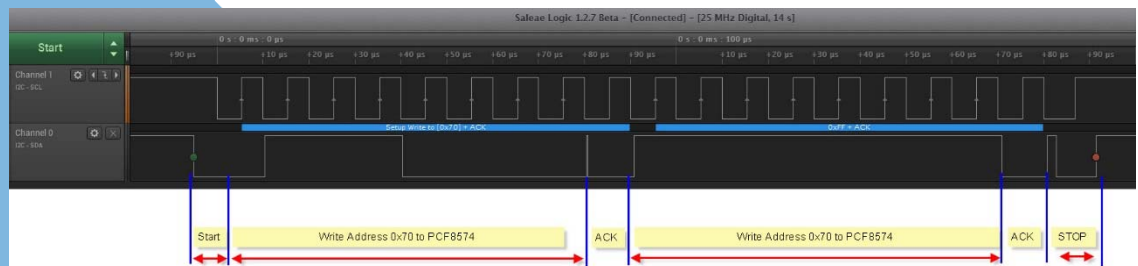




Mikrocontroller MCB32

I2C Bus: Anwendung Sensor BME280



MCB32 - Embedded Programmierung I2C und Sensor BME280

Version: 2107.01

Diese Dokumentation kann ohne Vorankündigung jederzeit angepasst, verbessert und erweitert werden.
Wünsche und Fehler an: info@mcb32.ch

1 Inhalt

| | | |
|-----------|---|-----------|
| 1 | Inhalt | 2 |
| 2 | Gefahren Portbeschaltung | 4 |
| 3 | Details I2C / BME280 mit MCB32 | 4 |
| 3.1 | Beschreibung BME280 | 4 |
| 3.2 | Kalibration | 5 |
| 3.3 | Auflösung und Berechnung der Messwerte | 5 |
| 3.4 | I2C Adresse des BME280 | 6 |
| 3.5 | Power ON (PON) BME280 und seine Betriebsarten | 6 |
| 3.6 | Betriebsarten | 7 |
| 3.7 | Register Map BME280 | 7 |
| 4 | Treiber | 8 |
| 5 | I2C Bus | 9 |
| 5.1 | Bit transfer | 9 |
| 5.2 | Start and stop conditions | 9 |
| 5.3 | Acknowledge (ACK) | 9 |
| 5.4 | Write Mode | 10 |
| 5.5 | Hardwarefunktionen MCB32 | 11 |
| 5.5.1 | 5x GPIO General Purpose Input Output | 11 |
| 5.5.2 | 2 x ADC Analog Digital Converter (0...3.3V, 8Bit Auflösung) | 11 |
| 5.5.3 | 2 x DAC Digital Analog Converter (0...3.3V, 8Bit Auflösung) | 11 |
| 5.6 | Serielle Schnittstellen [2] | 12 |
| 5.6.1 | USART1 & USART2 | 12 |
| 5.7 | Interrupt Funktionen | 13 |
| 5.7.1 | 4 x Externe Interrupt Requests (Vereinfacht, jede pinNr nur 1x) | 13 |
| 5.7.2 | 6 x Interne Interrupt Requests | 13 |
| 5.7.3 | Interrupt Handler (ACHTUNG: die Namen/Bezeichner sind vorgeschrieben) | 13 |
| 6 | Grafikprogrammierung | 14 |
| 6.1.1 | Fonts | 14 |
| 6.1.2 | Grafikfunktionen [3] | 14 |
| 6.1.3 | Touch-Funktionen | 15 |
| 6.1.4 | Touchscreen Textfunktionen | 15 |
| 6.2 | Farbliste | 16 |
| 6.3 | Musterprogramm für Grafikfunktionen | 17 |
| 7 | Übung I2C | 18 |
| 7.1 | Auftrag | 18 |
| 8 | Anhang Grafikhardware | 20 |
| 8.1 | Hintergrund | 20 |
| 8.2 | Hardwarenahe Beschreibung der Displayansteuerung | 20 |
| 9 | Anhang: Umstellung von C51-Code auf ARM32-Code | 21 |
| 9.1 | Wichtig für das Funktionieren neuer Projekte | 21 |
| 10 | Anhang Touchscreen Kontrolle am µC-Board MCB32 | 22 |
| 10.1.1 | Touchscreen Kontrolle aus dem Quellcode | 22 |
| 11 | Anhang Anschlüsse am µC-Board MCB32 | 23 |
| 11.1.1 | STLINK, Schalter, Potentiometer, P0, P1 | 23 |
| 11.1.2 | Button 0 / Wakeup (Pin:PA0); nicht gedrückt PA_0=0 | 25 |
| 11.1.3 | Button 1 / Tamper (Pin:PC13); nicht gedrückt PC_13=1 | 25 |
| 11.1.4 | Potentiometer (PC4) // resp. P0_4 (Library) | 25 |
| | 25 | |
| | 25 | |
| 11.1.5 | LED von Port P1 auf Board aktivieren | 26 |
| 11.1.6 | Übersicht über die Hardwarestruktur eines Pins | 26 |
| 11.2 | Port PA Pin 0..7 | 26 |
| 11.2.1 | Steckerbelegung für 10pol. Stecker PA[0..7] | 27 |
| 11.2.2 | Original Belegung PA[0..7] | 27 |
| 12 | Anhang: Interrupt Vektorliste und Servicefunktionsaufrufe | 28 |
| 13 | Anhang: SysTick Timer | 30 |



| | | |
|----|------------------------------------|----|
| 14 | Anhang: Port Pin Liste MCB32 | 30 |
| 15 | Referenzen | 32 |

2 Gefahren Portbeschaltung

Achtung: Die Ports des MCB32 dürfen nicht mit mehr als 3,3V beschaltet werden. Falsche Handhabung führt zur Zerstörung des Kontrollers. Mit einem geeigneten Treiberbaustein/ Levelshifter oder mit Schutzwiderständen kann dieses Problem umgangen werden.

3 Details I2C / BME280 mit MCB32

3.1 Beschreibung BME280

Der BME280 ist ein integrierter Umgebungs-sensor, der speziell für mobile Anwendungen entwickelt wurde, bei denen Größe und geringer Stromverbrauch wichtige Designbedingungen sind. Das Gerät kombiniert individuelle hochlineare und hochgenaue Sensoren für Druck, Feuchte und Temperatur in einem 8-poligen Metalgehäuse.

Niedrige Stromaufnahme (3,6 μ A @1Hz), Langzeitstabilität und hohe EMV-Robustheit entwickelt sind weitere Merkmale. Der Feuchtigkeitssensor zeichnet sich durch eine extrem schnelle Reaktionszeit aus, die die Leistungsanforderungen für neuartige Anwendungen unterstützt, wie z.B.: Kontexterkenkung und hohe Genauigkeit über einen weiten Temperaturbereich. Der Drucksensor ist ein absoluter barometrischer Drucksensor, der sich durch eine aussergewöhnlich hohe Genauigkeit und Auflösung bei sehr geringem Rauschen auszeichnet.

Der integrierte Temperatursensor wurde für sehr geringes Rauschen und hohe Auflösung optimiert. Es wird hauptsächlich zur Temperaturkompensation der Druck- und Feuchtesensoren eingesetzt und kann zur Messung der Umgebungstemperatur verwendet werden.

Der BME280 unterstützt zudem unterschiedlichste Betriebsarten um das Gerät hinsichtlich Stromverbrauch, Auflösung und Filterleistung zu optimieren.

| BME280 Technical data [1] | |
|---|--|
| Package dimensions | 8-Pin LGA with metal 2.5 x 2.5 x 0.93 mm ³ |
| Operation range (full accuracy) | Pressure: 300 ... 1100 hPa Temperature: -40 ... +85 °C |
| Supply voltage V _{DDIO} Supply voltage V _{DD} | 1.2 ... 3.6 V 1.7 ... 3.6 V |
| Interface | I ² C and SPI |
| Average current consumption (typ.) (1Hz data refresh rate) | 1.8 μ A @ 1 Hz (H, T) 2.8 μ A @ 1 Hz (P, T) 3.6 μ A @ 1 Hz (H, P, T) T = temperature |
| Average current consumption in sleep mode | 0.1 μ A |
| Humidity sensor Response time ($\tau_{63\%}$) Accuracy tolerance Hysteresis | 1 s ± 3 % relative humidity ≤ 2 % relative humidity |
| Pressure sensor RMS Noise Sensitivity Error Temperature coefficient offset | 0.2 Pa (equiv. to 1.7 cm) ± 0.25 % (equiv. to 1 m at 400 m height change) ± 1.5 Pa/K (equiv. to ± 12.6 cm at 1 °C temperature change) |

3.2 Kalibration

Der BME280 misst mit Hilfe eines AD-Wandlers seine Sensoren. Jedes Sensorelement verhält sich unterschiedlich und nichtlinear. Die Daten für die Kalibration der Messwerte sind im BME280 abgespeichert. Diese Daten müssen vor der Verarbeitung gelesen werden. Danach stehen sie für die Berechnung der Messwerte zur Verfügung. Es ist darauf zu achten, dass diese Kalibrier-Daten nach jedem Reset / PON gelesen werden. Damit werden immer die für den aktuellen Sensor gültigen Werte zur Berechnung der Messwerte verwendet.

Die tatsächlichen Druck- und die tatsächliche Temperatur-Werte müssen dann mit Hilfe dieses Satzes von Kalibrierparametern berechnet werden. Die empfohlene Berechnung erfolgt mit Festpunktarithmetik und ist Bosch Manual des BME280 beschrieben. Jedes Kompensationsword vom NVM (non volatile memory) ist 16Bit signed oder unsigned im 2er Komplement. Siehe Tabelle im Bosch Manual.

3.3 Auflösung und Berechnung der Messwerte

Die Auflösung beträgt für die Feuchte 16Bit, die Temperatur 20Bit und für den Luftdruck 20Bit. Die Berechnung der realen Messwerte (Temp, Hum, Press) passiert im Empfänger der Daten (uP, MCU oder PC usw). Die Berechnung basiert auf den Messwerten vom AD-Wandler, den Kalibrationsdaten aus dem Sensor und den Vorschriften (Algorithmen) welche Bosch angibt.

Soll ein Messwert nicht gelesen werden kann er übersprungen werden (Skipped).

Das wird im Oversamplingregister angegeben (skipped) :

Bsp: im ctrl_meas Register 0xF4 wird: osrs_p[2:0] =000 → der Druck wird nicht gemessen.

3.4 I2C Adresse des BME280

Der Sensor wird am MCB32 am I2C Bus mit 100kHz Clock (400kHz ist möglich) betrieben.

Die 7-Bit-Geräteadresse lautet 111011x (also 0x76, 0x77) welche nach links geschiftet im Sendebyte verpackt werden muss. Das letzte Bit ist durch den SDO-Wert (Leitung am BME280) definiert und kann während des Betriebs geändert werden. Das Verbinden von SDO mit GND führt zu der Slave-Adresse 1110110 (0x76); das Verbinden mit VDD führt zu der Slave-Adresse 1110111 (0x77). Der SDO-Pin kann nicht potentialfrei gelassen werden; wenn er potentialfrei bleibt, ist die I²C-Adresse undefiniert.

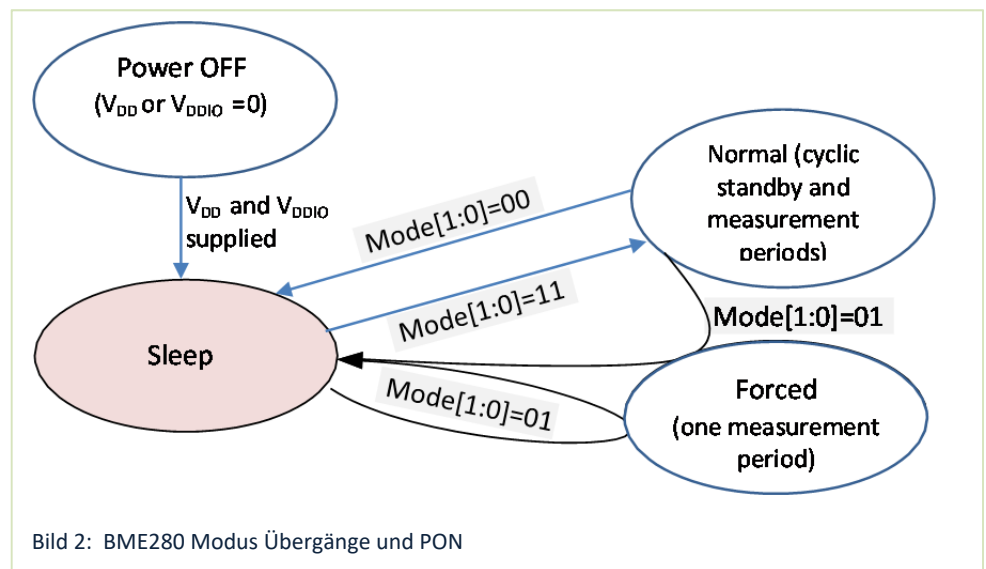
Siehe dazu nächstes Bild. (ACKS = Acknowledge by slave)



3.5 Power ON (PON) BME280 und seine Betriebsarten

Die unterstützten Betriebsarten (Mode) sind in Bild 2 dargestellt. Nach PON ist der Sensor im Sleep Modus.

Wenn das Gerät eine Messung durchführt, wird die Ausführung von Befehlen für das Wechseln der Betriebsart bis zum Ende der aktuell laufenden Messperiode verzögert. Weitere „Moduswechselbefehle“ oder andere Schreibbefehle in das Register ctrl_hum werden ignoriert, bis der „Moduswechselbefehl“ ausgeführt wurde.



3.6 Betriebsarten

Die Betriebsarten sind im Register 0xF4 abgelegt.

| Register 0xF4 | | |
|---------------|-------------|---|
| mode[1:0] | Mode | |
| 00 | Sleep mode | Single measure, Sensors goes Sleep after measure. |
| 01 | Forced mode | |
| 10 | Forced mode | Perpetual Mode based on Settings |
| 11 | Normal mode | |

Bosch empfiehlt ja nach Anwendung unterschiedliche Betriebsarten. Siehe folgende Tabelle.

| | Wettermonitoring | Feuchte Messung | Haus (Indoor) Anwendung |
|---|--------------------------------|-------------------|--|
| Datenrate | tief 1/60Hz | tief 1Hz | hoch 25Hz |
| Mode | Forced or Normal 1 Sample /Min | Forced 1 Sample/s | Normal 25 Samples/s |
| T Standby | | | 0.5ms |
| Filter IIR | OFF | OFF | ON filtercoeff16 |
| Oversampling | press * 1 | press * 0 (Skip) | press * 16 |
| | temp * 1 | temp * 1 | temp * 2 |
| | hum * 1 | hum * 1 | hum * 1 |
| Sample Code for Setup: <code>struct bme280_dev dev;</code> | | | dev.settings.osr_h = BME280_OVERSAMPLING_1X; dev.settings.osr_p = BME280_OVERSAMPLING_16X; dev.settings.osr_t = BME280_OVERSAMPLING_2X; dev.settings.filter = BME280_FILTER_COEFF_16; |
| Current Consumption | 0.16uA | 2.9uA | 633uA |

3.7 Register Map BME280

Die folgende Tabelle gibt eine Übersicht über die verschiedenen Register des BME280.

| BME 280 Sensor Register Tabelle | | | | | | | | | | | |
|---------------------------------|-------------|------------------|------|------|-------------|--------------|-------------|-------------|------|-------------|------|
| Register Name | Address | bit7 | bit6 | bit5 | bit4 | bit3 | bit2 | bit1 | bit0 | Reset state | |
| hum lsb | 0xFE | hum lsb<7:0> | | | | | | | | 0x00 | |
| hum msb | 0xFD | hum msb<7:0> | | | | | | | | 0x80 | |
| temp xlsb | 0xFC | temp xlsb<7:4> | | | | 0 | 0 | 0 | 0 | 0x00 | |
| temp lsb | 0xFB | temp lsb<7:0> | | | | | | | | 0x00 | |
| temp msb | 0xFA | temp msb<7:0> | | | | | | | | 0x80 | |
| press xlsb | 0xF9 | press xlsb<7:4> | | | | 0 | 0 | 0 | 0 | 0x00 | |
| press lsb | 0xF8 | press lsb<7:0> | | | | | | | | 0x00 | |
| press msb | 0xF7 | press msb<7:0> | | | | | | | | 0x80 | |
| config | 0xF5 | t sb[2:0] | | | filter[2:0] | | | spi3w en[0] | | 0x00 | |
| ctrl meas | 0xF4 | osrs t[2:0] | | | osrs p[2:0] | | | mode[1:0] | | 0x00 | |
| status | 0xF3 | measurin | | | | im update[0] | | | 0x00 | | |
| ctrl hum | 0xF2 | | | | | | osrs h[2:0] | | | | 0x00 |
| calib26..calib41 | 0xE1...0xF0 | calibration data | | | | | | | | individual | |
| reset | 0xE0 | reset[7:0] | | | | | | | | 0x00 | |
| id | 0xD0 | chip id[7:0] | | | | | | | | 0x60 | |
| calib00..calib25 | 0x88...0xA1 | calibration data | | | | | | | | individual | |

Reservierte Register sind nicht angegeben

| | | | | | | | |
|-----------|--------------------|------------------|-------------------|----------------|------------------|-----------|------------|
| Register: | Reserved registers | Calibration data | Control registers | Data registers | Status registers | Chip ID | Reset |
| Type: | do not change | read only | read / write | read only | read only | read only | write only |

4 Treiber

Bosch stellt eine vollständige Bibliothek für die Messung mit dem BME280 zur Verfügung. Diese Lib kann für das MCB32 einfach integriert werden.

```
1  /**
2  * @file      BME280_I2C.c
3  * @author    rma / Cityline AG
4  * @date      2.4.2018
5  * @version   1.1
6  * @note      BME280 I2C Prototyp
7  * @note      BME280 chip from Bosch
8  * @note      Remapping of I2C bus: YES
9  *            default: PB6 - I2C1_SCL; PB7 - I2C1_SDA
10 *            remap:   PB8 - I2C1_SCL; PB9 - I2C1_SDA
11 * @link      https://github.com/BoschSensortec/BME280_driver
12 * @note
13 *****/
14
15 #include <stm32f10x.h>           // uC-Typ bei uVision4
16 #include "TouchPOPl.h"         // P0-,Pl-,Touchscreen
17 #include "i2c.h"
18 #include "delays_cz.h"
19 #include <stdio.h>              // lib for sprintf .....
20 #include <string.h>
21 #include <stdlib.h>
22 #include "global_directives.h" // used for conditional compiling
23 #include "..\I2C_BME280\bme280_CZ.h" // BOSCH Files from BOSCH Github
24
```

Bild 3: Einbindung BME280 Lib Bosch

5 I2C Bus

Der I2C Bus ist ein 2 Leitungs- und 2 Wege-Bus für die Kommunikation zwischen verschiedenen Chips auf einem Bus. Die 2 Leitungen sind die SDA (serial data line) und die SCL Leitung (serial clock line).

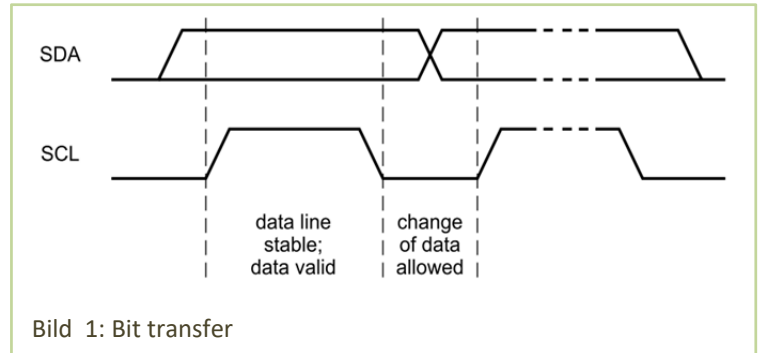
Beide Leitungen müssen mit einem Pull-Up Widerstand an VCC angeschlossen werden.

Ein Datentransfer kann nur durchgeführt, initiiert werden wenn der Bus nicht beschäftigt (Busy) ist.

5.1 Bit transfer

Ein Datenbit wird während einem Clockpuls transferiert. Die Daten auf der SDA Leitung müssen während der High-Phase des Clocks stabil bleiben um als Daten akzeptiert zu werden. Andernfalls werden die Änderungen als Kontrollsignale interpretiert.

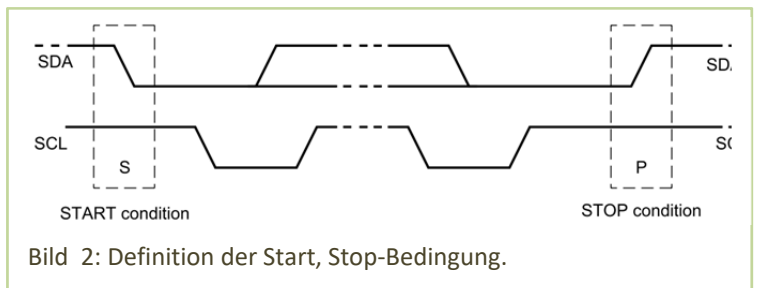
Siehe Bild 1: Bit transfer



5.2 Start and stop conditions

Beide Leitungen (SDA, SCL) bleiben High wenn der Bus nicht beschäftigt ist. Ein High zu Low- Übergang (Transition) während dem der Clock High ist wird als START-Bedingung (start condition (S)) bezeichnet. Ein Low zu High-Übergang wird als Stop-Bedingung bezeichnet (stop condition (P)).

Siehe Bild 2: Definition der Start, Stop-Bedingung..



5.3 Acknowledge (ACK)

Die Anzahl der Datenbytes, welche zwischen einer Start und Stopp-Bedingung gesendet werden ist nicht limitiert. Jedes Byte wird von einem ACK abgeschlossen. Siehe Bild 3: Acknowledgment on the I2C-bus...

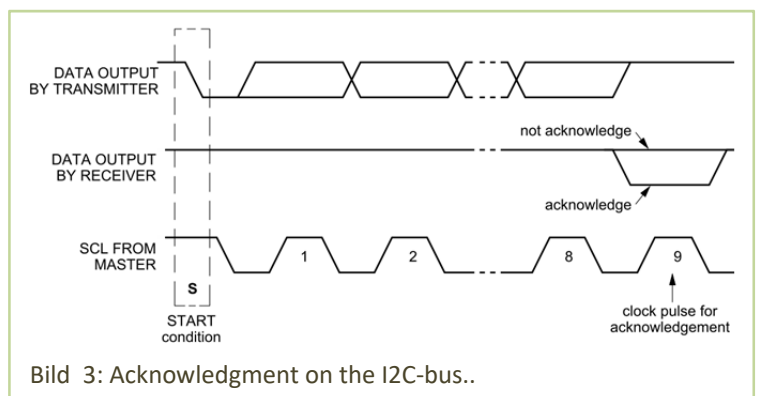
Das ACK Bit ist ein HIGH Pegel welcher vom Transmitter-Chip währenddem der Master einen geeigneten Clock Puls generiert.

Ein Slave-Empfänger, welcher adressiert wird, muss nach dem Empfang jedes Bytes ein ACK erzeugen.

Auch ein Master generiert ein ACK nach dem Empfang eines Bytes welches vom Slave Transmitter gesendet (clocked) wurde.

Der Chip, welcher das ACK-Signal erzeugt muss die SDA Leitung während dem ACK-Clock auf GND ziehen (pull down). Die SDA Leitung muss während dem ACK-clock-pulse stabil sein.

Ein Master-Receiver muss das Ende der Daten gegenüber dem Transmitter signalisieren indem er nach dem Aussenden des letzten Bytes, welches vom Slave stammt, kein ACK erzeugt. In diesem Zustand muss der Transmitter die Datenleitung auf HIGH lassen um dem Master das Erzeugen der Stopp-Bedingung zu ermöglichen.



5.4 Write Mode

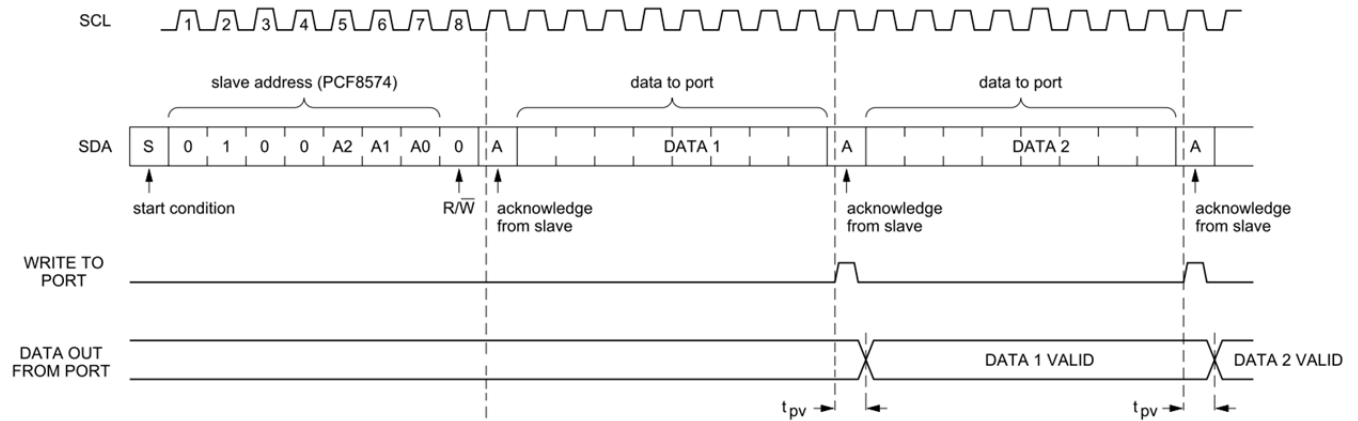


Bild 4: Write Modus

5.5 Hardwarefunktionen MCB32

Einfache Nutzung der integrierten, peripheren Funktionen des MCB32 ohne Registerkenntnisse. Zu beachten ist die Grundkonfiguration (Default nach Reset) der Ports:

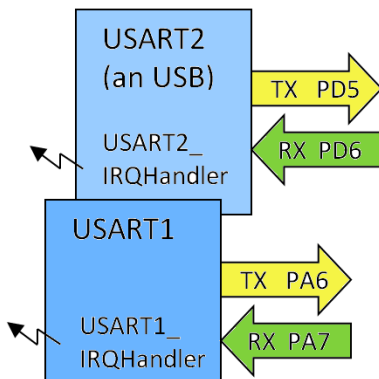
- **IN:** floating 3.3v
- **Out:** Open Drain wegen Kurzschlussgefahr. **Nicht 5 V tolerant**

| Peripheriefunktionen: | | | Version und Stand: 2107.01 |
|--|---|--------------------------|--------------------------------|
| Funktion | Beschreibung / Beispiel | Parameter | Hinweis zur Hardware |
| 5.5.1 5x GPIO General Purpose Input Output | | | |
| <code>GPIOInit("PxH/L", "yyyyyyyy");</code> | Initialisiert den entsprechenden Port (A-E) y: 0 Input (Pull Up) y: 1 Output (50MHz, open Drain) H/L High oder LOW-Byte vom Port x | x:A-E | <p>Anordnung 10pol Stecker</p> |
| <code>GPIOInit("PAL", "1111000"); // 4Out,4In</code> | | | |
| <code>GPIOPutByte("PxH/L", Byte);</code> | Siehe bei <code>GPIOInit("PxH/L", "yyyyyyyy");</code> Schreibe 0xAA auf Port E, High Byte: <code>GPIOPutByte("PEH", 0xAA); // 0xAA->PEH</code> | x:A-E | |
| <code>char GPIOGetByte("PxH/L");</code> | Liest den Wert von Port A vom L -Byte und speichert in Var. <i>char</i> . <code>b = GPIOGetByte("PAL"); // PAL->var b</code> | x:A-E | |
| 5.5.2 2 x ADC Analog Digital Converter (0...3.3V, 8Bit Auflösung) | | | |
| Achtung. Diese einfachen Funktionen unterstützen nur den 8Bit Betrieb. Für 12Bit Betrieb sowie mehr Features bitte APP-Note oder STM32F107 Referenz-Manual studieren. | | | |
| <code>ADCInit(1/2, "Pin");</code> | Wählt einen von 16 möglichen Pins als Eingang und einer der beiden ADC als Wandler. Startet den ADC im kontinuierlichen Betrieb. <code>ADCInit(1, "PC4"); // ADC1 an PC4</code> | Pin: siehe rechts | |
| <code>char ADCGetVal(1/2);</code> | Liest den gewählten Pin via ADC-Kanal 1 oder 2 ein und liefert den 8Bit Wert zur Variablen (char). <code>var = ADCGetVal(1); // AnaLog von PC4</code> | | |
| | | | Version und Stand: 2107.01 |
| 5.5.3 2 x DAC Digital Analog Converter (0...3.3V, 8Bit Auflösung) | | | |
| Je 1 Digital_Analog-Kanal auf je 1 Leitung ausgehen. Ausgänge sind PA4 und PA5. | | | |
| <code>DACInit(1/2);</code> | Wählt entweder Kanal1 (PA4) oder Kanal2 (PA5) für die Ausgabe des analogen Signales aus. Die Werte werden direkt nach dem Laden desselbigen ausgegeben. Das heisst in dieser einfachen Version werden keine Trigger unterstützt. <code>DACInit(1); // DAC1 an PA4</code> | Pin: siehe rechts | |
| <code>DACPutVal(1/2, 8Bit-Wert);</code> | Gibt den 8Bit-Wert (0..0xFF) auf DAC 1 aus. 12Bit Wert müssen direkt programmiert werden (siehe dazu Applikation – Note DAC) <code>DACPutVal(1, 100); // AnaLog Out PA4</code> | | |

5.6 Serielle Schnittstellen [2]

2 Schnittstellen für die RS232-Datenübertragung sind als echte RS232 Leitungen vorhanden. Das heisst die Signale werden von einem Treiber auf die Standardpegel (+/-15V gebracht, 0=+15V, 1=-15V).

Die Baudraten sind nach einem Reset und einem USARTInit() nicht bei beiden Schnittstellen gleich (siehe unten). USART1 hat 19200Bd und USART2 hat 9600Bd.

| Peripheriefunktionen: | | | Version und Stand: 2107.01 |
|---|---|--|---|
| Funktion | Beschreibung / Beispiel | Parameter | Hinweis zur Hardware |
| 5.6.1 USART1 & USART2 | | | |
| 2x USART | USART1 TX-> PB6, RX <- PB7 (via Remapping) USART2 TX-> PD5, RX <- PD6 | | |
| Universal Synchronous Asynchronous Receiver Transmitter USART2 ist geeignet für Kommunikation mit PC; | | | |
| default USART1: 19200Bd, 1,8,1,n //PCLK2 mit 72MHz max (MCB32 nach Reset) | | | |
| default USART2: 9600Bd, 1,8,1,n //PCLK1 mit 32MHz max (MCB32 nach Reset) | | | |
| USARTInit (1/2, " IRQPrio0..15); | Nr: 1, 2 Einer von 2 USART | |  |
| | IRQPrio: 0, 1..15 Interrupt Priorität; 0 = AUS, 1 höchste, 15 tiefste | | |
| | Aktiviert den UART-Clock und Initialisiert auf 19200Bd resp. 9600, 1Start-, 8 Daten-, 1 Stopbit, No Parity. | | |
| | Bei IRQPrio >0 löst ein ankommendes Zeichen einen Interrupt aus. | | |
| USARTInit(2, 0); // USART2 ohne Intr. | | | |
| USARTWrite (1/2, 'char'); | Schreibt den Wert resp. das Zeichen ‚CHAR‘ via USART1 oder 2 auf den seriellen Bus. (Ev. Delay zw. den Zeichen) | | |
| char USARTRead (1 / 2); | Wartet bis ein Zeichen an USART 1 oder 2 eintrifft und gibt Wert via Var. char zurück. | | |
| char USARTtoRead (1/2); | c = USARTRead(2); // Warte auf Zeichen .. | | |
| | Gibt den Status des USART Kanales zurück. 1 = Zeichen eingetroffen. 0= kein Zeichen im Buffer. | | |
| | c = USARTtoRead(2); // 0/1 in c | | |
| Beispiel: neue Baudrate für USART 1 programmieren | | | |
| USARTInit(1,0); | | // Schalte USART1 mit 19200Bd ein (Default), ohne IR | |
| USARTInit(2,0); | | // Schalte USART1 mit 9600Bd ein (Default), ohne IR | |
| USART1->BRR = 0x1D4C; | | // USART1: 9600Bd @ 72MHz benötigt einen Teiler von 468.75 | |
| | | // siehe RefManual Page 792 | |

5.7 Interrupt Funktionen

Version und Stand: 2107.01

5.7.1 4 x Externe Interrupt Requests (Vereinfacht, jede pinNr nur 1x)

ExtIRQInit
("Pin", Flanke0/1/2,
IRQPrio0..15);

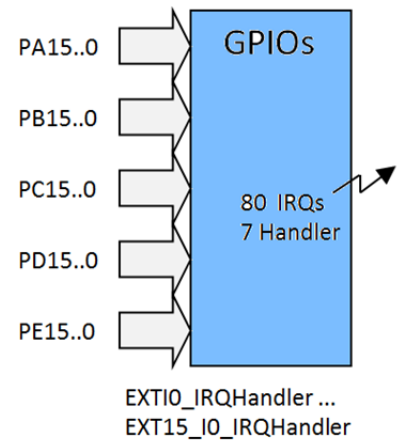
Pin[" "]: "PA0".."PE15"
Interrupt auslösender Pin

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 *EXTIO_IRQHandler()*..
EXTI15_10_IRQHandler().

// IRQ PA0,pos Flanke Priorität 4
ExtIRQInit("PA0",0,4);



5.7.2 6 x Interne Interrupt Requests

Priorität: 0= OFF, 1 = Höchste 15=Niedrigste

TimerInit
(2..5,n*100us,IRQPrio0..15);

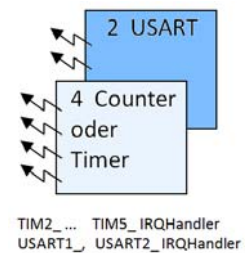
Siehe unter Timer

CounterInit
(2..5,"Pin",UD0/1,IRQPrio0..15);

Siehe unter Counter
CounterInit(2,"PA0",0,3); // Prio 3

USARTInit1/2, " IRQPrio0..15);

Siehe unter USART



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

```
// Beispiel
void TIM2_IRQHandler(void)
{
    IRQClearFlag("T2"); // Immer zuerst:
                        // IRQClearFlag(..)
                        // dann IntrService Programm
}
```

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: | EXTIO_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") |

6 Grafikprogrammierung

6.1.1 Fonts

Es steht 1 Font zur Verfügung. Ein „7*11“ ASCII Font.

Der Font kann in der Library „TouchGrafik.c“ individuell erweitert werden.

Achtung: Die Bildschirmkoordinaten sind 0,0 und 319,239.



Grafikfunktionen: Version und Stand: 2107.01


6.1.2 Grafikfunktionen [3]

| Funktion | Beschreibung | Parameter | Beispiel |
|---|---|------------------|---|
| InitTouchScreen () | Touchscreen ohne POP1 für Text, Grafik, Peripherie | - | <code>InitTouchScreen();</code> |
| InitTouchPOP1 ("0/1"); | Neben dem Display wird auch der Komfort für P0 und P1 initialisiert. Für Schulübungen mit Bitmanipulation. <i>Siehe Fehler! Verweisquelle konnte nicht gefunden werden. betreff SysTick_Handler().</i> | Siehe Kapitel 10 | Der Befehl: <code>InitTouchPOP1("1");</code> schaltet P0,P1 ein → |
| setScreenDir (DIR) | Setzt die Schreibrichtung des Displays | HOR und VER | <code>setScreenDir (HOR);</code> <code>setScreenDir (VER);</code> |
| char getScreenDir () | Gibt die Schreibrichtung zurück | HOR=0 und VER=1 | <code>if(getScreenDir()==VER)</code> <code>{clearScreen (BLUE);}</code> |
| clearScreen (color) | Löscht den Bildschirm mit der angegebenen Farbe. Funktioniert nur mit Einstellung setScreenDir(VER) . | long color | <code>clearScreen(BLACK);</code> |
| ACHTUNG: Bildschirm-Koordinaten fangen bei 0,0 an und enden bei 319,239. | | | |
| plotDot (X,Y,color) ○ | Zeichnet ein DOT an der Stelle X,Y mit der Farbe color. a,b,c unsigned int. | | <code>plotDot(120,120,WHITE);</code> |
| Circle (X,Y,Radius, Tick, Color, Fill) | Zeichnet einen Kreis an der Stelle X,Y mit dem Radius r. Die Kreislinie wird mit der Dicke „Tick: 0..100“ gezeichnet wenn Fill 0 ist. Wenn Fill =1 angegeben ist, so wird der Kreis gefüllt. | div | <code>circle(50,80,20,2,GREEN,0);</code> |
| ellipse (h,k,rx,ry,tick,color, fill) | h und k beschreiben den Mittelpunkt der Ellipse. rx und ry die Radien, Tick, Color und Fill wie beim Kreis, | | |
| rectan (x1,y1,x2,y2,tick,color, fill) | Zeichnet ein Rechteck von x1,y1 zu x2,y2. | | <code>rectan(100,150,140,180,1,RED,1);</code> |
| plotFilledRect (x1,y1,dx,dy,color) | Gefülltes Rechteck von x1, y1 nach x1+dx, y1+dy mit Farbe color | | <code>plotFilledRect (10, 20, 50, 60, RED);</code> |
| textxy (String,x,y,For_col, Back_Col) | Schreibt an der Stelle x,y, mit der Farbe For_col und der Hintergrund-Farbe den String. | | <code>textxy(" MCB32 Lib</code> <code>Version:", 2, 32,</code> <code>BLACK, YELLOW);</code> |
| line (x1,y1,x2,y2,thick,color) | Zeichne Linie von X1,y1 nach x2,y2 mit der Dicke und der Farbe | | <code>Line(5,110,315,110,2,WHITE);</code> |

Grafikfunktionen:

Version und Stand: 2107.01

6.1.3 Touch-Funktionen

| | | |
|-------------------------------|--|---|
| <code>getTSCxy ()</code> | Erfasst die x/y - Werte der berührten Position. | <code>getTSCxy();</code> |
| <code>getTSCx ()</code> | Gibt die x-Position der letzten Erfassung zurück. | <code>xPos = getTSCx();</code> |
| <code>getTSCy ()</code> | Gibt die y-Position der letzten Erfassung zurück. | <code>yPos = getTSCy();</code> |
| <code>getTSCtouched ()</code> | Touchscreenberührung: Rückgabe 0 / 1 0: unberührt 1: während Berührung  <i>Bemerkung: getTSCxy() je nach Fall zuerst ausführen.</i> | <pre>if (getTSCtouched ()) { }</pre> |

6.1.4 Touchscreen Textfunktionen

vertikal, 20 Zeilen à 30 Zeichen

horizontal, 15 Zeilen à 40 Zeichen

```
0: Text-, Variablenausgaben
1: -----

Variablenwerte dezimal:
0, -444, 1234567890

Variablenwerte binär:
1 Bit: 0,
8 Bit: 1010'1010
16 Bit: 1111'1000'1111'1000

Variablenwerte hexadezimal:
16 Bit: 0x2AFB

Textfarbenwechsel:
32 Bit: 1111'1000'1111'1000
      1111'1000'1111'1000
18:
19:
```

```
0: Text-, Variablenausgaben
1: -----

Variablenwerte dezimal:
0, -444, 1234567890

Variablenwerte binär:
1, 8, 16 Bit

32-Bit:
1111'1000'1111'1000:1111'1000'1111'1000

13:
14:
```

| Funktion | Beschreibung | Beispiel |
|---|--|--|
| <code>InitTouchScreen ();</code> | Initialisiert den Touchscreen ohne POP1 für Text, Grafik und Peripherie | <code>InitTouchScreen ();</code> |
| <code>setTextcolor (long color);</code> | Farbwechsel für nachfolgenden Text | <code>setTextcolor (WHITE);</code> |
| <code>print (char *txt);</code> | Schreibt Text hinter die letzte Position | <code>print ("Text");</code> |
| <code>println (char *txt);</code> | Schreibt Zeile hinter die letzte Position und springt an den nächste Zeilenanfang | <code>println ("Text");</code> <code>println ("");</code> |
| <code>printAt (char n, char *txt);</code> | Schreibt Text an den Anfang der Zeile mit Nummer n | <code>printAt (12, "Text");</code> |
| <code>printBin (char n, long num);</code> | Konstanten- und Variablenwerte im Binär-code wie 1111'0000 mit der Bitanzahl n | <code>printBin (8, 250);</code> <code>printBin (32, variable);</code> |
| <code>printHex (char n, long num);</code> | Konstanten- und Variablenwerte im Hex-code wie 0xFF00123E mit der Bitanzahl n | <code>printHex (8, 250);</code> <code>printHex (32, variable);</code> |
| <code>printDec (char form long num);</code> | Ganzzahlige Werte aller Typen mit Feldlänge und Vorzeichen in form - vorgegebene Feldlänge für Typ unsigned - vorgegebene Feldlänge mit Vorzeichen - wertabhängige Feldlänge, Typ unsigned - wertabhängige Feldlänge mit Vorzeichen da zu kurze Feldlängen erweitert werden | <code>printDec (12, variable);</code> <code>printDec (-8, 123456);</code> <code>printDec (1, variable);</code> <code>printDec (-1, -123456);</code> |

6.2 Farbliste

Die **nebenstehende** Farbliste zeigt die vordefinierten Farben. Weitere Farben müssen gemäss dem Muster:

RRRR`RGGG`GGGB`BBBB zusammengestellt werden.

Der 16 Bit Farbcode hat 32 Rot-, 64 Grün- und 32 Blauanteile

in Bit: **5** Bit **R**, **6** Bit **G**, **5** Bit **B**

RRRR`RGGG`GGGB`BBBB

Mischbeispiel:

long sattgrün = 63<<5; 0000`0111`1110`0000

long hellgrün = 15<<11 + 63<<5 + 15; 0111`1111`1110`1111

Die genauere Beschreibung befindet sich im ILI 9341 Manual.

| Definiere die Farbe für den Display: | | |
|--------------------------------------|---------|-------------------------|
| Color = RRRR`RGGG`GGGB`BBBB | | |
| Name | Color # | |
| #define no_bg | 0x0001 | // No Color Back Ground |
| #define BLACK | 0x0000 | |
| #define WHITE | 0xFFFF | |
| #define RED | 0x8000 | |
| #define GREEN | 0x0400 | |
| #define DARK_GREEN | 0x1C03 | // weber |
| #define BLUE | 0x0010 | |
| #define YELLOW | 0xFF00 | |
| #define DARK_YELLOW | 0x8403 | // weber |
| #define CYAN | 0x0410 | |
| #define MAGENTA | 0x8010 | |
| #define BROWN | 0xFC00 | |
| #define OLIVE | 0x8400 | |
| #define BRIGHT_RED | 0xF800 | |
| #define BRIGHT_GREEN | 0x07E0 | |
| #define BRIGHT_BLUE | 0x001F | |
| #define BRIGHT_YELLOW | 0xFFE0 | |
| #define BRIGHT_CYAN | 0x07FF | |
| #define BRIGHT_MAGENTA | 0xF81F | |
| #define LIGHT_GRAY | 0x8410 | |
| #define LIGHT_BLUE | 0x841F | |
| #define LIGHT_GREEN | 0x87F0 | |
| #define LIGHT_CYAN | 0x87FF | |
| #define LIGHT_RED | 0xFC10 | |
| #define LIGHT_MAGENTA | 0xFC1F | |
| #define DARK_GRAY | 0x4208 | |
| #define GRAY0 | 0xE71C | |
| #define GRAY1 | 0xC618 | |
| #define GRAY2 | 0xA514 | |
| #define GRAY3 | 0x630C | |
| #define GRAY4 | 0x4208 | |
| #define GRAY5 | 0x2104 | |
| #define GRAY6 | 0x3186 | |
| #define BLUE0 | 0x1086 | |
| #define BLUE1 | 0x3188 | |
| #define BLUE2 | 0x4314 | |
| #define BLUE3 | 0x861C | |
| #define CYAN0 | 0x3D34 | |
| #define CYAN1 | 0x1DF7 | |
| #define GREEN0 | 0x0200 | |
| #define GREEN1 | 0x0208 | |

6.3 Musterprogramm für Grafikfunktionen

Das folgende Programm zeigt die Möglichkeiten der Library.

(Änderungen jederzeit möglich. Siehe Dokumentation.)

```

/** @file grafikfunktionen_1.c
 * @brief Zeigt die grundlegenden Grafikfunktionen Version I von MCB32
 */
//=====Includes=====
#include <stm32f10x.h> // Mikrokontrollertyp
#include "TouchP0P1.h" // P0/P1,8Bit,Touchscreen und Grafik
#include <math.h> // lib für Sinus
#define PI 3.14159f // Konstante PI
//*****Implementation*****

int main(void) // Hauptprogramm
{
    long t; // Verzögerungsvariable
    float rad;
    unsigned char uc_val,color_toggle=0; // Hilfsvariablen;

    char LIBVer[]=dMCB32_LibVersion; // Option: Zeige Lib Version an
    InitTouchScreen(); // Init. der Display Hardware
    setScreenDir (HOR); // setze Richtung Display. 0,0 bei Resettaster
    textxy(" MCB32 Lib Version:", 2, 32, BLACK, YELLOW);
    textxy(LIBVer, 160, 32, WHITE, BLACK);
    printAt(2,"----- "); // Schreibe auf der 2ten Zeile den Text
    circle(50,80,20,2,GREEN,0); // Zeichne Kreis
    ellipse(100, 80, 10,20,1,YELLOW,1); // Zeichne Ellipse
    rectan(100,150,140,180,1,BRIGHT_RED,1); // Zeichne Rechteck
    line(5,110,315,110,2,WHITE); // Zeichne Linie
    for(uc_val =0;uc_val<80;uc_val++){ // Zeichne mit plotDot() ein Muster
        for (t=0; t<100;t++){
            plotDot(140+uc_val+t,115+t,uc_val*t*8);
        }
    }
    for (t=0; t<180;t++){ // zeichne Sinus mit plotDot
        rad = 4*t * PI / 180; // Berechnen des Bogenmaßwinkels
        plotDot(10+t,(210+12*sin((double)(rad))),WHITE);
    }
    plotFilledRect ( 300, 20, 10, 10, RED ); // zeichne ein gefülltes Rechteck
    GPIOInit("PEH",00000000);
    GPIOE->CRH &= 0x00000000; // Konfiguriere GPIOE für
    GPIOE->CRH |= 0x22222222; // General purpose output push-pull, 2MHz
    while(1){
        getTSCxy(); // initialisiert Touch, liest die Werte für getTSCx() und getTSCy() ein.
        printAt(8, "TSC:");
        if(getTSCx() <= 320){printDec(5, getTSCx());} // grenze Bereich für Rückgabewerte ein und gib sie aus.
        if(getTSCy() <= 320){printDec(5, getTSCy());}
        printAt(13, ""); printBin(1,getTSCtouched()); // Schreibe Berührungstatus auf den Screen
        uc_val = getTSCtouched(); // Hole Touchwert 0,1 Debugging
        GPIOPutByte("PEH",getTSCtouched()); // zeige via LED ob Touch gedrückt wurde
        if(uc_val==1){
            for (t=0; t<220;t++){
                rad = 4*t * PI / 180; // Berechnen des Bogenmaßwinkels
                if(color_toggle==0) {
                    plotDot(10+t,(210+12*sin((double)(rad))),BRIGHT_BLUE);
                } else {
                    plotDot(10+t,(210+12*sin((double)(rad))),WHITE);
                }
            }
            color_toggle=color_toggle^0x01; // Toggle Color für den nächsten Sinus. Spielerei
        }
    }
}

```



7 Übung I2C

7.1 Auftrag

Nehmen Sie das Programm mit einem MCB32 in Betrieb und laden Sie die richtige Lib.

- Alle Antworten direkt im Code. Bilder in einem Wordfile.
- Neben den unten aufgeführten Fragen hat es auch Fragen im Code, welche beantwortet werden müssen.
- Je mehr sie dokumentieren umso besser die Note.

Schliessen Sie danach einen I2C-Expander an das MCB32 an:

1. Welchen Port müssen Sie nehmen. Woher nehmen Sie die Speisung?

Studieren Sie das Datenblatt sowie den Expander um herauszufinden welches die richtige Adresse für den I2C-Expander ist.

2. Adresse? Und Beschreibung des Vorgehens für das Herausfinden.

Wenn die Schaltung funktioniert machen Sie Messungen mit dem Logianalyzer. Konfigurieren Sie den Analyzer so, dass er I2C direkt dekodiert und Ihnen die Daten in HEX anzeigt. Speichern Sie die Messungen in einem Wordfile mit Beschreibung pro Bild.

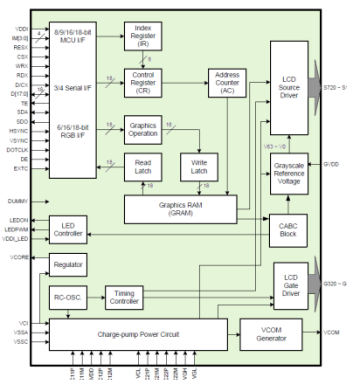
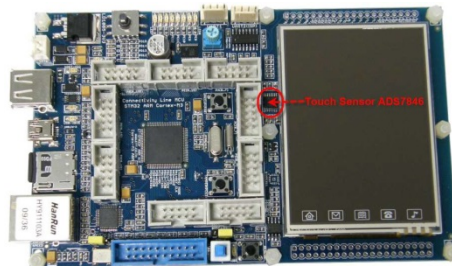
3. Erläutere alle wichtigen I2C Kommandos im Code mit Messung
4. Beschreibe den Code (ausser bei Printstatements) vollständig, was macht welche Zeile usw.

5. Programmiere, wenn die SW kommentiert und alle Messungen erledigt sind, ein Lauflicht auf dem I2C Expander. Schliessen Sie externe LEDS an oder machen Sie Messungen mit dem LogikAnalyzer um die Funktion zu zeigen.

8 Anhang Grafikhardware

8.1 Hintergrund

Der MCB32 Kit arbeitet mit einem TFT –LCD Color Grafik Display 320x240Pixel (3.2”) und einer Touch-Sensor (Folie). Der Grafik-Display wird über einen Chip ILI9341 angesteuert und der der Touch-Sensor über einen ADS7846. Beide Chips kommunizieren via die SPI Schnittstelle mit dem ARM-Prozessor resp. der Library.



Der ILI9341 Chip steuert den eigentlichen Bildschirm an. Der Chip hat 720 Source-Ausgänge und 320 Gate-Ausgänge um die einzelnen Dots (Pixels) ein und auszuschalten. Zudem können 172,8KByte RAM genutzt werden.

Der Chip wird über ein spezielles 9Bit SPI Interface angesteuert. Das heisst der Datentransfer ist beschränkt durch die serielle Datenübertragungsrate. Mehr Informationen zum Chip unter: <http://www.adafruit.com/datasheets/ILI9341.pdf> oder andere Quellen.

8.2 Hardwarenahe Beschreibung der Displayansteuerung

Verbindung Display TFT320x240 mit dem ARM: via GPIO Pins und SPI3 (9Bit)

| Pin | Beschreibung | Funktion | Port ARM |
|-----|------------------|------------|------------|
| CS | Chipselect | CS# | PC8 (Out) |
| SCL | Clock | SPI3:SCK3 | PC10 (Clk) |
| SDO | Data Out to ARM | SPI3:MISO3 | PC11 (Inp) |
| SDI | Data IN from ARM | SPI3:MOSI3 | PC12 (Out) |
| BL | IRQ to ARM | GPIO: IRQ | PD7 (Out) |

Verbindung Touch-Sensor ADS7846 mit dem ARM: via GPIO Pins

| Pin ADS7846 | Beschreibung | Funktion | Port ARM |
|-------------|------------------|-----------|-----------|
| CS | Chipselect | CS# | PE6 (Out) |
| DCLK | Clock | SPI:SCK | PE7 (Clk) |
| DOUT | Data Out to ARM | SPI:MISO | PE4 (Inp) |
| DIN | Data IN from ARM | SPI:MOSI | PE5 (Out) |
| PENIRQ | IRQ to ARM | GPIO: IRQ | PE3 (Inp) |

9 Anhang: Umstellung von C51-Code auf ARM32-Code

```

/*****
* Titel:          Umstellung von C51-Code auf ARM32-Code
* Datei:          C51toARM32.c / 14.1.14 / Version 1.0
* Ersteller:      R. Weber (BSU); E. Malacarne (TBZ)
* Funktion:       Die wichtigsten Umstellungen sind in den Kommentaren dokumentiert
*****/

```

```

#include <stm32f10x.h>           // Mikrokontrollertyp
#include " TouchPOP1.h"          // P0-, P1-Definition
                                // P0 = Input, P1= Output PE[15-8]

// Input und Outputbits an Ports benennen
#define Start P0_0              // Start = Input Port0[0]
#define Alarm P1_7              // 'Bit'-Variablentyp char
char bTemp = 0 ;                // Zeitvariable
long t;

```

neue #includes

Keine sfr und
sbit mehr!

```

int main ( void )               // Hauptprog., ohne return bei Keil
{
    InitTouchPOP1 ("1");        // Touchscreen aktiv,
                                // horizontal gedreht
                                // LSB rechts

```

Main verlangt int

InitTouchPOP1 ("0"),
wenn nur P0,P1 und
ohne Touchscreen

```

while(1)                        // Endlos-schleife
{
    P1_0 = 0;                   // Bitverarbeitung wie bisher
    Alarm = 1;                  // Zuweisung, Invertierung,
    bTemp = ! Start;            // &, &&, |, ||, ^, !, ==, !=

    while ( P1 < 100 )          // Byteverarbeitung wie bisher
        P1 += 2;                // Kurzformen wie bisher
    P1 = P0 & 0x0F;              // Maskierungen wie bisher

    for(t=120000; t>0; t--);    // Verzögerung 10ms
}
}

```

Verzögerung

vom Typ long mit
Wert 12 / μ s

9.1 Wichtig für das Funktionieren neuer Projekte

Wichtig: Bei der Erstellung eines neuen Projektes im Schulbereich, also Vorbereitung für 8Bit-Programme „Elektroniker“ mit Port P0, P1 und Touchscreen ist folgendes zu beachten.

Kopieren Sie in jedes neue Projektverzeichnis diesen zwei Dateien:

- TouchPOP1.h (REV C oder REV D)
- TouchPOP1.lib (REV C oder REV D)

10 Anhang Touchscreen Kontrolle am µC-Board MCB32

Beschreibung der Touchscreen Oberfläche mit P0 (=Eingabe-) und P1 (Ausgabe-Port)

The diagram shows a touchscreen interface with a grid of buttons. Annotations include:

- Betrachtung vertikal - horizontal drehen**: Points to the top of the screen.
- P0 und P1 vertauschen**: Points to the top right corner.
- Lage von MSB / LSB vertauschen**: Points to the right side of the screen.
- Wahl des gemeinsamen Schalter-/ Taster-Typs, Unterstrich zeigt den Typ**: Points to the top left corner.
- Wahl von 8 individuellen Schalter-/Tastertypen**: Points to the left side of the screen.
- Die Schalter- /Tastertypen sind:**: Points to the left side of the screen.
- Touch-Eingabe und Anzeige von P0**: Points to the center of the screen.
- Ausgabe-Anzeige von P1**: Points to the right side of the screen.

Die Schalter- /Tastertypen sind:

- 1: **Schalter** low Start
- 2: **Schalter** high Start
- 3: **Taster** high aktiv
- 4: **Taster** low aktiv

10.1.1 Touchscreen Kontrolle aus dem Quellcode

Der Projekt-Ordner muss TouchPOP1.h und TouchPOP1.lib enthalten:

Im Projekt-Manager sind die Quelldatei.c und die Lib „TouchPOP1.lib“ aufzunehmen.

```
/* Beschreibung der 8 Bit P0- und P1-Kontrolle über den Touchscreen des MCB32
*****/
#include <stm32f10x.h> // Mikrokontrollertyp
#include "TouchPOP1.h" // P0-, P1-Definition. Angepasst für REV C oder D

void main(void) // Hauptprogramm
{
    InitTouchPOP1 ("1"); // Touchscreen aktivieren. Bei „0“ ist SysTick
                        // -Timer abgeschaltet.
    while(1) { } // Benutzerprogramm
}
```

1) `InitTouchPOP1 ("0");`

Der Touchscreen bleibt ausgeschaltet

P0 ist als Input, P1 als Output konfiguriert

2) `InitTouchPOP1 ("1") .. ("1 r m p");`

Der Touchscreen wird aktiviert und konfiguriert, einfachste Konfiguration mit `InitTouchPOP1 ("1")`.

1...4: Gemeinsamer Schalter-/Tastertyp

p: P0 aussen, sonst mittig.

m: MSB oben/rechts, sonst unten/links.

r: Rotiert horizontal, sonst vertikal.

11 Anhang Anschlüsse am µC-Board MCB32

11.1.1 STLINK, Schalter, Potentiometer, P0, P1

Fremdspeisung
genau 5V! oder ...

USB Speisung (**roter Stecker**)

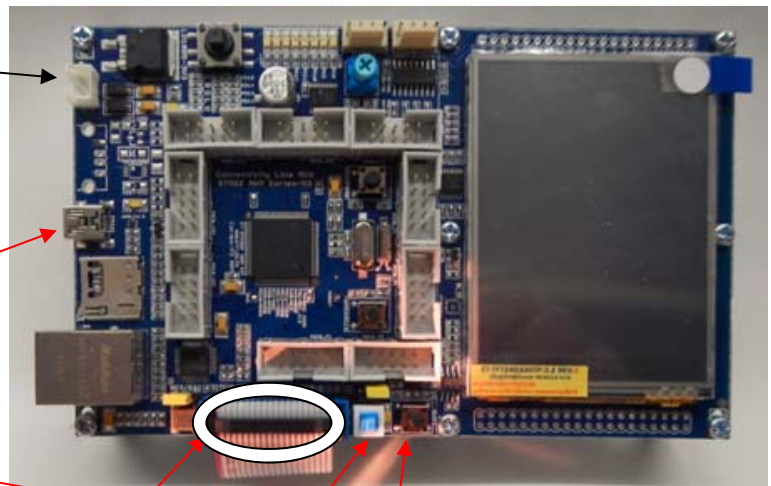
USB - ST-Link (**schwarzer**
USB Stecker)
Zielsystemdebugging,
Boardspeisung
wie oben

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

PE[8...15]



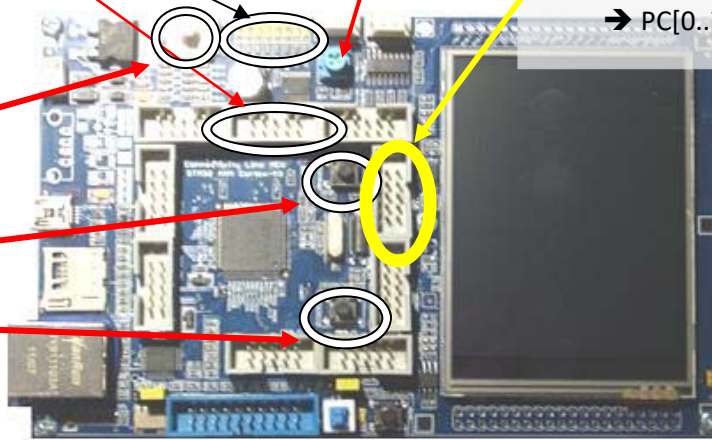
Pot auf Anschlag Uhrzeigersinn drehen, da gleichzeitig PO_4

P0: 8 externe Schalter
→ PC[0..7]

ControlStick

Button 0

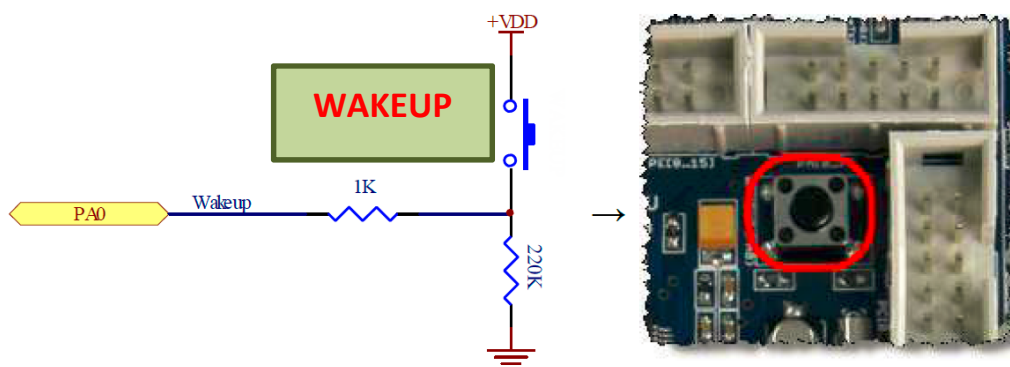
Button 1



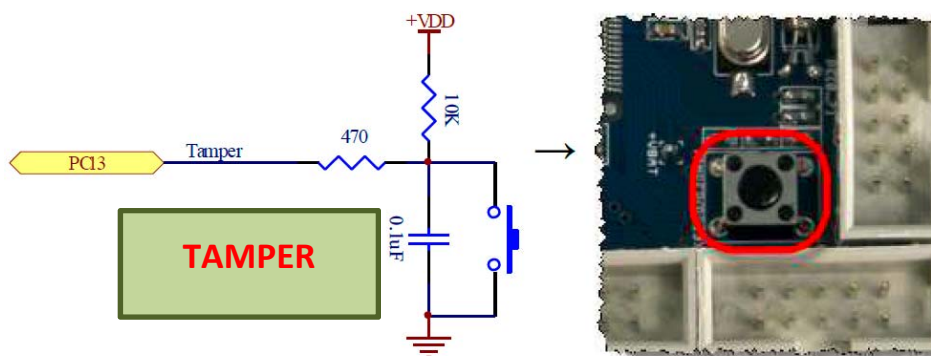

```
// In TouchP0P1.h definierte Pin-Bezeichnungen PA_0 .. PD_11, ohne Bezeichner wie Button .. !
char Button0      = PA_0;           // Bitwert 1/0, aktiv low, prellt wenig
char Button1      = PC_13;          // Bitwert 0/1, aktiv high

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
```

11.1.2 Button 0 / Wakeup (Pin:PA0); nicht gedrückt PA_0=0



11.1.3 Button 1 / Tamper (Pin:PC13); nicht gedrückt PC_13=1



11.1.4 Potentiometer (PC4) // resp. PO_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.



11.1.5 LED von Port P1 auf Board aktivieren

Mit den folgenden Befehlen wird Port PE[8..15] so gesetzt, dass die LEDs auf dem Board parallel zu dem Display auch aktiv angesteuert werden. Damit leuchten die LEDs gleich wie auf dem Display.

Achtung: im Falle, dass der Port GPIO im Mode Output-Push-Pull betrieben wird, dürfen keine externen Quellen oder Lasten ohne genaueres Wissen über die Vorgänge rund um den Port angeschlossen werden. Der Prozessor kann **Schaden** nehmen.

Siehe Beispiel Code weiter unten.

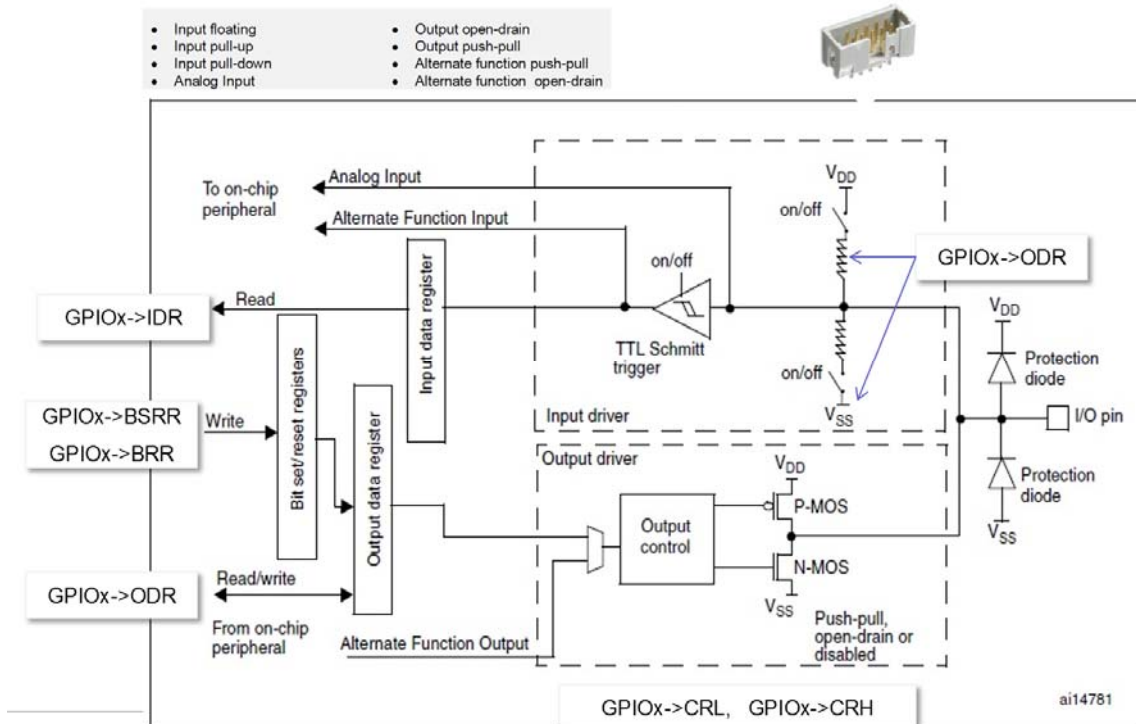
```
GPIOE->CRH   &= 0x00000000;    // Configure the GPIOE for
GPIOE->CRH   |= 0x22222222;    // General purpose output push-pull, 2MHz
```

```
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;                   // Zählung 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
    }
}
```

11.1.6 Übersicht über die Hardwarestruktur eines Pins

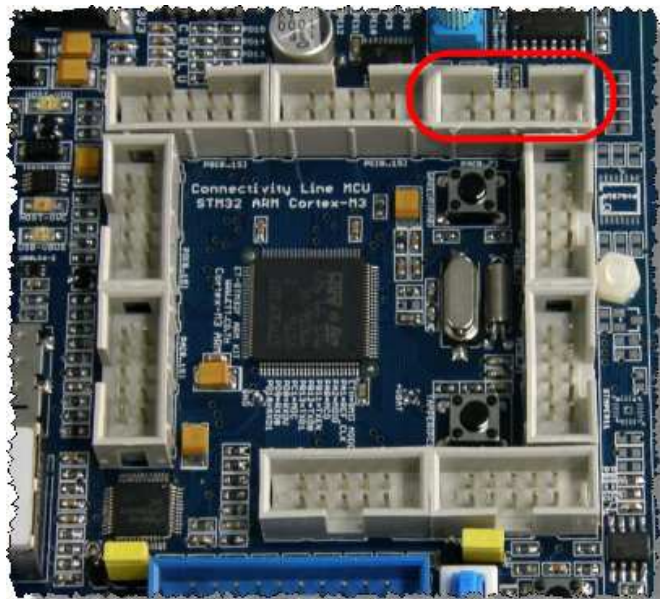


11.2 Port PA Pin 0..7

Im folgenden Abschnitt wird Port PA als Stellvertreter für die anderen Ports erklärt.

11.2.1 Steckerbelegung für 10pol. Stecker PA[0..7]

| PAL | PA [0..7] | |
|------|-------------|-----|
| PA0 | 1 2 | PA1 |
| PA2 | | PA3 |
| PA4 | | PA5 |
| PA6 | | PA7 |
| +3V3 | 9 10 | GND |



Die Belegung des 10poligen Steckers sieht wie oben am Beispiel des Steckers PAL abgebildet aus. Die roten Zahlen definieren die Adern des Flachbandkabels. Rote Ader = Pin1, daneben Ader 2 = Pin2 usw. .

11.2.2 Original Belegung PA[0..7]

Die Original Pin-Belegung von Stecker PA[0...7] ist wie in der nebenstehenden Tabelle. Einerseits zeigt die Tabelle die Funktion wie sie im Chip vorgesehen ist und die auf dem MCB32 dann ausgeführte Funktion.

Dies alles ist obsolet wenn die Funktionen nicht verwendet werden. Der Port kann dann als IO eingesetzt werden.

| Pin | STM32F107VC Funktion | MCB32 Module/Device | |
|-----|-------------------------|------------------------|--|
| PA0 | Wakeup | Switch Wakeup | |
| PA1 | RMII_REF_CLK | Ethernet LAN | |
| PA2 | RMII_MDIO | Ethernet LAN | |
| PA3 | - | - | |
| PA4 | - | - | |
| PA5 | SPI1_SCK | SD Card CLK | |
| PA6 | SPI1_MISO | SD Card DAT0 | |
| PA7 | SPI1_MOSI | SD Card CMD | |

12 Anhang: Interrupt Vektorliste und Servicefunktionsaufrufe



Interrupt and exception vectors

| Position | Priority | Type of priority | Acronym | Description | Address |
|----------|----------|------------------|---------------|--|-------------|
| - | - | - | - | Reserved | 0x0000_0000 |
| -3 | fixed | Reset | Reset | Reset | 0x0000_0004 |
| -2 | fixed | NMI | NMI | Non maskable interrupt. The RCC Clock Security System (CSS) is linked to the NMI vector. | 0x0000_0008 |
| -1 | fixed | HardFault | HardFault | All class of fault | 0x0000_000C |
| 0 | settable | MemManage | MemManage | Memory management | 0x0000_0010 |
| 1 | settable | BusFault | BusFault | Pre-fetch fault, memory access fault | 0x0000_0014 |
| 2 | settable | UsageFault | UsageFault | Undefined instruction or illegal state | 0x0000_0018 |
| | | Reserved | | 0x0000_001C - | 0x0000_002B |
| 3 | settable | SVCall | SVCall | System service call via SWI Instr | 0x0000_002C |
| 4 | settable | Debug Monitor | Debug Monitor | Debug Monitor | 0x0000_0030 |
| - | - | - | - | Reserved | 0x0000_0034 |
| 5 | settable | PendSV | PendSV | Pendable request for system service | 0x0000_0038 |
| 6 | settable | SysTick | SysTick | System tick timer | 0x0000_003C |
| 0 | 7 | settable | WWDG | Window Watchdog interrupt | 0x0000_0040 |
| 1 | 8 | settable | PVD | PVD through EXTI Line detection | 0x0000_0044 |
| 2 | 9 | settable | TAMPER | Tamper interrupt | 0x0000_0048 |
| 3 | 10 | settable | RTC | RTC global interrupt | 0x0000_004C |
| 4 | 11 | settable | FLASH | Flash global interrupt | 0x0000_0050 |
| 5 | 12 | settable | RCC | RCC global interrupt | 0x0000_0054 |
| 6 | 13 | settable | EXTI0 | EXTI Line0 interrupt | 0x0000_0058 |
| 7 | 14 | settable | EXTI1 | EXTI Line1 interrupt | 0x0000_005C |
| 8 | 15 | settable | EXTI2 | EXTI Line2 interrupt | 0x0000_0060 |
| 9 | 16 | settable | EXTI3 | EXTI Line3 interrupt | 0x0000_0064 |
| 10 | 17 | settable | EXTI4 | EXTI Line4 interrupt | 0x0000_0068 |
| 11 | 18 | settable | DMA1_Channel1 | DMA1 Channel1 global interrupt | 0x0000_006C |
| 12 | 19 | settable | DMA1_Channel2 | DMA1 Channel2 global interrupt | 0x0000_0070 |
| 13 | 20 | settable | DMA1_Channel3 | DMA1 Channel3 global interrupt | 0x0000_0074 |
| 14 | 21 | settable | DMA1_Channel4 | DMA1 Channel4 global interrupt | 0x0000_0078 |
| 15 | 22 | settable | DMA1_Channel5 | DMA1 Channel5 global interrupt | 0x0000_007C |
| 16 | 23 | settable | DMA1_Channel6 | DMA1 Channel6 global interrupt | 0x0000_0080 |
| 17 | 24 | settable | DMA1_Channel7 | DMA1 Channel7 global interrupt | 0x0000_0084 |
| 18 | 25 | settable | ADC1_2 | ADC1 and ADC2 global interrupt | 0x0000_0088 |
| 19 | 26 | settable | CAN1_TX | CAN1 TX interrupts | 0x0000_008C |
| 20 | 27 | settable | CAN1_RX0 | CAN1 RX0 interrupts | 0x0000_0090 |
| 21 | 28 | settable | CAN1_RX1 | CAN1 RX1 interrupt | 0x0000_0094 |
| 22 | 29 | settable | CAN1_SCE | CAN1 SCE interrupt | 0x0000_0098 |
| 23 | 30 | settable | EXTI9_5 | EXTI Line[9:5] interrupts | 0x0000_009C |
| 24 | 31 | settable | TIM1_BRK | TIM1 Break interrupt | 0x0000_00A0 |
| 25 | 32 | settable | TIM1_UP | TIM1 Update interrupt | 0x0000_00A4 |
| 26 | 33 | settable | TIM1_TRG_COM | TIM1 Trigger and Commutation | 0x0000_00A8 |
| 27 | 34 | settable | TIM1_CC | TIM1 Capture Compare interrupt | 0x0000_00AC |
| 28 | 35 | settable | TIM2 | TIM2 global interrupt | 0x0000_00B0 |
| 29 | 36 | settable | TIM3 | TIM3 global interrupt | 0x0000_00B4 |
| 30 | 37 | settable | TIM4 | TIM4 global interrupt | 0x0000_00B8 |
| 31 | 38 | settable | I2C1_EV | I2C1 event interrupt | 0x0000_00BC |
| 32 | 39 | settable | I2C1_ER | I2C1 error interrupt | 0x0000_00C0 |
| 33 | 40 | settable | I2C2_EV | I2C2 event interrupt | 0x0000_00C4 |
| 34 | 41 | settable | I2C2_ER | I2C2 error interrupt | 0x0000_00C8 |
| 35 | 42 | settable | SPI1 | SPI1 global interrupt | 0x0000_00CC |
| 36 | 43 | settable | SPI2 | SPI2 global interrupt | 0x0000_00D0 |
| 37 | 44 | settable | USART1 | USART1 global interrupt | 0x0000_00D4 |
| 38 | 45 | settable | USART2 | USART2 global interrupt | 0x0000_00D8 |
| 39 | 46 | settable | USART3 | USART3 global interrupt | 0x0000_00DC |
| 40 | 47 | settable | EXTI15_10 | EXTI Line[15:10] interrupts | 0x0000_00E0 |
| 41 | 48 | settable | RTCAlarm | RTC alarm through EXTI line inte | 0x0000_00E4 |
| 42 | 49 | settable | OTG_FS_WKUP | USB On-The-Go FS Wakeup | 0x0000_00E8 |
| - | - | - | Reserved | 0x0000_00EC - | 0x0000_0104 |
| 50 | 57 | settable | TIM5 | TIM5 global interrupt | 0x0000_0108 |
| 51 | 58 | settable | SPI3 | SPI3 global interrupt | 0x0000_010C |
| 52 | 59 | settable | UART4 | UART4 global interrupt | 0x0000_0110 |
| 53 | 60 | settable | UART5 | UART5 global interrupt | 0x0000_0114 |
| 54 | 61 | settable | TIM6 | TIM6 global interrupt | 0x0000_0118 |
| 55 | 62 | settable | TIM7 | TIM7 global interrupt | 0x0000_011C |
| 56 | 63 | settable | DMA2_Channel1 | DMA2 Channel1 global interrupt | 0x0000_0120 |
| 57 | 64 | settable | DMA2_Channel2 | DMA2 Channel2 global interrupt | 0x0000_0124 |
| 58 | 65 | settable | DMA2_Channel3 | DMA2 Channel3 global interrupt | 0x0000_0128 |

```
void EXTI0_IRQHandler(void)
```

```

WWDG_IRQHandler
PVD_IRQHandler
TAMPER_IRQHandler
RTC_IRQHandler
FLASH_IRQHandler
RCC_IRQHandler
EXTI0_IRQHandler
EXTI1_IRQHandler
EXTI2_IRQHandler
EXTI3_IRQHandler
EXTI4_IRQHandler
DMA1_Channel1_IRQHandler
DMA1_Channel2_IRQHandler
DMA1_Channel3_IRQHandler
DMA1_Channel4_IRQHandler
DMA1_Channel5_IRQHandler
DMA1_Channel6_IRQHandler
DMA1_Channel7_IRQHandler
ADC1_2_IRQHandler
CAN1_TX_IRQHandler
CAN1_RX0_IRQHandler
CAN1_RX1_IRQHandler
CAN1_SCE_IRQHandler
EXTI9_5_IRQHandler
TIM1_BRK_IRQHandler
TIM1_UP_IRQHandler
TIM1_TRG_COM_IRQHandler
TIM1_CC_IRQHandler
TIM2_IRQHandler
TIM3_IRQHandler
TIM4_IRQHandler
I2C1_EV_IRQHandler
I2C1_ER_IRQHandler
I2C2_EV_IRQHandler
I2C2_ER_IRQHandler
SPI1_IRQHandler
SPI2_IRQHandler
USART1_IRQHandler
USART2_IRQHandler
USART3_IRQHandler
EXTI15_10_IRQHandler

```

```

RTCAlarm_IRQHandler
OTG_FS_WKUP_IRQHandler
TIM5_IRQHandler
SPI3_IRQHandler
UART4_IRQHandler
UART5_IRQHandler
TIM6_IRQHandler
TIM7_IRQHandler
DMA2_Channel1_IRQHandler
DMA2_Channel2_IRQHandler
DMA2_Channel3_IRQHandler
DMA2_Channel4_IRQHandler
DMA2_Channel5_IRQHandler
ETH_IRQHandler
ETH_WKUP_IRQHandler
CAN2_TX_IRQHandler
CAN2_RX0_IRQHandler
CAN2_RX1_IRQHandler
CAN2_SCE_IRQHandler
OTG_FS_IRQHandler

```


13 Anhang: SysTick Timer

Alle Cortex-M Prozessoren enthalten einen 24bit Timer, mit dem man die Systemzeit misst. Der Timer zählt die Taktimpulse des Prozessors herunter und löst bei jedem Überlauf (0) einen Interrupt `SysTick_Handler()` aus welcher die gewünschten Schritte vornimmt. Das heisst der SysTick interruptfähig muss gemacht werden.

Da es sich um einen Interrupt handelt, muss auch eine zugehörige Serviceroutine geschrieben werden, die bei **Keil** einen festgelegten Namen hat:

```
void SysTick_Handler(void)
// SysTick Interrupt Handler
{    //...Insert function here }
```

Der Funktionsaufruf `SysTick_Config(SystemCoreClock/1000)` im Beispiel sorgt dafür, dass jede Millisekunde ein SysTick Interrupt ausgelöst wird.

```
#include <stdint.h>
#include "stm32f1xx.h"

uint32_t SystemCoreClock=8000000;
volatile uint32_t systick_count=0;

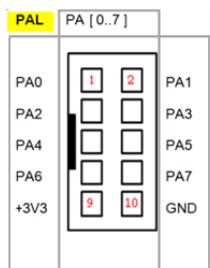
// Interrupt handler
void SysTick_Handler(void)
{
    systick_count++;
}

int main(void)
{
    // Initialize the timer: 1ms interval
    SysTick_Config(SystemCoreClock/1000);

    // Delay 2 seconds
    uint32_t start=systick_count;
    while (systick_count-start<2000);
    ...
}
```

14 Anhang: Port Pin Liste MCB32

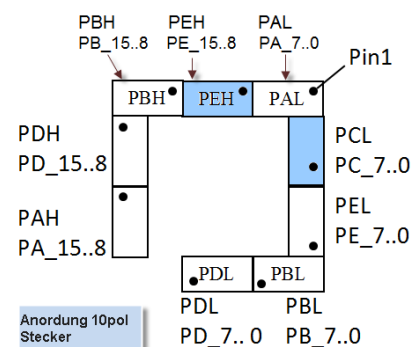
Die Nachfolgende Liste beschreibt die einzelnen Ports. Es ist zu beachten, dass für Versuche nur die Pins mit der Bezeichnung Free_xx benutzt werden sollen um die anderen, besetzten Funktionen nicht zu stören. Bei Abweichungen von dieser Regel ist jeder Benutzer verantwortlich für die Hardware- und Softwarefunktion. Port



PE8..15 (LEDs 0..7) kann auch als GPIO benutzt werden. Die LEDs sind via einen Treiber vom Port isoliert.

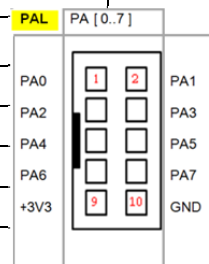
Bild Links: Portnummerierung von PAL

Achtung: Alle Ports dürfen nicht mit mehr als 3,3V beschaltet werden. Falsche Handhabung führt zur Zerstörung des Controllers. Die Garantie geht dabei verloren.



| Pin | Function | Devices | Pin | Function | Devices |
|-----|--------------|---------------|------|-----------|--------------------------|
| PA0 | Wakeup | Switch Wakeup | PA8 | MCO | Ethernet LAN |
| PA1 | RMII_REF_CLK | Ethernet LAN | PA9 | FS_VBUS | USB OTG/Device |
| PA2 | RMII_MDIO | Ethernet LAN | PA10 | FS_ID | USB OTG |
| PA3 | Free_1. | - | PA11 | FS_DM | USB Data HOST/OTG/Device |
| PA4 | Free_2. | - | PA12 | FS_DP | |
| PA5 | SPI1_SCK | SD Card CLK | PA13 | JTAG_TMS | JTAG |
| PA6 | SPI1_MISO | SD Card DAT0 | PA14 | JTAG_TCLK | JTAG |
| PA7 | SPI1_MOSI | SD Card CMD | PA15 | JTAG_TDI | JTAG |

| Pin | Function | Devices | Pin | Function | Devices |
|-----|------------|-------------------|------|------------------|-------------------|
| PB0 | Free_3. | - | PB8 | I2C1_SCL | 24C01, STMPE811 |
| PB1 | Free_4. | - | PB9 | I2C1_SDA | 24C01, STMPE811 |
| PB2 | BOOT1 | Jumper BOOT1 | PB10 | Free_5. | - |
| PB3 | JTAG_TDO | JTAG | PB11 | RMII_TXEN | Ethernet LAN |
| PB4 | JTAG_TRST | JTAG | PB12 | RMII_TXD0 | Ethernet LAN |
| PB5 | Free_6. | - | PB13 | RMII_TXD1 | Ethernet LAN |
| PB6 | USART1_TX | UART1 | PB14 | Free_7. | - |
| PB7 | USART1_RX | UART1 | PB15 | Free_8. | - |
| Pin | Function | Devices | Pin | Function | Devices |
| PC0 | Free_9. | - | PC8 | GPIO Out | GLCD CS# |
| PC1 | RMII_MDC | Ethernet LAN | PC9 | HOST_EN | USB HOST/OTG |
| PC2 | Free_10. | - | PC10 | SPI3_SCK | GLCD WR#/SCL |
| PC3 | Free_11. | - | PC11 | SPI3_MISO | GLCD SDO |
| PC4 | ADC14 | Volume VR1 | PC12 | SPI3_MOSI | GLCD SDI |
| PC5 | GPIO Out | SD Card / CD(CS#) | PC13 | Tamper | Switch Tamper |
| PC6 | Free_12. | - | PC14 | OSC32_IN | RTC X-TAL |
| PC7 | Free_13. | - | PC15 | OSC32_OUT | RTC X-TAL |
| Pin | Function | Devices | Pin | Function | Devices |
| PD0 | Free_14. | - | PD8 | RMII_CRS_DV | Ethernet LAN |
| PD1 | Free_15. | - | PD9 | RMII_RXD0 | Ethernet LAN |
| PD2 | Free_16. | - | PD10 | RMII_RXD1 | Ethernet LAN |
| PD3 | Free_17. | - | PD11 | GPIO Input | Joy Switch Up |
| PD4 | Free_18. | - | PD12 | GPIO Input | Joy Switch Left |
| PD5 | USART2_TX | UART2(ISP) | PD13 | GPIO Input | Joy Switch Down |
| PD6 | USART2_RX | UART2(ISP) | PD14 | GPIO Input | Joy Switch Right |
| PD7 | GPIO Out | GLCD BL LED | PD15 | GPIO Input | Joy Switch Select |
| Pin | Function | Devices | Pin | Function | Devices |
| PE0 | Free_19. | - | PE8 | GPIO Out/Free_21 | LED0 |
| PE1 | USB_OVRCR | USB HOST/OTG | PE9 | GPIO Out/Free_22 | LED1 |
| PE2 | Free_20. | - | PE10 | GPIO Out/Free_23 | LED2 |
| PE3 | GPIO Input | ADS7846 PEN# | PE11 | GPIO Out/Free_24 | LED3 |
| PE4 | GPIO Input | ADS7846 DOUT | PE12 | GPIO Out/Free_25 | LED4 |
| PE5 | GPIO Out | ADS7846 DIN | PE13 | GPIO Out/Free_26 | LED5 |
| PE6 | GPIO Out | ADS7846 CS# | PE14 | GPIO Out/Free_27 | LED6 |
| PE7 | GPIO Out | ADS7846 DCLK | PE15 | GPIO Out/Free_28 | LED7 |



15 Referenzen

- [1] Bosch, «Datenblatt / Datasheet Bosch BME280».
- [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, Hrsg., Heidelberg: Hütigh Jehle Rehm GmbH, 2014.