Formelsammlung

1. Ports: Eingabe und Ausgabe mit MBED

1. 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;

1. 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;

}

}

1. Bidirektionale Ports
   1. Parallel

int main()

{

**//PB als Bidirektionaler Port SE1 Sender Empfänger 1**

**PortIn se1In(PortB); //Eingang se1In**

**se1In.mode(PullUp);**

**PortInOut se1Out(PortB); //Ausgang se1Out**

**se1Out.output();**

**se1Out.mode(OpenDrain);**

//Sendetaste

DigitalIn eingabeTaste(PA\_1);

eingabeTaste.mode(PullDown);

//Testausgabe

PortOut testSE1(PortC);

while (true) {

if (eingabeTaste==1) **se1Out=0b10101010; //senden bei Taste**

else se1Out=0b11111111;

**testSE1=se1In; //empfangen**

}

}  
Der Port wird mit PortIn und mode(PullUp) als Eingang und gleichzeitig mit PortInOut, output() und mode(OpenDrain) als Ausgang 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**

**DigitalIn se1In(PB\_6); //Eingang: se1In**

**se1In.mode(PullUp); //mit PullUp**

**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 DigitalIn und mode(PullUp) als Eingang und gleichzeitig mit DigitalInOut, output() und mode(OpenDrain) als Ausgang 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

# 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; |

# 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; |

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:



Doppelklick

Library LCD\_i2c\_GSOE suchen

Import



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

Library mit #include „LCD.h“  
einbinden



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

RawSerial pc(USBTX, USBRX);

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