Thema 5 – ADC (Interrupt)

Maximilian Roll, Martin Platajs

Inhalt

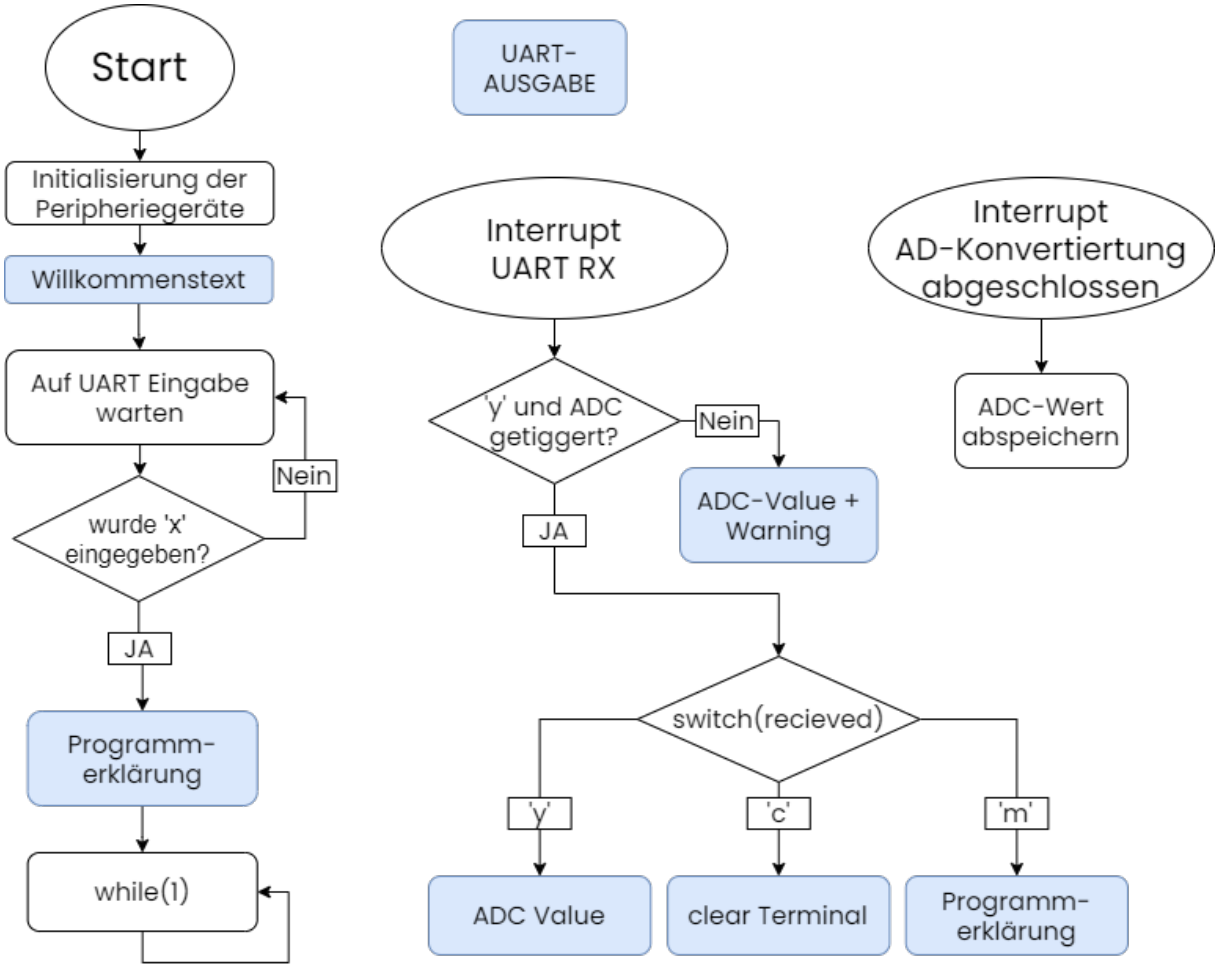
1 Aufgabenstellung

Es soll eine reale Anwendung mit der Hilfe des Microcontrollers STM32F10X realisiert werden. Die zu verwendenden Peripheriegeräte sind:

- Potentiometer auf DIL Adapter
- ADC im Continous-Conversion-Mode und Interrupt-gesteuert
- UART1 Interrupt-gesteuert

Zu programmieren ist ein UART-Request-Handler. Es sollen Anfragen via UART empfangen werden können, welche eine Funktion ausführen. Eine dieser Funktionen soll den Wert eines ADCs zurückgeben, der das analoges Signal eines Potentiometers, in ein digitales umwandelt. Die Konvertierung des ADCs soll durch einen externen Interrupt ausgelöst werden.

2 Blockschaltbild



3 Funktionsnachweis

Für den Funktionsnachweis das beigelegte Video ansehen.

4 Allgemeines

Für den externen Trigger des ADCs sind nur eine Handvoll Timer und die externe Line 11 zu Verfügung. Da der ADC im Continous Conversion Mode betrieben werden soll, mach hier ein Timer nur sehr wenig Sinn. Deshalb wurde der Port PB11 verwendet da es bei diesem Möglich ist diesen als Input zu definieren. Jedoch musste dafür die LED/Schalter Platine entfernt werden und mit einem Draht das Schalten Simulieren, weil PB11 auf einer LED angeschlossen ist. Die Anschlüsse PB11 (5), 3,3V(18) und Gnd(25) der Sub-D Buchse wurden verwendet.

5 Programmcode

5.1 Config.h

Hier werden die Peripheriegeräte initialisiert und die Libraries inkludiert.

5.1.1 Includes und Prototypen

```
      /* ----------- Includes -----------*/
#include <stdio.h>
#include "stm32f10x.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"
#include <stm32f10x_usart.h>
#include <stm32f10x_adc.h>
#include "stm32f10x_exti.h"


/* ----------- Prototypes -----------*/
static void RCC_Configuration(void);
static void NVIC_Configuration(void);
static void GPIO_Configuration(void);
static void USART_Configuration(void);
static void EXTI_Configuration(void);
static void ADC_Configuration(void);
static void USART_SendString (USART_TypeDef *USARTx, char *str);
```

Hier sind die Includes und Prototypen aufgelistet.

5.1.2 RCC

```
/*-------------------------------------------
  Reset and Clock Control, RCC initialization
 *-------------------------------------------*/
void RCC_Configuration(void){
  //enable clock for GPIOA, GPIOB, AFIO, ADC1 and USART1
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOB |
                         RCC_APB2Periph_AFIO | RCC_APB2Periph_ADC1 |
                          RCC_APB2Periph_USART1, ENABLE);
}
```

In der RCC-Initialisierung wird der Clock für GPIOA, GPIOB, AFIO, ADC1 und USART1 aktiviert.

5.1.3 NVIC

```
/*-------------------------------------------------------
  Nested Vectored Interrupt Controller, NVIC initialization
 *-------------------------------------------------------*/
void NVIC_Configuration(void){
  //Init NVIC for USART1; RX -> Interrupt (Request handling)
  NVIC_InitTypeDef NVIC_InitStructure;
  NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 3;
  NVIC_Init(&NVIC_InitStructure);

  //Init NVIC for ADC1_2; Conversion complete -> Interrupt (save value)
  NVIC_InitStructure.NVIC_IRQChannel = ADC1_2_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 3;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);
}
```

Hier wird der NVIC für den UART und ADC1_2 Interrupt initialisiert. Der UART Interrupt wird ausgelöst, wenn Daten über RX empfangen werden, und der ADC1_2 Interrupt wird ausgelöst, wenn die Konvertierung des ADCs abgeschlossen wurde.

5.1.4 GPIO

```
/*-------------------------------------------------
  General Purpose Input Output, GPIO initialization
 *-------------------------------------------------*/
void GPIO_Configuration(void){
    GPIO_InitTypeDef GPIO_InitStructure;

    // Set PA9 to alternate function push pull (Tx)
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    // Set PA10 to input floating (Rx)
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    // Set PB11 to input floating (external interrupt)
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_11;
    GPIO_Init(GPIOB, &GPIO_InitStructure);
```

```
    // Set PC4 to analog input (ADC1 channel 14)
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;
    GPIO_Init(GPIOC, &GPIO_InitStructure);
}
```

Hier werden alle verwendeten GPIO Ports initialisiert. PA9 ist die TX Leitung des UARTs und wird im Alternate-Function-Push-Pull Betrieb verwendet. Die RX Leitung des UARTs auf PA10 wird mit dem Input-floating Modus initialisiert. Für den externen Interrupt der den ADC triggern soll, wird mit dem Port PB11 und im Modus Input-floating initialisiert. Zuletzt wird der Pin für des Potentiometers mit PC4 und dem Modus analog-input eingestellt.

5.1.5 EXTI

```
/*---------------------------------------
  External Interrupt, EXTI initialization
 *---------------------------------------*/
 void EXTI_Configuration(void){
  EXTI_InitTypeDef EXTI_InitStructure;
  //Set external Interrupt Line to PB11
  GPIO_EXTILineConfig(GPIO_PortSourceGPIOB, GPIO_PinSource11);
  //setup external interrupt for ADC Conversion start
  EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Event;   //Mode: event (ADC1)
  EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;  //rising edge
  EXTI_InitStructure.EXTI_Line = EXTI_Line11;   //Line 11 (PB11)
  EXTI_InitStructure.EXTI_LineCmd = ENABLE;
  EXTI_Init(&EXTI_InitStructure);
 }
```

Hier wird das externe Interrupt für die Triggerung des ADCs initialisiert. Der Interrupt wird im event-Modus verwendet, da er mit dem ADC verknüpft ist. Außerdem soll er bei steigenden Flanken auslösen.

5.1.6 UART

```
/*-------------------------------------------------------------------------
----
  Universal Synchronous/Asynchronous Receiver Transmitter, USART initializat
ion
 *-------------------------------------------------------------------------
---*/
void USART_Configuration(void){
  USART_InitTypeDef USART_InitStructure;
  //Init USART1 to 9600 baud
  USART_InitStructure.USART_BaudRate = 9600;
  USART_InitStructure.USART_WordLength = USART_WordLength_8b;
  USART_InitStructure.USART_StopBits = USART_StopBits_1;
  USART_InitStructure.USART_Parity = USART_Parity_No;
  USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_
None;
  USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;

  USART_ClockInitTypeDef Usart_ClockInitStructure;
  USART_ClockInit(USART1, &Usart_ClockInitStructure);

    USART_Init(USART1, &USART_InitStructure);
    USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
    USART_Cmd(USART1, ENABLE);
}
```

Hier wird der UART mit einer Baudrate von 9600 initialisiert.

5.1.7 ADC

```
/*---------------------------------------
  Analog-Digital-Converter, ADC initialization
 *---------------------------------------*/
void ADC_Configuration(void){
   ADC_InitTypeDef ADC_InitStructure;
   //Init ADC1 with Continous Conversion Mode and external Trigger
   ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
   ADC_InitStructure.ADC_ScanConvMode = DISABLE;
   ADC_InitStructure.ADC_ContinuousConvMode = ENABLE; //Continous Conversion
 Mode
   ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_Ext_IT11_TI
M8_TRGO;  //external trigger line 11
   ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
   ADC_InitStructure.ADC_NbrOfChannel = 1;
   ADC_Init(ADC1, &ADC_InitStructure);

   ADC_RegularChannelConfig(ADC1, ADC_Channel_14, 1, ADC_SampleTime_28Cycles
5);
   ADC_ExternalTrigConvCmd(ADC1, ENABLE);
   ADC_ITConfig(ADC1, ADC_IT_EOC, ENABLE);
   ADC_Cmd(ADC1, ENABLE);

  ADC_ResetCalibration(ADC1);         //ADC Calibration
  while(ADC_GetResetCalibrationStatus(ADC1));
  ADC_StartCalibration(ADC1);
  while(ADC_GetCalibrationStatus(ADC1));
}
```

Hier wird der ADC im Continous Conversion Mode und mit externem Trigger initialisiert und kalibriert.

5.1.8 USART_SendString()

```
/*---------------------------------------------------------------
  Sends a string, character by character, over the specified UART
 *---------------------------------------------------------------*/
void USART_SendString(USART_TypeDef *USARTx, char *str)
{
  while (*str) {
    while (USART_GetFlagStatus(USARTx, USART_FLAG_TXE) == RESET);
    USART_SendData(USARTx, *str++);
  }
}
```

Diese Funktion wird zum Senden eines ganzen Strings über UART verwendet.

5.2 Main.c

In diesem File beginnt die Exekution des ganzen Projekts.

5.2.1 Includes und Variablen

```
/* ---------------------------- Includes ----------------------*/

#include "config.h"

/* ---------------------------- Varibales ----------------------*/

static char recieved;       //variable for value recieved from USART
static uint16_t value = 0;  //variable for value from ADC1 channel 14
char buffer[20];            //buffor for response messages
```

Hier wird das zuvor beschriebene Header-File includiert, und 3 Variablen werden angelegt. „Recieved“ um abzuspeichern, welches Zeichen über UART empfangen wurde. „value“ um den, vom ADC konvertierten Wert, abzuspeichern. „buffer“ dient als Buffer für Strings, die für das Senden via UART vorbereitet werden.

5.2.2 ADC1_2_IRQHandler()

```
/*-------------------------------------------------------------------------
  Interrupt service routine for ADC1 (and ADC2), saves ADC-value to "value"
 *-------------------------------------------------------------------------
*/
void ADC1_2_IRQHandler()    //interrupt service routine for ADC1 (and ADC2)
{
    value = ADC_GetConversionValue(ADC1);
}
```

Hier wurde die Interrupt-Service-Routine des ADCs ausprogrammiert. Dieser Interrupt wird jedes Mal aufgerufen, wenn die Konvertierung eines Wertes abgeschlossen wurde. Innerhalb dieser Routine wird der konvertierte Wert in der Variable „value“ abgespeichert.

5.2.3 USART1_IRQHandler()

```
/*-------------------------------------------------------------------
  Interrupt service routine for USART RX, checks for USART-requests
 *-------------------------------------------------------------------*/
void USART1_IRQHandler()
{
    recieved = USART_ReceiveData(USART1);
    if(recieved == 'y' && ADC_GetFlagStatus(ADC1,ADC_FLAG_STRT) == RESET){
            sprintf(buffer,"\r\nADC Conversion not triggered yet!\r\nADC-
Value: %d\r\n",value);
            USART_SendString(USART1, buffer);
        }
        else{
        switch(recieved){

        case 'y':
            sprintf(buffer,"\r\nADC-Value: %d\r\n",value);
            USART_SendString(USART1, buffer);
            break;

        case 'c':
            USART_SendData(USART1,12);
            break;

        case 'm':
            sprintf(buffer,"\r\n-> Press y to get ADC Value\r\n-
> Press c to clear Console\r\n-
> Press m to display above mentioned Commands\r\n");
            USART_SendString(USART1, buffer);
            break;
        }
    }
}
```

Hier wurde die Interrupt-Service-Routine des UARTs ausprogrammiert. Dieser Interrupt wird jedes Mal aufgerufen, wenn über die RX Leitung ein Zeichen empfangen wurde. In dieser Routine werden die einzelnen Befehle ausprogrammiert. So gibt ‚y' eine Warnung mit dem ADC-Wert aus, wenn der ADC noch nicht getriggert wurde. Ist dieser getriggert, wird mit ‚y' der ADC-Wert zurückgegeben, mit ‚c' das Terminal gecleared, und mit ‚m' nochmals alle Befehle angezeigt.

5.2.4 Hauptprogramm

```
/*-------------------------------------------------------------------
  Main; initialize peripherals; send welcome text to USART1
 *-------------------------------------------------------------------*/
int main(){

    RCC_Configuration();    //load RCC Configuration
    NVIC_Configuration();   //load NVIC Configuration
    GPIO_Configuration();   //load GPIO Configuration
    USART_Configuration();  //load USART Configuration
    EXTI_Configuration();   //load EXTI Configuration
    ADC_Configuration();    //load ADC Configuration

    USART_SendData(USART1,12);  //clear console
    USART_SendString(USART1,"\r\n*****************\r\n*  ADC-
Request  *\r\n*****************\r\n\r\n"); //welcome text
    USART_SendString(USART1,"press x to start\r\n\r\n");    //press x for fu
rther execution
    while(recieved != 'x'){}                                //wait for x to
be pressed
    USART_SendString(USART1,"-
> Make sure ADC is triggered!\r\n");      // remove LED/Switches circuit boa
rd
    USART_SendString(USART1,"-
> Press y to get ADC Value\r\n");         // for further information see the
 protocol
    USART_SendString(USART1,"-> Press c to clear Console\r\n");
    USART_SendString(USART1,"-
> Press m to display above mentioned Commands\r\n");

    while(1);
}
```

Im Hauptprogramm werden zunächst alle Initialisierungen durchgeführt. Anschließend wird man aufgefordert über UART ein ‚x' zu übertragen, um das Programm zu starten. Danach folgt ein Willkommenstext mit Erklärung zu den Befehlen. Anschließend geht das Programm in eine Endlosschleife.