Formelsammlung

Inhalt

[1. Ports: Eingabe und Ausgabe mit MBED 3](#_Toc50998633)

[Ausgabe 3](#_Toc50998634)

[Eingabe 4](#_Toc50998635)

[Bidirektionale Ports 5](#_Toc50998636)

[2. Externer Interrupt 7](#_Toc50998637)

[3. Basic Timer TIM6 und TIM7 8](#_Toc50998638)

[Liste der Timerbefehle 8](#_Toc50998639)

[4. Timerinterrupt 9](#_Toc50998640)

[4.1 Timerinterrupt: Initialisierung 9](#_Toc50998641)

[4.2. Timerinterrupt: Interrupt Service Routine 9](#_Toc50998642)

[4.3. Ticker 10](#_Toc50998643)

[4.4. Timeout 11](#_Toc50998644)

[5. Ausgabe auf LCD-Display 12](#_Toc50998645)

[6. Terminal 13](#_Toc50998646)

[7. Zustandsdiagramm 14](#_Toc50998647)

[7.1. Zustandsdefinition in C/CPP 14](#_Toc50998648)

[7.2. Zustandsvariable C/CPP (MBED) 14](#_Toc50998649)

[7.3. Der Start-Pseudozustand 15](#_Toc50998650)

[7.4. Aktivitäten 15](#_Toc50998651)

[7.5. Zustandsübergang mit Wächterbedingung 15](#_Toc50998652)

[7.6. Zustandsübergang mit Ereignis und Wächterbedingung 16](#_Toc50998653)

[7.7. Internes Ereignis und Selbsttransition 17](#_Toc50998654)

[7.8. Varianten von Transitionen 18](#_Toc50998655)

[8. Blockdiagramm 19](#_Toc50998656)

[9. Serial Peripheral Interface (SPI) 20](#_Toc50998657)

[9.1 Signale 20](#_Toc50998658)

[9.2 Mögliche Anschlüsse: 20](#_Toc50998659)

[9.3 Deklaration 20](#_Toc50998660)

[9.4 Initialisierung 20](#_Toc50998661)

[9.5 Verwendung 20](#_Toc50998662)

[10. Inter-Integrated Circuit (I2C) 21](#_Toc50998663)

[10.1 Signale 21](#_Toc50998664)

[10.2 Mögliche Anschlüsse 21](#_Toc50998665)

[10.3 Deklaration 21](#_Toc50998666)

[10.4 Initialisierung 21](#_Toc50998667)

[10.5 Verwendung 22](#_Toc50998668)

[11. Universal Asynchronous Receiver Transmitter (UART) 23](#_Toc50998669)

[11.1 UART Instruktionen 23](#_Toc50998670)

[12. PWM 24](#_Toc50998671)

[13. Analog – Digital – Wandlung 26](#_Toc50998672)

# 1. Ports: Eingabe und Ausgabe mit MBED

## Ausgabe

* 1. 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 --)

* 1. 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 Einstellugen 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

1. 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.
  1. 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 Einstellugen 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

1. 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.

1. 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(); //= Augang**

**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 |
| InterruptIn name(Port); | Ein Port wird als Interruptquelle mit Namen *name* festgelegt Beispiel: InterruptIn taste(PA\_1); |
| name.mode(PullDown); | Der Interrupteingang bekommt einen PullDown |
| name.rise(&isr) | Bei einer **steigenden** Flanke am Port wird das Unterprogramm mit dem Namen isr (Interrupt Service Routine) automatisch aufgerufen |
| name.fall(&isr) | Bei einer **fallenden** Flanke am Port wird das Unterprogramm mit dem Namen isr (Interrupt Service Routine) automatisch aufgerufen |
| name.enable\_irq() | Freigabe des Interrupts. Nach dieser Anweisung reagiert der Mikrocontroller auf die fallende oder steigende Flanke indem er automatisch die ISR aufruft |
| Name.disable\_irq() | Sperre des Interrupts. Der Mikrocontroller reagiert nicht mehr auf Ereignisse am Port. |
| \_\_enable\_irq() | Globale Interruptfreigabe |
| \_\_disable\_irq() | 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

Timer TIMx

NVIC

Nested Vector Interrupt Controller

UIE

0

UIF

TIMx->SR=0;

//UIF =0 (Update Interrupt Flag)

//UIF zurücksetzen auf Anfangswert

TIMx->DIER=1;

//UIE = 1 (Update Interrupt Enable)

NVIC\_SetVector(TIMx\_IRQn, (uint32\_t)&isrTIM6);

RAM

TIMx\_IRQn

0

HAL\_NVIC\_EnableIRQ(TIMx\_IRQn);

TIM6: x=6

TIM7: x=7

Vektor-

tabelle

TIM6\_IRQn

TIM7\_IRQn

isrTIM6

isrTIM7

|  |  |
| --- | --- |
| Befehl | C |
| ISR in die Vektortabelle eintragen | NVIC\_SetVector(TIMx\_IRQn, (uint32\_t)&isrTIM6); |
| Interrupt freigeben NVIC | HAL\_NVIC\_EnableIRQ(TIMx\_IRQn); |
| Timerüberlaufflag UIF=0 zurücksetzen | TIMx->SR=0; |
| Interrupt freigeben Timer UIE=1 | TIMx->DIER=1; |

### 4.2. Timerinterrupt: Interrupt Service Routine

Timer TIMx

NVIC

Nested Vector Interrupt Controller

UIE

1->0

UIF

TIMx->SR=0;

//UIF =0 (Update Interrupt Flag)

//UIF zurücksetzen auf Anfangswert

HAL\_NVIC\_ClearPendingIRQ(TIMx\_IRQn);

TIMx\_IRQn

1->0

TIM6: x=6 TIM7: x=7

|  |  |
| --- | --- |
| Befehl | C |
| Pendingbit rücksetzen NVIC | HAL\_NVIC\_ClearPendingIRQ(TIMx\_IRQn); |
| Überlaufflag rücksetzen Timer UIF=0 | TIMx->SR=0; |

### 4.3. Ticker

[Die MBED-Bibliothek enthält eine einfache Implementierung des Timerinterrupts: Den Ticker](https://os.mbed.com/docs/mbed-os/v6.2/apis/ticker.html)

|  |  |
| --- | --- |
| void | [attach](https://os.mbed.com/docs/mbed-os/v6.2/mbed-os-api-doxy/classmbed_1_1_ticker_base.html#aecc275cbe328728bbee4d153e5b6b5b5) (&func, float t) |
|  | Attach a function to be called by the [Ticker](https://os.mbed.com/docs/mbed-os/v6.2/mbed-os-api-doxy/classmbed_1_1_ticker.html), specifying the interval in seconds. [More...](https://os.mbed.com/docs/mbed-os/v6.2/mbed-os-api-doxy/classmbed_1_1_ticker.html#aecc275cbe328728bbee4d153e5b6b5b5) |
| void | [attach\_us](https://os.mbed.com/docs/mbed-os/v6.2/mbed-os-api-doxy/classmbed_1_1_ticker_base.html#acf096ae7c6360241732a629df59b1ebd) (&func, int t) |
|  | Attach a function to be called by the [Ticker](https://os.mbed.com/docs/mbed-os/v6.2/mbed-os-api-doxy/classmbed_1_1_ticker.html), specifying the interval in microseconds. [More...](https://os.mbed.com/docs/mbed-os/v6.2/mbed-os-api-doxy/classmbed_1_1_ticker.html#acf096ae7c6360241732a629df59b1ebd) |
| void | [detach](https://os.mbed.com/docs/mbed-os/v6.2/mbed-os-api-doxy/classmbed_1_1_ticker_base.html#ac295bade8aee589f6718dfa79edc2a34) () |
|  | Detach the function. [More...](https://os.mbed.com/docs/mbed-os/v6.2/mbed-os-api-doxy/classmbed_1_1_ticker.html#ac295bade8aee589f6718dfa79edc2a34) |

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:

main

Zeit

1ms

1ms

1ms

meineTickerOp()

meineTickerOp()

meineTickerOp()

main

main

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](https://os.mbed.com/docs/mbed-os/v6.2/apis/timeout.html) des Timerinterrupts: Den Timeout

|  |  |
| --- | --- |
| void | [attach](https://os.mbed.com/docs/mbed-os/v6.2/mbed-os-api-doxy/classmbed_1_1_ticker_base.html#aecc275cbe328728bbee4d153e5b6b5b5) (&func, float t) |
|  | Attach a function to be called by the [Timeout](https://os.mbed.com/docs/mbed-os/v6.2/apis/timeout.html), specifying the interval in seconds. |
| void | [attach\_us](https://os.mbed.com/docs/mbed-os/v6.2/mbed-os-api-doxy/classmbed_1_1_ticker_base.html#acf096ae7c6360241732a629df59b1ebd) (&func, int t) |
|  | Attach a function to be called by the [Timeout](https://os.mbed.com/docs/mbed-os/v6.2/apis/timeout.html), specifying the interval in microseconds. |
| void | [detach](https://os.mbed.com/docs/mbed-os/v6.2/mbed-os-api-doxy/classmbed_1_1_ticker_base.html#ac295bade8aee589f6718dfa79edc2a34) () |
|  | 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:

main

Zeit

1ms

attach

main

meineTimeoutOp()

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



Doppelklick

Library LCD\_i2c\_GSOE suchen

Import



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(*Formatstrinng,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*

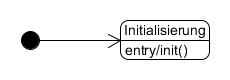
Oder:

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.

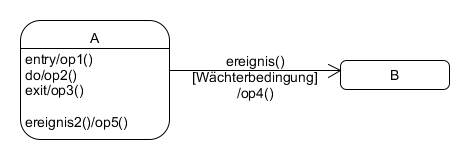
### 7.3. Der Start-Pseudozustand

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

|  |  |
| --- | --- |
|  | int main() {  while(true)  {  switch (zustand)  {  case A: if (*Wächterbedingung*)  {  exitA();  transitionAB();  zustand=B;  entryB();  }  break;  }  }  } |

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

|  |  |
| --- | --- |
|  | void ereignis(void)  {  switch (zustand)  {  case A: if (*Wächterbedingung*)  {  exitA();  transitionAB();  zustand=B;  entryB();  }  break;  }  } |

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 | |
|  | void ereignis(void)  {  switch (zustand)  {  case A: tunochwas();  zustand=A;  tuwas();  }  break;  }  } |

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 | |
|  | void ereignis(void)  { switch (zustand)  { case A: bearbeite();  break;  }  } |

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.

1. Transition ohne Beschriftung bewirkt unmittelbaren Zustandswechsel nachdem die entry-Aktivität beendet wurde.
2. Transition mit Wächterbedingung. Sobald die Wächterbedingung erfüllt wird, erfolgt der Zustandswechsel
3. Transition mit Wächterbedingung und Aktivität. Sobald die Wächterbedingung erfüllt ist erfolgt der Zustandswechsel- Beim Zustandswechsel wird die Aktivität ausgeführt
4. Transition mit Ereignis. Beim Ereignis handelt es sich um eine ISR. Beim Eintreten des Ereignisses erfolgt der Zustandswechsel
5. Wie d), zusätzlich wird beim Zustandswechsel noch die Aktivität ausgeführt
6. Wie d) zusätzlich wird die Wächterbedingung geprüft
7. Wie f) zusätzlich wird die Aktivität ausgeführt

# 8. Blockdiagramm

Mul/Div

Shift

ALU

PSR

Memory Interface

NVIC

Bus Interconnect (Bus Matrix)

Instruction Bus

Data Bus

ARM Cortex M3

R0

R15

Instruction Fetch Unit

Befehlsregister

Decoder/  
Leitwerk

ARM Cortex M3

ICode

DCode

System

Flash-ROM

SRAM

Advanced High Performance Bus (AHB)

GPIOA

GPIOB

GPIOC

Bridge

TIM6

TIM7

EXTI

…

APB

Busmatrix

# 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..

Mikrocontroller

SPI

Master

SPI

Slave

MOSI

MISO

SCLK

MOSI

MISO

SCLK

### 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..

Mikrocontroller

I2C

Master

I2C

Slave

SCL

SDA

SCL

SDA

I2C

Slave

SDA

SCL

I2C

Slave

SDA

SCL

…

### 10.1 Signale

SCL: Serial Clock: Taktleitung

SDA: Serial Data: Datenleitung bidirektional

### 10.2 Mögliche Anschlüsse

|  |  |  |
| --- | --- | --- |
|  | **SCL** | **SDA** |
| I2C1 | PB\_6  PB\_8 | PB\_7  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-bitadresse = 7-Bitadresse \* 2
* char\* daten: Ein char-array, das die zu lesenden Daten aufnehmen soll
* int laenge: Die Anzahl der zu lesenende 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-bitadresse = 7-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)

Mikrocontroller

UART

Modul

TX

RX

RX

TX

# 11.1 UART Instruktionen

**Deklaration:**

BufferedSerial *name*(PinName tx, PinName rx, int [baud](https://os.mbed.com/docs/mbed-os/v6.2/mbed-os-api-doxy/classmbed_1_1_serial_base.html#a9afb7aa9321cd71a8a26a673157583d2))

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:**

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 |

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

Mit char\* daten: Char-Array mit den Versendedaten

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();

# 12. PWM

|  |  |
| --- | --- |
| Public Member Functions | |
|  | [PwmOut](https://os.mbed.com/docs/mbed-os/v6.2/mbed-os-api-doxy/classmbed_1_1_pwm_out.html#ae90809b159b1af641a257b3505c56c41) (PinName pin) Beispiel: PwmOut licht(PC\_6);  Create a [PwmOut](https://os.mbed.com/docs/mbed-os/v6.2/mbed-os-api-doxy/classmbed_1_1_pwm_out.html) connected to the specified pin. [More...](https://os.mbed.com/docs/mbed-os/v6.2/mbed-os-api-doxy/classmbed_1_1_pwm_out.html#ae90809b159b1af641a257b3505c56c41) 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](https://os.mbed.com/docs/mbed-os/v6.2/mbed-os-api-doxy/classmbed_1_1_pwm_out.html#a04593bbcefdddb53406c0943a711f293) (float value) Beispiel: licht.write(0.5); oder alternativ licht=0.5 |
| Set the output duty-cycle, specified as a percentage (float) [More...](https://os.mbed.com/docs/mbed-os/v6.2/mbed-os-api-doxy/classmbed_1_1_pwm_out.html#a04593bbcefdddb53406c0943a711f293) |
| float | [read](https://os.mbed.com/docs/mbed-os/v6.2/mbed-os-api-doxy/classmbed_1_1_pwm_out.html#a868d9d1513fbaf34c4d5d15a7fa4baa8) () Beispiel: float helligkeit=licht.read(); oder float helligkeit=licht; |
|  | Return the current output duty-cycle setting, measured as a percentage (float) [More...](https://os.mbed.com/docs/mbed-os/v6.2/mbed-os-api-doxy/classmbed_1_1_pwm_out.html#a868d9d1513fbaf34c4d5d15a7fa4baa8) |
| void | [period](https://os.mbed.com/docs/mbed-os/v6.2/mbed-os-api-doxy/classmbed_1_1_pwm_out.html#a7307df34e279d3346e2f56157c58f705) (float seconds) |
|  | Set the PWM period, specified in seconds (float), keeping the duty cycle the same. [More...](https://os.mbed.com/docs/mbed-os/v6.2/mbed-os-api-doxy/classmbed_1_1_pwm_out.html#a7307df34e279d3346e2f56157c58f705) |
| void | [period\_ms](https://os.mbed.com/docs/mbed-os/v6.2/mbed-os-api-doxy/classmbed_1_1_pwm_out.html#a255e860d28f3c096398a9920355599a8) (int ms) |
|  | Set the PWM period, specified in milliseconds (int), keeping the duty cycle the same. [More...](https://os.mbed.com/docs/mbed-os/v6.2/mbed-os-api-doxy/classmbed_1_1_pwm_out.html#a255e860d28f3c096398a9920355599a8) |
| void | [period\_us](https://os.mbed.com/docs/mbed-os/v6.2/mbed-os-api-doxy/classmbed_1_1_pwm_out.html#a2561e8891d8585134d870680c133dbb0) (int us) |
|  | Set the PWM period, specified in microseconds (int), keeping the duty cycle the same. [More...](https://os.mbed.com/docs/mbed-os/v6.2/mbed-os-api-doxy/classmbed_1_1_pwm_out.html#a2561e8891d8585134d870680c133dbb0) |
| void | [pulsewidth](https://os.mbed.com/docs/mbed-os/v6.2/mbed-os-api-doxy/classmbed_1_1_pwm_out.html#add3053f7f15f458c214b3f9c70a02f16) (float seconds) |
|  | Set the PWM pulsewidth, specified in seconds (float), keeping the period the same. [More...](https://os.mbed.com/docs/mbed-os/v6.2/mbed-os-api-doxy/classmbed_1_1_pwm_out.html#add3053f7f15f458c214b3f9c70a02f16) |
| void | [pulsewidth\_ms](https://os.mbed.com/docs/mbed-os/v6.2/mbed-os-api-doxy/classmbed_1_1_pwm_out.html#a7bc30e2d1f1c1ea873af9f57e19137a2) (int ms) |
|  | Set the PWM pulsewidth, specified in milliseconds (int), keeping the period the same. [More...](https://os.mbed.com/docs/mbed-os/v6.2/mbed-os-api-doxy/classmbed_1_1_pwm_out.html#a7bc30e2d1f1c1ea873af9f57e19137a2) |
| void | [pulsewidth\_us](https://os.mbed.com/docs/mbed-os/v6.2/mbed-os-api-doxy/classmbed_1_1_pwm_out.html#ad0919da77d53eed74e75496eb9cb1137) (int us) |
|  | Set the PWM pulsewidth, specified in microseconds (int), keeping the period the same. [More...](https://os.mbed.com/docs/mbed-os/v6.2/mbed-os-api-doxy/classmbed_1_1_pwm_out.html#ad0919da77d53eed74e75496eb9cb1137) |

pulsewidth

periode

t

0

1

pin

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.

A

D

Bitzahl z.B. 12

12

0

1

0

0

0

1

0

1

0

1

1

1

0..3,3V

Rohwandelwert

Ue

|  |  |
| --- | --- |
| AnalogIn | AnalogIn *meinAnalogin*(PinName pin, float); |
|  | Create an [AnalogIn](https://os.mbed.com/docs/mbed-os/v6.2/mbed-os-api-doxy/classmbed_1_1_analog_in.html), connected to the specified pin. |
| float | *float x=meinAnalogin*.[read](https://os.mbed.com/docs/mbed-os/v6.2/mbed-os-api-doxy/classmbed_1_1_analog_in.html#a868d9d1513fbaf34c4d5d15a7fa4baa8) (); |
|  | Read the input voltage, represented as a float in the range [0.0, 1.0].  Äquivalent: *float x= meinAnalogIn;* |
| unsigned short | *unsigned short x=meinAnalogin*.[read\_u16](https://os.mbed.com/docs/mbed-os/v6.2/mbed-os-api-doxy/classmbed_1_1_analog_in.html#a475e3e9fa76c6ea79ad235829926d0a3) (); |
|  | Read the input voltage, represented as an  unsigned short in the range [0x0, 0xFFFF]. |

Berechnungsformeln:

Rohwandelwert= (Ue / 3,3V) \* 4095

Wandelwert float x= Ue/3,3V

Wandelwert unsigned short x= (Ue / 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.