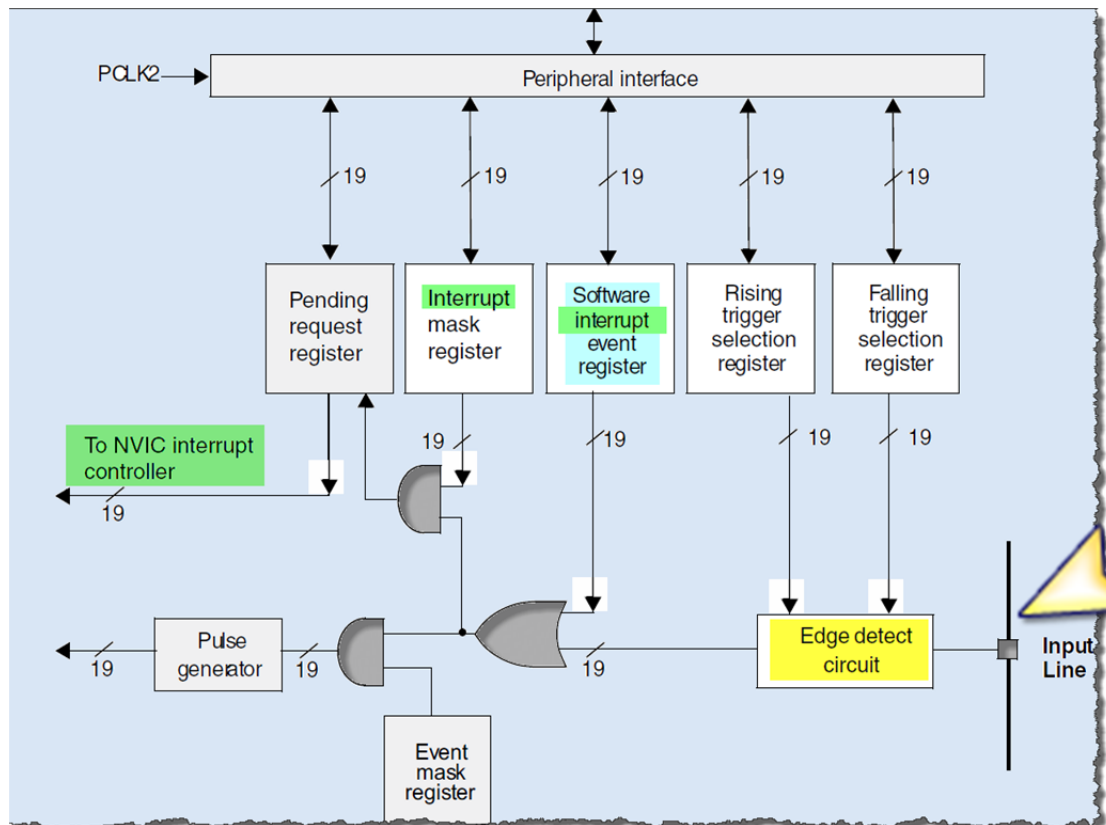


# Mikrocontroller MCB32

## Interrupts



## MCB32 - Embedded Programmierung

### Lib Befehle

Version: 2203.01

Bitte beachten. Diese Unterlagen können ohne Vorankündigung jederzeit angepasst, verbessert und erweitert werden. Wir bitten Sie Wünsche und auch Fehler zu melden. ([info@mcb32.ch](mailto:info@mcb32.ch))

Version C: Print mit **transparenter** Bodenplatte. Muss mit der LIB für Ver. C betrieben werden.

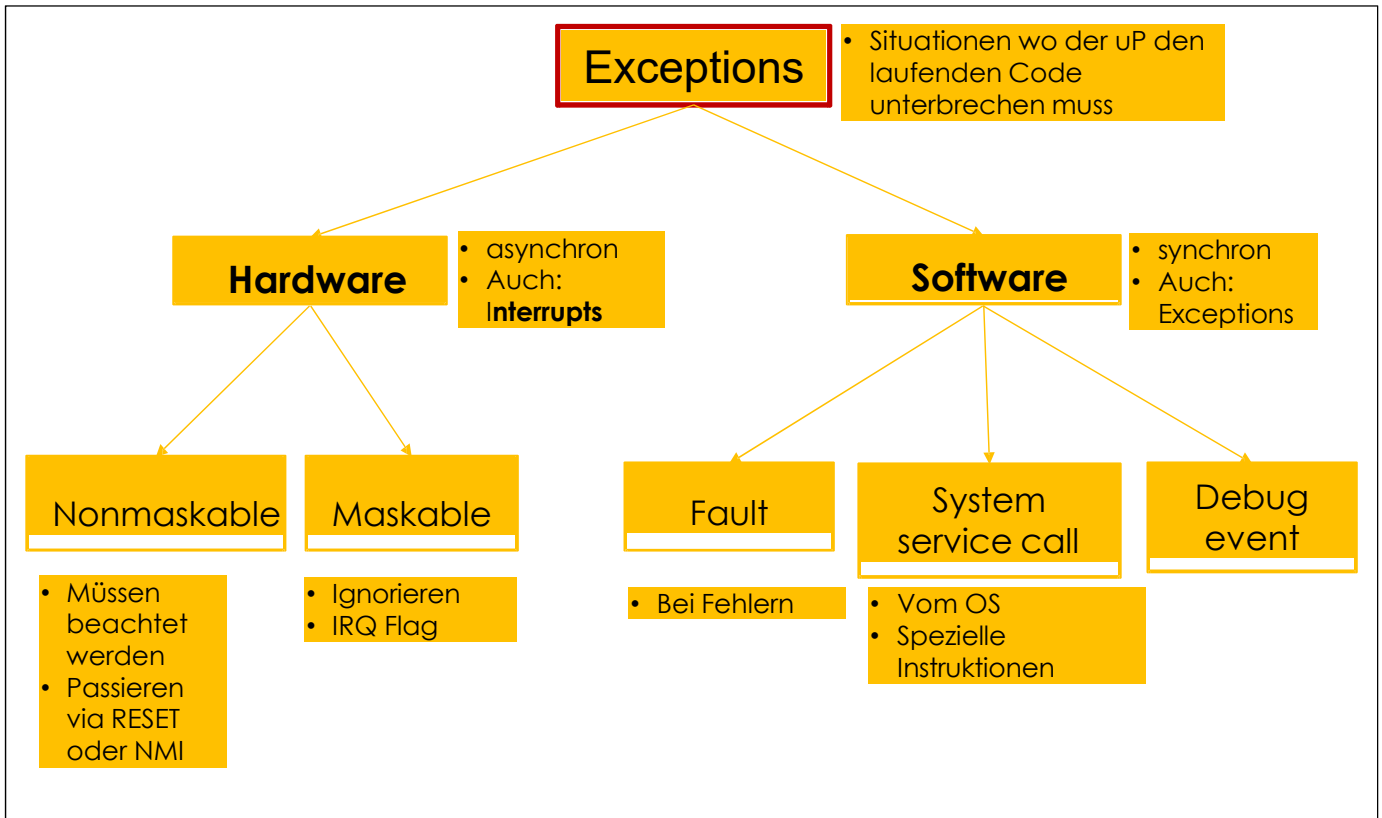
Version D: Print mit **grüner** Bodenplatte. Muss mit der LIB für Ver. D betrieben werden.

# 1 Interrupt Steuerung

## 1.1 Einleitung Theorie

### 1.1.1 Typen von Interrupts

Die nachfolgende Skizze zeigt die Einordnung der Interrupts (IR). Wir sehen, IR können von der Software wie auch von der Hardware ausgelöst werden.



### 1.1.2 Interrupt Request [1]

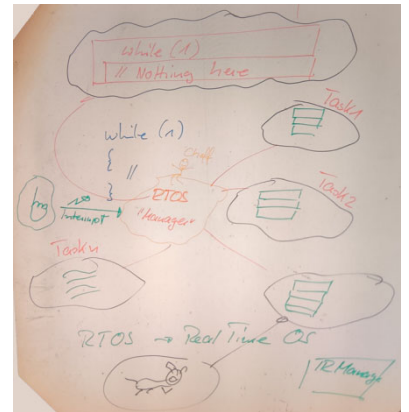
Normalerweise arbeitet ein Prozessor ( $\mu$ C oder CPU) ein Programm der Reihe nach ab. Praktisch alle CPUs und  $\mu$ Cs haben aber die Möglichkeit unabhängig vom normalen Programmfluss auf einige Ereignisse zu reagieren, die Interrupts. Typische Interruptquellen sind der Überlauf eines Timers, das Ende einer AD Wandlung und ein empfangenes Zeichen an einer Schnittstelle.

Wenn ein Interrupt Signal vorliegt, pausiert das normale Programm, und die ISR (**I**nterrupt **S**ervice **R**outine) wird stattdessen aufgerufen. Danach wird das normale Programm wieder fortgesetzt.

Mit den Interrupts kann die CPU mehrere Dinge sozusagen „gleichzeitig“ erledigen. In der ISR (je nach CPU auch direkt durch die Hardware), wird der Zustand der MCU, also die Register gesichert und am Ende der ISR wieder hergestellt. Das Hauptprogramm bemerkt so wenig von der ISR, außer der unvermeidlichen Verzögerung und den gewünschten Effekten in der ISR.

Weil die ISR eben doch manchmal stört, erlauben es fast alle CPUs, dass das Programm den Aufruf der ISR durch eine geeignete Steuerung, meist ein Interruptflag unterbinden kann. Der Interrupt wird dann "maskiert". In der Regel wird das auslösende Signal, das den Interrupt auslösen möchte, gespeichert und die ISR wird nachgeholt werden, wenn das Programm es wieder zulässt. Für wichtige Signale (z.B. Übertemperatur, Unterspannung), die nicht unterdrückt werden sollen gibt es teilweise nicht maskierbare Interrupts, die sich nicht so einfach abschalten lassen.

Bei Mikrocontrollern sind viele der Interruptquellen interne Module wie Timer, UART, I2C oder A/D, D/A. Je nach Prozessor gibt es je nach Interruptquelle eine eigene ISR.



### 1.1.3 Anwendungen für Interrupts

Die klassische Anwendung für Interrupts ist die Reaktion auf IO-Ereignisse. Die ISR zur UART kann z.B. die Daten von der RS232 Schnittstelle in einen Puffer speichern, oder aus einem Puffer senden.

Für Vorgänge die in regelmäßigen Zeitabständen erledigt werden sollen, eignen sich Timer Interrupts. Der Timer kann z.B. alle 10 ms ein Interrupt auslösen in dem dann z.B. die interne Uhrzeit aktualisiert wird oder Tasten abgefragt werden.

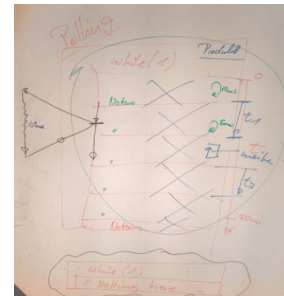
Wenn die CPU in einen Stromsparmodus versetzt wird, dient zu Aufwachen in der Regel auch ein Interrupt. Dadurch kann man z.B. in der Zeit, die man auf eine AD Wandlung wartet, Strom sparen.

### 1.1.4 Unterschiede zum Polling

Alternativ zur Verwendung von Interrupts lässt sich das Auftreten eines externen Ereignisses auch durch das regelmäßige Abfragen des externen Gerätes erkennen. Diese Technik nennt sich Polling (der Prozessor *frägt* das externe Gerät).

#### Vorteile des Pollings gegenüber der Verwendung von Interrupts:

- Es ist einfacher zu implementieren, vor allem für Anfänger, da die Abfrage des Geräts im Hauptprogramm erfolgen kann.
- Es reagiert meist schneller auf das externe Ereignis, sofern nur auf ein Ereignis gewartet wird. Bei Interrupts wird der aktuelle Befehl des Hauptprogramms noch fertig ausgeführt, dann wird die Rücksprungadresse gesichert und in die ISR verzweigt. Diese wiederum muss die zu benutzenden Register auf dem Stack sichern, bevor sie mit der eigentlichen Reaktion auf das externe Ereignis beginnen kann. Beim Polling kann nach Erkennung des Ereignisses sofort reagiert werden.
- Es ist weniger Hardwareaufwand bei den Geräten notwendig, da diese nicht in der Lage sein müssen, auf das Auftreten eines Ereignisses mit einer Interrupt-Anfrage zu reagieren. So braucht z.B. ein Temperatursensor für Polling im einfachsten Fall nur einen A/D-Wandler. Bei der Verwendung von Interrupts muss er jedoch neben der Möglichkeit, die Temperatur zu messen, zusätzlich auf die Änderung der Temperatur mit einer Interruptanfrage reagieren können.
- Auch Eingänge ohne Interruptfunktion können genutzt werden.



#### Vorteile von Interrupts gegenüber Polling:

- Das Hauptprogramm wird einfacher und besser verständlich, weil es sich auf das Wesentliche konzentriert, ohne ständig oder von unterschiedlichen Codestellen aus, Ereignisse abzufragen zu müssen.
- Es können leicht mehrere mögliche Signale überwacht werden.
- Das Auftreten des externen Ereignisses wird immer überwacht. Beim Polling geschieht dies nur zu den Zeiten, zu denen das Hauptprogramm danach fragt.
- Das Hauptprogramm kann andere Aufgaben übernehmen als das externe Gerät zu überwachen. Beim Polling wird häufig sämtliche Rechenleistung für die Abfrage der Geräte aufgewendet.
- Die Kommunikationsleitungen, Datenbusse und externen Geräte werden entlastet. Im Gegensatz zum Polling wird nur mit dem Gerät kommuniziert, wenn tatsächlich ein externes Ereignis stattfindet.
- Es ist oft stromsparender. Hat das Hauptprogramm keine Aufgaben mehr zu erledigen, so kann es Teile der CPU bis zum nächsten Auftreten eines Interrupts in einen stromsparenden Sleep-Mode versetzen. In diesem Modus wird die CPU angehalten und beim nächsten Interrupt wieder aktiviert. Fast alle modernen, interruptfähigen Prozessoren unterstützen einen solchen Modus.

In den allermeisten Fällen überwiegen die Vorteile der Verwendung von Interrupts deutlich!

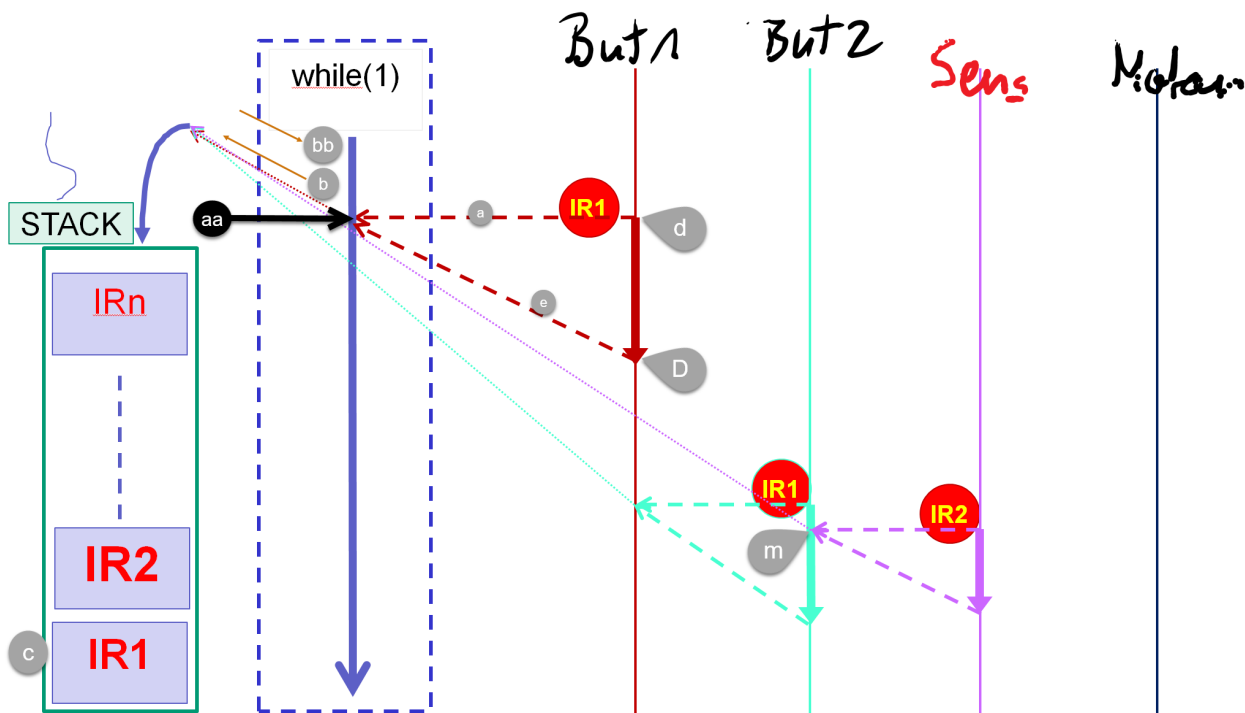
### 1.1.6 Interrupt-Prioritäten

Im einfachsten Fall sind während der Ausführung der ISR weitere Interrupts unterdrückt. Bei einigen Prozessoren gibt es Interrupts mit verschiedener Priorität. Dann kann ein Interrupt eine ISR niedrigerer Priorität unterbrechen. Davon zu unterscheiden sind Prioritäten bei der Signalisierung der Interrupts: Wenn mehrere Interrupts auf die Ausführung warten (z.B. weil Interrupts vorher gesperrt waren), wird darüber festgelegt welche ISR zuerst ausgeführt wird, sofern es mehr als eine ISR gibt.

### 1.1.7 Was passiert bei einem Interrupt

Der Prozessor wird bei aktuellem Prozess (WHILE(1)) unterbrochen. Das heisst der Prozess WHILE(1) muss den aktuellen Stand seiner Umgebung, Register, Programmcounter PC, sowie alle wichtigen Informationen für ein reibungsloses Weiterarbeiten nach dem Interrupt abspeichern. Der Speicher, wo die Daten abgespeichert werden heisst Stack-Speicher. (Stapel). Beim MCB32 ist er mit der verwendeten Einstellung 400Bytes gross.

**Beispiel:** Der Button 1 kann durch Drücken einen Interrupt IR1 (a) auslösen. Sofort werden die aktuellen Daten (aa) via Stack (b) im Speicherbereich IR1 (c) abgespeichert. Nun läuft (d) die Interrupt-Service Routine (ISR) für den Button1. Wenn diese beendet wird (Ende des braunen Pfeils bei (D)) wird die ISR abgeschlossen und der Hauptprozess (WHILE(1)) wird nach dem Abholen und „Restaurieren“ der Daten (bb) am unterbrochenen Ort (aa) fortgesetzt.



Der Stapelspeicher ist ein spezieller Bereich welcher vom Nutzer reserviert wird. Die Daten landen nach der First-In-Last-Out-Methode (FILO) in diesem Speicher. Die zuerst abgespeicherten Daten werden zuletzt abgeholt.

Dies ist beim Ereignis Button2 gut zu sehen. Diese ISR wird durch den Sensor mit IR2 bei (m) unterbrochen. Also müssen die Daten, wie beim Beispiel, vom aktuellen Zustand der Button2-Umgebung gespeichert werden. Diese Daten landen dann im Bereich IR2 auf dem Stack. Wenn die ISR des Sensors fertig ist, werden die Daten aus dem Bereich IR2 zurückgeholt und die ISR für den Button2 wird bei (m) fortgesetzt. Diese Überlegung kann nun beliebig fortgesetzt werden.

Ist einleuchtend, dass hier eine gute Übersicht nötig ist. Sogenannten RTOS (Real-Time Operating Systems) übernehmen die Verwaltung dieser Unterbrechungen.

## 1.2 Einleitung für den STM32F107

Der STM32F107 hat mehr als 60 interne (Schnittstellen, RTC, Timer, Watchdog, ...) Interrupt-Quellen und 20 externe (Pins). Dabei unterbricht ein Interrupt die laufende Programmabarbeitung für die Ausführung der Interruptfunktion. Die Interrupts können mit Prioritäten versehen werden. 16 Interrupt-Levels kennt der Chip. Interrupts mit höherer Priorität unterbrechen Interrupts mit niedrigerer Priorität. Die Interruptstruktur ist entsprechend aufwändig. Siehe auch Anhang mit Blockschema.

### 1.2.1 Beispiel mit AD-Wandler

Wenn wir z.Bsp einen AD-Wandler einsetzen wollen gilt in etwa folgendes:

Jeweils am Ende der AD-Wandlung löst der Wandler den End Of Conversion Interrupt (EOCI) aus. Damit kann eine Interruptroutine angestossen werden, welche den gewandelten Wert aus dem Register liest. Dadurch erhält man den „neuen“ Wert augenblicklich. Eine zeitliche Abstimmung ist dadurch möglich.

D.h:

- ➔ Wir brauchen eine Interruptfunktion, wobei der Name der Funktion der Link ist auf die Vektortabelle (Einsprungsadresse in Interruptfunktion). Der ADCI hat die Interruptnummer **18** (siehe table 61 im reference manual) .
- ➔ EOCIE (end of conversion interrupt enable) muss freigegeben sein, siehe **6.1**.
- ➔ Der “Nested vectored interrupt controller (NVIC)” managed die Interruptquellen. Dazu müssen wir im Minimum den EOCI des ADC freigegeben. Folgende Tabelle ist im „programming manual“ zu finden.

#### Also:

Am Ende der AD-Wandlung löst der Wandler den End Of Conversion Interrupt (EOCI) aus. Damit kann eine Interruptroutine angestossen werden, welche den gewandelten Wert aus dem Register liest. Dadurch erhält man den „neuen“ Wert augenblicklich. Eine zeitliche Abstimmung ist dadurch möglich.

Das dafür nötige Register finden Sie auf der nächsten Seite.

Table 41. Mapping of interrupts to the interrupt variables

Interrupts	CMSIS array elements <sup>(1)</sup>				
	Set-enable	Clear-enable	Set-pending	Clear-pending	Active Bit
0-31	ISER[0]	ICER[0]	ISPR[0]	ICPR[0]	IABR[0]
32-63	ISER[1]	ICER[1]	ISPR[1]	ICPR[1]	IABR[1]
64-67	ISER[2]	ICER[2]	ISPR[2]	ICPR[2]	IABR[2]

1. Each array element corresponds to a single NVIC register, for example the element ICER[1] corresponds to the ICER1 register.

Der Interrupt für den **ADC** muss im ISER[0] Register „ge-enabled“ werden. Das Register hat folgenden Aufbau.

#### 4.3.2 Interrupt set-enable registers (NVIC\_ISERx)

Address offset: 0x00 - 0x0B

Reset value: 0x0000 0000

Required privilege: Privileged

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SETENA[31:16]															
rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SETENA[15:0]															
rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs

Bits 31:0 **SETENA[31:0]**: Interrupt set-enable bits.

**Write:**

- 0: No effect
- 1: Enable interrupt

**Read:**

- 0: Interrupt disabled
- 1: Interrupt enabled.

See [Table 41: Mapping of interrupts to the interrupt variables on page 119](#) for the correspondence of interrupts to each register bit.

If a pending interrupt is enabled, the NVIC activates the interrupt based on its priority. If an interrupt is not enabled, asserting its interrupt signal changes the interrupt state to pending, but the NVIC never activates the interrupt, regardless of its priority.

## 2 Interrupts im STM32F107xx

### 2.1 Einleitung

Für externe Interrupts stehen insgesamt 16 Interrupt Lines zur Verfügung. Jede Line kann mit einem beliebigen Port verknüpft werden, wobei die Pin-Nummer bereits festgelegt ist. EXTI0 kann mit PA0, PB0, ... verschaltet werden, jedoch beispielsweise nicht mit PA1.

Anmerkung: Es sind noch weitere EXTI Lines vorhanden, deren Quellen nicht konfigurierbar sind – siehe Reference Manual.

### 2.2 Funktionsdiagramm Interruptselector

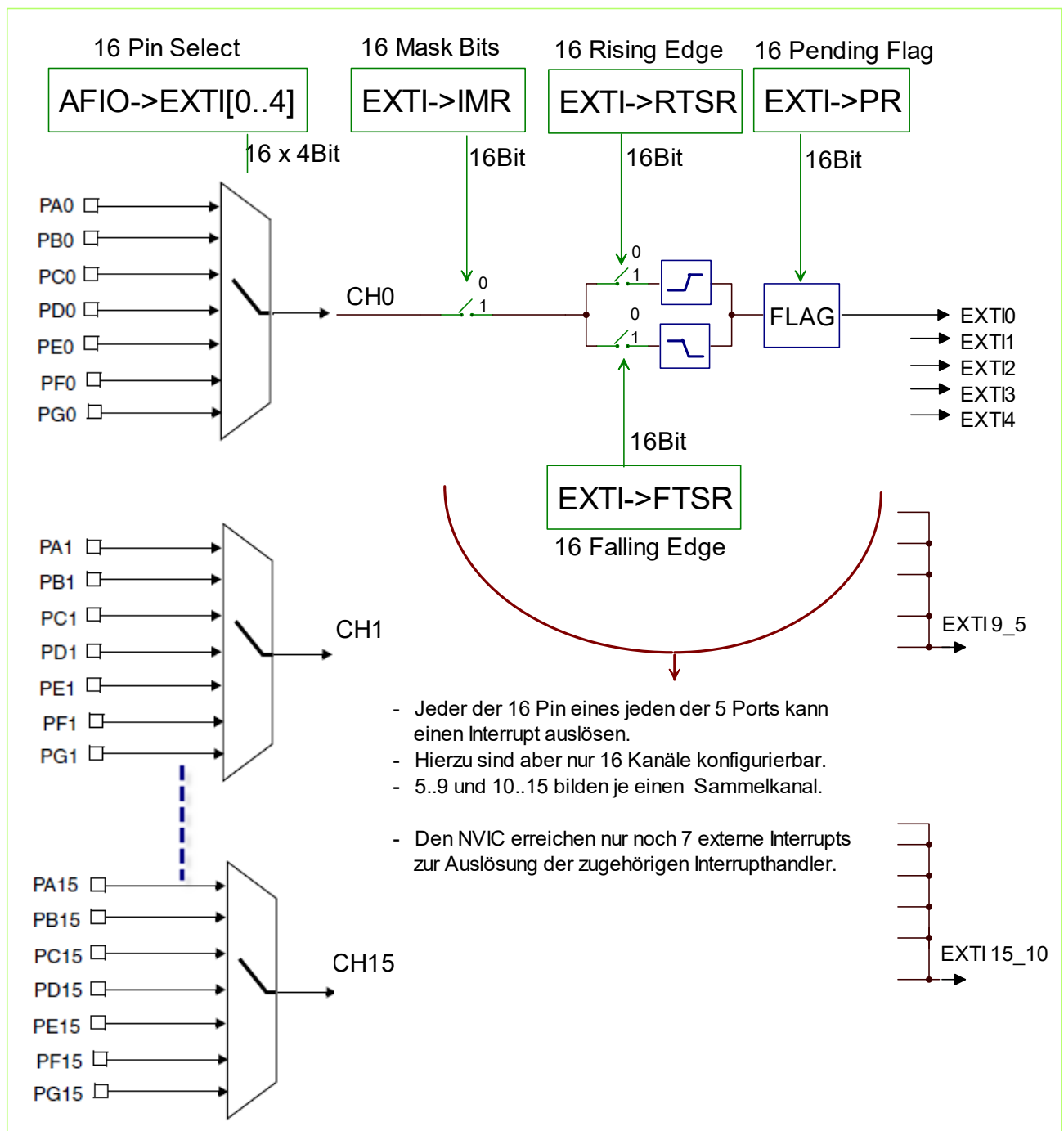


Abbildung 1: Interrupt Funktionsdiagramm



Nach dem Interrupt-Selector folgt der NVIC-Block. Der NVIC (Nested Vector Interrupt Controller) ist für die Verwaltung der Interrupts verantwortlich. Genauere Information zum NVIC findet man im Programming Manual (nicht im Reference-Manual).

Beim Hardware Design ist darauf zu achten dass nur EXTI0-EXTI4 separate Interrupt-Vektoren besitzen. Die restlichen EXTI-Lines werden in EXTI9\_5 und EXTI15\_10 zusammengefasst.

Im Anhang findet man die Interrupt-Vektorliste und die Definition der Serviceaufrufe.

An dieser Stelle sei ein Beispiel für den Interrupt-Handler für den „EXTI Line0 interrupt“ aufgeführt:

#### **Beispiel Interrupthandler**

```
void EXTI0_IRQHandler(void)
```

### 3 Registerauszug STM32F107 Interrupts

#### 1. External interrupt configuration register 1 (AFIO\_EXTICR1)

8.4.3

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI3[3:0]				EXTI2[3:0]				EXTI1[3:0]				EXTI0[3:0]			
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

Bits 31:16 Reserved

Bits 15:0 **EXTIx[3:0]**: EXTI x configuration (x= 0 to 3)

These bits are written by software to select the source input for EXTIx external interrupt.

0000: PA[x] pin    0100: PE[x] pin  
 0001: PB[x] pin    0101: PF[x] pin  
 0010: PC[x] pin    0110: PG[x] pin  
 0011: PD[x] pin

**Abbildung 2:** External interrupt configuration register 1 (AFIO\_EXTICR1) Reset value: 0x0000 [2, p. p 191]

#### 2. Interrupt mask register (EXTI\_IMR)

9.3.1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved												MR19	MR18	MR17	MR16
												r/w	r/w	r/w	r/w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MR15	MR14	MR13	MR12	MR11	MR10	MR9	MR8	MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

Bits 31:20 Reserved, must be kept at reset value (0).

Bits 19:0 **MRx**: Interrupt Mask on line x

0: Interrupt request from Line x is masked

1: Interrupt request from Line x is not masked

Note: Bit 19 is used in connectivity line devices only and is reserved otherwise.

**Abbildung 3:** Interrupt mask register (EXTI\_IMR) Reset value: 0x0000 . [2, p. p 210]

#### 3. Rising trigger selection register (EXTI\_RTSTR)

9.3.3

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved												TR19	TR18	TR17	TR16
												r/w	r/w	r/w	r/w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

Bits 19:0 **TRx**: Rising trigger event configuration bit of line x

0: Rising trigger disabled (for Event and Interrupt) for input line

1: Rising trigger enabled (for Event and Interrupt) for input line.

Bits 31:20 Reserved, must be kept at reset value (0)

**Abbildung 4:** Rising trigger selection register (EXTI\_RTSTR) Reset value: 0x0000 . [2, p. p 211]

## 4. Falling trigger selection register (EXTI\_FTSR) 9.3.4

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved												TR19	TR18	TR17	TR16
												rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 19:0 **TRx**: Falling trigger event configuration bit of line x

Bits 31:20 Reserved, must be kept at reset value (0)

0: Falling trigger disabled (for Event and Interrupt) for input line

1: Falling trigger enabled (for Event and Interrupt) for input line.

**Abbildung 5:** Rising trigger selection register (EXTI\_FTSR) Reset value: 0x0000 . [2, p. p 211]

## 5. Pending register (EXTI\_PR) 9.3.6

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved												PR19	PR18	PR17	PR16
												rc_w1	rc_w1	rc_w1	rc_w1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PR15	PR14	PR13	PR12	PR11	PR10	PR9	PR8	PR7	PR6	PR5	PR4	PR3	PR2	PR1	PR0
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Bits 19:0 **PRx**: Pending bit

Bits 31:20 Reserved, must be kept at reset value (0).

0: No trigger request occurred

1: selected trigger request occurred

This bit is set when the selected edge event arrives on the external interrupt line. This bit is cleared by writing a '1' into the bit.

**Abbildung 6:** Pending register (EXTI\_PR) Reset value: undefined. [2, p. p 212]

## 4 Mustercodes

### 1. Mustercode für den externen Interrupt ExtIO (Button 0):

Die LEDs zählen die pos. Flanken von Button0 PA0 mit P1-Anzeige am Touchscreen; Prellen beachten

**Lösung:**  
Code für Int  
EXTIO

```
#include "TouchP0P1.h"           // P0-, P1-Definition

void EXTIO_IRQHandler(void)      // Vorgeg. Handlername
{
    EXTIO->PR |= 1;               // Zuerst: Loesche
    P1++;                         // Intr Flag 1 = Pin0
                                // Zaehler erhoehen
}

int main(void)                   // Hauptprogramm
{
    InitTouchP0P1("1");          // P0P1-Touchscreen ON
                                // Ports initialisieren:
    RCC->APB2ENR |= 1<<2;         // Enable GPIOA clock, Input
    RCC->APB2ENR |= 1<<6;         // Enable GPIOE clock, LEDs
    GPIOA->CRL  &= 0xFFFFF00;    // Configure the GPIOA 0..7
    GPIOA->CRL  |= 0x00000088;    // Pullup Input PA_0, PA_1

    RCC->APB2ENR |= 1;           // Alternate Funct. AFIO ON
    AFIO ->EXTICR[0] = 0x00;     // Pin0 von PA0 auf EXTIO
    EXTI->IMR  |= 1;             // EXTI0 ON-Maske ON
    EXTI->RTSR |= 1;             // Steigende Triggerflanke
    EXTI->FTSR |= 1;             // Fallende Triggerflanke
    NVIC->ISER[0] |= 1<<6;       // Enable IntrNr 6 für EXTI0
    while(1)                     // Endlosschleufe
    {
                                // Nichts tun
    }
}
```

**Abbildung 7:** Die LEDS zählen die pos. Flanken von Button0 (PA0), Prellen.

### 2. Mustercode für diverse Interrupts und blinkende LEDs

- Toggelt LED14 im Hauptprogramm und toggelt LED15 vom Interrupt „SysTick“ und toggelt LED13 vom Interrupt „ControlStick UP“ EXTIO15\_10\_IRQn.
- Output LED15 löst denselben Intr.EXTIO15\_10\_IRQn aus und toggelt LED12

```

#include "stm32f10x.h"

void InitEXTI(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    EXTI_InitTypeDef EXTI_InitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;
    // enable AFIO clock: IN GPIOD_12 ControlStick_UP
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE);
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12;
    GPIO_Init(GPIOD, &GPIO_InitStructure);
    // Connect EXTI Lines to Button Pins
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
    GPIO_EXTILineConfig(GPIO_PortSourceGPIOD, GPIO_PinSource12);
    GPIO_EXTILineConfig(GPIO_PortSourceGPIOE, GPIO_PinSource15);
    // NEU!!! select EXTI line15, interrupt mode, rising edge, enable EXTI line
    EXTI_InitStructure.EXTI_Line = EXTI_Line15;
    EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
    EXTI_InitStructure.EXTI_LineCmd = ENABLE;
    EXTI_Init(&EXTI_InitStructure);
    //select EXTI line12, interrupt mode, rising edge, enable EXTI line
    EXTI_InitStructure.EXTI_Line = EXTI_Line12;
    EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
    EXTI_InitStructure.EXTI_LineCmd = ENABLE;
    EXTI_Init(&EXTI_InitStructure);
    // configure NVIC channel, lowest priority, lowest subpriority, enable channel
    NVIC_InitStructure.NVIC_IRQChannel = EXTI15_10_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x0F;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x0F;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
//-----
void EXTI15_10_IRQHandler(void)
{
    static BitAction Bit1 = Bit_SET;
    static BitAction Bit2 = Bit_SET;
    if(EXTI_GetITStatus(EXTI_Line12) != RESET)    // Check EXTI_Line12
    {
        Bit1 = (BitAction)!Bit1;
        GPIO_WriteBit(GPIOE, GPIO_Pin_13, Bit1);
        EXTI_ClearITPendingBit(EXTI_Line12);    // Clear pending Intr
    }
    if(EXTI_GetITStatus(EXTI_Line15) != RESET)    // Check EXTI_Line15
    {
        Bit2 = (BitAction)!Bit2;
        GPIO_WriteBit(GPIOE, GPIO_Pin_12, Bit2);
        EXTI_ClearITPendingBit(EXTI_Line15);    // Clear pending Intr
    }
}

```

**Abbildung 8:** Musterprogramm für div. Interrupts weiter nächste Seite

```

//-----
void InitLEDs(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOE, ENABLE);
    GPIO_InitStructure.GPIO_Pin =
        GPIO_Pin_10|GPIO_Pin_11|GPIO_Pin_12|GPIO_Pin_13|GPIO_Pin_14|GPIO_Pin_15;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOE, &GPIO_InitStructure);
}
//-----
void SysTick_Handler(void)
{
    static BitAction val = Bit_SET;
    val = (BitAction)!val;
    GPIO_WriteBit(GPIOE, GPIO_Pin_15, val);
}
//-----
int main(void)
{
    long t; int dataByte=0;
    SystemInit();
    InitLEDs();
    InitEXTI();
    SysTick_Config(15000000); // SysTick interrupt
    while (1)
    {
        dataByte = GPIO_ReadInputDataBit(GPIOE, GPIO_Pin_13); // Lese Status von LED13
        GPIO_SetBits(GPIOE, GPIO_Pin_14);
        if(dataByte==0){ // Wenn LED13 AUS dann
            GPIO_SetBits(GPIOE, GPIO_Pin_14); // Blinke mit LED14

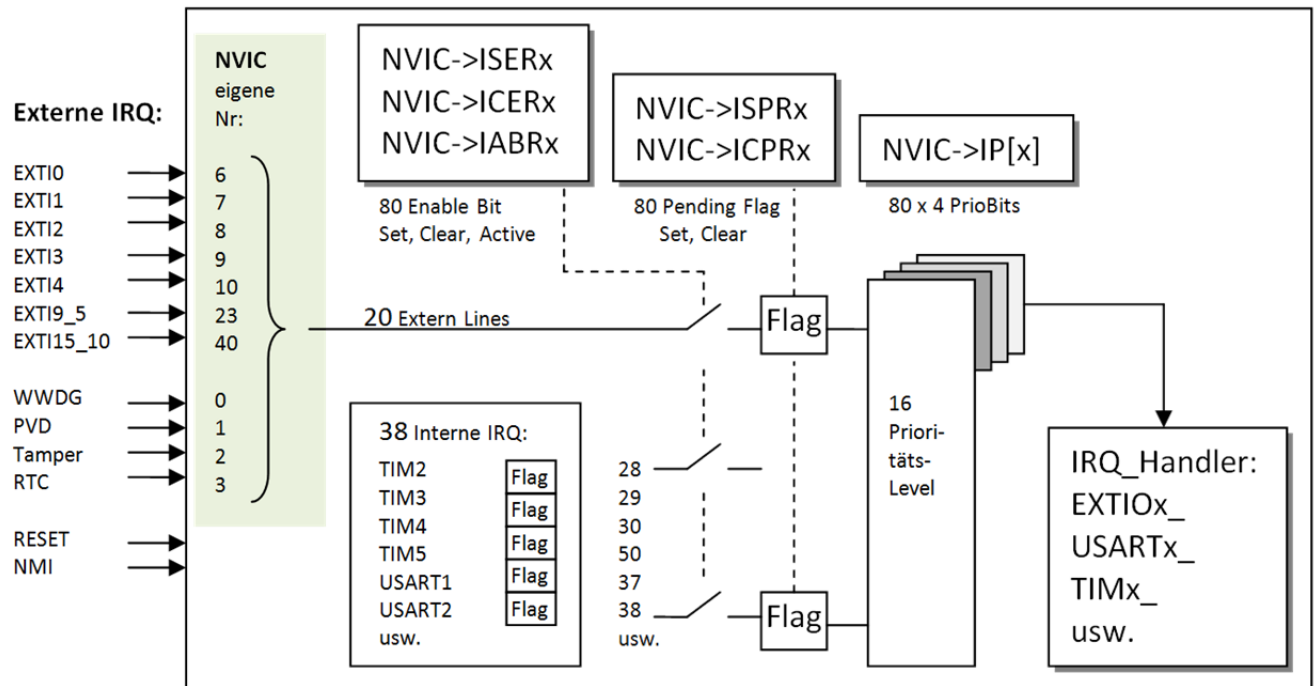
            for(t=0;t<500000;t++){
                GPIO_ResetBits(GPIOE, GPIO_Pin_14);
                for(t=0;t<400000;t++){
                }
            }
            else{ // sonst mit LED10
                GPIO_ResetBits(GPIOE, GPIO_Pin_14); // Loesche LED14
                GPIO_SetBits(GPIOE, GPIO_Pin_10);
                for(t=0;t<200000;t++){
                }
                GPIO_ResetBits(GPIOE, GPIO_Pin_10);
                for(t=0;t<100000;t++){
                }
            }
        }
    }
}

```

**Abbildung 9:** Musterprogramm für div. Interrupts Fortsetzung

## 4.1 Nested Vector Interrupt Controller NVIC

für insgesamt 58 Interrupts.



**Abbildung 10:** NVIC Blockdiagramm

Der NVIC→IP[x]-Block legt den Prioritäts-Level des Interrupts zwischen 0 und 15 fest. Interrupts mit einer niedrigeren Nummer besitzen eine höhere Priorität. Wird ein Interrupt mit einer höheren Priorität ausgelöst, während ein Interrupt einer niedrigeren Priorität abgearbeitet wird, so wird letzterer unterbrochen und die Abarbeitung des Interrupt Handlers des höher priorisierten Interrupts gestartet.

## 5 Anhang: Interrupt Vektorliste und Servicefunktionsaufrufe

Die Namen sind in einer Vektor-Tabelle im File: *startup\_stm32f10x\_cl.s* definiert. Dort wird auch der Bereich des Stack-Speichers festgelegt.

NVIC NR	Priority	Type of priority	Acronym	Description	Address	Interrupt Handler Service Name
	-	-	-	Reserved	0x0000_0000	
	-3	fixed	Reset	Reset	0x0000_0004	
	-2	fixed	NMI	Non maskable interrupt. The RCC Clock Security System (CSS) is linked to the NMI vector.	0x0000_0008	
	-1	fixed	HardFault	All class of fault	0x0000_000C	
	0	settable	MemManage	Memory management	0x0000_0010	
	1	settable	BusFault	Pre-fetch fault, memory access fault	0x0000_0014	
	2	settable	UsageFault	Undefined instruction or illegal state	0x0000_0018	
	-	-	-	Reserved	0x0000_001C - 0x0000_002B	
	3	settable	SVCall	System service call via SWI instruction	0x0000_002C	
	4	settable	Debug Monitor	Debug Monitor	0x0000_0030	
	-	-	-	Reserved	0x0000_0034	<b>Beispiel Interrupthandler</b>  <code>void EXTIx_IRQHandler(void)</code>
	5	settable	PendSV	Pendable request for system service	0x0000_0038	
	6	settable	SysTick	System tick timer	0x0000_003C	
0	7	settable	WWDG	Window Watchdog interrupt	0x0000_0040	WWDG_IRQHandler
1	8	settable	PVD	PVD through EXTI Line detection interrupt	0x0000_0044	PVD_IRQHandler
2	9	settable	TAMPER	Tamper interrupt	0x0000_0048	TAMPER_IRQHandler
3	10	settable	RTC	RTC global interrupt	0x0000_004C	RTC_IRQHandler
4	11	settable	FLASH	Flash global interrupt	0x0000_0050	FLASH_IRQHandler
5	12	settable	RCC	RCC global interrupt	0x0000_0054	RCC_IRQHandler
6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000_0058	EXTI0_IRQHandler
7	14	settable	EXTI1	EXTI Line1 interrupt	0x0000_005C	EXTI1_IRQHandler
8	15	settable	EXTI2	EXTI Line2 interrupt	0x0000_0060	EXTI2_IRQHandler
9	16	settable	EXTI3	EXTI Line3 interrupt	0x0000_0064	EXTI3_IRQHandler
10	17	settable	EXTI4	EXTI Line4 interrupt	0x0000_0068	EXTI4_IRQHandler
11	18	settable	DMA1_Channel1	DMA1 Channel1 global interrupt	0x0000_006C	DMA1_Channel1_IRQHandler
12	19	settable	DMA1_Channel2	DMA1 Channel2 global interrupt	0x0000_0070	DMA1_Channel2_IRQHandler
13	20	settable	DMA1_Channel3	DMA1 Channel3 global interrupt	0x0000_0074	DMA1_Channel3_IRQHandler
14	21	settable	DMA1_Channel4	DMA1 Channel4 global interrupt	0x0000_0078	DMA1_Channel4_IRQHandler
15	22	settable	DMA1_Channel5	DMA1 Channel5 global interrupt	0x0000_007C	DMA1_Channel5_IRQHandler



## Einsatz von Interrupts

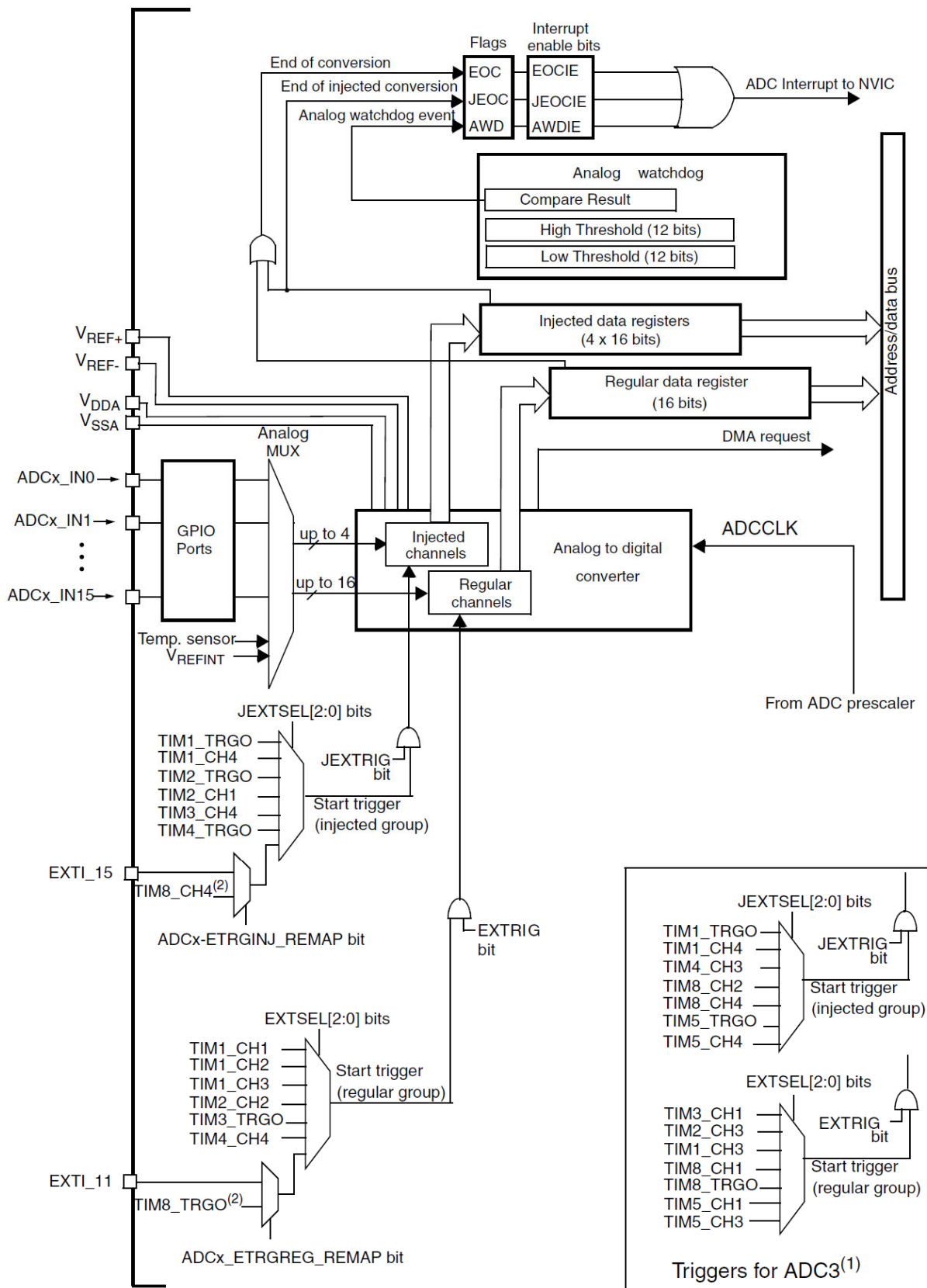
NVIC NR	Priority	Type of priority	Acronym	Description	Address	Interrupt Handler Service Name
16	23	settable	DMA1_Channel6	DMA1 Channel6 global interrupt	0x0000_0080	DMA1_Channel6_IRQHandler
17	24	settable	DMA1_Channel7	DMA1 Channel7 global interrupt	0x0000_0084	DMA1_Channel7_IRQHandler
18	25	settable	ADC1_2	ADC1 and ADC2 global interrupt	0x0000_0088	ADC1_2_IRQHandler
19	26	settable	CAN1_TX	CAN1 TX interrupts	0x0000_008C	CAN1_TX_IRQHandler
20	27	settable	CAN1_RX0	CAN1 RX0 interrupts	0x0000_0090	CAN1_RX0_IRQHandler
21	28	settable	CAN1_RX1	CAN1 RX1 interrupt	0x0000_0094	CAN1_RX1_IRQHandler
22	29	settable	CAN1_SCE	CAN1 SCE interrupt	0x0000_0098	CAN1_SCE_IRQHandler
23	30	settable	EXTI9_5	EXTI Line[9:5] interrupts	0x0000_009C	EXTI9_5_IRQHandler
24	31	settable	TIM1_BRK	TIM1 Break interrupt	0x0000_00A0	TIM1_BRK_IRQHandler
25	32	settable	TIM1_UP	TIM1 Update interrupt	0x0000_00A4	TIM1_UP_IRQHandler
26	33	settable	TIM1_TRG_COM	TIM1 Trigger and Commutation interrupts	0x0000_00A8	TIM1_TRG_COM_IRQHandler
27	34	settable	TIM1_CC	TIM1 Capture Compare interrupt	0x0000_00AC	TIM1_CC_IRQHandler
28	35	settable	TIM2	TIM2 global interrupt	0x0000_00B0	TIM2_IRQHandler
29	36	settable	TIM3	TIM3 global interrupt	0x0000_00B4	TIM3_IRQHandler
30	37	settable	TIM4	TIM4 global interrupt	0x0000_00B8	TIM4_IRQHandler
31	38	settable	I2C1_EV	I2C1 event interrupt	0x0000_00BC	I2C1_EV_IRQHandler
32	39	settable	I2C1_ER	I2C1 error interrupt	0x0000_00C0	I2C1_ER_IRQHandler
33	40	settable	I2C2_EV	I2C2 event interrupt	0x0000_00C4	I2C2_EV_IRQHandler
34	41	settable	I2C2_ER	I2C2 error interrupt	0x0000_00C8	I2C2_ER_IRQHandler
35	42	settable	SPI1	SPI1 global interrupt	0x0000_00CC	SPI1_IRQHandler
36	43	settable	SPI2	SPI2 global interrupt	0x0000_00D0	SPI2_IRQHandler
37	44	settable	USART1	USART1 global interrupt	0x0000_00D4	USART1_IRQHandler
38	45	settable	USART2	USART2 global interrupt	0x0000_00D8	USART2_IRQHandler
39	46	settable	USART3	USART3 global interrupt	0x0000_00DC	USART3_IRQHandler
40	47	settable	EXTI15_10	EXTI Line[15:10] interrupts	0x0000_00E0	EXTI15_10_IRQHandler
41	48	settable	RTCAlarm	RTC alarm through EXTI line interrupt	0x0000_00E4	RTCAlarm_IRQHandler
42	49	settable	OTG_FS_WKUP	USB On-The-Go FS Wakeup through EXTI line interrupt	0x0000_00E8	OTG_FS_WKUP_IRQHandler
-	-	-	-	Reserved	0x0000_00EC - 0x0000_0104	-__IRQHandler
50	57	settable	TIM5	TIM5 global interrupt	0x0000_0108	TIM5_IRQHandler
51	58	settable	SPI3	SPI3 global interrupt	0x0000_010C	SPI3_IRQHandler
52	59	settable	UART4	UART4 global interrupt	0x0000_0110	UART4_IRQHandler
53	60	settable	UART5	UART5 global interrupt	0x0000_0114	UART5_IRQHandler
54	61	settable	TIM6	TIM6 global interrupt	0x0000_0118	TIM6_IRQHandler
55	62	settable	TIM7	TIM7 global interrupt	0x0000_011C	TIM7_IRQHandler
56	63	settable	DMA2_Channel1	DMA2 Channel1 global interrupt	0x0000_0120	DMA2_Channel1_IRQHandler

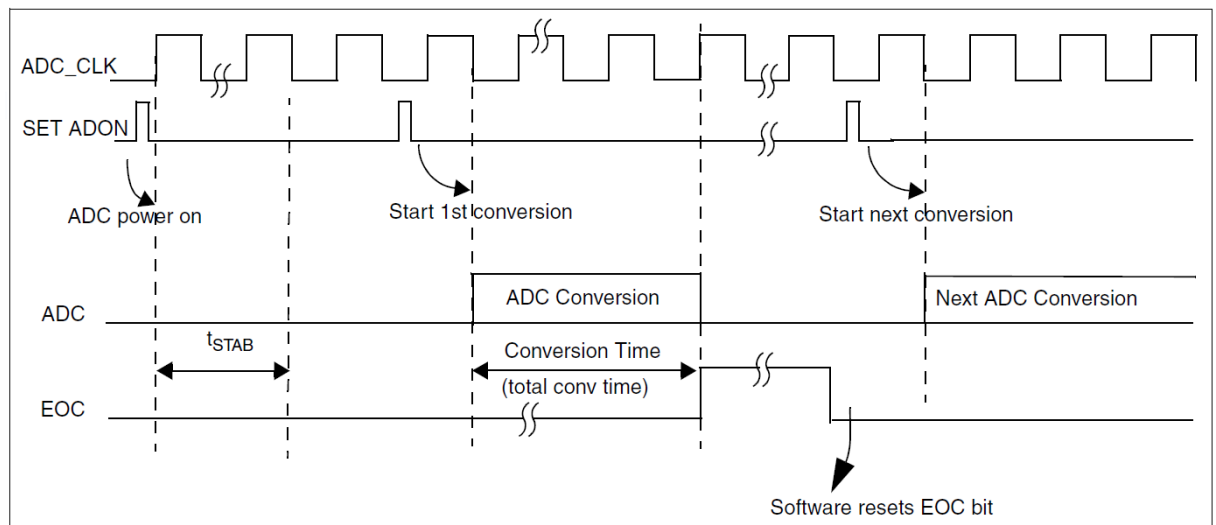
## Einsatz von Interrupts

NVIC NR	Priority	Type of priority	Acronym	Description	Address	Interrupt Handler Service Name
57	64	settable	DMA2_Channel2	DMA2 Channel2 global interrupt	0x0000_0124	DMA2_Channel2_IRQHandler
58	65	settable	DMA2_Channel3	DMA2 Channel3 global interrupt	0x0000_0128	DMA2_Channel3_IRQHandler
59	66	settable	DMA2_Channel4	DMA2 Channel4 global interrupt	0x0000_012C	DMA2_Channel4_IRQHandler
60	67	settable	DMA2_Channel5	DMA2 Channel5 global interrupt	0x0000_0130	DMA2_Channel5_IRQHandler
61	68	settable	ETH	Ethernet global interrupt	0x0000_0134	ETH_IRQHandler
62	69	settable	ETH_WKUP	Ethernet Wakeup through EXTI line interrupt	0x0000_0138	ETH_WKUP_IRQHandler
63	70	settable	CAN2_TX	CAN2 TX interrupts	0x0000_013C	CAN2_TX_IRQHandler
64	71	settable	CAN2_RX0	CAN2 RX0 interrupts	0x0000_0140	CAN2_RX0_IRQHandler
65	72	settable	CAN2_RX1	CAN2 RX1 interrupt	0x0000_0144	CAN2_RX1_IRQHandler
66	73	settable	CAN2_SCE	CAN2 SCE interrupt	0x0000_0148	CAN2_SCE_IRQHandler
67	74	settable	OTG_FS	USB On The Go FS global interrupt	0x0000_014C	OTG_FS_IRQHandler

## 6 Anhang Informationen zu ADC

### 6.1 Blockschaltbild





## 7 Aufgaben

Aufgabe #	Auftrag
Inter. 1	Suche im Internet nach Quellen rund um die Interruptsteuerung für den STm32F10xx Proz.
Inter. 2	Was macht folgender Code ?  Schreibe ihn ab und dokumentiere ihn vollständig. Funktioniert er?

```
// ??????
#include <stm32f10x.h>

void ADC1_2_IRQHandler(void)    // Warum dieser Name
{
    GPIOE->ODR = (ADC1->DR)<<4    // Was machen wir hier, warum <<4
}
// -----
int main(void)
{
    unsigned long i;

    //-----
    RCC->APB2ENR |= 1<<6;
    GPIOE->CRH = 0x11111111;
    //-----
    //Initialisierung Alternate Functions/peripheral clock/ADC
    //-----
    RCC->APB2ENR = RCC->APB2ENR | 0x00000211;
    RCC->CFGR    |= (2<<14);
    ADC1->SQR3    = (14<<0);
    ADC1->CR2     |= (7<<17);
    ADC1->CR2     |= (1<<20);
    ADC1->CR2     |= (1<<1);
    ADC1->CR1     |= (1<<5);    //EOCIE
    NVIC->ISER[0] |= (1<<18);    //interrupt set enable register for adc
    ADC1->CR2     |= (1<<0);    //ADC ON
    //-----

    for (i=0;i<1000000;i++);    //delay between ADC-ON and start conversion->see manual
    ADC1->CR2 |= (1<<22);    //start conversion
    while(1)    //forever
    {
        //nothing to do!
    }
}
```

Inter. 3	Die Beispiele aus der Vorlage zu Laufen bringen. Siehe <b>Mustercodes</b>
Inter. 4	Versuche mit Hilfe der Befehlsliste, einer IRs für Timer5 und dem Timer5 einen Blinker (1s Blink Zeit) auf PEH zu programmieren.

## 8 Library TouchP0P1.Lib

Die bekannte Library umfasst neben den Befehlen für die Grafikansteuerung auch Befehle für das einfache Handling der Hardware (AD-DA Wandler usw.) sowie Befehle für das Handling der Interrupts.

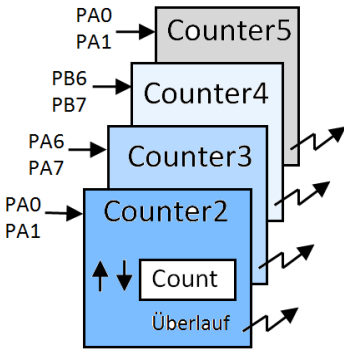
### 8.1 Ressourcennutzung

TouchP0P1.lib 8..17KByte	P0/P1 definiert an Ports	P0/P1 auf Screen	Touch, Grafik, Text, Periphere Funktionen	Sys-Timer belegt	Sys-Tick_Handler belegt
InitTouchP0P1 ("1..");	ja	ja	ja	1ms	aktiv
InitTouchP0P ("0");	ja	nein	ja	1ms	passiv

### 8.2 Interrupt Funktionen

		Version und Stand: 2203.01
4 x Externe Interrupt Requests (Vereinfacht, jede pinNr nur 1x)		
<b>ExtIRQInit</b> ("Pin", Flanke0/1/2, IRQPrio0..15);	Pin[" "]: "PA0"..PE15" Interrupt auslösender Pin  Flanke: 0,1,2 Auslösende Flanke: 0 pos, 1 neg, 2 beide  IRQPrio: 0, 1..15 Interruptpriorität 0 = AUS, 1 höchste, 15 tiefste.  Px0..Px4 lösen 5 separate Interrupts aus. Px5..Px9 und Px10..Px15 je einen Interrupt. Das Programm springt in die zugehörigen 7 IRQ-Handler <i>EXTI0_IRQHandler()</i> .. <i>EXTI15_10_IRQHandler()</i> .  <pre>// IRQ PA0,pos Flanke Priorität 4 ExtIRQInit("PA0",0,4);</pre>	<p>EXTI0_IRQHandler ... EXTI15_I0_IRQHandler</p>
6 x Interne Interrupt Requests		
Priorität: 0= OFF, 1 = Höchste .... 15=Niedrigste		
<b>TimerInit</b> (2..5,n*100us,IRQPrio0..15);	2..5 = Timer Nummer.  n muss >=1 sein, in 100us Schritten  Interrupt-Priorität; 0 = AUS, 1 höchste, 15 tiefste  Aktiviert den TimerClock und zählt mit dem Systemclock. Bei Ablauf wird sein Flag 1 und wenn IRQPrio >0 löst jeder Timer seinen eigenen Interrupt aus.  <pre>TimerInit (2, 5000, 0); // T2 auf 500mS,                         // ohne Interrupt</pre>	
int <b>TimerGetTime</b> (2..5);	Liefert einen 16Bit Wert zurück.	
	<pre>var = TimerGetTime(2); // Zeit in n*100us</pre>	
char <b>TimerGetFlag</b> (2..5);	Liefert 0 oder 1 zurück. 1= Überlauf	
	<pre>var = TimerGetFlag(2); //0 / 1=Überlauf</pre>	

## 8.2.1 4 x General Purpose Counter

	<p>Nr: 2..5 Einer von 4 Countern Pin:"PA0" ... Fest zugeteilte Zählpins: PA0, PA1, PA6, PA7, PB6, PB7 U<sub>pDn</sub>: 0, 1 0: Aufwärts-, 1 Abwärtszähler IRQPrio: 0, 1..15 Interruptpriorität; 0 = Off, 1 höchste, 15 tiefste.</p> <p>Aktiviert den Counterclock und zählt die Pinimpulse. Bei Überlauf und wenn IRQPrio &gt; 0 löst jeder Counter seinen eigenen Interrupt aus.</p>		
<b>CounterInit</b> (2..5, "Pin", U <sub>pDn</sub> 0/1, IRQPrio0..15);	<pre>// Counter 2 &lt;-PA1, UP ohne Interrupt CounterInit(2, "PA1", 0, 0);</pre>		
<b>CounterPutCount</b> (2..5, ival);	<pre>CounterInit(2, "PA0", 0, 3); // Prio 3 Int_val: 0..65535. Schreibt den übergebenen 16Bit-Wert in den Zähler 2..5 CounterPutCount(2, 0); // Count = 0</pre>		
int <b>CounterGetCount</b> (2..5);	<pre>Liefert einen 16Bit Zählwert zurück ivar = CounterGetCount(2); // return count</pre>		

<b>USARTInit</b> 1/2, " IRQPrio0..15);	Siehe unter USART	 TIM2_... TIM5_IRQHandler USART1_ USART2_IRQHandler
--	-------------------	--

## Interrupt Handler (ACHTUNG: die Namen/Bezeichner sind vorgeschrieben)

<pre>// Beispiel void TIM2_IRQHandler(void) {     IRQClearFlag("T2");     // Immer zuerst:     // IRQClearFlag(..)     // dann IntrService Programm }</pre>	Siehe auch folgende Tabelle für das Handling der ISR
---	--

Interrupt Quelle	Namen der Service-Routinen ISR	IRQClearFlag Bezeichner
Timer/Counter-IRQ:	TIM2_IRQHandler ... TIM5_IRQHandler	-> IRQClearFlag ("T2") ... ("T5")
USART-IRQ:	USART1_IRQHandler, USART2_IRQHandler	-> IRQClearFlag ("U1"), ("U2")
Ext. IRQ 0..4:	EXTI0_IRQHandler... EXTI4_IRQHandler	-> IRQClearFlag ("PA0") ... ("PE4")
Ext. IRQ 5..9:	EXTI9_5_IRQHandler (gemeinsam)	-> IRQClearFlag ("PA5") ... ("PE9")
Ext. IRQ 10..15:	EXTI15_10_IRQHandler (gemeinsam)	-> IRQClearFlag ("PA10") ... ("PE15")



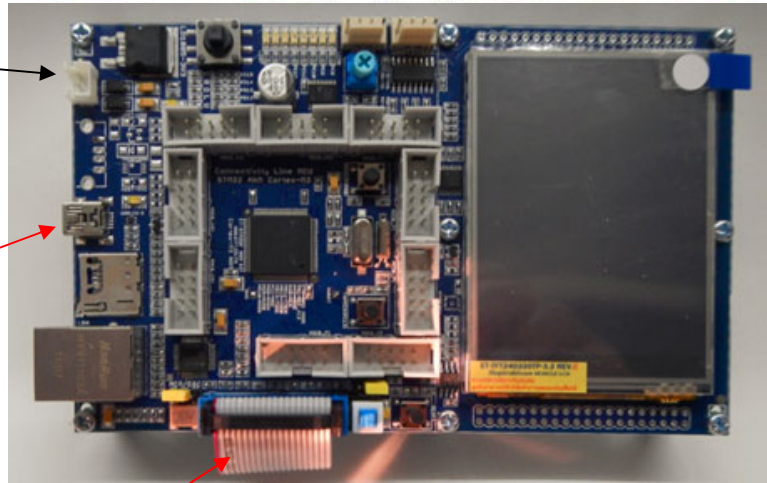
## 10 Anhang Anschlüsse am µC-Board MCB32

### Entwicklungsanschlüsse

Fremdspeisung  
genau 5V! oder ...

USB Speisung (**roter Stecker**)

USB - ST-Link (**schwar-  
zer** USB Stecker)  
Zielsystemdebugging,  
Boardspeisung  
wie oben



Bootloadschalter: Ungedrückt lassen **LED OFF**

**Reset**-Taster: Programmrestart

Digitale Ein- und Ausgaben am µC-Board MCB32. **ACHTUNG** mit Potentiometer (Pot)

P1: 8x onboard LEDs  
parallel zu 8x extern LEDs

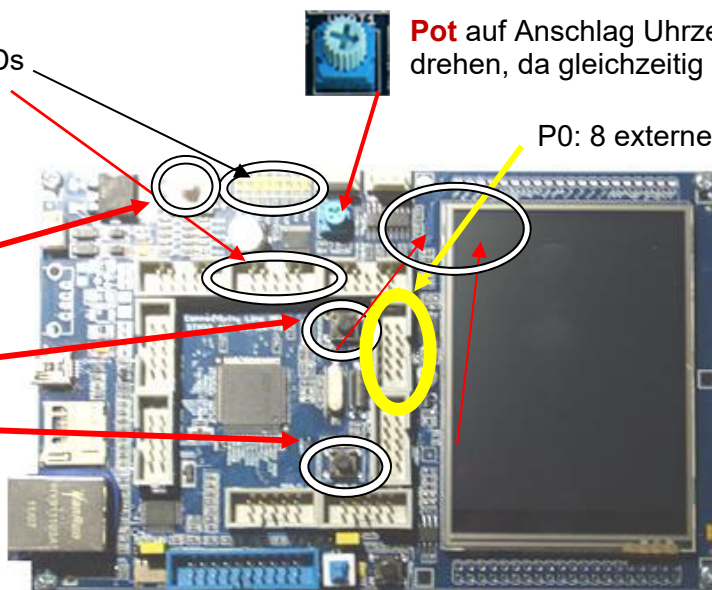
**Pot** auf Anschlag Uhrzeigersinn  
drehen, da gleichzeitig P0\_4

P0: 8 externe Schalter

ControlStick

Button 1

Button 0



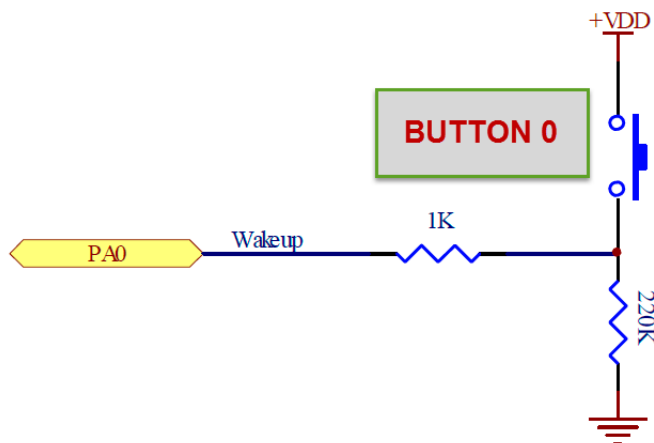
// In TouchP0P1.h definierte Pin-Bezeichnungen PA\_0 .. PD\_11, ohne Bezeichner wie Button .. !

```
char Button0    = PA_0;           // Bitwert 1/0, aktiv low, prellt wenig
char Button1    = PC_13;

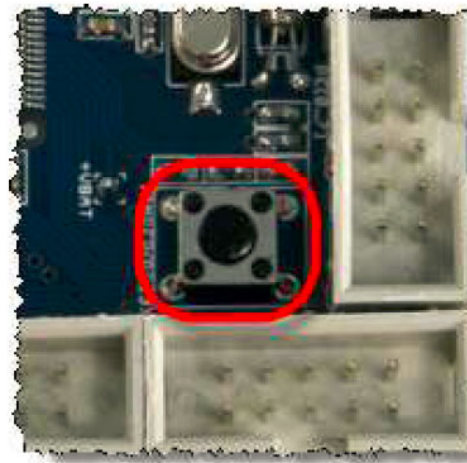
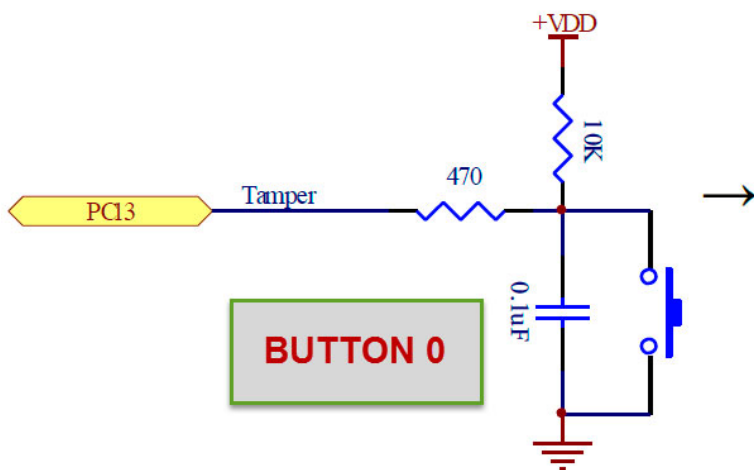
char Stick      = PD_High;        // als Byte 0xF8 open, aktiv low, alle entprellt
char StickSelect = PD_15;        // Bitwert 1/0; Bytewert 0x80
char StickDown  = PD_14;        //          1/0;          0x40
char StickLeft  = PD_13;        //          1/0;          0x20
char StickUp    = PD_12;        //          1/0;          0x10
char StickRight = PD_11;        //          1/0;          0x08
```



## Button 0 (PA 0)

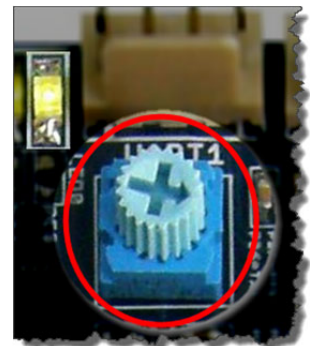
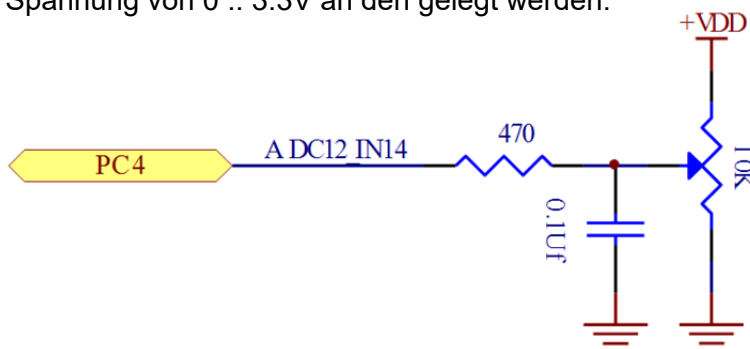


## Button 1 (PC 13)



## Potentiometer (PC 4) // resp. P0\_4 (Library)

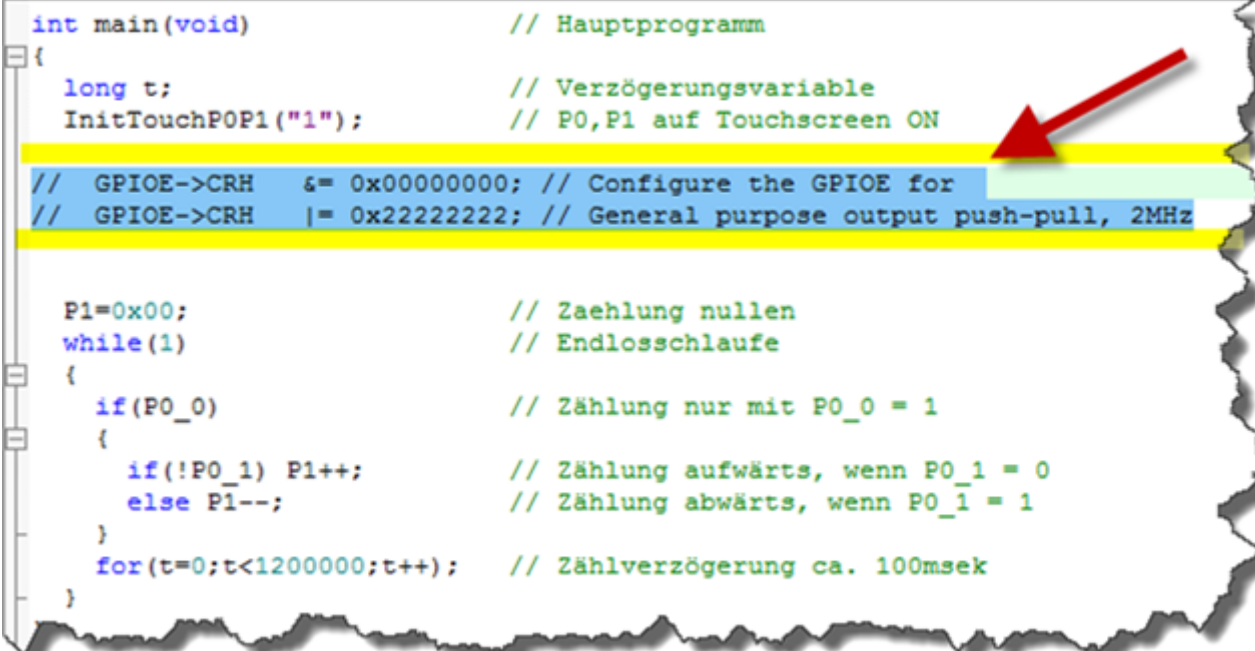
Wenn der Port PC4 als Analog-Input (AD-Wandler) geschaltet ist kann mit dem Potentiometer eine Spannung von 0 .. 3.3V an den gelegt werden.



## LED auf Board aktivieren

Mit den folgenden Befehlen wird Port PE[8..15] so gesetzt dass die LEDs auf dem Board auch angesteuert werden. Siehe Beispiel Code weiter unten.

```
GPIOE->CRH   &= 0x00000000;    // Configure the GPIOE for
GPIOE->CRH   |= 0x22222222;    // General purpose output push-pull, 2MHz
```



```
int main(void)                // Hauptprogramm
{
    long t;                    // Verzögerungsvariable
    InitTouchPOP1("1");        // P0,P1 auf Touchscreen ON

    // GPIOE->CRH   &= 0x00000000; // Configure the GPIOE for
    // GPIOE->CRH   |= 0x22222222; // General purpose output push-pull, 2MHz

    P1=0x00;                    // Zaehlung nullen
    while(1)                    // Endlosschleife
    {
        if(P0_0)                // Zählung nur mit P0_0 = 1
        {
            if(!P0_1) P1++;      // Zählung aufwärts, wenn P0_1 = 0
            else P1--;           // Zählung abwärts, wenn P0_1 = 1
        }
        for(t=0;t<1200000;t++); // Zählverzögerung ca. 100msek
    }
}
```

## 11 Anhang: Referenzen [3]

- [1] <http://rn-wissen.de/Interrupt>, «<http://rn-wissen.de>,» 30 10 2017. [Online]. Available: <http://rn-wissen.de/wiki/index.php?title=Interrupt>. [Zugriff am 30 10 2017].
- [2] ST, «ARM\_STM\_Reference manual\_V2014\_REV15,» ST, 2014.
- [3] R. Weber, «Projektvorlagen (div) MCB32,» 2013ff.
- [4] J. Yiu, The definitive Guide to ARM Cortex-M3 and M4 Processors, 3 Hrsg., Bd. 1, Elsevier, Hrsg., Oxford: Elsevier, 2014.
- [5] R. Jesse, Arm Cortex M3 Mikrocontroller. Einstieg und Praxis, 1 Hrsg., [www.mitp.de](http://www.mitp.de), Hrsg., Heidelberg: Hütigh Jehle Rehm GmbH, 2014.
- [6] Diller, «System Timer,» 06 07 2014. [Online]. Available: [http://www.diller-technologies.de/stm32.html#system\\_timer](http://www.diller-technologies.de/stm32.html#system_timer). [Zugriff am 06 07 2014].
- [7] A. Limited, «DDI0337E\_cortex\_m3\_r1p1\_trm.pdf,» ARM Limited, [http://infocenter.arm.com/help/topic/com.arm.doc.ddi0337e/DDI0337E\\_cortex\\_m3\\_r1p1\\_trm.pdf](http://infocenter.arm.com/help/topic/com.arm.doc.ddi0337e/DDI0337E_cortex_m3_r1p1_trm.pdf), 2005, 2006 ARM Limited.
- [8] A. C. Group, «[http://www.vr.ncue.edu.tw/esa/b1011/CMSIS\\_Core.htm](http://www.vr.ncue.edu.tw/esa/b1011/CMSIS_Core.htm),» 2007. [Online].
- [9] E. Malacarne, *Glossar Malacarne*, V11 Hrsg., Rüti: Cityline AG, 2014.
- [10] R. Weber, «General Purpos Input Output,» 2014.
- [11] STM, «STM32F10x Standard Peripherals Firmware Library,» STM, 2010ff.
- [12] E. Schellenberg und E. Frei, «Programmieren im Fach HST,» TBZ, Zürich, 2010ff.
- [13] R. Weber, «Robert Weber / Berufsschulzentrum Uster / Projektvorlagen MCB32 (div),» Uster, 2015.