

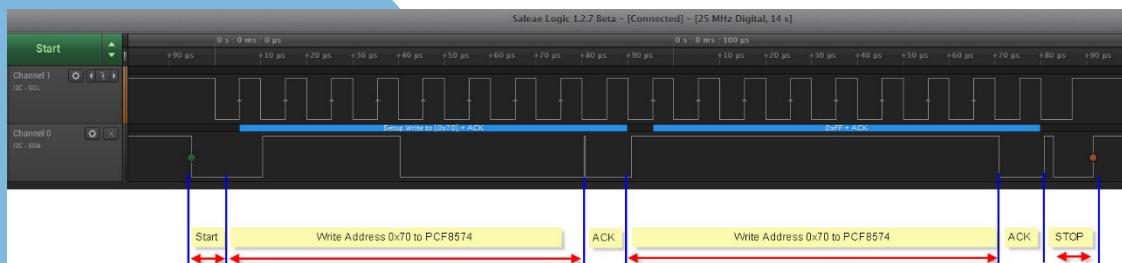
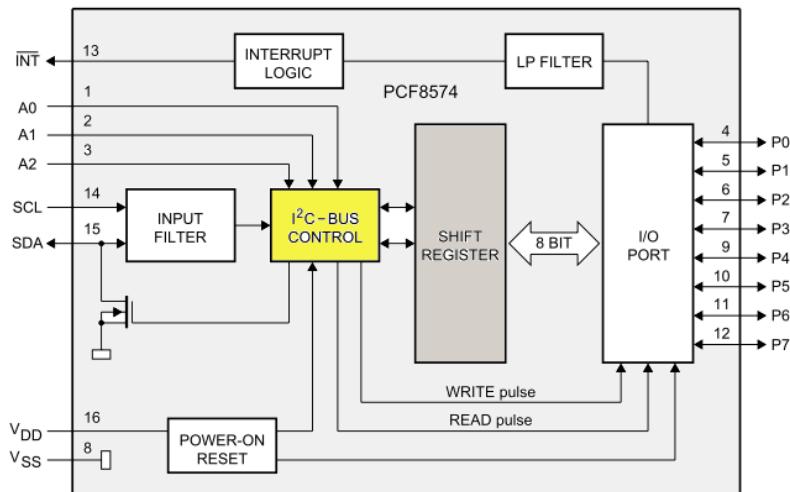


Mikrocontroller MCB32

I²C Bus: Einführung / Anwendung

Remote 8-bit I/O expander for I²C-bus

PCF8574



MCB32 - Embedded Programmierung I²C

Version: 1946.01

Diese Dokumentation kann ohne Vorankündigung jederzeit angepasst, verbessert und erweitert werden. Wünsche und Fehler an: 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 Inhalt

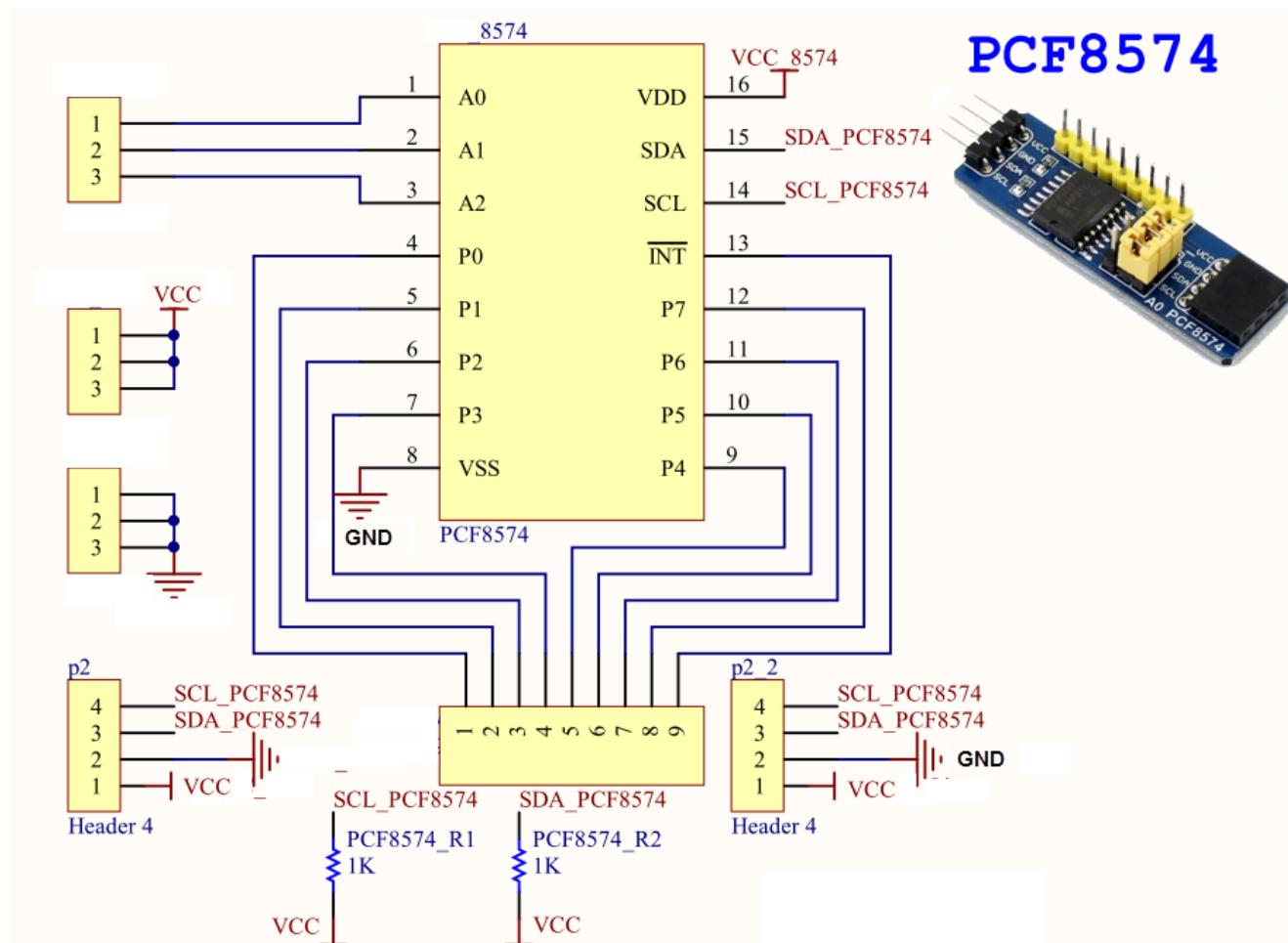
| | | |
|-----------|--|-----------|
| 1 | Inhalt | 2 |
| 2 | Gefahren Portbeschaltung | 3 |
| 3 | Hardwarefunktionen I2C mit MCB32 | 3 |
| 3.1 | Hardware, Schema | 3 |
| 3.2 | I2C Bus | 3 |
| 3.2.1 | Bit transfer | 4 |
| 3.2.2 | Start and stop conditions | 4 |
| 3.2.3 | Acknowledge (ACK) | 4 |
| 3.2.4 | Write Mode | 5 |
| 3.3 | IC PCF 8574 | 5 |
| 3.4 | Hardwarefunktionen | 7 |
| 3.4.1 | 5x GPIO General Purpose Input Output | 7 |
| 3.4.2 | 2 x ADC Analog Digital Converter (0...3.3V, 8Bit Auflösung) | 7 |
| 3.4.3 | 2 x DAC Digital Analog Converter (0...3.3V, 8Bit Auflösung) | 8 |
| 3.5 | Serielle Schnittstellen [2] | 9 |
| 3.5.1 | USART1 & USART2 | 9 |
| 3.6 | Interrupt Funktionen | 10 |
| 3.6.1 | 4 x Externe Interrupt Requests (Vereinfacht, jede pinNr nur 1x) | 10 |
| 3.6.2 | 6 x Interne Interrupt Requests | 10 |
| 3.6.3 | Interrupt Handler (ACHTUNG: die Namen/Bezeichner sind vorgeschrrieben) | 10 |
| 4 | Grafikprogrammierung | 11 |
| 4.1.1 | Fonts | 11 |
| 4.1.2 | Grafikfunktionen [1] | 11 |
| 4.1.3 | Touch-Funktionen | 12 |
| 4.1.4 | Touchscreen Textfunktionen | 12 |
| 4.2 | Farbliste | 13 |
| 4.3 | Musterprogramm für Grafikfunktionen | 14 |
| 5 | Übung I2C | 14 |
| 5.1 | Auftrag | 15 |
| 6 | Anhang Grafikhardware | 17 |
| 6.1 | Hintergrund | 17 |
| 6.2 | Hardwarenahe Beschreibung der Displayansteuerung | 17 |
| 7 | Anhang: Umstellung von C51-Code auf ARM32-Code | 18 |
| 7.1 | Wichtig für das Funktionieren neuer Projekte | 18 |
| 8 | Anhang Touchscreen Kontrolle am µC-Board MCB32 | 19 |
| 8.1.1 | Touchscreen Kontrolle aus dem Quellcode | 19 |
| 9 | Anhang Anschlüsse am µC-Board MCB32 | 20 |
| 9.1.1 | STLINK, Schalter, Potentiometer, P0, P1 | 20 |
| 9.1.2 | Button 0 / Wakeup (Pin:PA0); nicht gedrückt PA_0=0 | 21 |
| 9.1.3 | Button 1 / Tamper (Pin:PC13); nicht gedrückt PC_13=1 | 21 |
| 9.1.4 | Potentiometer (PC4) // resp. P0_4 (Library) | 21 |
| 9.1.5 | LED von Port P1 auf Board aktivieren | 22 |
| 9.1.6 | Übersicht über die Hardwarestruktur eines Pins | 22 |
| 9.2 | Port PA Pin 0..7 | 22 |
| 9.2.1 | Steckerbelegung für 10pol. Stecker PA[0..7] | 23 |
| 9.2.2 | Original Belegung PA[0..7] | 23 |
| 10 | Anhang: Interrupt Vektorliste und Servicefunktionsaufrufe | 24 |
| 11 | Anhang: SysTick Timer | 25 |
| 12 | Anhang: Port Pin Liste MCB32 | 25 |
| 13 | Referenzen | 28 |

2 Gefahren Portbeschaltung

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

3 Hardwarefunktionen I2C mit MCB32

3.1 Hardware, Schema



3.2 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.

3.2.1 Bit transfer

Ein Datenbit wird während einem Clock-puls 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 Kontrolle signale interpretiert.

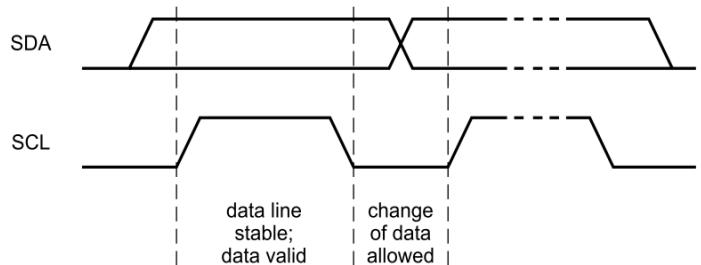


Bild 1: Bit transfer

3.2.2 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..

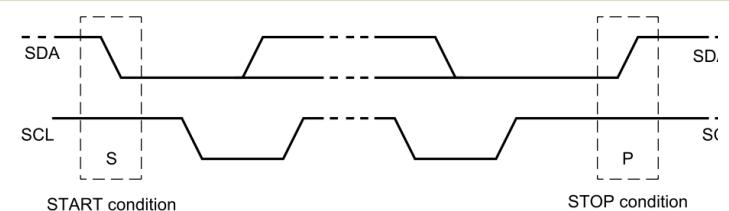


Bild 2: Definition der Start, Stop-Bedingung.

3.2.3 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 1: Acknowledgment on the I2C-bus...

Das ACK Bit ist ein HIGH Pegel welcher vom Transmitter-Chip während dem 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.

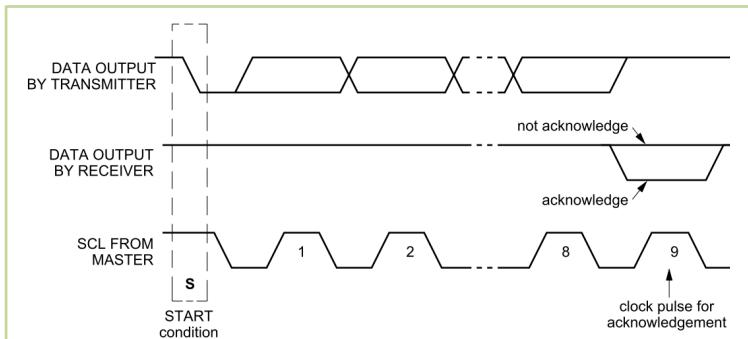


Bild 3: Acknowledgment on the I2C-bus..

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. **The device that acknowledges has to pull down the SDA line during the acknowledge clock pulse, so that the SDA line is stable LOW during the HIGH period of the acknowledge related clock pulse, set-up and hold times must be taken into account. A master receiver must signal an end of data to the**

transmitter by not generating an acknowledge on the last byte that has been clocked out of the slave. In this event the transmitter must leave the data line HIGH to enable the master to generate a stop condition.

3.2.4 Write Mode

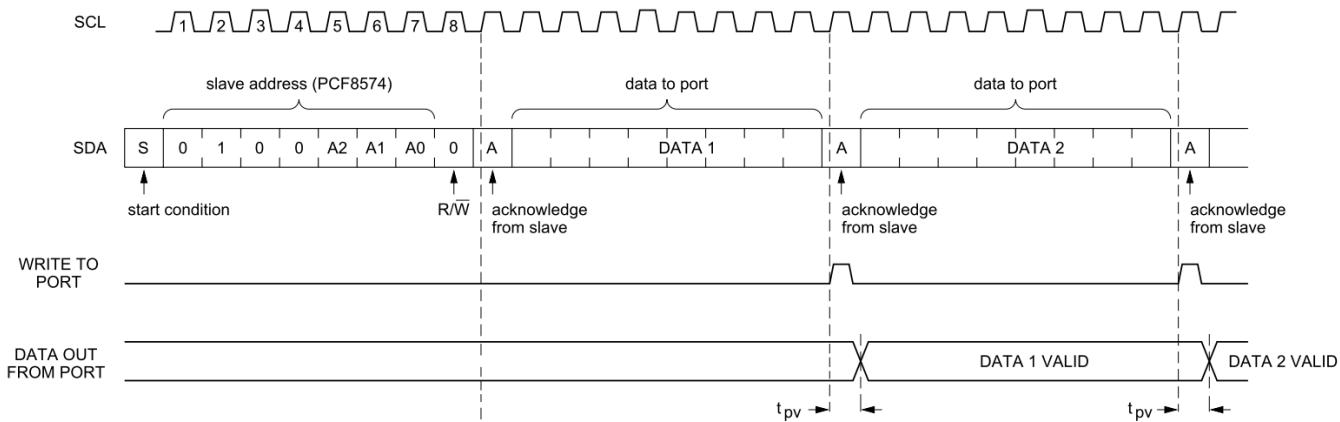


Bild 4: Write Modus

3.3 IC PCF 8574

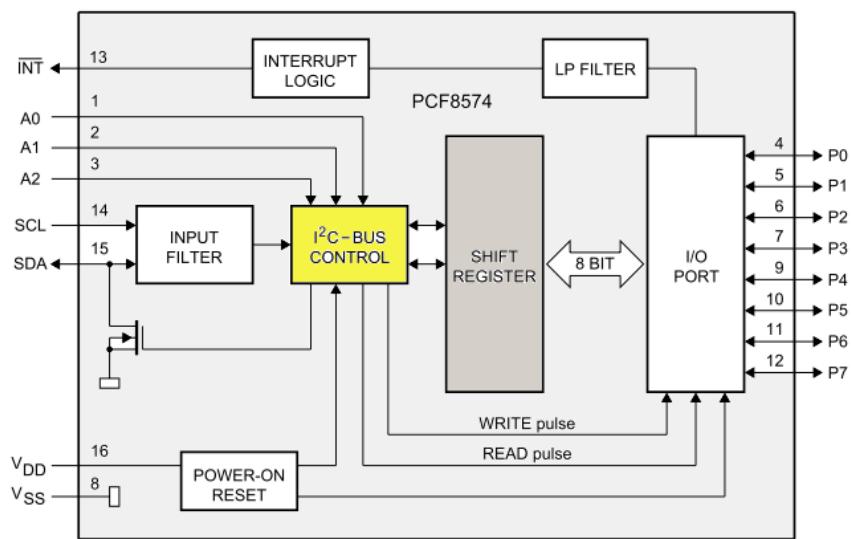
Der 8574 war einer der ersten I2C Bus Chips welcher von Philips / Heute NXP auf den Markt gebracht wurde. Er ermöglicht das Erzeugen eines 8Bit-IO Systems indem via den seriellen I2C-Bus die Daten (1Byte) geschrieben resp. gelesen werden.

Das Blockschema sieht einfach aus:

1. Die Daten gelangen via den I2C-Controller zum Schieberegister (SR) welches die Daten von Seriell- nach Parallel wandelt und vice-versa.
2. Der IO-Port ist statisch, d.h. die Daten werden gelatched. (FF)
3. 3 Adress-Pins erlauben das Adressieren der Chips. Es können also gleichzeitig 8 Chips an einem Bus angeschlossen werden.

Remote 8-bit I/O expander for I²C-bus

PCF8574



3.4 Hardwarefunktionen

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: 1946.01 |
|--|--|-----------|----------------------------|
| Funktion | Beschreibung / Beispiel | Parameter | Hinweis zur Hardware |
| 3.4.1 5x GPIO General Purpose Input Output | | | |
| <code>GPIOInit("PxH/L", "yyyyyyy");</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 <code>GPIOInit("PAL", "11110000"); // 40ut,4In</code> | x:A-E | |
| <code>GPIOPutByte("PxH/L ", Byte);</code> | Siehe bei <code>GPIOInit("PxH/L", "yyyyyyy");</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. char. <code>b = GPIOGetByte("PAL"); // PAL->var b</code> | x:A-E | |
| 3.4.2 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. | | | |

| | | | |
|----------------------------|--|-------------------------|--|
| ADCInit(1/2,"Pin"); | 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 | |
| char ADCGetVal(1/2); | 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: 1946.01

3.4.3 2 x DAC Digital Analog Converter (0...3.3V, 8Bit Auflösung)

Je 1 Digital_Analog-Kanal auf je 1 Leitung ausgeben. Ausgänge sind PA4 und PA5.

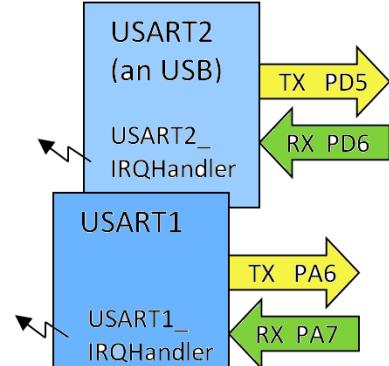
| | | | |
|---------------------------|--|-------------------------|----------|
| DACInit(1/2); | Wählt entweder Kanal1 (PA4) oder Kanal2 (PA5) für die Ausgabe des analogen Signals aus. Die Werte werden direkt nach dem Laden desselbigen ausgegeben. Das heisst in dieser einfachen Version werden keine Trigger unterstützt. <code>DACInit(1); // DAC1 an PA4</code> | Pin: siehe rechts | |
| DACPutVal(1/2,8Bit-Wert); | 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> | | |

3.5 Serielle Schnittstellen [1]

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: 1946.01 | |
|---|---|----------------------------|----------------------|
| Funktion | Beschreibung / Beispiel | Parameter | Hinweis zur Hardware |
| 3.5.1 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 IRQPrio: 0, 1..15 Einer von 2 USART 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) USARTWrite(2, 'c'); // Sende Zeichen 'c' | | |
| char USARTRead (1 / 2); | Wartet bis ein Zeichen an USART 1 oder 2 eintrifft und gibt Wert via Var. char zurück. c = USARTRead(2); // Warte auf Zeichen .. | | |
| char USARTToString (1/2); | Gibt den Status des USART Kanals zurück. 1 = Zeichen eingetroffen. 0= kein Zeichen im Buffer. c = USARTToString(2); // 0/1 in c | | |
| | | | |
| Beispiel: neue Baudrate für USART 1 programmieren | | | |
| <pre>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</pre> | | | |



3.6 Interrupt Funktionen

| | | Version und Stand: 1946.01 |
|--|--|--|
| 3.6.1 4 x Externe Interrupt Requests (Vereinfacht, jede pinNr nur 1x) | | |
| ExtIRQInit (<i>"Pin"</i> , Flanke0/1/2, IRQPrio0..15); | <p>Pin[" "]: "PA0".."PE15" Interrupt auslösender Pin</p> <p>Flanke: 0,1,2 Auslösende Flanke: 0 pos, 1 neg, 2 beide</p> <p>IRQPrio: 0, 1..15 Interruptpriorität 0 = AUS, 1 höchste, 15 tiefste.</p> <p>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>.</p> <pre>// IRQ PA0,pos Flanke Priorität 4 ExtIRQInit("PA0",0,4);</pre> | <p>PA15.0 PB15.0 PC15.0 PD15.0 PE15.0</p> <p>GPIOs</p> <p>80 IRQs 7 Handler</p> <p>EXTI0_IRQHandler ... EXTI15_10_IRQHandler</p> |
| 3.6.2 6 x Interne Interrupt Requests | | |
| TimerInit (2..5,n*100us,IRQPrio0..15); | Siehe unter Timer | <p>2 USART</p> <p>4 Counter</p> <p>oder</p> <p>Timer</p> <p>TIM2_ ... TIM5_IRQHandler USART1_ USART2_IRQHandler</p> |
| CounterInit (2..5,"Pin",UD0/1,IRQPrio0..15); | Siehe unter Counter | |
| USARTInit1/2 , " IRQPrio0..15); | CounterInit(2,"PA0",0,3); // Prio 3 Siehe unter USART | |
| 3.6.3 Interrupt Handler (ACHTUNG: die Namen/Bezeichner sind vorgeschrieben) | | |
| <pre>// Beispiel void TIM2_IRQHandler(void) { // Immer zuerst: IRQClearFlag("T2"); // 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") |

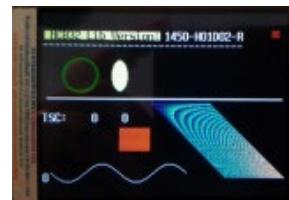
4 Grafikprogrammierung

4.1.1 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: 1946.01

4.1.2 Grafikfunktionen [2]

| Funktion | Beschreibung | Parameter | Beispiel |
|------------------------|--|-----------------|--|
| InitTouchScreen () | Touchscreen ohne P0P1 für Text, Grafik, Peripherie | - | InitTouchScreen(); |
| InitTouchP0P1 ("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 8 | Der Befehl: InitTouchP0P1("1"); schaltet P0,P1 ein → |
| setScreenDir (DIR) | Setzt die Schreibrichtung des Displays | HOR und VER | setScreenDir (HOR); setScreenDir (VER); |
| char getScreenDir() | Gibt die Schreibrichtung zurück | HOR=0 und VER=1 | if(getScreenDir()==VER) {clearScreen (BLUE);} |
| clearScreen (color) | Löscht den Bildschirm mit der angegeben Farbe. Funktioniert nur mit Einstellung setScreenDir(VER). | long color | clearScreen(BLACK); |

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. | | plotDot(120,120,WHITE); |
| 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 | circle(50,80,20,2, GREEN,0); |
| 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. | | rectan(100,150,140,180,1, RED,1); |
| plotFilledRect (x1,y1,dx,dy,color) | Gefülltes Rechteck von x1, y1 nach x1+dx, y1+dy mit Farbe color | | plotFilledRect (10, 20, 50, 60, RED); |
| 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. | | textxy(" MCB32 Lib Version:", 2, 32, BLACK, YELLOW); |
| line (x1,y1,x2,y2,thick,color) | Zeichne Linie von X1,y1 nach x2,y2 mit der Dicke und der Farbe | | line(5,110,315,110,2,WHITE); |

Grafikfunktionen:

Version und Stand: 1946.01

4.1.3 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> |  | <code>if (getTSCtouched ()) { }</code> |

4.1.4 Touchscreen Textfunktionen

| vertikal, 20 Zeilen à 30 Zeichen | horizontal, 15 Zeilen à 40 Zeichen |
|--|---|
| <pre>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:</pre> | <pre>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:</pre> |

| Funktion | Beschreibung | Beispiel |
|---|--|--|
| <code>void InitTouchScreen (void);</code> | Initialisiert den Touchscreen ohne P0P1 für Text, Grafik und Peripherie | <code>InitTouchScreen ();</code> |
| <code>void setScreenDir (char dir);</code> | Display Ausrichtung wechseln horizontal / vertikal | <code>setScreenDir (HOR);</code> <code>setScreenDir (VER);</code> |
| <code>void clearScreen (long color);</code> <code>void setTextColor (long color);</code> | Füllt den Touchscreen mit color Farbwechsel für nachfolgenden Text | <code>clearScreen (BLACK);</code> <code>setTextcolor (WHITE);</code> |
| <code>void print (char *txt);</code> <code>void println (char *txt);</code> | Schreibt Text hinter die letzte Position Schreibt Zeile hinter die letzte Position und springt an den nächsten Zeilenanfang | <code>print ("Text");</code> <code>println ("Text");</code> <code>println ("");</code> |
| <code>void printAt (char n, char *txt);</code> | Schreibt Text an den Anfang der Zeile mit Nummer n | <code>printAt (12, "Text");</code> |
| <code>void printBin (char n, long num);</code> | Konstanten- und Variablenwerte im Binärcode wie 1111'0000 mit der Bitanzahl n | <code>printBin (8, 250);</code> <code>printBin (32, variable);</code> |
| <code>void printHex (char n, long num);</code> | Konstanten- und Variablenwerte im Hexcode wie 0xFF00123E mit der Bitanzahl n | <code>printHex (8, 250);</code> <code>printHex (32, variable);</code> |

| | | |
|--|---|--|
| <code>void 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); printDec (-8, 123456); printDec (1, variable); printDec (-1, -123456);</code> |
| <code>void textxy (char x,char y, *txt, long forcol, long backcol);</code> | Schreibt an x y Text mit Vorder-, Hintergrundfarbe | <code>textxy (10, 20, "Text", RED, BLACK);</code> |

4.2 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: | |
|--------------------------------------|--------------------------------|
| 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 CYANO | 0x3D34 |
| #define CYAN1 | 0x1DF7 |
| #define GREEN0 | 0x0200 |
| #define GREEN1 | 0x0208 |

4.3 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>                                // Mikrocontrollertyp
#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;                                           // Hilfsvariablen;
    unsigned char uc_va1,color_toggle=0;

    char LIBVer[] = dMCB32_LibVersion;                    // Option: Zeige Lib Version an
    InitTouchScreen();                                    // Init. der Display Hardware
    setScreenDir (HOR);                                  // setze Richtung Display. 0,0 bei Resetztaste
    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,WHITE,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_va1 =0;uc_va1<80;uc_va1++){
        for (t=0; t<100;t++){
            plotDot(140+uc_va1+t,115+t,uc_va1*t*8);   // Zeichne mit plotDot() ein Muster
        }
    }
    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);                           // Konfiguriere GPIOE für
    GPIOE->CRH  &= 0x00000000;                         // General purpose output push-pull, 2MHz
    GPIOE->CRH  |= 0x22222222;
    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 Berührungsstatus auf den Screen
        uc_va1 = getTSCtouched();                         // Hole Touchwert 0,1 Debugging
        GPIOPutByte("PEH",getTSCtouched());               // zeige via LED ob Touch gedrückt wurde
        if(uc_va1==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
        }
    }
}
```



5 Übung I2C

5.1 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.

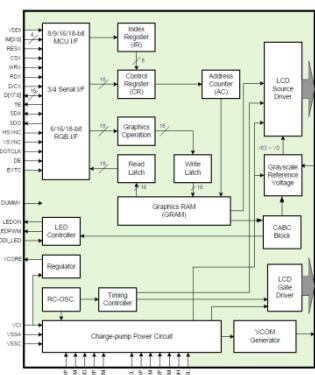
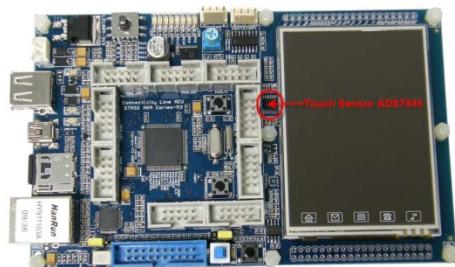
5. Programmiere, wenn die SW kommentiert und alle Messungen erledigt sind, ein Laufflicht auf dem I2C Expander. Schliessen Sie externe LEDS an oder machen Sie Messungen mit dem LogikAnalyzer um die Funktion zu zeigen.

Alles Abspeichern für Notengebung.

6 Anhang Grafikhardware

6.1 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 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.

6.2 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:莫斯I3 | 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:莫斯I | PE5 (Out) |
| PENIRQ | IRQ to ARM | GPIO: IRQ | PE3 (Inp) |

7 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>           // Mikrocontrollertyp
#include "TouchP0P1.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

```
int main ( void )              // Hauptprog., ohne return bei Keil
{
  InitTouchP0P1 ("1");         // Touchscreen aktiv,
                                // horizontal gedreht
                                // LSB rechts

  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
  }
}
```

**Keine sfr und
sbit mehr!**

Main verlangt int

InitTouchP0P1 ("0"),
wenn nur P0,P1 und
ohne Touchscreen

Verzögerung
vom Typ long mit
Wert 12 / μ s

7.1 Wichtig für das Funktionieren neuer Projekte

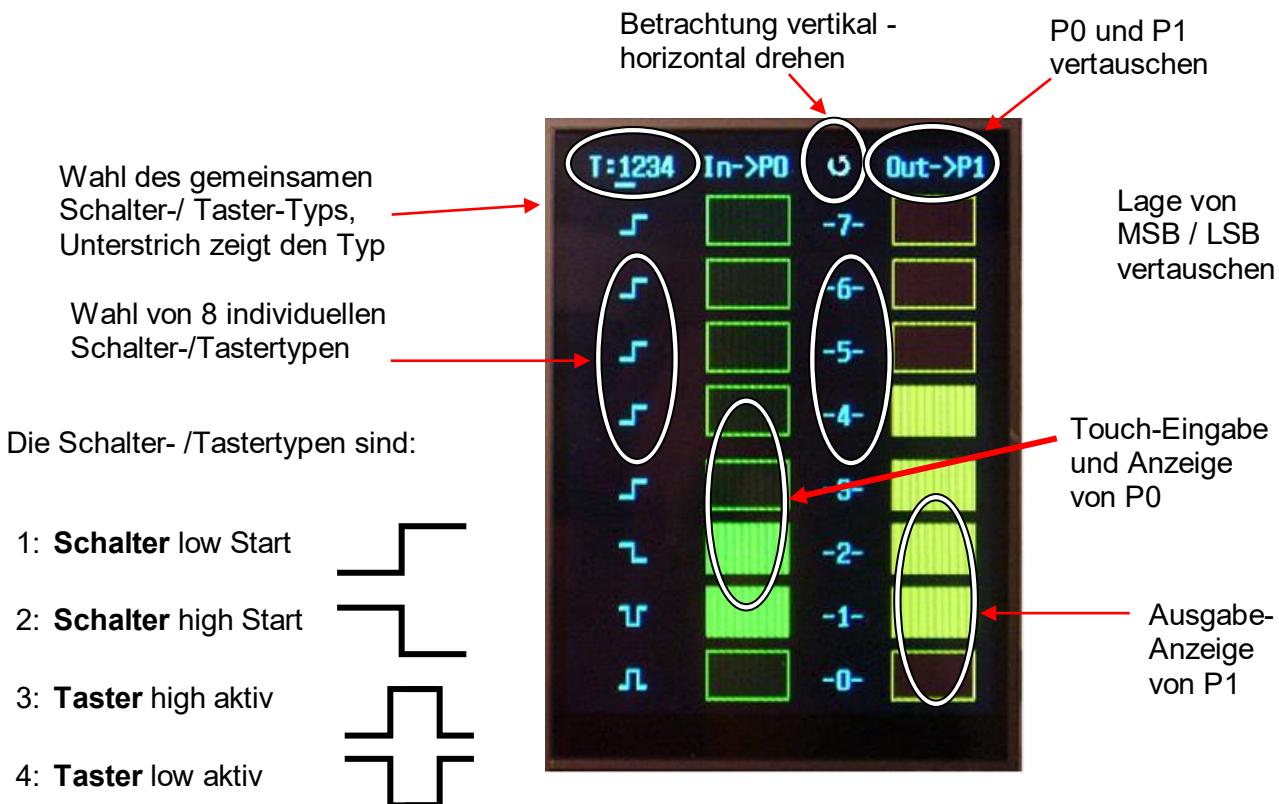
Wichtig: Bei der Erstellung eines neuen Projektes im Schulbereich, also Vorbereitung für 8Bit-Programme „Elektroniker“ mit Port P0, P1 und Touchscreen ist folgendes zu beachten.

Kopieren Sie in jedes neue Projektverzeichnis diesen zwei Dateien:

- TouchP0P1.h (REV C oder REV D)
- TouchP0P1.lib (REV C oder REV D)

8 Anhang Touchscreen Kontrolle am µC-Board MCB32

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



8.1.1 Touchscreen Kontrolle aus dem Quellcode

Der Projekt-Ordner muss TouchP0P1.h und TouchP0P1.lib enthalten:

Im Projekt-Manager sind [zum stm32f10x.h](#) die Quelldatei.c und die Lib „TouchP0P1.lib“ aufzunehmen.

```
/* Beschreibung der 8 Bit P0- und P1-Kontrolle über den Touchscreen des MCB32
***** */
#include <stm32f10x.h>           // Mikrocontrollertyp
#include "TouchP0P1.h"             // P0-, P1-Definition. Angepasst für REV C oder D

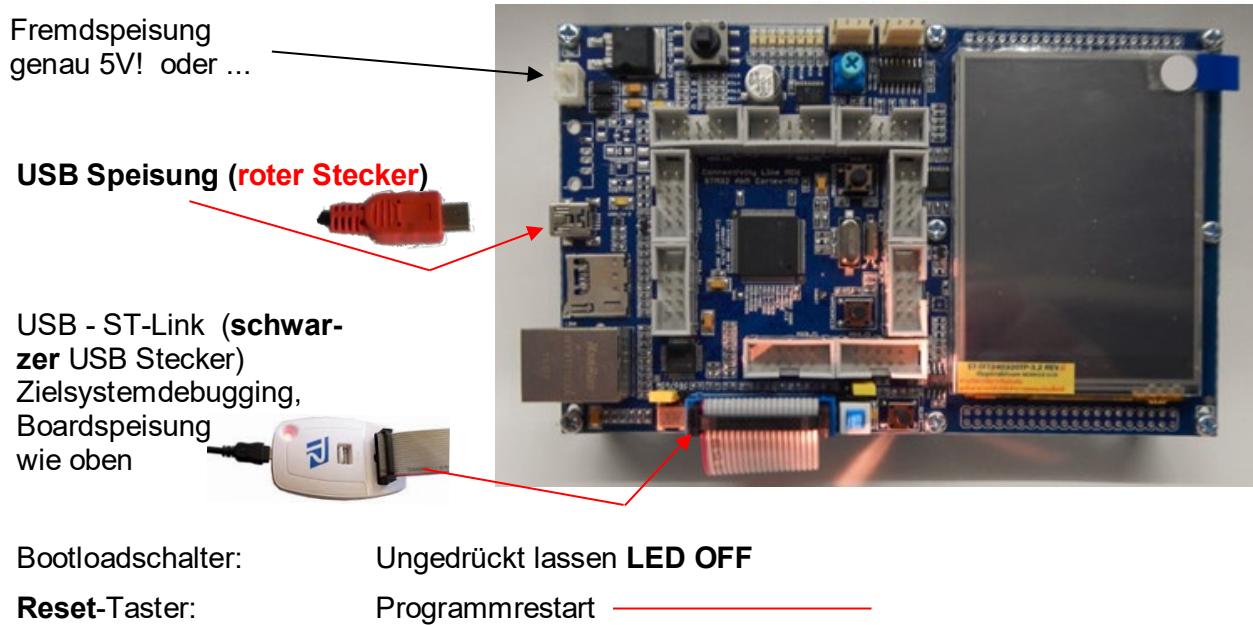
void main(void)                  // Hauptprogramm
{
    InitTouchP0P1 ("1");          // Touchscreen aktivieren. Bei „0“ ist SysTick
                                  // -Timer abgeschaltet.
    while(1) { }                 // Benutzerprogramm
}

1) InitTouchP0P1 ("0");          Der Touchscreen bleibt ausgeschaltet
                                  P0 ist als Input, P1 als Output konfiguriert

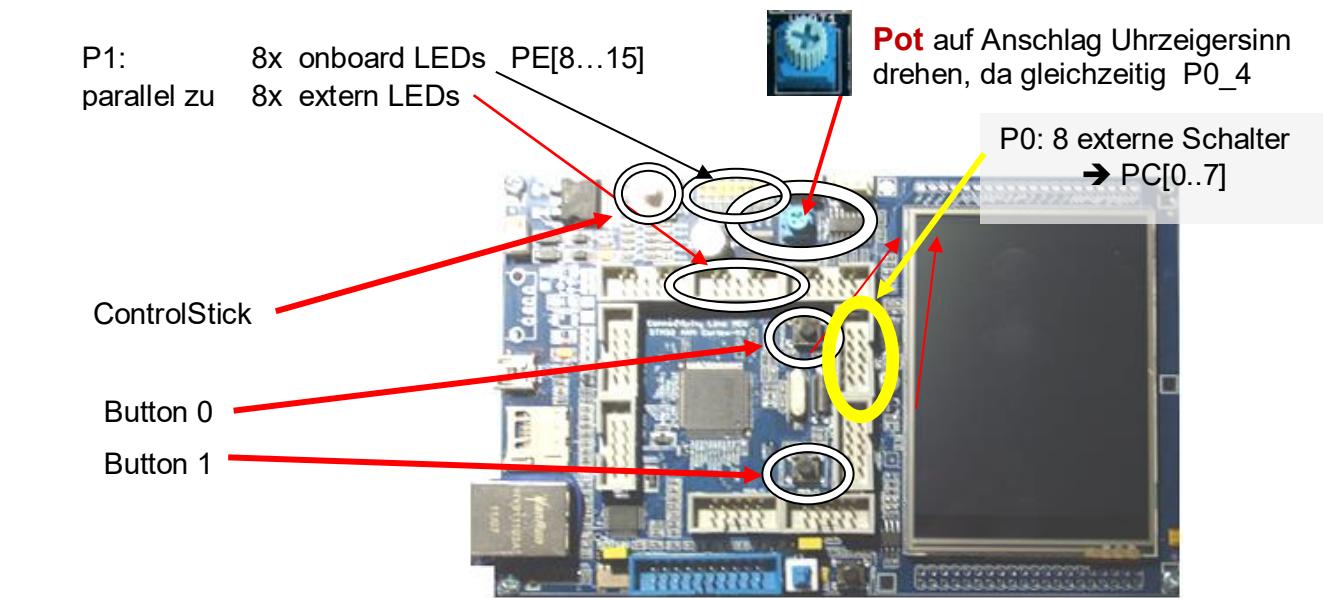
2) InitTouchP0P1 ("1") .. ("1 r m p"); Der Touchscreen wird aktiviert und konfiguriert,
                                         einfache Konfiguration mit InitTouchP0P1 ("1").
                                         1...4: Gemeinsamer Schalter-/Tastertyp
                                         p:   P0 aussen, sonst mittig.
                                         m:   MSB oben/rechts, sonst unten-links.
                                         r:   Rotiert horizontal, sonst vertikal.
```

9 Anhang Anschlüsse am µC-Board MCB32

9.1.1 STLINK, Schalter, Potentiometer, P0, P1



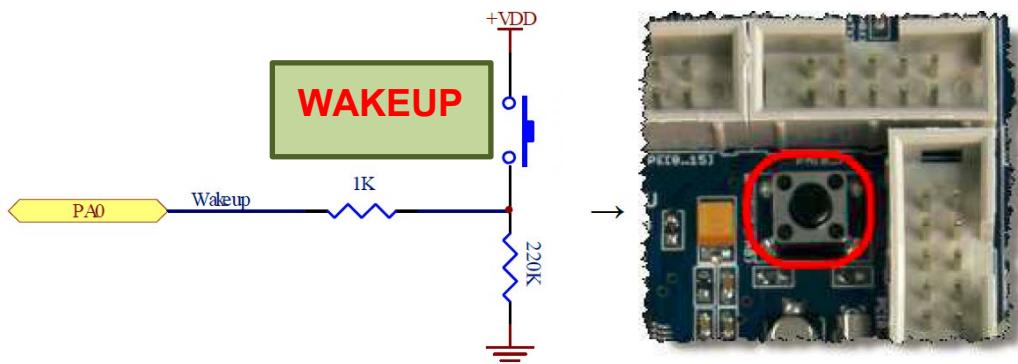
Digitale Ein- und Ausgaben am µC-Board MCB32. **ACHTUNG** mit Potentiometer (Pot)



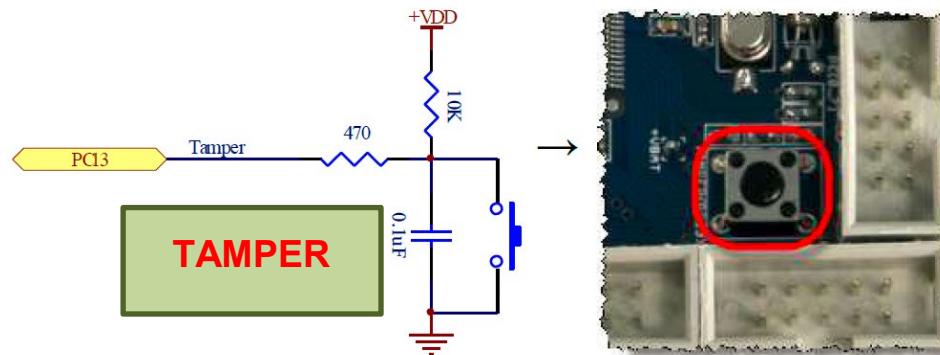
```
// In TouchP0P1.h definierte Pin-Bezeichnungen PA_0 .. PD_11, ohne Bezeichner wie Button .. !
char Button0      = PA_0;           // Bitwert 1/0, aktiv low, prellt wenig
char Button1      = PC_13;          // Bitwert 0/1, aktiv high

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
```

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



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



9.1.4 Potentiometer (PC4) // 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.



9.1.5 LED von Port P1 auf Board aktivieren

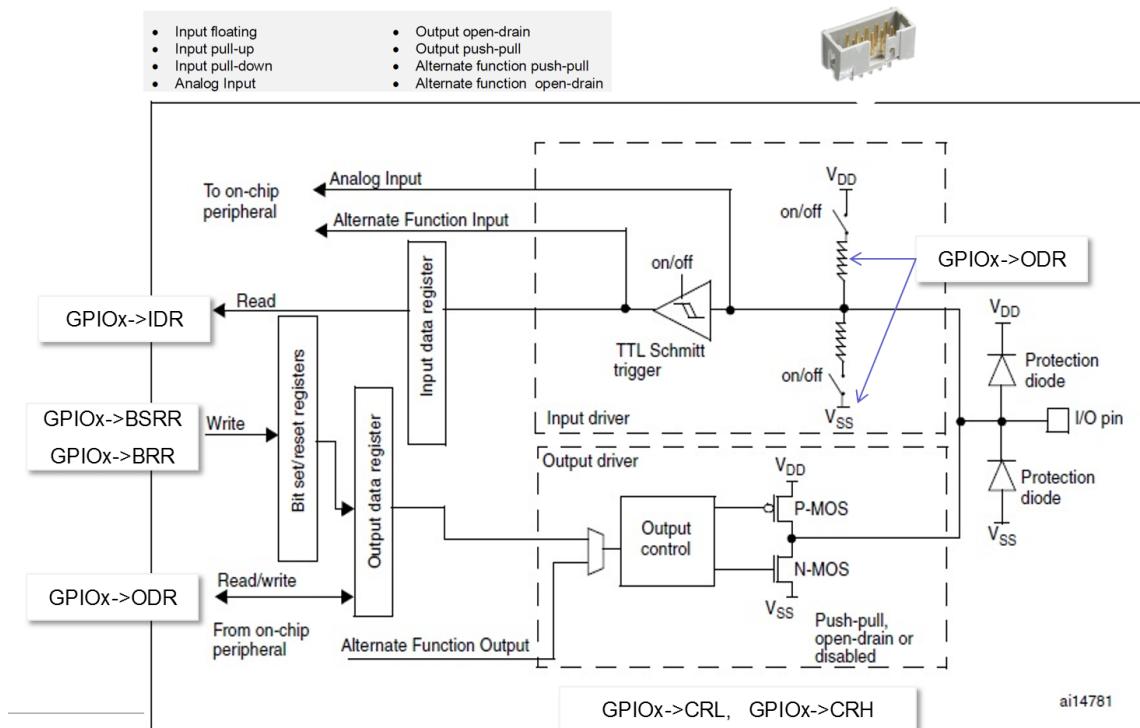
Mit den folgenden Befehlen wird Port PE[8..15] so gesetzt, dass die LEDs auf dem Board parallel zu dem Display auch aktiv angesteuert werden. Damit leuchten die LEDs gleich wie auf dem Display.

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
```

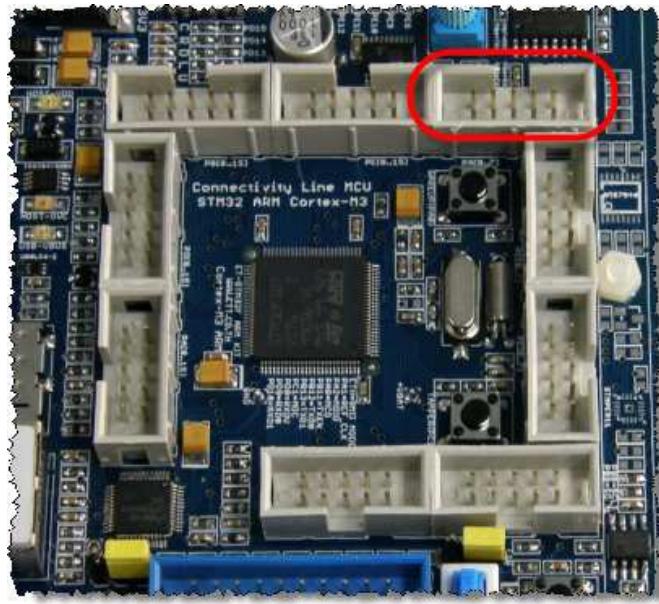
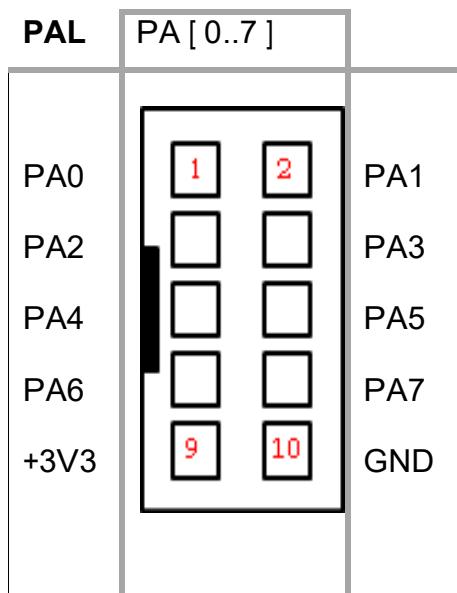
9.1.6 Übersicht über die Hardwarestruktur eines Pins



9.2 Port PA Pin 0..7

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

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



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. .

9.2.2 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 |



10 Anhang: Interrupt Vektorliste und Servicefunktionsaufrufe

Interrupt and exception vectors

| Position | Priority | Type of priority | Acronym | Description | Address |
|----------|----------|------------------|---------------|--|-------------|
| - | - | - | | 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 Instr | 0x0000_002C |
| 4 | settable | Debug Monitor | | Debug Monitor | 0x0000_0030 |
| - | - | - | | Reserved | 0x0000_0034 |
| 5 | settable | PendSV | | Pendable request for system service | 0x0000_0038 |
| 6 | settable | SysTick | | System tick timer | 0x0000_003C |
| 0 | 7 | settable | WWWDG | 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 Line1 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 | I ² C1 event interrupt | 0x0000_00BC |
| 32 | 39 | settable | I2C1_ER | I ² C1 error interrupt | 0x0000_00C0 |
| 33 | 40 | settable | I2C2_EV | I ² C2 event interrupt | 0x0000_00C4 |
| 34 | 41 | settable | I2C2_ER | I ² C2 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 int | 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 EXTIO_IRQHandler(void)
```

```
WWDG_IRQHandler
PVD_IRQHandler
TAMPER_IRQHandler
RTC_IRQHandler
FLASH_IRQHandler
RCC_IRQHandler
EXTIO_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
RTCAutoWakeup_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
```

NVIC Nummerierung

11 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);
    ...
}
```

12 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.

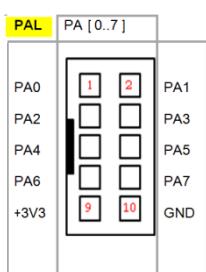
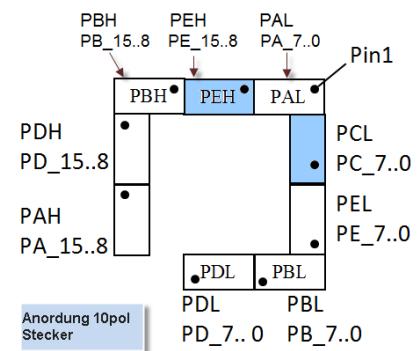


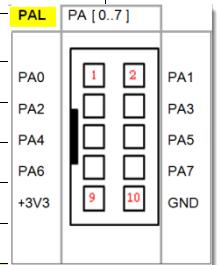
Bild Links: Portnummierung von PAL

Achtung: Alle Ports dürfen nicht mit mehr als 3,3V beschaltet werden. Falsche Handhabung führt zur Zerstörung des Kontrollers. 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_RXD0 | Ethernet LAN |
| PB5 | Free_6. | - | PB13 | RMII_RXD1 | 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 |



13 Referenzen

- [1] ST, «ARM_STM_Reference manual_V2014_REV15,» ST, 2014.
- [2] R. Weber, «Projektvorlagen (div) MCB32,» 2013ff.
- [3] J. Yiu, The definitive Guide to ARM Cortex-M3 and M4 Processors, 3 Hrsg., Bd. 1, Elsevier, Hrsg., Oxford: Elsevier, 2014.
- [4] R. Jesse, Arm Cortex M3 Mikrocontroller. Einstieg und Praxis, 1 Hrsg., www.mitp.de, Hrsg., Heidelberg: Hüthig Jehle Rehm GmbH, 2014.

14 Anhang Wichtige Dokumente

Die folgende Liste zeigt auf die wichtigsten Dokumente welche im WEB zu finden sind. Beim Suchen lassen sich noch viele nützliche Links finden.

- Datenblatt{ XE "Datenblatt" } ([STM32F107VC](#)) Beschreibung des konkreten Chips für Pinbelegung etc.
- Reference Manual ([STM32F107VC](#)) (>1000Seiten in Englisch)
Ausführliche Beschreibung der Module einer Familie. Unter Umständen sind nicht alle Module im eingesetzten Chip vorhanden – siehe Datenblatt.
- Programming Manual{ XE "Programming Manual" } ([Cortex-M3](#))
Enthält beispielsweise Informationen zum Interrupt Controller (NVIC{ XE "NVIC" }).
- Standard Peripheral Library ([STM32F10x](#))
Im Gegensatz zu anderen MCUs sollen die Register der STM32{ XE "STM32" } nicht direkt angesprochen werden. Dafür dienen die Funktionen der Standard Peripheral Library.
Sie ist auf <http://www.st.com/> zusammen mit einer Dokumentation (Datei: stm32f10x_stdperiph_lib_um.chm) herunterladbar.