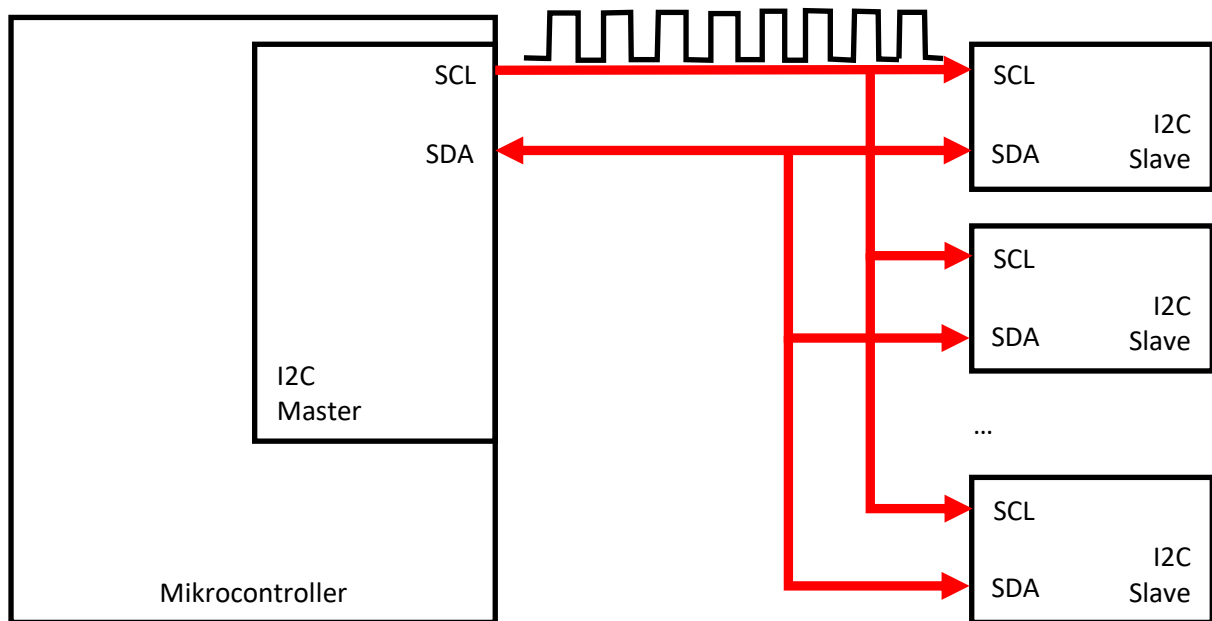
mit:

int sendung: Zahlenwert der zum Slave geschickt wird

int empfang: Zahlenwert, der vom Slave zurückgeschickt wird

10. Inter-Integrated Circuit (I2C)

Die I2C-Schnittstelle dient der Kommunikation des Mikrocontrollers mit „Modulen“ auf der Platine. Module können sein: Anzeigen, Speicher, Realtimeclock (RTC) usw..



10.1 Signale

SCL: Serial Clock: Taktleitung

SDA: Serial Data: Datenleitung bidirektional

10.2 Mögliche Anschlüsse

	SCL	SDA
I2C1	PB_6	PB_7
	PB_8	PB_9
I2C2	PB_10	PB_11

10.3 Deklaration

I2C name(SDAPin,SCLpin)

z.B. I2C i2c(PB_11,PB_10);

10.4 Initialisierung

Bei Bedarf kann die Taktfrequenz eingestellt werden
frequency(int hz), mit hz = Taktfrequenz in Hertz

z.B.

i2c.frequency(10000);

10.5 Verwendung

Daten von einem Modul lesen:

```
int read( int adresse, char* daten, int laenge, bool weiter)
```

Mit:

- int adresse: 8-Bitadresse des Kommunikationspartners. Bei vielen Modulen ist eine 7-Bitadresse angegeben. Wenn das so ist, muss der Wert verdoppelt werden.
 $8\text{-bitadresse} = 7\text{-Bitadresse} * 2$
- char* daten: Ein char-array, das die zu lesenden Daten aufnehmen soll
- int laenge: Die Anzahl der zu lesende Bytes
- bool weiter: Bei 1 oder true endet die Übertragung nicht mit einer Stop-Bedingung. Die Übertragung wird nicht abgeschlossen.

Beispiel: `i2c.read(0xA0,daten,8,false);`

Die gelesenen Daten werden im Array `char daten[8];` gespeichert

Daten zu einem Modul senden:

```
int write( int adresse, char* daten, int laenge, bool weiter)
```

Mit:

- int adresse: 8-Bitadresse des Kommunikationspartners. Bei vielen Modulen ist eine 7-Bitadresse angegeben. Wenn das so ist, muss der Wert verdoppelt werden.
 $8\text{-bitadresse} = 7\text{-Bitadresse} * 2$
- char* daten: Ein char-array, das die zu sendenden Daten enthält.
- int laenge: Die Anzahl der zu sendende 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

Beispiel:

```
i2c.write(0xA0,speicheradresse,2,true);
```

```
i2c.read(0xA0,daten,8,false);
```

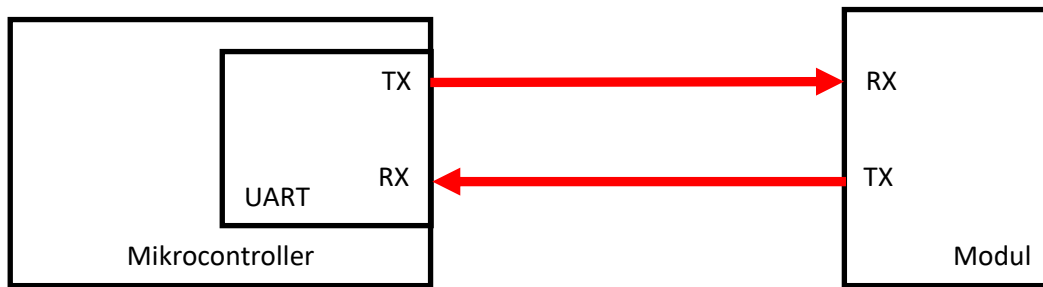
Mit `char speicheradresse[2]={0x00,0x02}; //speicheradresse 16Bit 0x0002`

Mit `i2c.write(0xA0,(char*)&x,sizeof(x));`

Kann eine Variable `x` beliebigen Typs gesendet, oder mit `read`, auch empfangen werden.

Der Returnwert ist 0 bei erfolgreicher Datenübertragung und 1 bei fehlerhafter Datenübertragung.

11. Universal Asynchronous Receiver Transmitter (UART)



11.1 UART Instruktionen

Deklaration:

BufferedSerial *name*(PinName tx, PinName rx, int [baud](#))

z.B. Deklaration einer UART-Schnittstelle mit den Anschlüssen tx=PB_10, rx=PB_11, Übertragungsgeschwindigkeit 9600 Bit/s:

```
BufferedSerial hc05(PB_10,PB_11,9600);
```

Daten empfangen:

Int *anz*=*Name*.read(char* daten,int length)

Mit char* daten: Char-Array zur Aufnahme der Daten

Int length: Maximale Bytezahl

Return int anz: Anzahl der empfangenen Datenbytes

z.B.:

```
char daten[16];
```

```
int anz;
```

```
anz = hc05.read(daten,16);
```

Daten senden:

Int *anz*=*Name*.write(char* daten,int length)

Mit char* daten: Char-Array mit den Versendedenaten

Int length: Maximale Bytezahl zum versenden

Return int anz: Anzahl der versendeten Datenbytes

z.B.:

```
char daten[16];
```

```
int anz;
```

```
anz = hc05.write(daten,16);
```

Prüfen ob Daten verfügbar sind:

```
bool verfuegbar=hc5.readable();
```

Prüfen ob Daten versendbar sind:

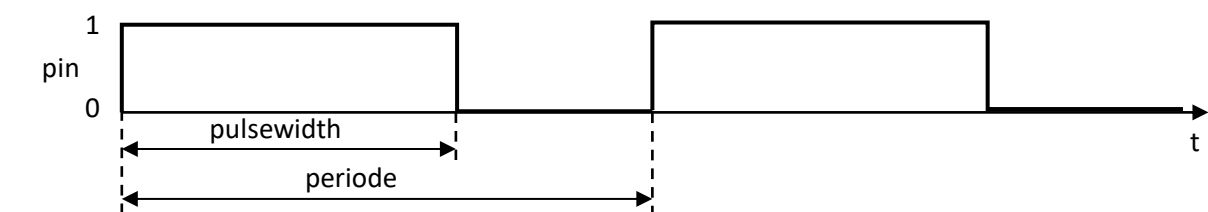
```
bool versendbar=hc5.writeable();
```

Mögliche Anschlüsse:

	RX	TX
UART3	PB_11	PB_10
UART1	PA_10	PA_9
UART1	PB_7	PB_6
UART3/4	PC_11	PC_10

12. PWM

Public Member Functions	
PwmOut (PinName pin) Beispiel: PwmOut licht(PC_6);	
Create a PwmOut connected to the specified pin. More... Mögliche PinNamen: PC_6, PC_7, PC_8, PC_9, PB_0..PB_15, PA_0..PA_3, PA_5..PA_7, PA_15	
void write (float value) Beispiel: licht.write(0.5); oder alternativ licht=0.5	
Set the output duty-cycle, specified as a percentage (float) More...	
float read () Beispiel: float helligkeit=licht.read(); oder float helligkeit=licht;	
Return the current output duty-cycle setting, measured as a percentage (float) More...	
void period (float seconds)	
Set the PWM period, specified in seconds (float), keeping the duty cycle the same. More...	
void period_ms (int ms)	
Set the PWM period, specified in milliseconds (int), keeping the duty cycle the same. More...	
void period_us (int us)	
Set the PWM period, specified in microseconds (int), keeping the duty cycle the same. More...	
void pulsewidth (float seconds)	
Set the PWM pulsewidth, specified in seconds (float), keeping the period the same. More...	
void pulsewidth_ms (int ms)	
Set the PWM pulsewidth, specified in milliseconds (int), keeping the period the same. More...	
void pulsewidth_us (int us)	
Set the PWM pulsewidth, specified in microseconds (int), keeping the period the same. More...	



Die Periodendauer kann in

- Sekunden: licht.period(WertInSekunden);
- Millisekunden: licht.period_ms(WertInMilliSekunden); oder
- Mikrosekunden: licht.period_us(WertInMikroSekunden); angegeben werden

Die Impulsbreite (pulsewidth) kann auf 2 Arten angegeben werden:

1. Absolut

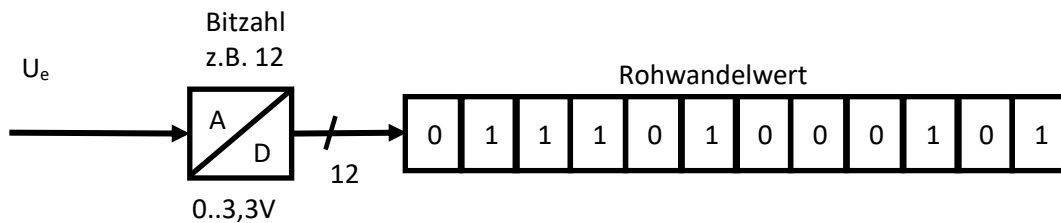
- In Sekunden: `licht.pulsewidth(WertInSekunden);`
- In Millisekunden: `licht.pulsewidth_ms(WertInMilliSekunden);`
- In Mikrosekunden: `licht.pulsewidth_us(WertInMikroSekunden);`

2. Relativ (Tastgrad g)

- `licht=relativerWert;` bzw. `licht.write(relativerWert)`
mit relativer Wert 0.0(=0%) bis 1.0(=100%)

13. Analog – Digital – Wandlung

Die Analogeingänge dienen der Messung von Sensorsignalen.



AnalogIn	<code>AnalogIn meinAnalogin(PinName pin, float);</code> Create an AnalogIn , connected to the specified pin.
float	<code>float x=meinAnalogin.read ();</code> Read the input voltage, represented as a float in the range [0.0, 1.0]. Äquivalent: <code>float x= meinAnalogIn;</code>
unsigned short	<code>unsigned short x=meinAnalogin.read_u16 ();</code> Read the input voltage, represented as an unsigned short in the range [0x0, 0xFFFF].

Berechnungsformeln:

$\text{Rohwandelwert} = (U_e / 3,3V) * 4095$

Wandelwert float $x = U_e / 3,3V$

Wandelwert unsigned short $x = (U_e / 3,3V) * 65535$

Erlaubte Ports (PinName):

PA_0 .. PA_7, PB_0 .. PB_2, PB_12 .. PB_15, PC_0..PC_5

Hinweis:

Bei MBED kann die Verwendung von Analogeingängen in Interrupt-Service-Routinen (ISR) (Callback) Fehler hervorrufen. Lösung: Einlesen des Analogeingangs in der Endlosschleife in eine globale Variable.