




Mikrocontroller MCB32

TouchLIB Befehle

2.1.2 Grafikfunktionen [1]

Grafikfunktionen:

Funktion	Beschreibung		
<code>InitTouchScreen ()</code>	Touchscreen ohne P0P1 für Text, Grafik, Peripherie		
<code>setScreenDir (DIR)</code>	Setzt die Schreibrichtung des Displays	VER	<code>setScreenDir (VER);</code>
<code>char getScreenDir()</code>	Gibt die Schreibrichtung zurück	HOR=0 und VER=1	<code>if(getScreenDir()==VER) {clearScreen (BLUE);}</code>
<code>clearScreen (color)</code>	Löscht den Bildschirm mit der angegeben Farbe. Funktioniert nur mit Einstellung <code>setScreenDir(VER)</code> .	long color	<code>clearScreen(BLACK);</code>
<code>InitTouchP0P1 ()</code>	Hier wird neben dem Display auch der Komfort für P0 und P1 initialisiert. Schulübungen mit Bitmanipulation.		<code>InitTouchP0P1("1");</code>
<code>plotDot (X,Y,color) ○</code>	Zeichnet ein DOT an der Stelle X,Y mit der Farbe color. a,b,c unsigned int.		<code>plotDot(</code>
<code>Circle (X,Y,Radius, Tick, Color, Fill)</code>	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. Fill =1 füllt angegeben ist so wird der Kreis gefüllt den Kreis. 	div	<code>circle(50,</code>

MCB32 - Embedded Programmierung Lib Befehle

Version: 2002.30

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	Library TouchPOP1.Lib	4
3.1	Resourcennutzung für P0, P1 Betrieb (8-Bit Kapselung)	4
3.1.1	TouchPOP1. Lib	4
3.2	Hardwarefunktionen	4
3.2.1	Port P0 und P1	5
3.2.2	Weitere, belegte Pins	5
3.2.3	Peripheriefunktionen	6
3.2.4	5x GPIO General Purpose Input Output	6
3.2.5	2 x ADC Analog Digital Converter (0...3.3V, 8Bit Auflösung)	6
3.2.6	2 x DAC Digital Analog Converter (0...3.3V, 8Bit Auflösung)	6
3.3	Serielle Schnittstellen [2]	7
3.3.1	USART1 & USART2	7
3.3.2	Elektrische Eigenschaften / Pegel RS232/USART Schnittstelle [3]	8
3.3.3	Pin Belegung 4pol Stecker auf MCB32	8
3.4	Die Timer des STM32F107xx (Kapitel 15 REF Manual)	9
3.4.1	Überblick über die Timer	9
3.4.2	Blockdiagramm der "General-purpose TIM2 ..TIM5"	9
3.4.3	4 x General Purpose Timer (20us6.5s)	10
3.4.4	4 x General Purpose Counter	11
3.5	Interrupt Funktionen	12
3.5.1	4 x Externe Interrupt Requests (Vereinfacht, jede pinNr nur 1x)	12
3.5.2	6 x Interne Interrupt Requests	12
3.5.3	Interrupt Handler (ACHTUNG: die Namen/Bezeichner sind vorgeschrieben)	12
4	Grafikprogrammierung	13
4.1	Fonts	13
4.2	Koordinaten Bildschirm	13
4.3	Lib Befehle für Grafikfunktionen und sprintf()	13
4.3.1	Grafikfunktionen [1]	14
4.3.2	Touch-Funktionen	15
4.3.3	Touchscreen Textfunktionen	15
4.4	Farbliste	16
4.5	Musterprogramm für Grafikfunktionen	17
5	Anhang Grafikhardware	17
5.1	Hintergrund	18
5.2	Hardwarenahe Beschreibung der Displayansteuerung	18
6	Anhang: Umstellung von C51-Code auf ARM32-Code	19
6.1	Wichtig für das Funktionieren neuer Projekte	19
7	Anhang Touchscreen Kontrolle am µC-Board MCB32	20
7.1.1	Touchscreen Kontrolle aus dem Quellcode	20
8	Anhang Anschlüsse am µC-Board MCB32	21
8.1.1	STLINK, Schalter, Potentiometer, P0, P1	21
8.1.2	Button 0 / Wakeup (Pin:PA0); nicht gedrückt PA_0=0	22
8.1.3	Button 1 / Tamper (Pin:PC13); nicht gedrückt PC_13=1	22
8.1.4	Potentiometer (PC4) // resp. P0_4 (Library)	22
	22	
	22	
8.1.5	LED von Port P1 auf Board aktivieren	23
8.1.6	Übersicht über die Hardwarestruktur eines Pins	23
8.2	Port PA Pin 0..7	23
8.2.1	Steckerbelegung für 10pol. Stecker PA[0..7]	24
8.2.2	Original Belegung PA[0..7]	24
9	Anhang: Interrupt Vektorliste und Servicefunktionsaufrufe	25
10	Anhang: SysTick Timer	26
11	Anhang: Port Pin Liste MCB32	27
12	Referenzen	29

2 Gefahren Portbeschaltung

Achtung: Die Ports 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 Library TouchP0P1.Lib

Mit der Library, welche beim MCB32-Kit mitgeliefert wird kann der Benutzer ohne Detailkenntnisse des verwendeten Prozessors relativ schnell in mit der Programmierung anfangen und sich ein Wissen aufbauen. [1]

Die Library umfasst neben den Befehlen für den einfachen Betrieb der P0-P1-Ports zusätzlich Befehle für die Grafikansteuerung sowie auch Befehle für das einfache Handling der Hardware (AD-DA Wandler usw.)



3.1 Ressourcennutzung für P0, P1 Betrieb (8-Bit Kapselung)

Für den Einstieg in die Programmierung mit Hardware (1-4 Semester) wird der Touchscreen benutzt. Dort wird die Ein- und die Ausgabe auf die Ports P0/P1 grafisch dargestellt.

Die folgende Tabelle beschreibt den Init-Befehl für das Setup des Touchscreens.

Mit InitTouchP0P1 ("1") wird der Betrieb mit dem Port P0 und P1 aufgesetzt. Dabei läuft im Hintergrund ein 1ms Timer welcher das Handling der Ein- und Ausgaben übernimmt. Damit steht auch die Funktion *delay_ms(ms)* zur Verfügung. Sobald externe Schalter und LED angeschlossen werden, kann dieser Betrieb mit dem Befehl InitTouchP0P ("0") abgeschaltet werden.

Mit InitTouchP0P1 ("0"); ist der SysTick_Handler() nicht aktiv, daher keine Behandlung von *delay_ms(ms)*.

3.1.1 TouchP0P1. Lib					
TouchP0P1.lib 8..17KByte	P0/P1 definiert an Ports	P0/P1 auf Screen 	Touch, Grafik, Text, Periphere Funktionen 	Sys-Timer belegt	SysTick_Handler Ein/Aus
InitTouchP0P1 ("1.."); ¹	ja	ja	ja	1ms	aktiv
InitTouchP0P1 ("0"); ²	ja	nein	ja	--	aus

Der SysTick_Handler() wird via einen Interrupt aufgerufen, wenn der System-Timer den Wert 0 erreicht.

3.2 Hardwarefunktionen

¹ *SysTick_Config (72* 1030); // SysTick_Handle() all 1000us/ (Initializes the System Timer and its interrupt, and starts the System Tick Timer. Hinweis: Die Funktion SysTick_Config() erwartet als Parameter den Zählerwert für den SysTick-Timer. Mit dem Parameter SystemCoreClock/100 legen wir fest, dass nach xyz Systemtaktten der SysTick-Interrupt ausgelöst und der SysTick_Handler aufgerufen wird.*

² *SysTick_Handler() wird nicht gebraucht. delay_ms(ms) steht nicht zur Verfügung.*

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**

3.2.1 Port P0 und P1

Die beiden Ports P0 und P1 sind wie folgt definiert:

Eingabe Port P0: PC[0..7] resp. PC_{Low}

Ausgabe Port P1: PE[8..15] resp. PEH_{igh} und 8x LEDs direkt an PEH

3.2.2 Weitere, belegte Pins

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

3.2.3 Peripheriefunktionen

Peripheriefunktionen:			Version und Stand: 2002.30
-----------------------	--	--	----------------------------

Funktion	Beschreibung / Beispiel	Parameter	Hinweis zur Hardware
3.2.4 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	
<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	

3.2.5 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.	Pin: siehe rechts	
<code>ADCInit(1, "PC4"); // ADC1 an PC4</code>			
<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: 2002.30
--	--	----------------------------

3.2.6 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.

<code>DACInit(1/2);</code>	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.	Pin: siehe rechts	
<code>DACInit(1); // DAC1 an PA4</code>			
<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>		

3.3 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: 2002.30
Funktion	Beschreibung / <i>Beispiel</i>	Pa- rame- ter	Hinweis zur Hardware
3.3.1 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 36MHz 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		<pre>graph TD subgraph USART2 [USART2] direction TB U2[USART2 (an USB)] U2_I[USART2_ IRQHandler] end subgraph USART1 [USART1] direction TB U1[USART1] U1_I[USART1_ IRQHandler] end U2 -- TX PD5 --> TX2[] TX2 --> TX2_label[TX PD5] TX2_label --> U2_I U2_I -- RX PD6 --> RX2[] RX2_label[RX PD6] --> RX2 --> U2 U1 -- TX PA6 --> TX1[] TX1_label[TX PA6] --> TX1 --> U1_I U1_I -- RX PA7 --> RX1[] RX1_label[RX PA7] --> RX1 --> U1</pre>
	Aktiviert den UART-Clock und Initialisiert auf19200Bd 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 USARTtoRead (1/2);	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			

3.3.2 Elektrische Eigenschaften / Pegel RS232/USART Schnittstelle [3]

Achtung. Die Signalspannung darf nie zwischen +3 -3V zu liegen kommen.

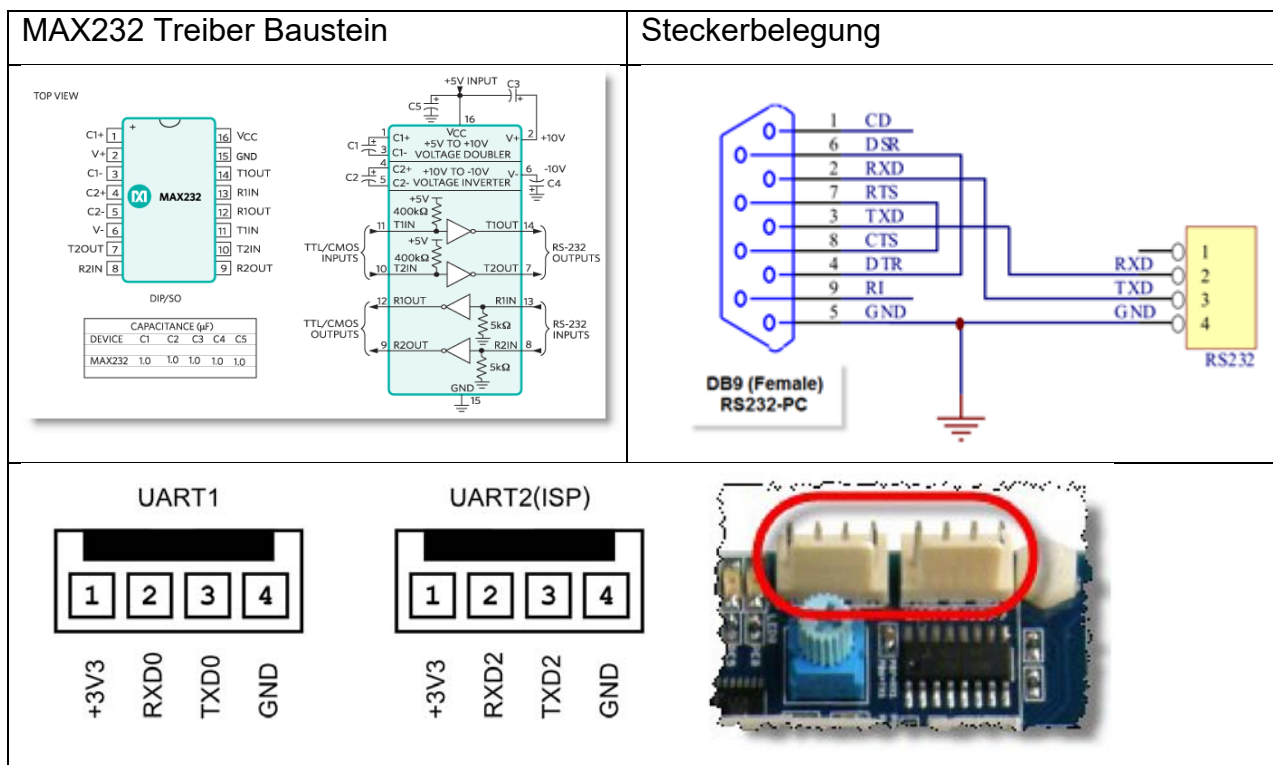
Logik Pegel uP	Signalpegel auf der RS232 Leitung	Spannung auf der Datenleitung	Spannung auf den Steuer- und Meldeleitungen
1	L (low)	-3V -15V	+3V +15V
0	H (high)	+3V +15V	-3V -15V

Bei RS232 Verbindungen ist die maximale Kabellänge vom Kabeltyp, von der Übertragungsrate sowie der Signalstärke abhängig. Je höher die Rate desto kürzer das Kabel. Im Normalfall ist ein Kabel zw. 6 – 8 m lang. Bei optimalen Voraussetzungen kann es auch bis zu 40m sein.

3.3.3 Pin Belegung 4pol Stecker auf MCB32

Pin	Bezeichnung MCB32	Verbunden mit Pin Nr. MAX232	Pegel und Spannung auf der Datenleitung
1	VDD		
2	Receive Data RxD	Pin 13 und Pin 8	H (high) / +3V +15V L (low) / -3V -15V
3	Transmitt Data TxD	Pin 14 und Pin 7	
4	GND		GND

Nachfolgende die Zusammenhänge aus Sicht der Hardware



3.4.3 4 x General Purpose **Timer** (20us6.5s)**Normal Resolution Timer:** $\Delta t = 100\mu s$

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

Prototyp:
void TimerInit(char Nr, int Time100us, char IRQPrio);

2..5 = Timer Nummer.

n muss ≥ 2 und ≤ 65000 sein,
 $n \cdot 100\mu s = \text{Zeit}$ **Bereich:** 200us ... 6.5sInterrupt-Priorität: 0 = AUS, 1 höchste,
15 tiefste

Aktiviert den TimerClock und zählt mit dem Systemclock.
Bei Ablauf wird das Timer Flag 1 und wenn IRQPrio > 0 ist, löst jeder Timer seinen eigenen Interrupt aus.

TimerInit (2, 5000, 0); // T2 auf 500ms,
// ohne Interrupt

High Resolution Timer: $\Delta t = 10\mu s$

TimerInitHr
(2..5, n*10us, IRQPrio0..15);

Prototyp:
void TimerInitHr(char Nr, int Time10us, char IRQPrio);

2..5 = Timer Nummer.

n muss ≥ 2 und ≤ 65000 sein,
 $n \cdot 10\mu s = \text{Zeit}$ **Bereich:** 20us ... 650msMehr Infos siehe oben bei **TimerInit**

TimerInitHr (5, 10, 0); // T5 auf 100us
// ohne Interrupt

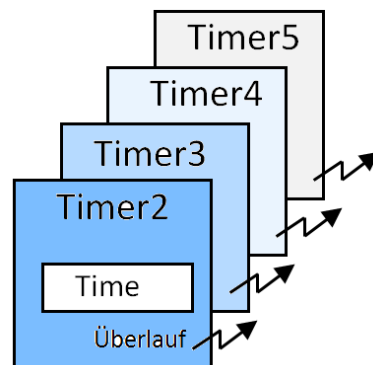
char **TimerGetFlag**(2..5);

Liefert 0 oder 1 zurück. 1= Überlauf

var = TimerGetFlag(2); // 0 / 1=Überlauf

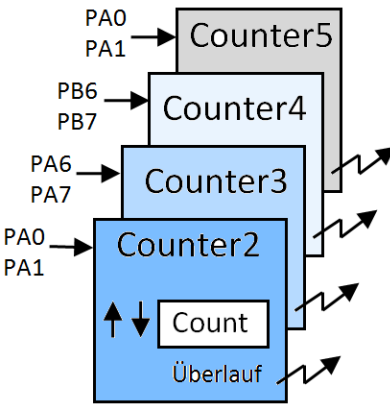
int **TimerGetTime**(2..5);

Liefert einen 16Bit Wert zurück.

var = TimerGetTime(2); // Zeit in $n \cdot 100\mu s$ 

```
//-----
// Auszug ausLib-Code für TimerInitHr (Hr steht für HighResolution =dt=10us ab 20us .... 650ms))
//-----
void TimerInitHr(char Nr, int Time10us, char IRQPrio)
{
    if(Time10us==1){Time10us=2;} // kleinst möglicher Wert ist 20us, 10us geht nicht
    switch(Nr)
    {
        case 2: RCC->APB1ENR |= 1<<0; // Timer2 Clock ON
                TIM2->PSC = 720-1; // 72MHz:720=100kHz
                TIM2->ARR = Time10us-1; // 2500x10us = 25ms
                TIM2->CNT = 0x0000; // Zaehlstart nullen
                TIM2->SMCR &= 0xFF00; // Timer2 = interner Clock
                if(IRQPrio>0)
                {
                    TIM2->DIER |= 1; // UIE Intr Enable
                    NVIC->IP[28] = IRQPrio<<4;
                    NVIC->ISER[0] |= 1<<28; // Intr Set Enable
                }
                TIM2->CR1 |= 1; // Timer2 ON
            break;
        ..... USW
        case 5: RCC->APB1ENR |= 1<<3; // Timer5 Clock ON
                TIM5->PSC = 720-1; // 72MHz:720=100kHz
                TIM5->ARR = Time10us-1; // 2500x10us = 25ms
                TIM5->CNT = 0x0000; // Zaehlstart nullen
                TIM5->SMCR &= 0xFF00; // Timer5 = interner Clock
                TIM5->CR1 |= 1; // Timer5 ON
                if(IRQPrio>0)
                {
                    TIM5->DIER |= 1; // UIE Intr Enable
                    NVIC->IP[50] = IRQPrio<<4;
                    NVIC->ISER[1] |= 1<<(50-32); // Intr Set Enable
                }
            break;
    }
}
//----- Ende of TimerInitHr() -----
```

3.4.4 4 x General Purpose Counter

<p>CounterInit (2..5, "Pin", UpDn0/1, IRQPrio0..15);</p>	<p>Nr: 2..5 Einer von 4 Countern Pin: "PA0" ... Fest zugeteilte Zählpins: PA0, PA1, PA6, PA7, PB6, PB7 UpDn: 0, 1 0: Aufwärts-, 1 Abwärtszähler IRQPrio: 0, 1..15 Interruptpriorität; 0 = Off, 1 höchste, 15 tiefste. Aktiviert den Counterclock und zählt die Pinimpulse. Bei Überlauf und wenn IRQPrio > 0 löst jeder Counter seinen eigenen Interrupt aus.</p>	
<p>CounterPutCount(2..5, ival);</p>	<p>Int_val: 0..65535. Schreibt den übergebenen 16Bit-Wert in den Zähler 2..5</p>	
<p>int CounterGetCount(2..5);</p>	<p>Liefert einen 16Bit Zählwert zurück ivar = CounterGetCount(2); // return count</p>	

```
//-----
// Beispiel 1 für Interrupt mit Timer 5
//-----

#include "TouchP0P1.h" // P0-,P1-,Touchscreen
unsigned char b=0;

void TIM5_IRQHandler(void) // IRQ Handler
{
    // Immer zuerst:
    IRQClearFlag("T5"); // IRQClearFlag(..)
    b= (ADCGetVal(1)); // Lese ADC alle 1s
}

// -----
int main(void) // Hauptprogramm
{
    InitTouchP0P1("1"); //P0P1-Touchscreen ON
    ADCInit(1,"PC4"); // Potentiometer
    // Set PEH to Push Pull
    GPIOE->CRH &= 0x00000000;
    GPIOE->CRH |= 0x22222222;
    TimerInit(5,10000,1); // Lies pro s ein Wert
    // ----- Endlosschleife
    while(1)
    { // Zeige Counter im H-Nibble und ADC im L-Nibble

        P1=((TimerGetTime(5)%255)<<4) | ((b>>4)&0x07);
    }
}
```

Bild 2: MusterCode für Timer 5 mit Interrupt

```
//-----
// Beispiel 2 für Interrupt mit Timer 5 und UART2
//-----

#include "TouchP0P1.h" // P0-,P1-,Touchscreen
unsigned char b=0;

void TIM5_IRQHandler(void) // IRQ Handler
{
    // Immer zuerst:
    IRQClearFlag("T5"); // IRQClearFlag(..)
    USARTWrite(2,ADCGetVal(1)); // SENDE ADC via Uart
}

// -----
int main(void) // Hauptprogramm
{
    InitTouchP0P1("1"); //P0P1-Touchscreen ON
    USARTInit(2,0); // USART 2 mit 96008N1 init.
    ADCInit(1,"PC4"); // Potentiometer
    // Set PEH to Push Pull
    GPIOE->CRH &= 0x00000000;
    GPIOE->CRH |= 0x22222222;
    TimerInit(5,10000,1); // Lies pro s ein Wert
    // ----- Endlosschleife
    while(1)
    { // Zeige Counter im H-Nibble und ADC im L-Nibble

        P1=((TimerGetTime(5)%255)<<4) | ((b>>4)&0x07);
    }
}
```

3.5 Interrupt Funktionen

Version und Stand: 2002.30

3.5.1 4 x Externe Interrupt Requests (Vereinfacht, jede pinNr nur 1x)

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

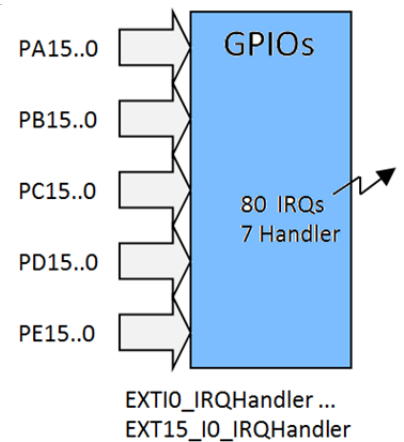
Pin[" "]: "PA0"..PE15"
Interrupt auslösender Pin

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 *EXTI0_IRQHandler()*..
EXTI15_10_IRQHandler().

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



3.5.2 6 x Interne Interrupt Requests

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

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

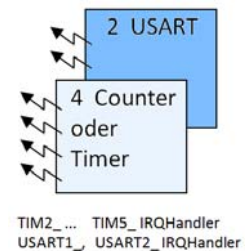
Siehe unter Timer

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



3.5.3 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:	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 Fonts

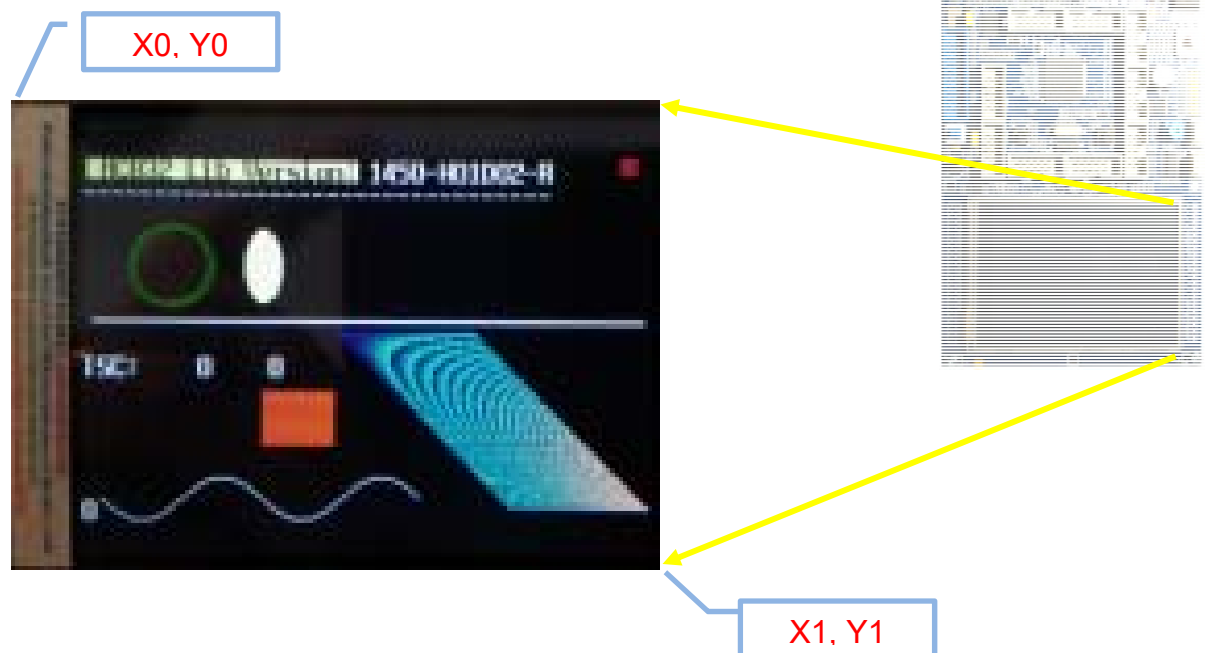
Es steht 1 Font zur Verfügung. Ein „7*11“ ASCII Font.
Der Font kann in der Library „TouchGrafik.c“ individuell erweitert werden.



4.2 Koordinaten Bildschirm

Für einige Funktionen sind die Koordinaten von Wichtigkeit. Das nachfolgende Beispiel zeigt die Anordnung wenn die Richtung für die Darstellung von Schrift auf **HOR** eingestellt ist.

Wenn `setScreenDir (HOR);` dann sieht die Situation wie folgt aus :







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

4.3 Lib Befehle für Grafikfunktionen und `sprintf()`

Grafikfunktionen:

Version und Stand: 2002.30

4.3.1 Grafikfunktionen [1]

Funktion	Beschreibung	Parameter	Beispiel
<code>InitTouchScreen()</code>	Touchscreen ohne P0P1 für Text, Grafik, Peripherie	-	<code>InitTouchScreen();</code>
<code>InitTouchP0P1 ("0/1");</code>	Neben dem Display wird auch der Komfort für P0 und P1 initialisiert. Für Schulübungen mit Bitmanipulation. <i>Siehe 3.1 betreff SysTick_Handler().</i>	Siehe Kapitel 7	Der Befehl: <code>InitTouchP0P1("1");</code> schaltet P0,P1 ein → 
<code>setScreenDir (DIR)</code>	Setzt die Schreibrichtung des Displays	HOR und VER	<code>setScreenDir (HOR);</code> <code>setScreenDir (VER);</code>
<code>char getScreenDir()</code>	Gibt die Schreibrichtung zurück	HOR=0 und VER=1	<code>if(getScreenDir()==VER)</code> <code>{clearScreen (BLUE);}</code>
<code>clearScreen (color)</code>	Löscht den Bildschirm mit der angegebenen Farbe. Funktioniert nur mit Einstellung <code>setScreenDir(VER)</code> .	long color	<code>clearScreen(BLACK);</code>
ACHTUNG: Bildschirm-Koordinaten fangen bei 0,0 an und enden bei 319,239.			
<code>plotDot (X,Y,color) ○</code>	Zeichnet ein DOT an der Stelle X,Y mit der Farbe color. a,b,c unsigned int.		<code>plotDot(120,120,WHITE);</code>
<code>Circle (X,Y,Radius, Tick, Color, Fill)</code>	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>
<code>ellipse (h,k,rx,ry,tick,color, fill)</code>	h und k beschreiben den Mittelpunkt der Ellipse. rx und ry die Radien, Tick, Color und Fill wie beim Kreis, 		
<code>rectan (x1,y1,x2,y2,tick,color, fill)</code>	Zeichnet ein Rechteck von x1,y1 zu x2,y2. Siehe 4.2.		<code>rectan(100,150,140,180,1, RED,1);</code>
<code>plotFilledRect (x1,y1,dx,dy,color)</code>	Gefülltes Rechteck von x1, y1 nach x1+dx, y1+dy mit Farbe color		<code>plotFilledRect (10, 20, 50, 60, RED);</code>
<code>textxy (String,x,y,For_col, Back_Col)</code>	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>
<code>line (x1,y1,x2,y2,thick,color)</code>	Zeichne Linie von X1,y1 nach x2,y2 mit der Dicke und der Farbe 		<code>Line(5,110,315,110,2,WHITE);</code>

Bemerkung: Printf ist in der Bibliothek nicht vorgesehen. Dafür kann aber der Befehl `sprintf()` genutzt werden. Er funktioniert wie `printf()`, ausser dass das Resultat der Ausgabe in einem Buffer landet welcher von uns genutzt werden kann: siehe Muster


```
#include <stdio.h>
char wertausgabe[20];
unsigned char Sensor1 =0x20;
sprintf(wertausgabe,"Der Sensor misst: %x", Sensor1);
textxy(wertausgabe, 10, 180, YELLOW, BLUE);

// lib für sprintf()
// Array, Buffer für sprintf(wertausgabe," ")
// Variable für Sensor 1 (Feuchte)
// Emuliere printf → Res. in buffer
// gib buffer aus mit MCB32 Lib
```

Grafikfunktionen:

Version und Stand: 2002.30

4.3.2 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>

4.3.3 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 P0P1 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>

4.4 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	0x87FD	
#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	

4.5 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;

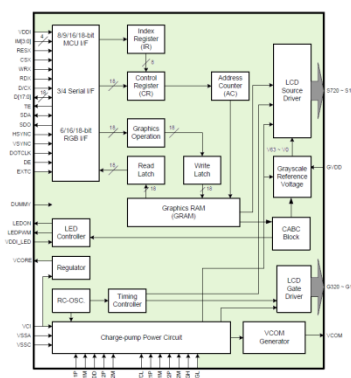
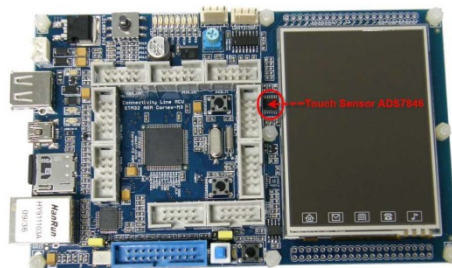
    char LIBVer[]=dMCB32_LibVersion; // Option: Zeige Lib Version an
    InitTouchScreen(); // Init. der Display Hardware
    setScreenDir (HOR); // setze Richtung Display. 0,0 bei Resettaster
    textxy(" MCB32 Lib Version:", 2, 32, BLACK, YELLOW);
    textxy(LIBVer, 160, 32, WHITE, BLACK);
    printAt(2,"----- "); // 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
        }
    }
}

```

5 Anhang Grafikhardware

5.1 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.

5.2 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)

6 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 " 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              // Start = Input Port1[7]
char bTemp = 0 ;                // 'Bit'-Variablentyp char
long t;                          // Zeitvariable

```

neue #includes

Keine sfr und
sbit mehr!

```

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

```

Main verlangt int

InitTouchP0P1 ("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 / µs

6.1 Wichtig für das Funktionieren neuer Projekte

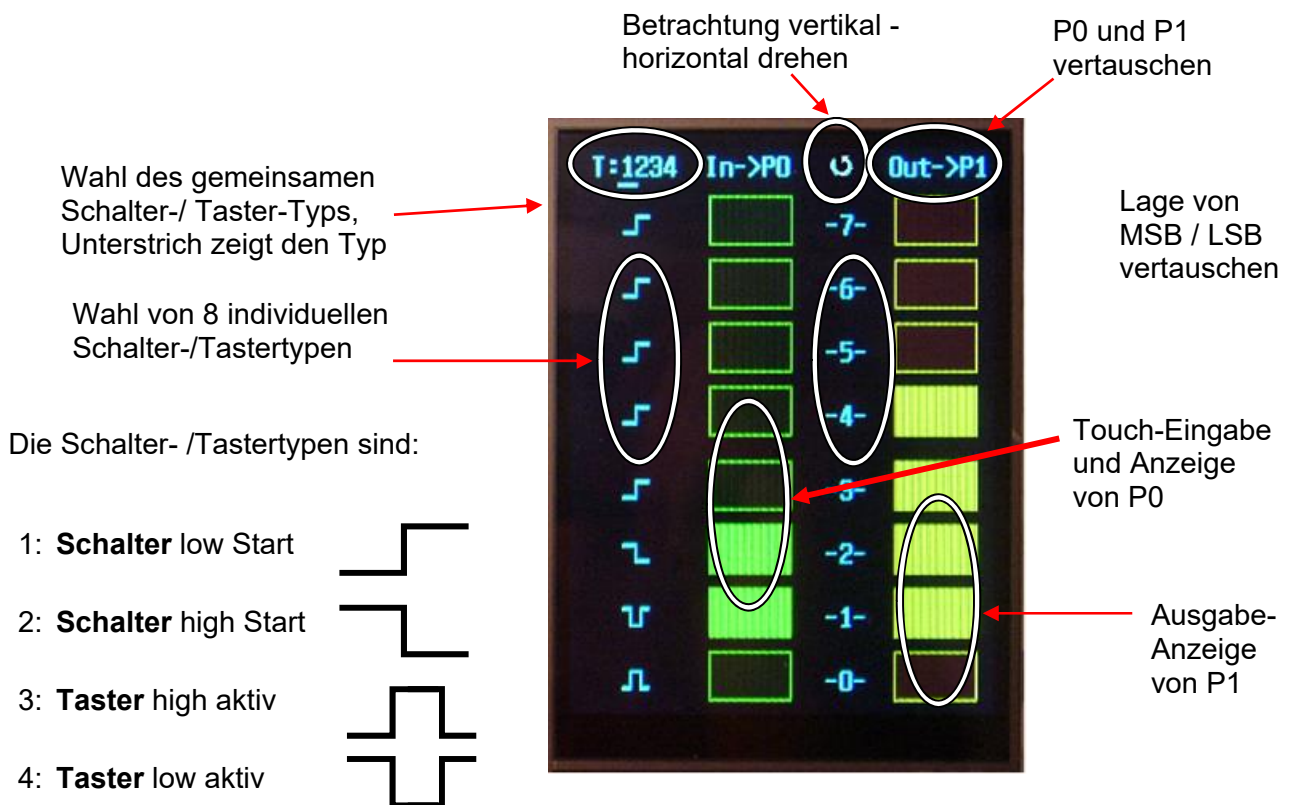
Wichtig: Bei der Erstellung eines neuen Projektes im Schulbereich, also Vorbereitung für 8Bit-Programme „Elektroniker“ mit Port P0, P1 und Touchscreen ist folgendes zu beachten.

Kopieren Sie in jedes neue Projektverzeichnis diesen zwei Dateien:

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

7 Anhang Touchscreen Kontrolle am µC-Board MCB32

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



7.1.1 Touchscreen Kontrolle aus dem Quellcode

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

Im Projekt-Manager sind 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> // Mikrokontrollertyp
#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, einfachste 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.

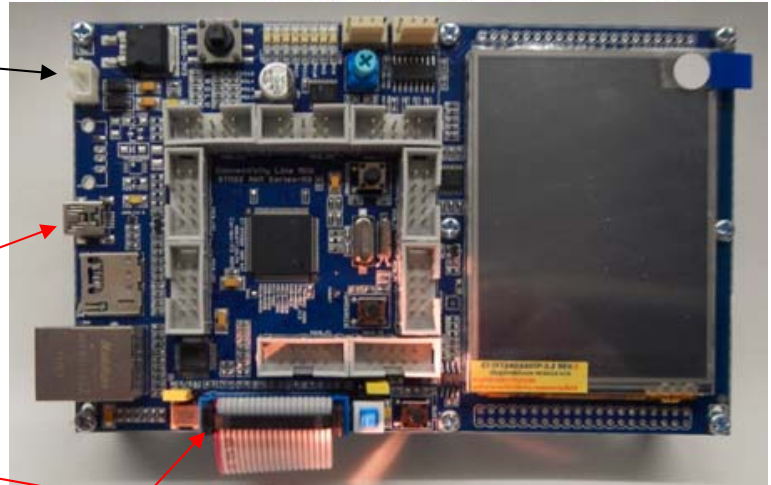
8 Anhang Anschlüsse am μ C-Board MCB32

8.1.1 STLINK, Schalter, Potentiometer, P0, P1

Fremdspeisung
genau 5V! oder ...

USB Speisung (**roter Stecker**)

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



Bootloadschalter:

Ungedrückt lassen **LED OFF**

Reset-Taster:

Programmrestart

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

P1: 8x onboard LEDs PE[8...15]
parallel zu 8x extern LEDs

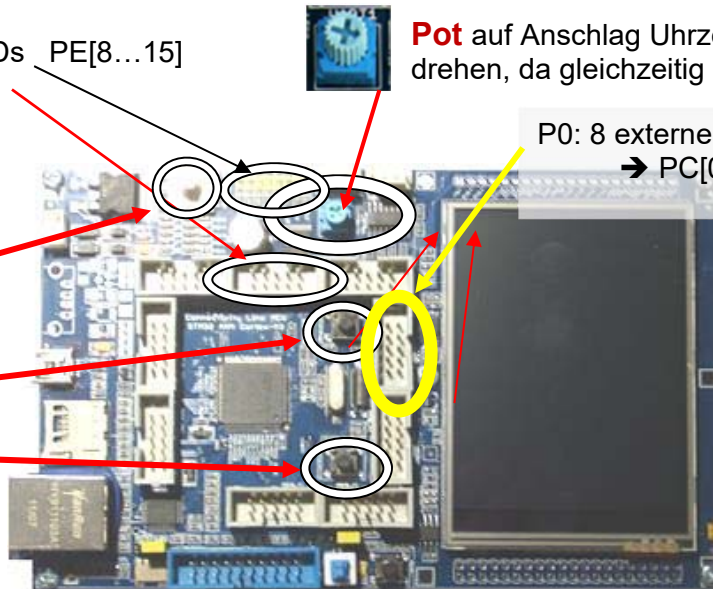
Pot auf Anschlag Uhrzeigersinn
drehen, da gleichzeitig P0_4

P0: 8 externe Schalter
→ PC[0..7]

ControlStick

Button 0

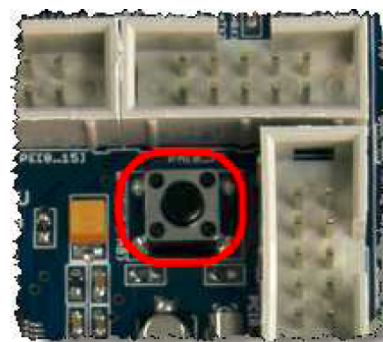
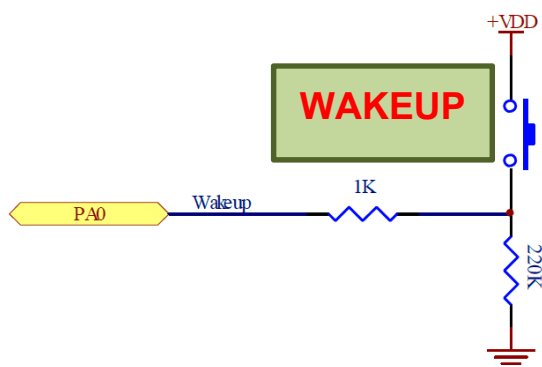
Button 1



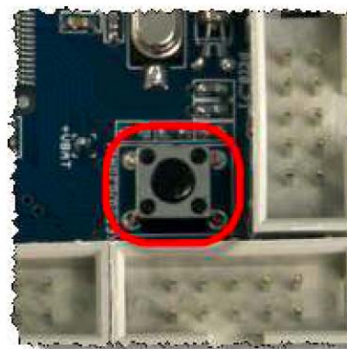
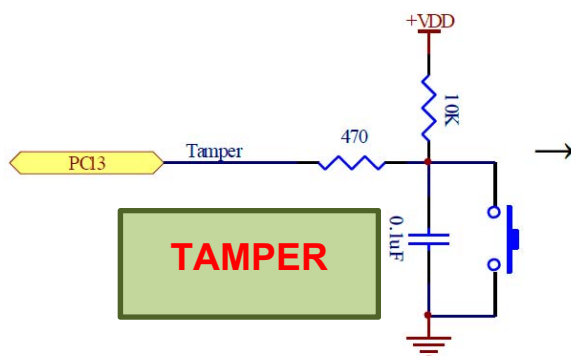

```
// In TouchP0P1.h definierte Pin-Bezeichnungen PA_0 .. PD_11, ohne Bezeichner wie Button .. !
char Button0      = PA_0;           // Bitwert 1/0, aktiv low, prellt wenig
char Button1      = PC_13;          // Bitwert 0/1, aktiv high

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
```

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



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



8.1.4 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.



8.1.5 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.

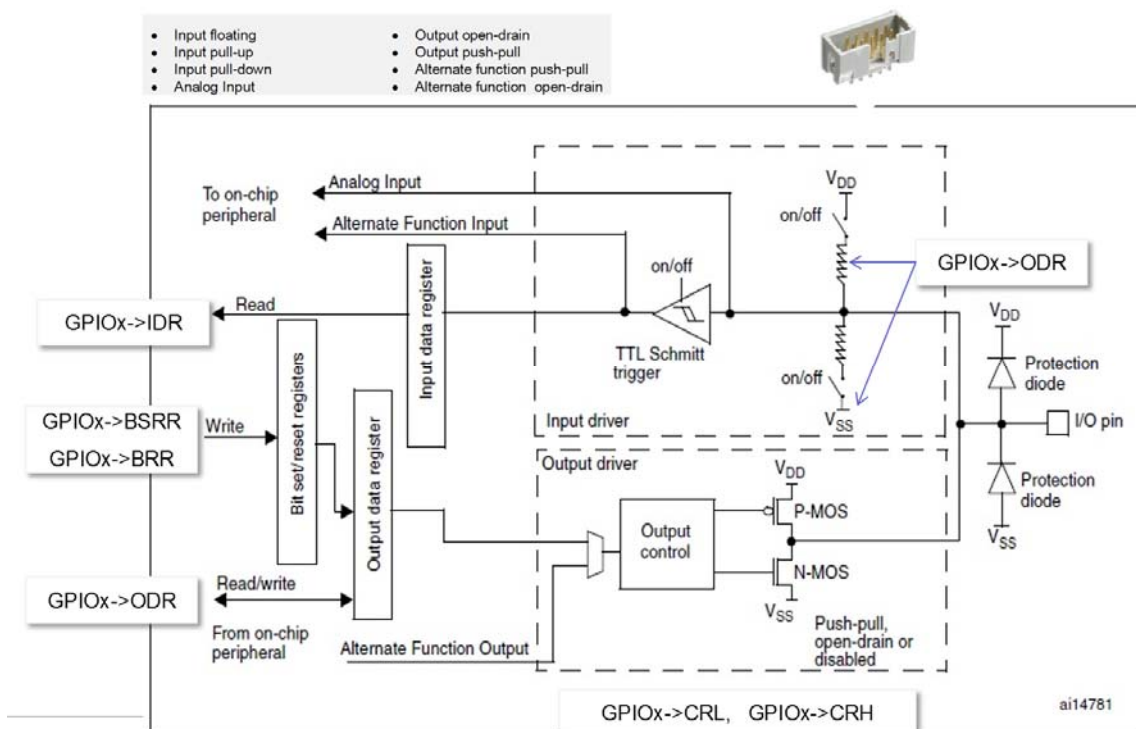
```
GPIO->CRH   &= 0x00000000;    // Configure the GPIOE for
GPIO->CRH   |= 0x22222222;    // General purpose output push-pull, 2MHz
```

```
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
    }
}
```

8.1.6 Übersicht über die Hardwarestruktur eines Pins

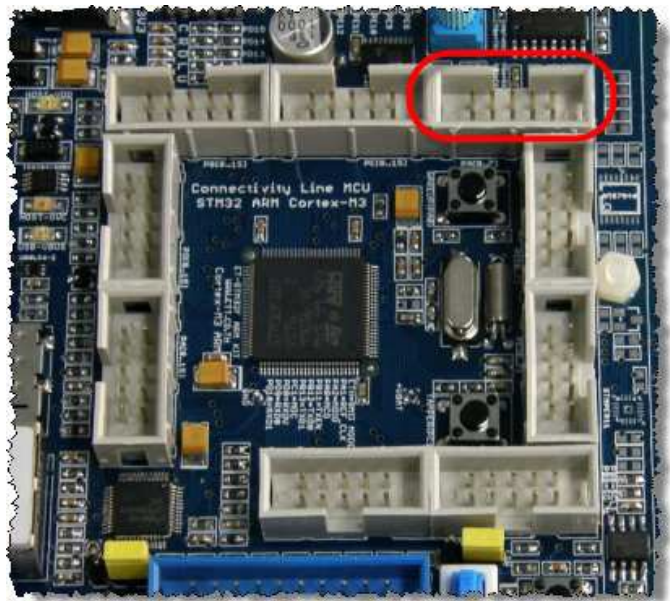


8.2 Port PA Pin 0..7

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

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

PAL	PA [0..7]	
PA0	<div>12</div>	PA1
PA2	<div></div>	PA3
PA4	<div></div>	PA5
PA6	<div></div>	PA7
+3V3	<div>910</div>	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. .

8.2.2 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	

9 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
	7	settable	WWDG	Window Watchdog interrupt	0x0000_0040
	8	settable	PVD	PVD through EXTI Line detection	0x0000_0044
	9	settable	TAMPER	Tamper interrupt	0x0000_0048
	10	settable	RTC	RTC global interrupt	0x0000_004C
	11	settable	FLASH	Flash global interrupt	0x0000_0050
	12	settable	RCC	RCC global interrupt	0x0000_0054
	13	settable	EXTIO	EXTI Line0 interrupt	0x0000_0058
	14	settable	EXTI1	EXTI Line1 interrupt	0x0000_005C
	15	settable	EXTI2	EXTI Line2 interrupt	0x0000_0060
	16	settable	EXTI3	EXTI Line3 interrupt	0x0000_0064
	17	settable	EXTI4	EXTI Line4 interrupt	0x0000_0068
	18	settable	DMA1_Channel1	DMA1 Channel1 global interrupt	0x0000_006C
	19	settable	DMA1_Channel2	DMA1 Channel2 global interrupt	0x0000_0070
	20	settable	DMA1_Channel3	DMA1 Channel3 global interrupt	0x0000_0074
	21	settable	DMA1_Channel4	DMA1 Channel4 global interrupt	0x0000_0078
	22	settable	DMA1_Channel5	DMA1 Channel5 global interrupt	0x0000_007C
	23	settable	DMA1_Channel6	DMA1 Channel6 global interrupt	0x0000_0080
	24	settable	DMA1_Channel7	DMA1 Channel7 global interrupt	0x0000_0084
	25	settable	ADC1_2	ADC1 and ADC2 global interrupt	0x0000_0088
	26	settable	CAN1_TX	CAN1 TX interrupts	0x0000_008C
	27	settable	CAN1_RX0	CAN1 RX0 interrupts	0x0000_0090
	28	settable	CAN1_RX1	CAN1 RX1 interrupt	0x0000_0094
	29	settable	CAN1_SCE	CAN1 SCE interrupt	0x0000_0098
	30	settable	EXTI9_5	EXTI Line[9:5] interrupts	0x0000_009C
	31	settable	TIM1_BRK	TIM1 Break interrupt	0x0000_00A0
	32	settable	TIM1_UP	TIM1 Update interrupt	0x0000_00A4
	33	settable	TIM1_TRG_COM	TIM1 Trigger and Commutation	0x0000_00A8
	34	settable	TIM1_CC	TIM1 Capture Compare interrupt	0x0000_00AC
	35	settable	TIM2	TIM2 global interrupt	0x0000_00B0
	36	settable	TIM3	TIM3 global interrupt	0x0000_00B4
	37	settable	TIM4	TIM4 global interrupt	0x0000_00B8
	38	settable	I2C1_EV	I ² C1 event interrupt	0x0000_00BC
	39	settable	I2C1_ER	I ² C1 error interrupt	0x0000_00C0
	40	settable	I2C2_EV	I ² C2 event interrupt	0x0000_00C4
	41	settable	I2C2_ER	I ² C2 error interrupt	0x0000_00C8
	42	settable	SPI1	SPI1 global interrupt	0x0000_00CC
	43	settable	SPI2	SPI2 global interrupt	0x0000_00D0
	44	settable	USART1	USART1 global interrupt	0x0000_00D4
	45	settable	USART2	USART2 global interrupt	0x0000_00D8
	46	settable	USART3	USART3 global interrupt	0x0000_00DC
	47	settable	EXTI15_10	EXTI Line[15:10] interrupts	0x0000_00E0
	48	settable	RTCAAlarm	RTC alarm through EXTI line inte	0x0000_00E4
	49	settable	OTG_FS_WKUP	USB On-The-Go FS Wakeup	0x0000_00E8
	-	-	Reserved	0x0000_00EC -	0x0000_0104
	50	settable	TIM5	TIM5 global interrupt	0x0000_0108
	51	settable	SPI3	SPI3 global interrupt	0x0000_010C
	52	settable	UART4	UART4 global interrupt	0x0000_0110
	53	settable	UART5	UART5 global interrupt	0x0000_0114
	54	settable	TIM6	TIM6 global interrupt	0x0000_0118
	55	settable	TIM7	TIM7 global interrupt	0x0000_011C
	56	settable	DMA2_Channel1	DMA2 Channel1 global interrupt	0x0000_0120
	57	settable	DMA2_Channel2	DMA2 Channel2 global interrupt	0x0000_0124
	58	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
```

```
RTCAAlarm_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
```

10 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 muss interruptfähig 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 `Sys-`

`Tick_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);
    ...
}
```

11 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.

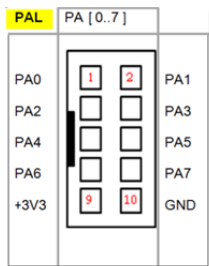
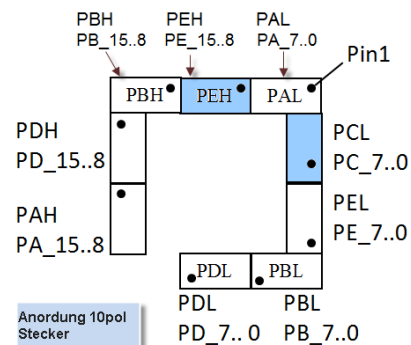


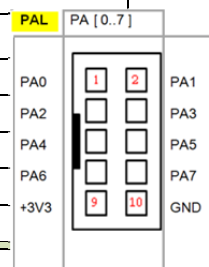
Bild Links: Portnummerierung von PAL

Achtung: Alle Ports dürfen nicht mit mehr als 3,3V beschaltet werden. Falsche Handhabung führt zur Zerstörung des 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_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	Display: GLCD WR#/SCL
PC3	Free_11.	-	PC11	SPI3_MISO	Display: GLCD SDO
PC4	ADC14	Volume VR1	PC12	SPI3_MOSI	Display: 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	Display: 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



12 Referenzen

- [1] R. Weber, «Projektvorlagen (div) MCB32,» 2013ff.
- [2] ST, «ARM_STM_Reference manual_V2014_REV15,» ST, 2014.
- [3] EKomp, «www.elektronik-kompodium.de,» [Online]. Available: <http://www.elektronik-kompodium.de/sites/com/0310301.htm>. [Zugriff am 12.2.2016].
- [4] R. Jesse, Arm Cortex M3 Mikrocontroller. Einstieg und Praxis, 1. Hrsg., www.mitp.de, Hrsg., Heidelberg: Hütigh Jehle Rehm GmbH, 2014.
- [5] J. Yiu, The definitive Guide to ARM Cortex-M3 and M4 Processors, 3. Hrsg., Bd. 1, Elsevier, Hrsg., Oxford: Elsevier, 2014.