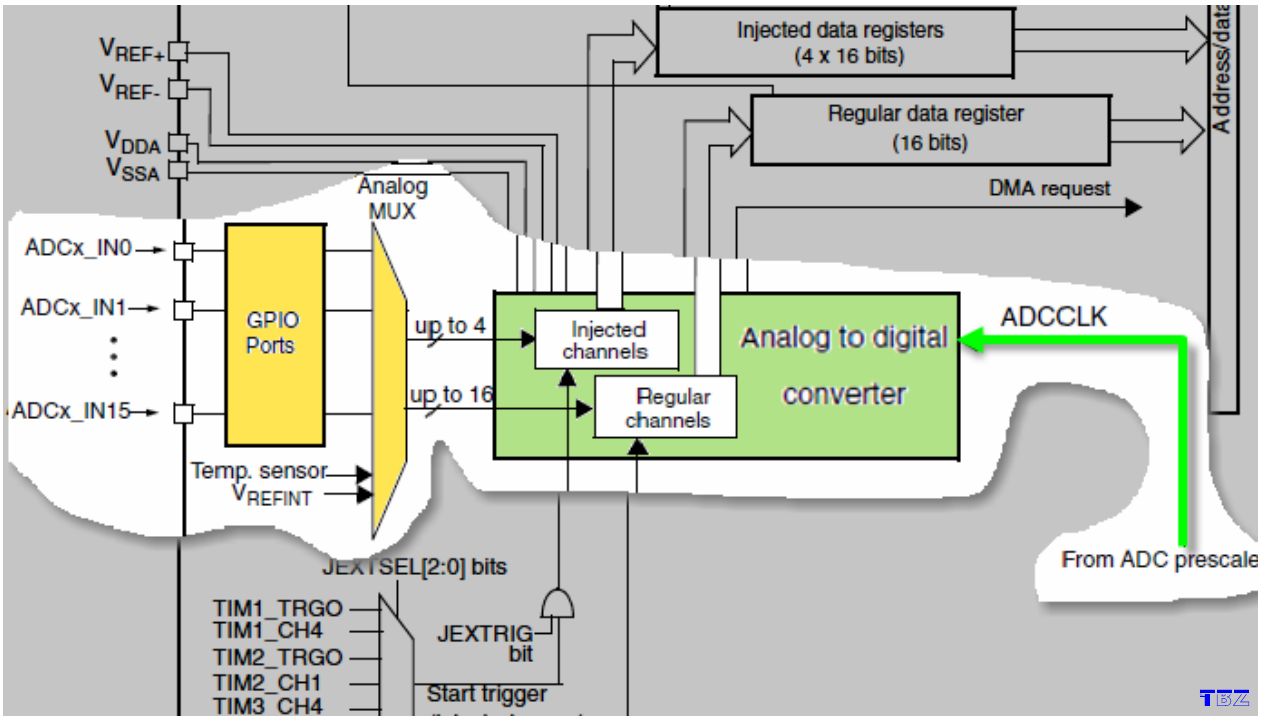
MCB32 Applikation Note
AD/DA Wandler mit MCB32



MCB32 – Embedded Programmierung

Ziel: Einstieg in die AD-Wandler Programmierung für Einsteiger erleichtern

Version: **1.1520d**

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

1 Inhaltsverzeichnis

1 Inhaltsverzeichnis 2
3 Einleitung Beschreibung Wandler 3
3.1 ADC 2 * 12bit 3
3.2 ADC Pin Beschreibung [2], [3], [4] 3
3.2.1 APB2 peripheral clock enable register (RCC_APB2ENR) [1, p. Kap.8.3.7] 7
3.2.2 ADC control register 1 (ADC_CR1) [1, p. Kap11.12.2] 7
3.2.3 ADC control register 2 (ADC_CR2) [1, p. Kap11.12.3] 7
3.2.4 ADC regular sequence register 3 (ADC_SQR3) [1, p. Kap11.12.11] 8
3.2.5 ADC regular data register (ADC_DR) [1, p. Kap11.12.14] 8
4 DAC 10
5 Anhang Port PA Pin 0..7 11
5.1 Anhang Anschlüsse am µC-Board MCB32 12
6 Anhang Übersicht Board 14
6.1 Tabelle mit Beschreibung der MCB32-Funktionen 14
9 Anhang Literaturverzeichnis und Links 16
10 Anhang Wichtige Dokumente 16

3 Einleitung Beschreibung Wandler

3.1 ADC 2 * 12bit

Der ADC ist ein 12bit „succesive approximation“- Wandler. 18 Kanäle erlauben, via einen Multiplexer, das Messen der Signale an den Pins. Die Eingangsspannung hat einen Bereich von: $V_{REF-} < V_{IN} < V_{REF+}$. 2 der 18Kanäle messen Intern die Temperatur und die Ref-Spannung.

Einzel-, kontinuierliche- , scan- oder nicht-kontinuierliche Messungen, programmiert via die Register, erlauben eine Vielfalt von Messarten.

Das Resultat der Messung gelangt links- oder rechtsbündig in ein 16bit Register.

Ein nettes Feature ist die Möglichkeit des analogen Watch-Dogs. Vom Benutzer vorgegebene Grenzen sind überwacht und lösen im Bedarfsfall einen IRQ (Interrupt Request) aus. [1, p. Kap.11ff]. Die analogen Ein- und Ausgänge sind auf dem Board zum Teil für andere Funktion reserviert. Es gilt immer noch: jeder Pin kann mehrere Funktionen übernehmen. Der Input Clock wird vom Clock-Signal PCLK2 abgeleitet. Dieser Clock darf 14MHz nicht überschreiten.

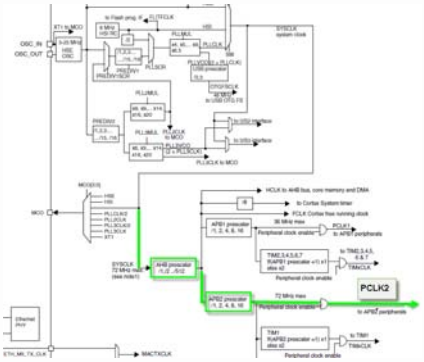


Abbildung 1: Auszug aus Clock-Blockschema

3.2 ADC Pin Beschreibung [2], [3], [4]

Die Pins des ADC arbeiten innerhalb eines bestimmten Arbeitsbereiches. Dieser Bereich hängt von der angelegten Speisespannung ab. Sollen beliebige Bereiche einer Spannung gemessen werden muss eine vorgeschaltete Elektronik (im einfachsten Fall ein Spannungsteiler) den Bereich anpassen.

1. Technische Daten eines ADC Pins (ADCx_IN : x=0 .. 15) [1, p. 217 / Tab 65]

Name	Signal type	Remarks
V_{REF+}	Input, analog reference positive	The higher/positive reference voltage for the ADC, $2.4\ V \le V_{REF+} \le V_{DDA}$
V_{DDA} (1)	Input, analog supply	Analog power supply equal to V_{DD} and $2.4\ V \le V_{DDA} \le 3.6\ V$
V_{REF-}	Input, analog reference negative	The lower/negative reference voltage for the ADC, $V_{REF-} = V_{SSA}$
V_{SSA} (1)	Input, analog supply ground	Ground for analog power supply equal to V_{SS}
ADCx_IN[15:0]	Analog signals	Up to 21 analog channels(2)

1. V_{DDA} and V_{SSA} have to be connected to V_{DD} and V_{SS}, respectively.

Abbildung 2: ADC Pins, Spannungsbereiche (Table 65) für STM32F107 Chip mit 16 Kanälen

2. Abbildung 3: ADC Blockschema von ADC1 sowie ADC2

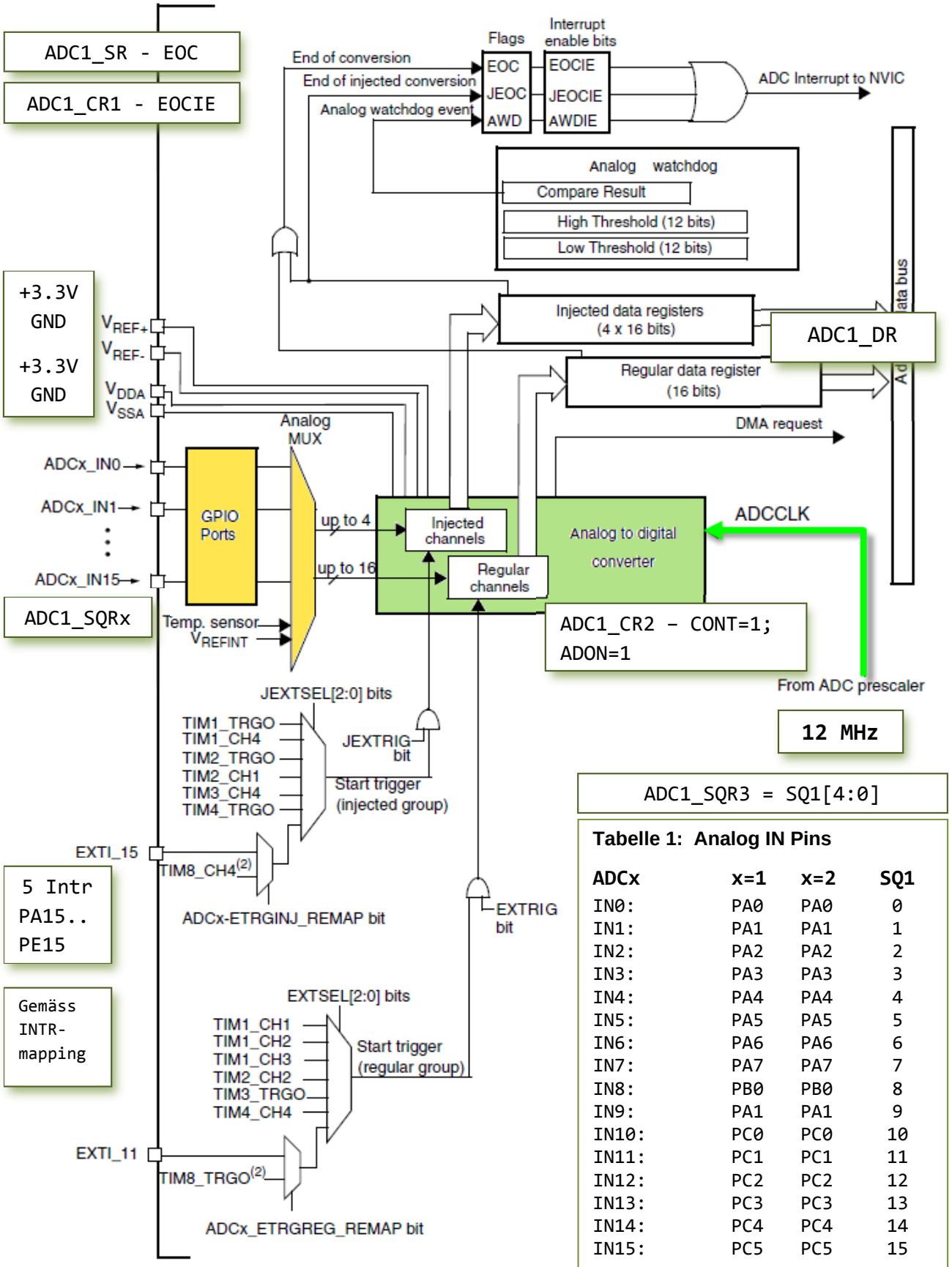
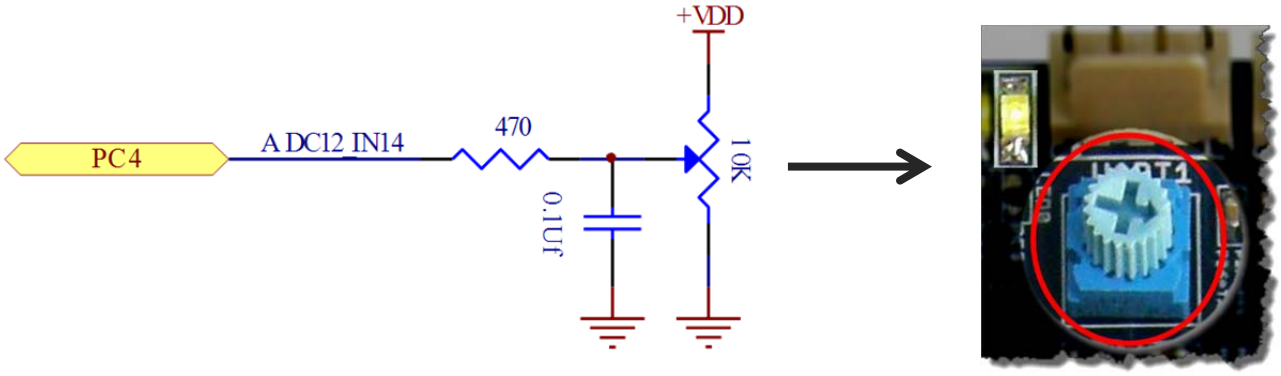


Tabelle 1: Analog IN Pins

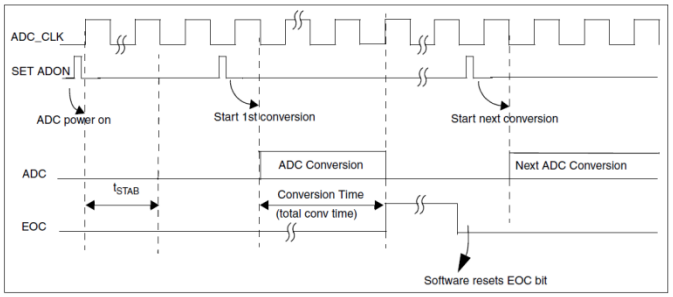
ADCx	x=1	x=2	SQ1
IN0:	PA0	PA0	0
IN1:	PA1	PA1	1
IN2:	PA2	PA2	2
IN3:	PA3	PA3	3
IN4:	PA4	PA4	4
IN5:	PA5	PA5	5
IN6:	PA6	PA6	6
IN7:	PA7	PA7	7
IN8:	PB0	PB0	8
IN9:	PA1	PA1	9
IN10:	PC0	PC0	10
IN11:	PC1	PC1	11
IN12:	PC2	PC2	12
IN13:	PC3	PC3	13
IN14:	PC4	PC4	14
IN15:	PC5	PC5	15

So nun wollen wir, ohne Zuhilfenahme einer speziellen Bibliothek, versuchen das Potentiometer auf der Anzeige auszugeben. Wenn der Port PC4 als Analog-Input (AD-Wandler) geschaltet ist, kann mit dem Potentiometer eine Spannung von 0 .. 3.3V an den Eingang PC4 gelegt werden.



Wir wollen dazu den AD-Wandler No1 benutzen und den Kanal kontinuierlich abtasten. Das heisst sobald eine Messung beendet ist, wird die nächste Messung gestartet. Also, nach jeder Messung passiert folgendes:

- Die Daten werden im 16Bit Datenregister (ADC_DR) gespeichert.
- Das EOC (End of Conversion) Flag wird gesetzt
- Mit IR: ein Interrupt würde generiert wenn EOCIE gesetzt ist. [1, p. Kap 11.].



1. Beispiel mit einfachem Setup:

Aufgabe: Port PC4 als analoger Eingang auf den ADC Wandler Kanal 1 zu schalten und dann kontinuierlich zu messen.

Wenn ein Pin als analoger Pin genutzt wird, so werden alle notwendigen Schaltungen vorgenommen um den Eingang für die Aufgabe vorzubereiten.

In unserem Fall wird der Pin hochohmig geschaltet. [1, p. Kap 8.1.10]

Also setzen wir das Port Konfiguration-Register (GPIOC_CRL) entsprechend dem Manual.

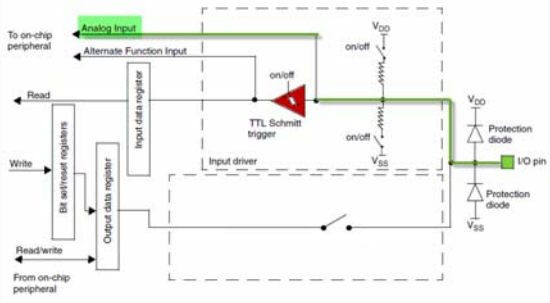


Abbildung 4: Pin High Impedance analog HW Setup (p166)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF7[1:0]		MODE7[1:0]		CNF6[1:0]		MODE6[1:0]		CNF5[1:0]		MODE5[1:0]		CNF4[1:0]		MODE4[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	**0**	**0**	**0**	**0**
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF3[1:0]		MODE3[1:0]		CNF2[1:0]		MODE2[1:0]		CNF1[1:0]		MODE1[1:0]		CNF0[1:0]		MODE0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:30, 27:26, 23:22, 19:18, 15:14, 11:10, 7:6, 3:2 — **CNFy[1:0]:** Port x configuration bits (y= 0 .. 7)
These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table on page 161*.
In input mode (MODE[1:0]=00):
00: Analog mode
01: Floating input (reset state)
10: Input with pull-up / pull-down
11: Reserved

Bits 29:28, 25:24, 21:20, 17:16, 13:12, 9:8, 5:4, 1:0 — **MODEy[1:0]:** Port x mode bits (y= 0 .. 7)
These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table on page 161*.
00: Input mode (reset state)
01: Output mode, max speed 10 MHz.
10: Output mode, max speed 2 MHz.
11: Output mode, max speed 50 MHz.

Abbildung 5: Pin High Impedance analog Configuration (p171)

Bit 16-19 werden auf 0 gesetzt. Siehe Bild oben. (**GPIOC->CRL&=0xFFF0FFFF**;)

Nun fehlt noch der Clock. Dieser wird über das „ APB2 Peripheral clock enable Register (RCC_APB2ENR)“ gesteuert. Mit der 1 bei Bit 9 wird der Clock eingeschaltet.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADC3 EN	USAR T1EN	TIM8 EN	SPI1 EN	TIM1 EN	ADC2 EN	ADC1 EN	IOPG EN	IOPF EN	IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	Res.	AFIO EN
rw	rw	rw	rw	rw	rw	**1** rw	rw	rw	rw	rw	rw	rw	rw		rw

Das GPIO Setup ist nun abgeschlossen.

2. AD Wandler No1: Setup im Detail

Nachdem das IO-Setup un der Clock gesetzt sind muss der AD-Wandler noch fertig konfiguriert werden. Wir wollen kontinuierlich wandeln. Die entsprechenden Register finden wir im Manual.

3.2.1 APB2 peripheral clock enable register (RCC_APB2ENR) [1, p. Kap.8.3.7]

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	USART 1EN	Res.	SPI1 EN	TIM1 EN	ADC2 EN	ADC1 EN	Reserved		IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	Res.	AFIO EN
	rw		rw	rw	rw	rw			rw	rw	rw	rw	rw		rw

Bit 9 in diesem Register muss gesetzt sein wir mit ADC1 wandeln wollen:
RCC->APB2ENR |= 1<<9;

3.2.2 ADC control register 1 (ADC_CR1) [1, p. Kap11.12.2]

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								AWDEN	JAWDEN	Reserved		DUALMOD[3:0]			
								rw	rw			rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DISCNUM[2:0]			JDISCEN	DISCEN	JAUTO	AWDSGL	SCAN	JEOCIE	AWDIE	EOCIE	AWDCH[4:0]				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Um keine Interrupts zu erzeugen wird das Bit EOCIE auf 0 gesetzt:
(ADC1->CR1 &= 0xFFFFFFDF;)

Bit 5 **EOCIE:** Interrupt enable for EOC
This bit is set and cleared by software to enable/disable the End of Conversion interrupt.
0: EOC interrupt disabled
1: EOC interrupt enabled. An interrupt is generated when the EOC bit is set.

3.2.3 ADC control register 2 (ADC_CR2) [1, p. Kap11.12.3]

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								TSVREFE	SWSTART	JSWSTART	EXTTRIG	EXTSEL[2:0]			Res.
								rw	rw	rw	rw	rw	rw	rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JEXTTRIG	JEXTSEL[2:0]			ALIGN	Reserved		DMA	Reserved				RSTCAL	CAL	CONT	ADON
rw	rw	rw	rw	rw	Res.		rw					rw	rw	1	1

WICHTIG !! →

ADON 1. Mal: wakeup

ADON 2. Mal: start ADC

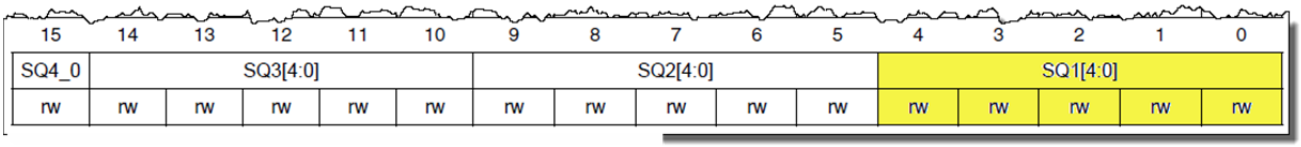
Bit 1 **CONT:** Continuous conversion
0: Single conversion mode
1: Continuous conversion mode

Bit 0 **ADON:** A/D converter ON / OFF
0: Disable ADC conversion/calibration and go to power down mode.
1: Enable ADC and to start conversion

Nun wird das 2 ADC –Control –Register gesetzt: **(ADC1->CR2 = 0x00000003;)**

Der STM32F107 hat, wie wir aus dem Blockschema herauslesen, eine komplexe Logik. Dazu gehört auch dass die Kanäle in Sequenzen arbeiten könnten. Für unsere Beispiel wollen wir aber nur eine Sequenz fahren und den Kanal 14 (PC4) einlesen. Dies teilen wir der Logik über das Register ADC_SQR3 in den Bits 0..4 mit. In Abbildung 3: ADC Blockschema von ADC1 sowie ADC2 und Tabelle 1: Analog IN Pins finden wir den Wert 14 für PC4. Diesen Wert benötigen wir für das zu beschreibende Register.

3.2.4 ADC regular sequence register 3 (ADC_SQR3) [1, p. Kap11.12.11]



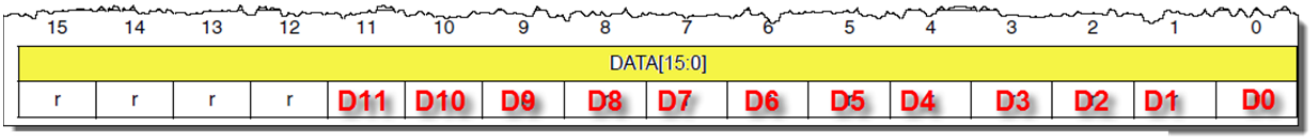
Bits 4:0 **SQ1[4:0]**: 1st conversion in regular sequence

Wie erläutert muss im SQ1 der Wert 14 stehen: **ADC1->SQR3 = 14;**

Um den ADC zu starten (siehe auch oben unter WICHTIG) wird das Register ADC1_CR2 ein 2tes Mal mit dem Wert 1 beschrieben. Dann startet der AD Wandler und liefert das Resultat im Register ADC_DR nach der Wandlung ab.

Das ALIGN bit im ADC_CR2 register selektiert die Ausrichtung der Datenbits nach der Wandlung (alignment of data). Die Daten sind rechts oder links justiert (Data can be left or right aligned). In unserem Fall haben wir eine 0 ins ALIGN-Bit geschrieben was bedeutet, die Bits sind rechts justiert. Bit D0 steht im Bit 0 vom Register ADC_DR.

3.2.5 ADC regular data register (ADC_DR) [1, p. Kap11.12.14]



Bits 15:0 **DATA[15:0]**: Regular data
These bits are read only. They contain the conversion result from the regular channels. The data is left or right-aligned

Das Auslesen des Register geschieht wie folgt: **P1 = ADC1->DR>>4;** Wir schieben die Bits von ADC_DR 4 mal nach rechts und schreiben das Resultat in P1. P1 kann 8 Bits darstellen, darum der Schiebebefehl.

So nun haben wir die wichtigsten Schritte aufgebaut und erklärt. Das Musterprogramm sieht wie folgt aus.

Lösung: AD Wandlung von Poti an PC4

```
#include "TouchP0P1.h"

int main(void)
{
 InitTouchP0P1("1");

 GPIOC->CRL   &= 0xFFF0FFFF;// PC4 analog IN, floating
 RCC->APB2ENR |= 1<<9;      // ADC1 Clock (ADC2 1<<10)
 ADC1->CR1    &= 0xFFFFFFDF;// IRQ OFF, keine Interrupts
 ADC1->CR2    = 0x00000003; // ADC Cont & ADC ON 1.Mal
 ADC1->SQR3   = 14;         // ADCxIN_0 .. 15, Bsp. PC4
 ADC1->CR2    |= 1<<0;      // ADC ON 2.Mal

 while(1)                   // Endlosschlaufe
 {                          // Aus Analog 12 Bit 11..0
   P1 = ADC1->DR>>4;        // obere 8 Bit 11..4
 }                          // zu P1-LEDs 7..0
}
```

Abbildung 6: Musterprogramm für AD Wandler

Bei ersten Beispiel wurde nicht eigentlich auf das Ende der Wandlung gewarte. Dies spiel bei einfachen Aufgaben keine grosse Rolle. Die Werte „flackern“ vielleicht ein bisschen mehr. Besser ist aber wenn das Ende der Wandlung abgefangen wird, entweder durch das Abfragen des EOC-Bits im ADC-Status Register oder indem der ADC einen Interupt auslöst. Das folgende Beispiel zeigt die Lösung mit EOC.

Lösung2: AD Wandlung von Poti an PC4 mit EOC-Abfrage

```
#include "TouchP0P1.h"

int main(void)
{
 InitTouchP0P1("1");

 GPIOC->CRL   &= 0xFFF0FFFF;      // PC4 analog IN, floating
 RCC->APB2ENR |= 1<<9;            // ADC1 Clock (ADC2 1<<10)
 ADC1->CR1    &= 0xFFFFFFDF;      // IRQ OFF, keine Interrupts
 ADC1->CR2    = 0x00000003;       // ADC Cont & ADC ON 1.Mal
 ADC1->SQR3   = 14;               // ADCxIN_0 .. 15, Bsp. PC4
 ADC1->CR2    |= 1<<0;            // ADC ON 2.Mal

 while(1)
 {
  if((ADC1->SR & 0x00000002)!=0) // Solange EOC 0 ist warten
   {                             // EOC ist 1 -> lese Bits
    P1 = ADC1->DR>>4;            // obere 8 Bit 11..4 zu P1
   }                             // und EOC löschen
 }
}
```

Abbildung 7: Musterprogramm für AD Wandler mit EOC

Lösung3: AD Wandlung von Port_A mit EOC-Abfrage. Aktuelles Beispiel ist mit **PA0**.

```
int main(void)
{
 InitTouchP0P1("1");               // P0P1-Touchscreen ON
 RCC->APB2ENR |= 1<<2;             // Enable PORTA clock
 GPIOA->CRL   &= 0x00000000;    // PA7..0 analog IN, floating
 RCC->APB2ENR |= 1<<9;             // ADC1 Clock (ADC2 1<<10)
 ADC1->CR2     = 0x00000003;       // ADC Cont & ADC ON 1.Mal
 ADC1->SQR3    = 0;                // ADCxIN_0 .. 15, zB. PA0
 ADC1->CR2   |= 1<<0;              // ADC ON 2.Mal

 while(1)
 {
  if ((ADC1->SR & 0x00000002) !=0) // Solange EOC 0 ist warten
  {                                // EOC ist 1 -> lese Bits
   P1 = ADC1->DR>>4;               // obere 8 Bit 11..4
  }                                // zu P1-LEDs 7..0
 }
}
```

Abbildung 8: Musterprogramm für AD Wandler von Port A mit EOC

Lösung3: AD Wandlung von Port_A mit EOC-Abfrage. Aktuelles Beispiel ist mit **PA0**.

```
#include <stm32f10x.h>    // David Haag

void ADC1_2_IRQHandler(void)    // Interrupt Service Routine ISR
{
GPIOE->ODR = (ADC1->DR)<<4;
//  ADC1->SR &= ~(1<<1);
}

int main(void)
{
 unsigned long i;
 //-------------------------------------------------
 //Initialisierung LED-Port
 //-------------------------------------------------
 RCC->APB2ENR |= 1<<6;
 GPIOE->CRH = 0x11111111;
 //-------------------------------------------------
 //Initialisierung Alternate Functions/peripheral clock/ADC
 //-------------------------------------------------------------
 RCC->APB2ENR = RCC->APB2ENR | 0x00000211;   //set bit0,bit4 and bit 9
                                             //enable peripheral clock GPIOC (ADC),
                                             //alternate function, clock ADC1
 RCC->CFGR |=   (2<<14);                     //divide by 6 (default divide by 2)
 ADC1->SQR3 =   (14<<0);         //channel 14 als einzigen Kanal->kein shiften nötig
 ADC1->CR2 |=   (7<<17);                     //start conversion by sw
 ADC1->CR2      |=          (1<<20);         //enable extern trigger (per sw=extern)
 ADC1->CR2      |=          (1<<1);          //continuous
 ADC1->CR1      |=          (1<<5);          //EOCIE
 NVIC->ISER[0] |= (1<<18);                  //interrupt set enable register for adc
 ADC1->CR2      |=          (1<<0);          //ADC ON
 ///-----------------------------------------------------------
 for (i=0;i<1000000;i++);    //delay between ADC-ON and start conversion->see manual
 ADC1->CR2 |= (1<<22);                                 //start conversion
 while(1)                                    //forever
   {                                         //nothing to do!
   }
}
```

Abbildung 9: Musterprogramm für AD Wandler mit Interrupt und Poti PC4

Obiges Beispiel zeigt den Einsatz einer Service Interrupt Routine sowie des IR-Mechanismus. Der AD Wandler löst am Ende der Wandlung einen IR aus. Dieser startet die ISR und führt die dort abgelegten Befehle aus.

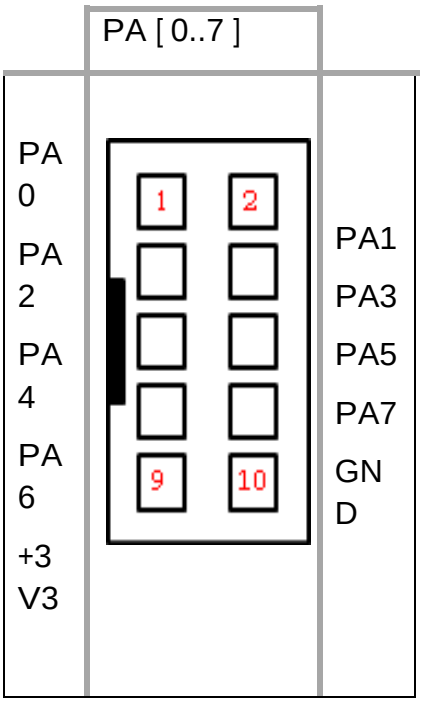
4 DAC

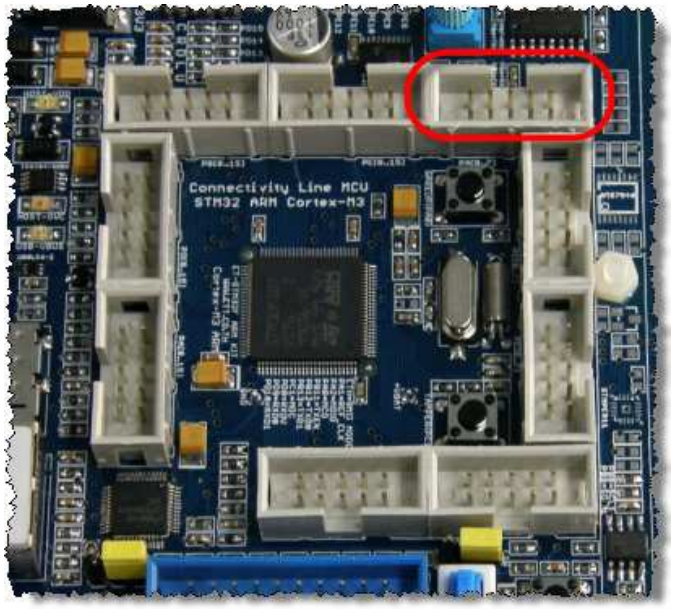
Siehe AP-Note für DA-Wandler

5 Anhang Port PA Pin 0..7

Im folgenden Versuch wird Port PA benutzt. Der Port wird als Eingang geschaltet werden. Die anliegenden Signale werden dann an Port PE durchgeschaltet. Wichtig: Eine Logik 1 schaltet die auf dem Board befindlichen LEDs ein. (Port PW[8..15])

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





Die Belegung des 10poligen Steckers sieht wie oben abgebildet aus. Die roten Zahlen definieren die Adern des Flachbandkabels mit roter Ader = Pin1.

2. Original Belegung PA[0..7]

Die Original Pin-Belegung von Stecker PA[0...7] ist wie in der nebenstehenden Tabelle. Diese Einstellung wird im Versucht aber nicht benötigt.

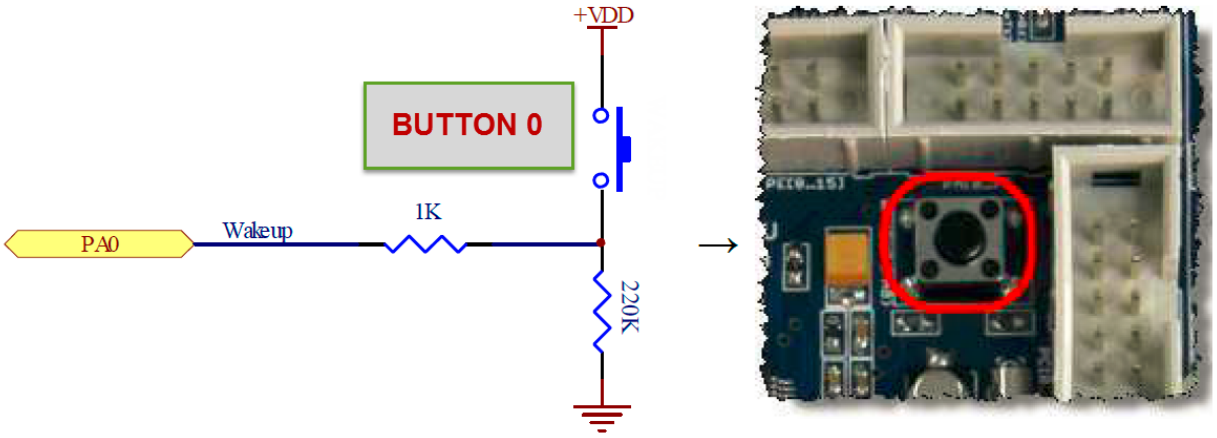
Pin	Funktion	Module/Device	
PA0	Wakeup	Switch Wakeup	
PA1	RMII_REF_CLK	Ethernet LAN	
PA2	RMII_MDIO	Ethernet LAN	
PA3	-	-	
PA4	-	-	
PA5	SPI1_SCK	SD Card CLK	
PA6	SPI1_MISO	SD Card DAT0	
PA7	SPI1_MOSI	SD Card CMD	

5.1 Anhang Anschlüsse am µC-Board MCB32

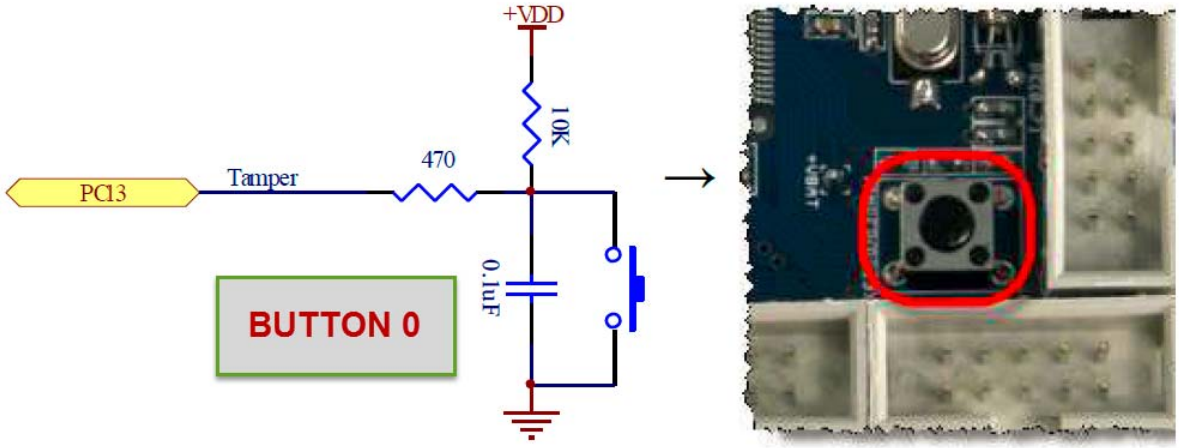
// In TouchP0P1.h definierte Pin-Bezeichnungen PA_0 .. PD_11, ohne Bezeichner wie Button .. !

```
char Button0     = PA_0;      // Bitwert 1/0, aktiv low, prellt wenig
char Button1     = PC_13;
char Stick       = PD_High;   // als Byte 0xF8 open, aktiv low, alle entprellt
char StickSelect = PD_15      // Bitwert  1/0;    Bytewert   0x80
char StickDown   = PD_14;     //          1/0;               0x40
char StickLeft   = PD_13;     //          1/0;               0x20
char StickUp     = PD_12;     //          1/0;               0x10
char StickRight  = PD_11;     //          1/0;               0x08
```

Button 0 (PA 0)

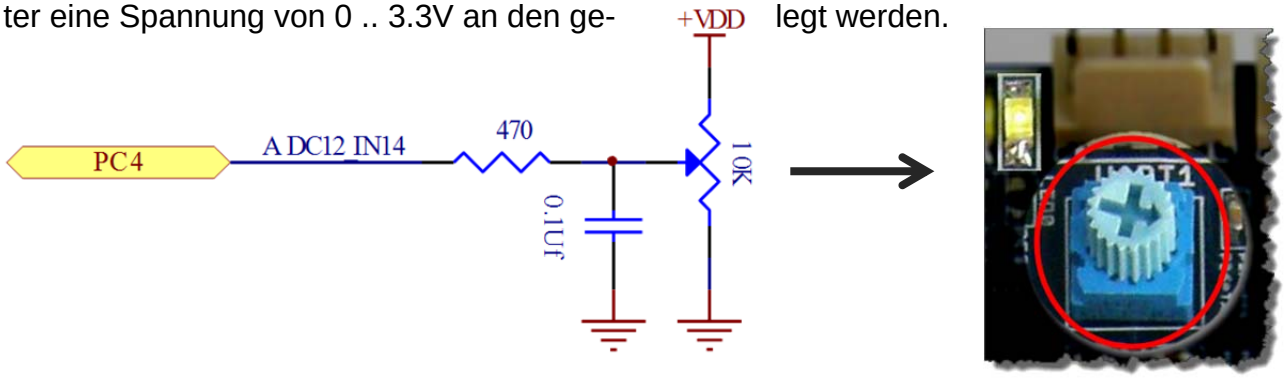


Button 1 (PC 13)

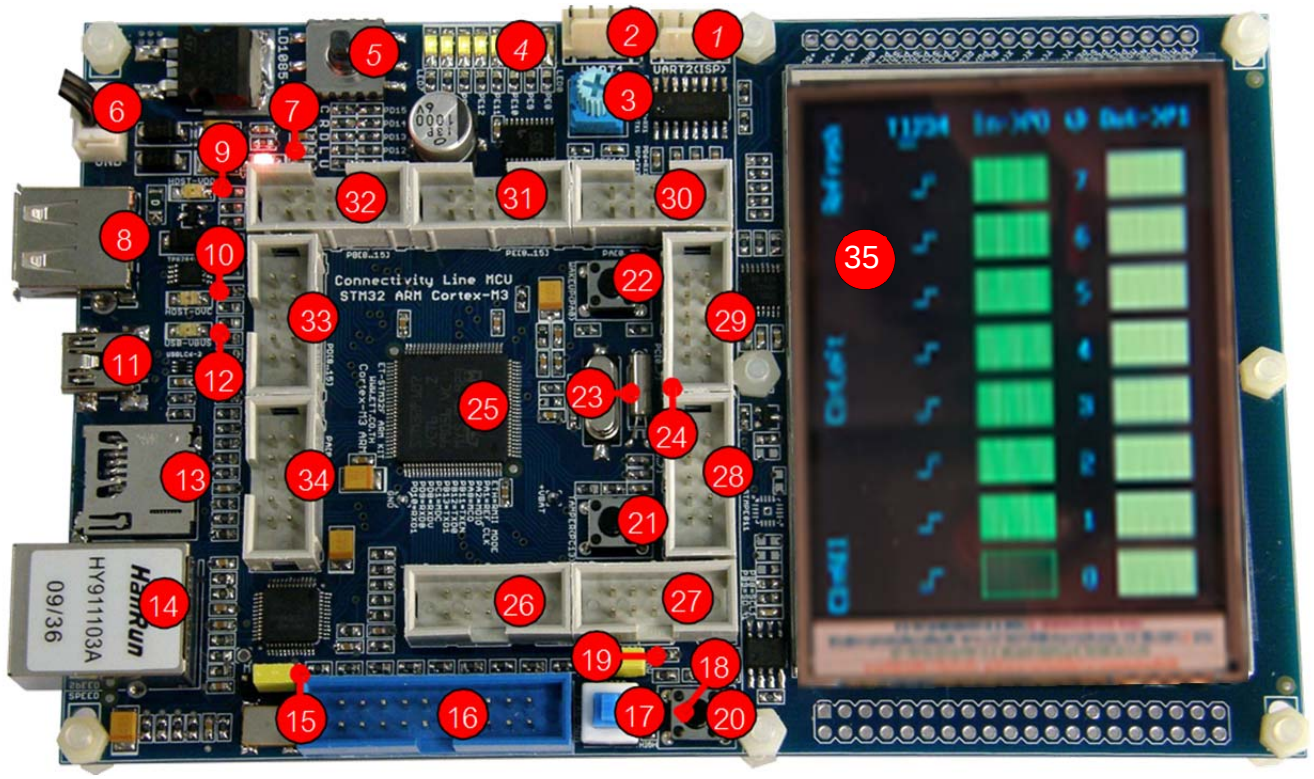


Potentiometer (PC4) // resp. P0_4 (Library)

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.



6 Anhang Übersicht Board



6.1 Tabelle mit Beschreibung der MCB32-Funktionen

#	Beschreibung der markierten Position (i)
1	RS232-2 (Uart2) Stecker. Empfängt HEX File für Bootloader
2	RS232-1 (Uart1) Stecker.
3	0…3,3Volt für A/D Wandler Test an (PC4/ADC14)
4	LED[0..7] angesteuert via PE[8..15].
5	5 Richtungs (Joy) Schalter
6	Stecker für +5Volt Speisung für das ganze Board
7	LED (+VDD: +3.3V) Status
8	USB Host
9	LED zeigt Status von USB-Host VDD.
10	LED zeigt Status von "Host Over Current"
11	Stecker "USB Device/OTG"
12	LED zeigt Status von "USB VBUS"

13	Micro SD Card Sockel
14	RJ45 Ethernet LAN.
15	Jumper(MCO/OSC) für die Auswahl des Clock-Signals für DP83848V.
16	JTAG Stecker für Real Time Debugging
17	Switch BOOT0 welcher zusammen mit Jumper BOOT1 den Boot-Modus der MCU bestimmt: Boot Loader (BOOT0=1, BOOT1=0) und Run (BOOT0=0,BOOT1=0).
18	LED zeigt Logic Status von BOOT0 = 1 (ON=Boot Loader, OFF=Run).
19	Jumper BOOT1(PB2); Ist im Normalfall auf LOW
20	Switch RESET.
21	Switch Tamper (PC13).
22	Switch Wakeup (PA0).
23	Quarz 25MHz für die Zeitbasis der MCU
24	Quarz 32.768KHz für die Zeitbasis der RTC (Real Time Clock)
25	MCU No.STM32F107VC T6
26	Stecker GPIO PD[0..7].
27	Stecker GPIO PB[0..7].
28	Stecker GPIO PE[0..7].
29	Stecker GPIO PC[0..7].
30	Stecker GPIO PA[0..7].
31	Stecker GPIO PE[8..15].
32	Stecker GPIO PB[8..15].
33	Stecker GPIO PD[8..15].
34	Stecker GPIO PA[8..15].
35	320x240 Dot TFT LCD mit Touch Screen Sensor. REV C oder REV D

9 Anhang Literaturverzeichnis und Links

[1] ST, «ARM_STM_Reference manual_V2014_REV15,» ST, 2014.
[2] R. Weber, «General Purpos Input Output,» 2014.
[3] R. Weber, «Projektvorlagen (div) MCB32,» 2013ff.
[4] ST, «STM32F107 Data Sheet_2014_REV7,» ST, 2014.
[5] J. Yiu, The definitive Guide to ARM Cortex-M3 and M4 Processors, 3 Hrsg., Bd. 1, Elsevier, Hrsg., Oxford: Elsevier, 2014.
[6] R. Jesse, Arm Cortex M3 Mikrocontroller. Einstieg und Praxis, 1 Hrsg., www.mitp.de, Hrsg., Heidelberg: Hütigh Jehle Rehm GmbH, 2014.
[7] Diller, «System Timer,» 06 07 2014. [Online]. Available: http://www.diller-technologies.de/stm32.html#system_timer. [Zugriff am 06 07 2014].
[8] A. Limited, «DDI0337E_cortex_m3_r1p1_trm.pdf,» ARM Limited, http://infocenter.arm.com/help/topic/com.arm.doc.ddi0337e/DDI0337E_cortex_m3_r1p1_trm.pdf, 2005, 2006 ARM Limited.
[9] A. C. Group, «http://www.vr.ncue.edu.tw/esa/b1011/CMSIS_Core.htm,» 2007. [Online].
[10] E. Malacarne, *Glossar Malacarne,* V11 Hrsg., Rüti: Cityline AG, 2014.
[11] E. F. E. Schellenberg, «Programmieren im Fach HST,» TBZ, Zürich, 2010ff.
[12] STM, «STM32F10x Standard Peripherals Firmware Library,» STM, 2010ff.

10 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 (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 (Cortex-M3)
  Enthält beispielsweise Informationen zum Interrupt Controller (NVIC).
- Standard Peripheral Library (STM32F10x)
  Im Gegensatz zu anderen MCUs sollen die Register der 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.