

---

# Laborprotokoll

## L05: Widerstandsmessgerät

---

Systemtechnik Labor  
4CHIT 2016/17, GruppeD

Martin Wölfer

Note:  
Betreuer: Weiser

Version 0.2  
Begonnen am 12. Mai 2017  
Beendet am 26. Mai 2017

# Inhaltsverzeichnis

<b>1 Einführung</b>	<b>1</b>
1.1 Aufgabenstellung . . . . .	1
1.2 Anmerkungen . . . . .	1
<b>2 Ergebnisse</b>	<b>2</b>
2.1 Static Library . . . . .	2
2.1.1 Erstellen . . . . .	2
2.1.2 Code file.h . . . . .	2
2.1.3 Code file.c . . . . .	2
2.1.4 Exportieren . . . . .	3
2.1.5 Einbinden . . . . .	3
2.1.6 Code use static library . . . . .	3
2.2 Interne Referenzspannung . . . . .	4
2.2.1 Code . . . . .	4
2.3 Interne Versorgungsspannung . . . . .	4
2.3.1 ADC Konfigurieren . . . . .	5
2.3.2 $V_{DDA}$ berechnen . . . . .	5
2.4 ADC als Widerstandsmessgerät . . . . .	6
2.4.1 Schaltung . . . . .	6
2.4.2 ADC konfigurieren . . . . .	6
2.4.3 Pins konfigurieren . . . . .	7
2.4.4 Werte berechnen . . . . .	7
2.4.5 Widerstabd R2 berechnen . . . . .	7
2.5 Ausgabe . . . . .	7

# 1 Einführung

## 1.1 Aufgabenstellung

Ziel dieser Übung ist es, die Basis für ein Widerstandsmessgerät zu schaffen. Ein Widerstand soll mittels des ADC gemessen werden und das Ergebnis soll über ITM ausgegeben werden. Gehe folgendermaßen vor:

- 1.) Erstelle eine static library für die ITM-Ausgabe und teste diese mit einem einfachen Testprogramm.
- 2.) Konfiguriere den ADC, misse die interne Referenzspannung und bestimme daraus die Versorgungsspannung. Dies benötigst du, um in der Folge Spannungsmessungen durchzuführen.
- 3.) Verbinde einen externen Pin mit einem fixen Widerstand und einem Testwiderstand in Serie. Misse dann alle drei Spannungspunkte (0V, 3V, und die Spannung zwischen den beiden Widerständen) mit dem ADC und bestimme daraus die Größe des Testwiderstandes.

## 1.2 Anmerkungen

- Punkt(1) ist analog zu Übung B08. Falls du Übung B06 noch nicht gemacht hast, so beginne damit.
- Beim Konfigurieren des ADC gehe schrittweise vor. Speichere immer den Rückgabewert von gerufenen Funktionen und gib ihn über ITM aus. Falls

## 2 Ergebnisse

### 2.1 Static Library

#### 2.1.1 Erstellen

Zuerst musste die Static Library erstellt werden. Dazu wurde ein neues Projekt angelegt, welches die Rolle einer statischen Bibliothek einnimmt, in welchem man ein header-file erzeugt in welchem die Methoden definiert werden. Diese Header-File heißt in meinem Fall `file.h`. Im tatsächlichen C-File wird dann diese Methode bzw. Methoden implementiert.

#### 2.1.2 Code `file.h`

```

1  /*
2   * file.h
3   *
4   * Created on: 23. Feb. 2017
5   * Author: Martin Woelfer
6   */
7
8 #ifndef FILE_H_
9 #define FILE_H_
10
11 void initItm();
12 int __io_putchar(int ch);
13
#ENDIF /* FILE_H_ */

```

#### 2.1.3 Code `file.c`

```

1  /*
2   * file.c
3   *
4   * Created on: 23. Feb. 2017
5   * Author: Martin Woelfer
6   */
7
8
9 #include "stm32f3xx.h"
10 #include "stm32f3_discovery.h"
11
12 #include <stdio.h>
13 #include "file.h"
14
15
16 void initItm(void){
17     // 0xE004 2004, Bit 5
18     SET_BIT(DBGMCU->CR, DBGMCU_CR_TRACE_IOEN);
19     // 0xE004 2004, Bit 6-7
20     CLEAR_BIT(DBGMCU->CR, DBGMCU_CR_TRACE_MODE);
21     // 0xE000 EDFC, Bit 24
22     SET_BIT(CoreDebug->DEMCR,CoreDebug_DEMCR_TRCENA_Msk);
23
24
25     TPI->ACPR = 0;
26     TPI->CSPSR = 0x1;
27     TPI->FFCR = 0x102;
28     TPI->SPPR = 0x02;
29
30

```

```

32     ITM->LAR = 0xC5ACCE55;
34     ITM->TCR = 0x00010005;
36     ITM->TER = 0x1;
37     ITM->TPR = 0x1;
38     setvbuf(stdout, NULL, _IONBF, 0);
39 }
40 int __io_putchar(int ch) {
41     return(ITM_SendChar(ch));
42 }
```

## 2.1.4 Exportieren

Damit die Files tatsächlich gefunden werden wenn man die static-library einbindet, muss man die ganzen Files der static-library noch exportieren. Dazu muss man in den **Properties** des Projektes auf **Paths and symbols** gehen und dort alles auswählen und auf **exportiere** drücken.

## 2.1.5 Einbinden

Der Sinn einer static-library liegt darin, dass einem diese viel Arbeit abnimmt. Daher sollte die Einbindung auch so einfach wie möglich erfolgen, und zwar muss man auf das Projektes welches die static-library verwenden soll wieder auf **Paths und Symbols** gehen und dort dann lediglich die static-library importieren. Zu beachten ist, dass man die richtige *Version* nimmt (Debug oder Active).

Nachdem muss man nur mehr das header-file importieren und kann nun z.B. eine einfache ITM-Ausgabe machen:

## 2.1.6 Code use static library

```

1 /**
2  * @file      main.c
3  * @author    Ac6
4  * @version   V1.0
5  * @date      01-December-2013
6  * @brief     Default main function.
7  */
8
9
10
11 #include "stm32f3xx.h"
12 #include "stm32f3_discovery.h"
13 #include <stdio.h>
14 #include "file.h"
15
16 int main(void)
17 {
18     HAL_Init();
19     BSP_LED_Init(LED_ORANGE);
20     initItm();
21     for(;;){
22         HAL_Delay(1000);
23         printf("Hallo Welt\n");
24         BSP_LED_Toggle(LED_ORANGE);
25     }
26 }
```

## 2.2 Interne Referenzspannung

Die interne Referenzspannung kann man sich ausrechnen nur durch auslesen des Speichers des Mikrocontrollers, es müssen nicht einmal jegliche Pins oder sonstiges konfiguriert werden.

Die interne Referenzspannung wird mit folgender Formel berechnet:

$$V_{refint} = 3300 * \frac{m_{refint\_cal}}{m_{max}}$$

Wobei  $m_{max}$  0xFFFF entspricht, der maximale Wert den die Testspannung erreichen kann (0-4300)

### 2.2.1 Code

```
2     uint16_t m_refint_cal= *((uint16_t*)0x1FFFF7BA);
      int v_ref_int = 3300* m_refint_cal/0xFFFF;
      printf("Die interne Referenzspannung betraegt %d \n", v_ref_int);
```

## 2.3 Interne Versorgungsspannung

Die interne Versorgungsspannung wird benötigt um die externen Spannungen richtig interpretieren zu können bzw. den ADC konfigurieren zu können.

Diese muss man deswegen auslesen / berechnen weil die je nach Board, Laptop und Port bei jedem anders sein könnte.

### 2.3.1 ADC Konfigurieren

Damit man den die Referenzspannung des ADCs auslesen kann, muss dieser zuerst *aktiviert* werden und konfiguriert:

```

1  ADC_HandleTypeDef hadc;
2    __ADC1_CLK_ENABLE();
3  hadc.Instance = ADC1;
4  hadc.DMA_Handle = 0;
5  hadc.ErrorCode = 0;
6  hadc.Lock = HAL_UNLOCKED;
7  hadc.State = HAL_ADC_STATE_RESET;

9  hadc.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV1;
hadc.Init.Resolution = ADC_RESOLUTION_12b;
11 hadc.Init.DataAlign = ADC_DATAALIGN_RIGHT;
hadc.Init.ScanConvMode = ADC_SCAN_ENABLE;
13 hadc.Init.EOCSelection = EOC_SINGLE_CONV;
hadc.Init.LowPowerAutoWait = DISABLE;
15 hadc.Init.ContinuousConvMode = DISABLE;
hadc.Init.DiscontinuousConvMode = DISABLE;
17 hadc.Init.ExternalTrigConv = ADC_SOFTWARE_START;
hadc.Init.DMAContinuousRequests = DISABLE;
19 hadc.Init.Overrun = OVR_DATA_OVERWRITTEN;
hadc.Init.NbrOfConversion = 1;
21 rc = HAL_ADC_Init(&hadc);
23
ADC_ChannelConfTypeDef sConfig1;
25 sConfig1.SamplingTime = ADC_SAMPLETIME_181CYCLES_5;
sConfig1.SingleDiff = ADC_SINGLE_ENDED;           // Ich messe nur ein PIN
27 sConfig1.OffsetNumber = ADC_OFFSET_NONE;         // Kein Offset
// sConfig1.Offset = 0; // unnoetig
29
// Config VrefInt
31 sConfig1.Channel = ADC_CHANNEL_VREFINT;
sConfig1.Rank = ADC_REGULAR_RANK_1;
33 HAL_ADC_ConfigChannel(&hadc, &sConfig1);

```

Damit wird nicht nur der ADC gegangen, sondern auch der Channel VREFINT, welcher im nächsten Schritt ausgelesen wird.

### 2.3.2 $V_{DDA}$ berechnen

Zuerst wird sich wieder die Kalibrierungsspannung gespeichert und anschließend wird, wie jedesmal wenn man einen Wert des ADCs auslesen will, zuerst `PollForConversion()` aufgerufen und danach `GetValue()` um den Wert für  $m_{refint}$  zu erhalten.

Danach wird mit der Formel  $V_{DDA} = 3300 * \frac{m_{refint\_cal}}{m_{refint}} V_{DDA}$  berechnet:

```

1 void setVDDA(void){
2     uint16_t m_refint_cal=*((uint16_t*)0x1FFFF7BA);
3     HAL_ADC_PollForConversion(&hadc, 100);
4     uint32_t m_refint=HAL_ADC_GetValue(&hadc);
5     VDDA=(3300*m_refint_cal)/m_refint;
6 }
7 void setVDDA(void){
8     uint16_t m_refint_cal=*((uint16_t*)0x1FFFF7BA);
9     HAL_ADC_PollForConversion(&hadc, 100);
10    uint32_t m_refint=HAL_ADC_GetValue(&hadc);
11    VDDA=(3300*m_refint_cal)/m_refint;
}

```

$V_{DDA}$  wird als globale Variable abgespeichert, da diese später noch in Verwendung kommt.

## 2.4 ADC als Widerstandsmessgerät

### 2.4.1 Schaltung

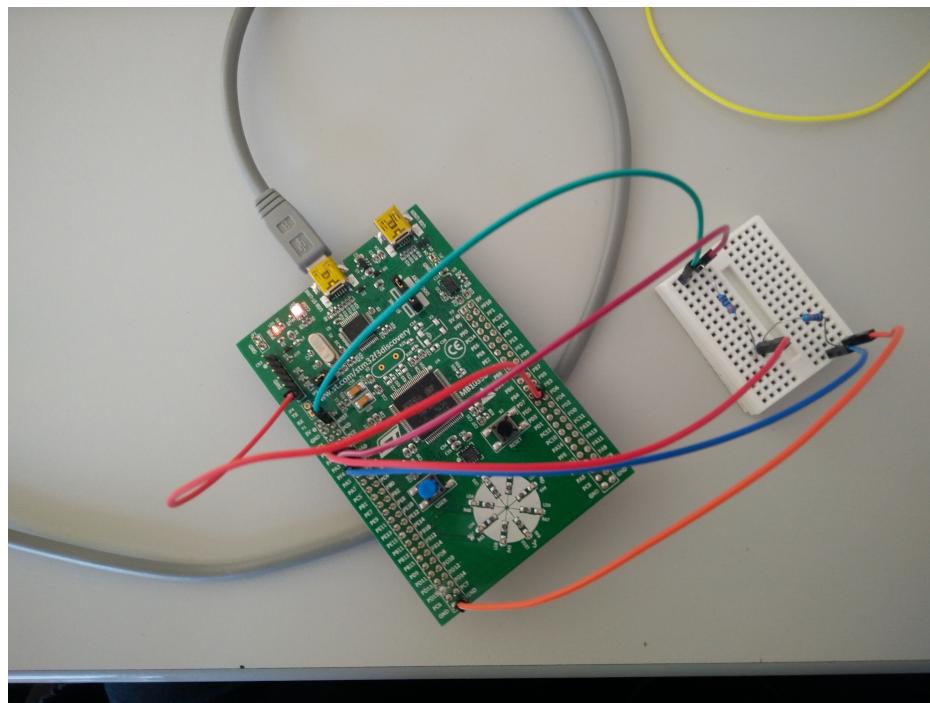


Abbildung 1: Die Schaltung mit den beiden Widerständen wurde aufgebaut

### 2.4.2 ADC konfigurieren

An der Konfiguration (Die oben beschrieben wurde) muss nichts geändert werden, außer dem Wert `NbrOfConversions`, welche statt auf 1 auf 4 gesetzt wird. Grund dafür ist, dass nun nicht nur mehr `VRefInt` ausgelesen wird, sondern noch an 3 externen Pins ein Wert angeschaut wird.

Zusätzlich dazu müssen natürlich noch die passenden Channel konfigurierten werden:

```

1 // Config U1 (PA0)
2 sConfig1.Channel = ADC_CHANNEL_1;
3 sConfig1.Rank = ADC_REGULAR_RANK_2;
4 HAL_ADC_ConfigChannel(&hadc, &sConfig1);
5
6 // Config U2 (PA1)
7 sConfig1.Channel = ADC_CHANNEL_2;
8 sConfig1.Rank = ADC_REGULAR_RANK_3;
9 HAL_ADC_ConfigChannel(&hadc, &sConfig1);
10
11 // Config U3 (PA2)
12 sConfig1.Channel = ADC_CHANNEL_3;
13 sConfig1.Rank = ADC_REGULAR_RANK_4;
14 HAL_ADC_ConfigChannel(&hadc, &sConfig1);

```

### 2.4.3 Pins konfigurieren

Und wenn noch externe Pins, also in meinem Fall PA0, PA1 und PA2 verwendet werden, müssen diese natürlich auch noch initialisiert werden:

```

1 // Init analog pins
2     GPIOA_CLK_ENABLE();
3     GPIO_InitTypeDef GPIO_InitStruct;
4     GPIO_InitStruct.Pin = GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2;
5     GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
6     GPIO_InitStruct.Speed = GPIO_SPEED_HIGH;
7     GPIO_InitStruct.Alternate = 0;
8     GPIO_InitStruct.Pull = GPIO_NOPULL;
9     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

```

### 2.4.4 Werte berechnen

Um die einzelnen Werte zu berechnen, wird in folgende Formel eingesetzt:

$$V_{test} = V_{DDA} * \frac{m_{test}}{m_{max}}$$

$m_{test}$  wird jedes mal neu ausgelesen, dies ist der wahre *Testwert*, der Rest wurde schon erklärt:

```

1 uint32_t calculateVTest(uint32_t m_test) {
2     return (VDDA * m_test) / 0xFFFF;
3 }

```

Um zu  $m_{test}$  zu gelangen, muss wieder jedesmal `PollForConversion()` und `GetValue()` aufgerufen werden, und mit diesen Werd die oben beschrieben Funktion aufgerufen werde:

```

1 HAL_ADC_PollForConversion(&hadc, 100);
2     uint32_t channel1 = calculateVTest(HAL_ADC_GetValue(&hadc));
3
4 HAL_ADC_PollForConversion(&hadc, 100);
5     uint32_t channel2 = calculateVTest(HAL_ADC_GetValue(&hadc));
6
7 HAL_ADC_PollForConversion(&hadc, 100);
8     uint32_t channel3 = calculateVTest(HAL_ADC_GetValue(&hadc));

```

### 2.4.5 Widerstabd R2 berechnen

Um R2 zu berechnen wird folgende Formel verwendet:

$$R_2 = R_1 * \frac{\Delta U_1}{\Delta U_2}$$

```

1 int delta_u1 = channel2 - channel1;
2 int delta_u2 = channel3 - channel2;
3
4 int R2 = (3300 * delta_u1) / delta_u2;

```

## 2.5 Ausgabe

Wenn nun die Schaltung richtig aufgebaut ist, man das Programm kompiliert und auf den Mikrocontroller spielt und anschließend den Utility Linker öffnet, sollte man folgendes sehen:

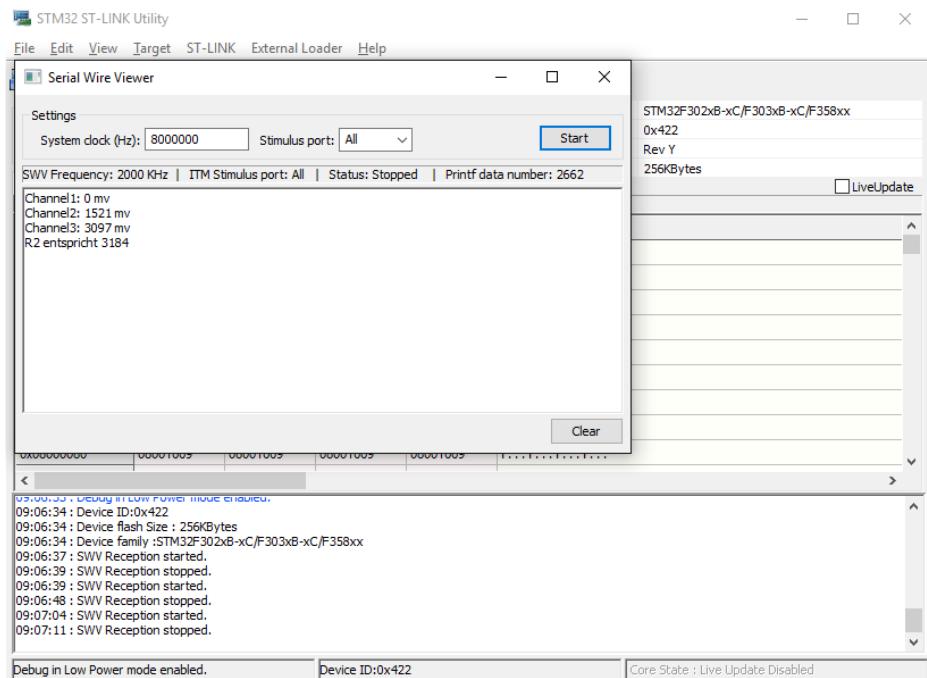


Abbildung 2: Es werden die Spannungen an den verschiedenen Channel in millivolt ausgegeben und der berechnete Widerstand

## Abbildungsverzeichnis

1	Die Schaltung mit den beiden Widerständen wurde aufgebaut . . . . .	6
2	Es werden die Spannungen an den verschiedenen Channel in millivolt ausgegeben und der berechnete Widerstand . . . . .	8