Mikroprozessorpraktikum

Konstantin Bork & Kean Seng Liew, Gruppe A, HWP8

09-01 ADC

A09-01.1

> Geben Sie den Wert der an PA0 angelegten Spannung im Sekundentakt und in der Einheit Volt auf der USART2 aus. Signalisieren Sie dabei mittels LED den Spannungswert in folgender Form: - Spannung größer gleich 1,4V: grüne LED an
>
> Die Messung soll im Polling erfolgen. Begründen Sie das Ausgabeformat der Spannung. Wieviel Vor- und Nachkomastellen sind technisch gerechtfertigt?

main.c

```
#include "main.h"
#include "aufgabe.h"

int main(void)
{
    // Initialisierung des Systems und des Clocksystems
    SystemInit();

    // LED und Tasten initialisieren
    init_leds();

    // USART2 und Timer initialisieren
    init_usart_2();
    init_adc();

    // Variablen für die Messung und die Ausgabe
    uint16_t messwert = 0;
    float spannung = 0.0f;

    while(1)
    {
        ADC_SoftwareStartConv(ADC1);
        if(ADC_GetFlagStatus(ADC1,ADC_FLAG_EOC)==SET)
        {
            messwert = ADC_GetConversionValue(ADC1);
            spannung = (messwert/4095.0f) * 3.3f;
        }

        // Schalte die LED an, wenn die Spannung größer als 1,4V ist
```

```
        if (spannung >= 1.4) {
            GR_LED_ON;
        } else {
            GR_LED_OFF;
        }

        // Ausgabe der gemessenen Spannung und Buffer leeren
        sprintf(usart2_tx_buffer, "Aktuelle Spannung: %.5f V\n", spannung);
        usart_2_print(usart2_tx_buffer);
        empty_buffers();

        wait_mSek(1000);
    }
}
```

Auszug aufgabe.h

```
// Aufgabe A09-01.1
void init_adc();
```

Auszug aufgabe.c

```
void init_adc()
{
    // ADC1 und GPIO Taktquelle einschalten
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1 , ENABLE);

    //GPIO - PA00 wird als Analogeingang Initialisieren
    GPIO_InitTypeDef GPIO_InitStructure;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL ;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    //ADC1 - ADC_CommonInitTypeDef
    ADC_CommonInitTypeDef ADC_CommonInitStructure;
    ADC_CommonInitStructure.ADC_Mode = ADC_Mode_Independent;
    ADC_CommonInitStructure.ADC_Prescaler = ADC_Prescaler_Div8;
    ADC_CommonInitStructure.ADC_DMAAccessMode = ADC_DMAAccessMode_Disabled;
    ADC_CommonInitStructure.ADC_TwoSamplingDelay = ADC_TwoSamplingDelay_5Cycles;
    ADC_CommonInit(&ADC_CommonInitStructure);

    //ADC1 - ADC_InitTypeDef
    ADC_InitTypeDef ADC_InitStructure;
    ADC_InitStructure.ADC_Resolution =  ADC_Resolution_12b;
```

```
    ADC_InitStructure.ADC_ScanConvMode = DISABLE;
    ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
    ADC_InitStructure.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_None;
    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_T1_CC1;
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
    ADC_InitStructure.ADC_NbrOfConversion = 1;
    ADC_Init((ADC_TypeDef *) ADC1_BASE, &ADC_InitStructure);

    //ADC1 - ADC_RegularChannelConfig (ADC1IN0-Leitung an PA0)
    ADC_RegularChannelConfig(ADC1, ADC_Channel_0, 1, ADC_SampleTime_3Cycles);

    //=== ADC1 Enable
    ADC_Cmd(ADC1, ENABLE);
}
```

Unsere Ausgabe hat eine Vor- und 5 Nachkommastellen. Die Anzahl der Vorkommastellen gibt uns die Möglichkeit, dass wir Spannungen von mindestens 1V anzeigen können, mehr als 3V dürfen für die Funktionalität des Boards nicht bereitgestellt werden. Aufgrund der gewählten Auflösung von 12 Bit und der Berechnung der eigentlichen Spannung sind maximal 16 Nachkommastellen möglich, wir haben uns aus Gründen der besseren Lesbarkeit für 5 Nachkommastellen entschieden.

A09.01-2

> Integrieren Sie eine Funktion die folgende Parameter erfasst. - die Batteriespannung - Temperatur des Mikrocontrollers
>
> Geben Sie im Rahmen der Anwendung die Parameter auf der USART2 aus.

main.c

```
#include "main.h"
#include "aufgabe.h"

int main(void)
{
    // Initialisierung des Systems und des Clocksystems
    SystemInit();

    // LED und Tasten initialisieren
    init_leds();

    // USART2 und Timer initialisieren
    init_usart_2();
    while(1)
```

```
    {
        init_adc();
        init_vbat_sensor();
        init_temp_sensor();
        wait_mSek(1000);
    }
}
```

Auszug aufgabe.h

```
// Aufgabe A09-01.1
void init_adc();

// Aufgabe A09-01.2
void init_vbat_sensor();
void init_temp_sensor();
```

Auszug aufgabe.c

```
void init_adc()
{
    // Resette ADC
    ADC_DeInit();

    // ADC1 und GPIO Taktquelle einschalten
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1 , ENABLE);

    //GPIO - PA00 wird als Analogeingang Initialisieren
    GPIO_InitTypeDef GPIO_InitStructure;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL ;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    //ADC1 - ADC_CommonInitTypeDef
    ADC_CommonInitTypeDef ADC_CommonInitStructure;
    ADC_CommonInitStructure.ADC_Mode = ADC_Mode_Independent;
    ADC_CommonInitStructure.ADC_Prescaler = ADC_Prescaler_Div8;
    ADC_CommonInitStructure.ADC_DMAAccessMode = ADC_DMAAccessMode_Disabled;
    ADC_CommonInitStructure.ADC_TwoSamplingDelay = ADC_TwoSamplingDelay_5Cycles;
    ADC_CommonInit(&ADC_CommonInitStructure);

    //ADC1 - ADC_InitTypeDef
    ADC_InitTypeDef ADC_InitStructure;
    ADC_InitStructure.ADC_Resolution =  ADC_Resolution_12b;
```

```
    ADC_InitStructure.ADC_ScanConvMode = DISABLE;
    ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
    ADC_InitStructure.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_None;
    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_T1_CC1;
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
    ADC_InitStructure.ADC_NbrOfConversion = 1;
    ADC_Init((ADC_TypeDef *) ADC1_BASE, &ADC_InitStructure);

    //ADC1 - ADC_RegularChannelConfig (ADC1IN0-Leitung an PA0)
    ADC_RegularChannelConfig(ADC1, ADC_Channel_0, 1, ADC_SampleTime_3Cycles);
    //=== ADC1 Enable
    ADC_Cmd(ADC1, ENABLE);

    // Starte die eigentliche Messung
    ADC_SoftwareStartConv(ADC1);

    uint16_t messwert = 0;
    float spannung = 0.0f;

    if(ADC_GetFlagStatus(ADC1,ADC_FLAG_EOC)==SET)
    {
        messwert = ADC_GetConversionValue(ADC1);
        spannung = (messwert/4095.0f) * 3.3f;
    }

    if(spannung >= 1.4) {
        GR_LED_ON;
    } else {
        GR_LED_OFF;
    }

    sprintf(usart2_tx_buffer, "Aktuelle Spannung: %.5f V\r\n", spannung);
    usart_2_print(usart2_tx_buffer);
    empty_buffers();

    //=== ADC1 Disable
    ADC_Cmd(ADC1, DISABLE);
}

void init_vbat_sensor()
{
    // Resette ADC
    ADC_DeInit();

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1 , ENABLE);
```

```
    //ADC1 - ADC_CommonInitTypeDef
    ADC_CommonInitTypeDef ADC_CommonInitStructure;
    ADC_CommonInitStructure.ADC_Mode = ADC_Mode_Independent;
    ADC_CommonInitStructure.ADC_Prescaler = ADC_Prescaler_Div8;
    ADC_CommonInitStructure.ADC_DMAAccessMode = ADC_DMAAccessMode_Disabled;
    ADC_CommonInitStructure.ADC_TwoSamplingDelay = ADC_TwoSamplingDelay_5Cycles;
    ADC_CommonInit(&ADC_CommonInitStructure);

    // ADC1 - ADC_InitTypeDef
    ADC_InitTypeDef ADC_InitStructure;
    ADC_InitStructure.ADC_Resolution =  ADC_Resolution_12b;
    ADC_InitStructure.ADC_ScanConvMode = DISABLE;
    ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
    ADC_InitStructure.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_None;
    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_T1_CC1;
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
    ADC_InitStructure.ADC_NbrOfConversion = 1;
    ADC_Init((ADC_TypeDef *) ADC1_BASE, &ADC_InitStructure);

    // ADC
    ADC_RegularChannelConfig(ADC1, ADC_Channel_Vbat, 1, ADC_SampleTime_3Cycles);
    // ADC freigeben
    ADC_Cmd(ADC1, ENABLE);
    // ADC
    ADC_VBATCmd(ENABLE);

    // Starte die eigentliche Messung
    ADC_SoftwareStartConv(ADC1);

    while(ADC_GetFlagStatus(ADC1,ADC_FLAG_EOC)==RESET);

    uint16_t messwert = ADC_GetConversionValue(ADC1);
    float spannung = 2.0f * (float) messwert * (3.3f / 4096.0f);

    sprintf(usart2_tx_buffer, "Aktuelle Batteriespannung: %.5f V\r\n", spannung);
    usart_2_print(usart2_tx_buffer);
    empty_buffers();

    // ADC Disable
    ADC_Cmd(ADC1, DISABLE);
    // Spannungsmessung ausschalten
    ADC_VBATCmd(DISABLE);
}

void init_temp_sensor()
{
```

```
// Resette ADC
ADC_DeInit();

RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1 , ENABLE);

//ADC1 - ADC_CommonInitTypeDef
ADC_CommonInitTypeDef ADC_CommonInitStructure;
ADC_CommonInitStructure.ADC_Mode = ADC_Mode_Independent;
ADC_CommonInitStructure.ADC_Prescaler = ADC_Prescaler_Div8;
ADC_CommonInitStructure.ADC_DMAAccessMode = ADC_DMAAccessMode_Disabled;
ADC_CommonInitStructure.ADC_TwoSamplingDelay = ADC_TwoSamplingDelay_5Cycles;
ADC_CommonInit(&ADC_CommonInitStructure);

// ADC1 - ADC_InitTypeDef
ADC_InitTypeDef ADC_InitStructure;
ADC_InitStructure.ADC_Resolution =  ADC_Resolution_12b;
ADC_InitStructure.ADC_ScanConvMode = DISABLE;
ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
ADC_InitStructure.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_None;
ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_T1_CC1;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStructure.ADC_NbrOfConversion = 1;
ADC_Init((ADC_TypeDef *) ADC1_BASE, &ADC_InitStructure);

// ADC1 - ADC_RegularChannelConfig (ADC_Channel_TempSensor)
ADC_RegularChannelConfig(ADC1, ADC_Channel_TempSensor, 1, ADC_SampleTime_480Cycles);

// ADC1 Enable
ADC_Cmd(ADC1, ENABLE);

// Temperatursensor einschalten
ADC_TempSensorVrefintCmd(ENABLE);

// Starte die eigentliche Messung
ADC_SoftwareStartConv(ADC1);

uint16_t messwert = 0;
float spannung = 0.0f;
float temp = 0.0f;

while(ADC_GetFlagStatus(ADC1,ADC_FLAG_EOC) == RESET);

messwert = ADC_GetConversionValue(ADC1);
spannung = (messwert/4095.0f) * 3.3f;
temp = (float) ((spannung - 0.76f)/0.0025f) + 25.0f;
```

```
    sprintf(usart2_tx_buffer, "Aktuelle Temperatur: %.5f C\r\n", temp);
    usart_2_print(usart2_tx_buffer);
    empty_buffers();

    // ADC1 Disable
    ADC_Cmd(ADC1, DISABLE);

    // Temperatursensor ausschalten
    ADC_TempSensorVrefintCmd(DISABLE);
}
```