Mikroprozessorpraktikum

Konstantin Bork, Kean Seng Liew, & Oliver Stein, Gruppe A, HWP8

03-01 USART

A03-01.1

Konfigurieren Sie im Rahmen der Funktion init_usart_2_tx() die USART2 mit folgender Einstellung: - Portleitung PA02 als TxD2 der USART2 - Baudrate 921600 Bit/s - 8 Datenbits - 1 Stopbit - keine Parität - keine Hardware Flußkontrolle - nur den USART Mode Tx konfigurieren

Realisieren Sie in einer Endlosschleife die zyklische Ausgabe der Zahlen 0...9 im Sekundentakt. Nutzen Sie zur Ansteuerung der USART die STM Library mit den Dateien stm32fxx_usart.h und stm32fxx_usart.c. Wenden Sie die folgende Funktionen und Flags an:

```
USART_SendData (USART_TypeDef *USARTx, uint16_t Data)
USART_GetFlagStatus (USART_TypeDef *USARTx, uint16_t USART_FLAG)
USART_FLAG_TC
```

Benötigte Variablen legen Sie bitte in aufgabe.c und aufgabe.h an, wenn diese global benötigt werden.
Schließen Sie das serielle Kabel an. Starten Sie TeraTerm und Überprüfen Sie die Funktionsfähigkeit des Programms.

aufgabe.h

```
#ifndef __aufgabe_h__
#define __aufgabe_h__

//################################################################################
//########## cmsis_lib include
//################################################################################
#include "stm32f4xx.h"
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_rcc.h"
#include "stm32f4xx_usart.h"

//################################################################################
//########## mpp_lib include
//################################################################################
```

```
#include "init.h"
#include "interrupts.h"
#include "led.h"
#include "taster.h"

//############################################################################
//########## Eigene Funktionen, Macros und Variablen
//############################################################################
//============================================================================
// Macros
//============================================================================
#define GR_LED_ON        (GPIO_SetBits(GPIOB, GPIO_Pin_2))
#define GR_LED_OFF       (GPIO_ResetBits(GPIOB, GPIO_Pin_2))
#define GR_LED_Toggle    (GPIO_ToggleBits(GPIOB, GPIO_Pin_2))

#define SEC_IN_USEC       1000000

//============================================================================
// Funktionen
//============================================================================
// Aufgabe A01-01
extern void init_leds();

// Aufgabe A01-02
extern void init_taste_1();
extern void init_taste_2();
extern int led_steuerung();

// Aufgabe A02-01
extern void init_PC09();
extern void fastMode();
extern void slowMode();

// Aufgabe A03-01
extern void init_usart_2_tx();
//============================================================================
#endif
```

Auszug aufgabe.c

```
// Aufgabe A03-01.1
void init_usart_2_tx() {
    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;

    // Taktsystem für die USART2 freigeben
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);

    // GPIO Port A Taktsystem freigeben
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);

    // USART2 TX an PA2 mit Alternativfunktion Konfigurieren
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
```

```
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    // USART2 TX mit PA2 verbinden
    GPIO_PinAFConfig(GPIOA, GPIO_PinSource2, GPIO_AF_USART2);

    // Datenprotokoll der USART einstellen
    USART_InitStructure.USART_BaudRate = 921600;
    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_Tx;
    USART_Init(USART2, &USART_InitStructure);

    // USART2 freigeben
    USART_Cmd(USART2, ENABLE); // enable USART2
}
```

main.c

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

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

    // SysTick initialisieren
    // jede ms erfolgt dann der Aufruf
    // des Handlers fuer den Interrupt SysTick_IRQn
    InitSysTick();

    // Initialisiere PA02 für USART
    init_usart_2_tx();

    int i;

    while(1)
    {
        for(i = 0;i < 10;i++)
        {
            USART_SendData(USART2, i);
            while (USART_GetFlagStatus(USART2, USART_FLAG_TC) == RESET){}
            wait_mSek(1000);
        }
    }
}
```

A03-01.2

Entwickeln Sie die Funktion usart_2_print(char * zeichenkette) zur Ausgabe einer

Zeichenkette auf die USART2. Generieren Sie dazu in einer Endlosschleife jeweils eine Zeichenkette mit einem Laufindex, einem Text und der Steuerzeichen Sequenz '\r\n' am Ende der Zeichenkette. Die Zeichenkette soll der Funktion übergeben werden. Innerhalb der Funktion erfolgen die Bestimmung der Zeichenkettenlänge und die zeichenweise Ausgabe auf die USART2.
Legen Sie bitte folgenden Variablen für die weitere Bearbeitung der Aufgabenstellungen an.

```
    char usart2_rx_buffer[50];
    char usart2_tx_buffer[50];
```

Auszug aufgabe.c

```
void usart_2_print(char* zeichenkette)
{
    int i = 0;
    for(i = 0;i < strlen(zeichenkette);i++)
    {
        USART_SendData(USART2, zeichenkette[i]);
        while (USART_GetFlagStatus(USART2, USART_FLAG_TC) == RESET){}
    }
}
```

Auszug aufgabe.h

```
//=============================================================================
// Variablen
//=============================================================================
// Aufgabe A03-01.2
extern char usart_rx_buffer[50];
extern char usart_tx_buffer[50];

//=============================================================================
// Funktionen
//=============================================================================
// Aufgabe A01-01
extern void init_leds();

// Aufgabe A01-02
extern void init_taste_1();
extern void init_taste_2();
extern int led_steuerung();

// Aufgabe A02-01
extern void init_PC09();
extern void fastMode();
extern void slowMode();

// Aufgabe A03-01
extern void init_usart_2_tx();
```

```
extern void usart_2_print();
```

A03-01.3

Ändern Sie die Konfiguration der USART2 in der Form, daß auch Zeichen empfangen werden können. Binden Sie diese Konfiguration in eine Funktion init_usart_2() ein. Entwickeln Sie für die while(1) Schleife in der main eine Empfangsroutine für die von der USART2 empfangenen Zeichen. Dazu soll die USART2 auf Zeichen im Polling abgefragt werden und die folgende Funktionalität realisiert: - Das Empfangene Zeichen "1" toggelt die grüne LED im 1 Sekundentakt und gibt auf der USART2 die Zeichenkette "grüne LED im 1 Sekundentakt\r\n" aus. - Das Empfangene Zeichen "2" toggelt die grüne LED im 2 Sekundentakt und gibt auf der USART2 die Zeichenkette "grüne LED im 2 Sekundentakt\r\n" aus. - Das Empfangene Zeichen "4" toggelt die grüne LED im 4 Sekundentakt und gibt auf der USART2 die Zeichenkette "grüne LED im 4 Sekundentakt\r\n" aus.

aufgabe.h

```
#ifndef __aufgabe_h__
#define __aufgabe_h__

//################################################################################
//########## cmsis_lib include
//################################################################################
#include "stm32f4xx.h"
//#include "misc.h"
//#include "stm32f4xx_adc.h"
//#include "stm32f4xx_can.h"
//#include "stm32f4xx_crc.h"
//#include "stm32f4xx_cryp_aes.h"
//#include "stm32f4xx_cryp_des.h"
//#include "stm32f4xx_cryp_tdes.h"
//#include "stm32f4xx_cryp.h"
//#include "stm32f4xx_dac.h"
//#include "stm32f4xx_dbgmcu.h"
//#include "stm32f4xx_dcmi.h"
//#include "stm32f4xx_dma.h"
//#include "stm32f4xx_exti.h"
//#include "stm32f4xx_flash.h"
//#include "stm32f4xx_fsmc.h"
#include "stm32f4xx_gpio.h"
//#include "stm32f4xx_hash_md5.h"
//#include "stm32f4xx_hash_sha1.h"
//#include "stm32f4xx_hash.h"
//#include "stm32f4xx_i2c.h"
//#include "stm32f4xx_iwdg.h"
//#include "stm32f4xx_pwr.h"
#include "stm32f4xx_rcc.h"
//#include "stm32f4xx_rng.h"
//#include "stm32f4xx_rtc.h"
```

```
//#include "stm32f4xx_sdio.h"
//#include "stm32f4xx_spi.h"
//#include "stm32f4xx_syscfg.h"
//#include "stm32f4xx_tim.h"
#include "stm32f4xx_usart.h"
//#include "stm32f4xx_wwdg.h"


//#################################################################################
//########## mpp_lib include
//#################################################################################
//#include "BME280.h"
//================================================================================
//#include "beeper.h"
//#include "client_ftp.h"
//#include "client_ntp.h"
//#include "global.h"
//#include "i2c.h"
#include "init.h"
#include "interrupts.h"
#include "led.h"
//#include "power.h"
//#include "rtc.h"
#include "taster.h"
//#include "usart.h"
//================================================================================
//#include "simplelink.h"
//#include "netapp.h"
//#include "CC3100.h"
//#include "CC3100_Board.h"
//================================================================================
//#include "dw1000_driver.h"
//#include "dw1000_ranging.h"
//#include "DW1000.h"
//================================================================================
//#include "MQTT.h"
//================================================================================
//#include "MPU9250.h"
//================================================================================
//#include "SDCARD.h"
//================================================================================
//#include "usbd_cdc_vcp.h"
//#include "usb_conf.h"
//================================================================================
//#include "CoOS.h"
//#include "mpu9250_driver.h"


//#################################################################################
//########## standart_lib include
//#################################################################################
//#include "stdio.h"
//#include "string.h"
//#include "math.h"


//#################################################################################
//########## Eigene Funktionen, Macros und Variablen
```

```
//#############################################################################


//=============================================================================
// Macros
//=============================================================================
#define GR_LED_ON        (GPIO_SetBits(GPIOB, GPIO_Pin_2))
#define GR_LED_OFF       (GPIO_ResetBits(GPIOB, GPIO_Pin_2))
#define GR_LED_Toggle    (GPIO_ToggleBits(GPIOB, GPIO_Pin_2))

#define SEC_IN_USEC       1000000

//=============================================================================
// Variablen
//=============================================================================
// Aufgabe A03-01
extern char usart_rx_buffer[50];
extern char usart_tx_buffer[50];

//=============================================================================
// Funktionen
//=============================================================================
// Aufgabe A01-01
extern void init_leds();

// Aufgabe A01-02
extern void init_taste_1();
extern void init_taste_2();
extern int led_steuerung();

// Aufgabe A02-01
extern void init_PC09();
extern void fastMode();
extern void slowMode();

// Aufgabe A03-01
extern void init_usart_2_tx();
extern void usart_2_print(char* zeichenkette);
extern void init_usart_2();
//=============================================================================
#endif
```

aufgabe.c

```
#include "aufgabe.h"

// Aufgabe A01-01.3
// Initialisiert die Portleitung der grünen LED und schaltet diese ein
void init_leds() {
    // Setzt GPIO Port auf den Reset Zustand zurück
    GPIO_DeInit(GPIOB);

    // Taktquelle für die Peripherie aktivieren
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);

    // Struct anlegen
```

```
    GPIO_InitTypeDef GPIO_InitStructure;

    // Struct Initialisieren setzt alle Leitungen auf
    // Eingang ohne PushPull
    GPIO_StructInit(&GPIO_InitStructure);

    // Die Funktionalität der Portleitungen festlegen

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;

    // Auswahl des I/O Mode
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT; //GPIO Output Mode

    // Auswahl der Speed
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz; // Low speed

    // Auswahl des Output Typs
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP; // PushPull

    // Auswahl des Push/Pull Typs
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL; // NoPull

    // PortLeitungen initialisieren
    GPIO_Init(GPIOB, &GPIO_InitStructure);

    // Schaltet die LED ein
    GR_LED_ON;

    // LED wurde initialisiert, LED ausschalten
    GR_LED_OFF;
}

// Aufgabe A01-02.2
// Funktion zur Initialisierung beider Tasten
void init_taste(uint16_t GPIO_Pin) {
    // Setzt GPIO Port auf den Reset Zustand zurück
    GPIO_DeInit(GPIOC);

    // Taktquelle für die Peripherie aktivieren
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);

    // Struct anlegen
    GPIO_InitTypeDef GPIO_InitStructure;

    // Struct Initialisieren setzt alle Leitungen auf
    // Eingang ohne PushPull
    GPIO_StructInit(&GPIO_InitStructure);

    // Die Funktionalität der Portleitungen festlegen

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin;

    // Auswahl des I/O Mode
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN; // GPIO Input Mode

    // Auswahl der Speed
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz; // Low speed
```

```
    // Auswahl des Output Typs
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP; // PushPull

    // Auswahl des Push/Pull Typs
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL; // NoPull

    // PortLeitungen initialisieren
    GPIO_Init(GPIOC, &GPIO_InitStructure);
}

// Initialisierung von Taste 1
void init_taste_1() {
    init_taste(GPIO_Pin_8);
}

// Initialisierung von Taste 2
void init_taste_2() {
    init_taste(GPIO_Pin_5);
}

// Aufgabe A01-02.3
// Programm zum Kontrollieren der grünen LED mit den beiden Tasten
int led_steuerung() {
    uint8_t Byte = 0;

    Byte = GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_5);

    if (Byte == Bit_SET) {
        return 1;
    }

    Byte = GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_8);

    if (Byte != Bit_SET) {
        return -1;
    }

    return 0;
}

// Aufgabe A02-01.1
// Initialisierung von Portleitung 9 für die SYSCLK
void init_PC09() {
    // Setzt GPIO Port auf den Reset Zustand zurück
    GPIO_DeInit(GPIOC);

    // Taktquelle für die Peripherie aktivieren
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);

    // Struct anlegen
    GPIO_InitTypeDef GPIO_InitStructure;

    // Struct Initialisieren setzt alle Leitungen auf
    // Eingang ohne PushPull
    GPIO_StructInit(&GPIO_InitStructure);

    // Benutze Pin 9 gemäß Aufgabe
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
```

```
    // Setze den Modus auf Alternate Function, da wir keine "richtigen" I/O-
Aufgaben lösen wollen
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;

    // Setze den Output Typ auf Push/Pull
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;

    // Verwende PullUp
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;

    // Setze die Frequenz des Ports auf 50 MHz
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;

    // Initialisiere den GPIO Port
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    // Konfiguriert PC09 für den Alternate Function Modus, verbindet MCO mit PC09
    GPIO_PinAFConfig(GPIOC, GPIO_Pin_9, GPIO_AF_MCO);

    // Konfiguriert MCO2 und PC09 so, dass die SYSCLK Taktquelle über MCO2 an PC09
ausgegeben wird
    RCC_MCO2Config(RCC_MCO2Source_SYSCLK, RCC_MCO2Div_1);
}

// Aufgabe A02-01.4
// PLLStartUp Hilfsfunktion
void RCC_WaitForPLLStartUp(void) {
    while ((RCC->CR & RCC_CR_PLLRDY) == 0) {
        __NOP();
    }
}

//==== Taktfrequenz 24MHz mit HSE-OSC=16MHz
void slowMode(void) {
    RCC_DeInit();

    RCC_HSEConfig(RCC_HSE_ON);
    if (RCC_WaitForHSEStartUp() == ERROR) {
        return;
    }
    // HSEOSC=16MHz SYSCLK=24MHz HCLK=24MHz
    // PCLK1=24 PCLK2=24MHz
    RCC_PLLConfig(RCC_PLLSource_HSE,    //RCC_PLLSource
                  16,     //PLLM
                  192,    //PLLN
                  8,      //PLLP
                  4       //PLLQ
    );
    RCC_PLLCmd(ENABLE);
    RCC_WaitForPLLStartUp();

    // Configures the AHB clock (HCLK)
    RCC_HCLKConfig(RCC_SYSCLK_Div1);
    // Low Speed APB clock (PCLK1)
    RCC_PCLK1Config(RCC_HCLK_Div1);
    // High Speed APB clock (PCLK2)
    RCC_PCLK2Config(RCC_HCLK_Div1);
```

```
    // select system clock source
    RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);
}

//==== Taktfrequenz 168MHz mit HSE-OSC=16MHz
void fastMode(void) {
    RCC_DeInit();

    RCC_HSEConfig(RCC_HSE_ON);
    if (RCC_WaitForHSEStartUp() == ERROR) {
        return;
    }
    // HSEOSC=16MHz SYSCLK=168MHz HCLK=168MHz
    // PCLK1=42MHz PCLK2=84MHz
    RCC_PLLConfig(RCC_PLLSource_HSE,    //RCC_PLLSource
                  16,     //PLLM
                  336,    //PLLN
                  2,      //PLLP
                  7       //PLLQ
    );
    RCC_PLLCmd(ENABLE);
    RCC_WaitForPLLStartUp();

    // Configures the AHB clock (HCLK)
    RCC_HCLKConfig(RCC_SYSCLK_Div1);
    // High Speed APB clock (PCLK1)
    RCC_PCLK1Config(RCC_HCLK_Div4);
    // High Speed APB clock (PCLK2)
    RCC_PCLK2Config(RCC_HCLK_Div2);


    // select system clock source
    RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);
}

// Aufgabe A03-01.1
void init_usart_2_tx() {
    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;

    // Taktsystem für die USART2 freigeben
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);

    // GPIO Port A Taktsystem freigeben
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);

    // USART2 TX an PA2 mit Alternativfunktion Konfigurieren
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    // USART2 TX mit PA2 verbinden
    GPIO_PinAFConfig(GPIOA, GPIO_PinSource2, GPIO_AF_USART2);

    // Datenprotokoll der USART einstellen
```

```
    USART_InitStructure.USART_BaudRate = 921600;
    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_Tx;
    USART_Init(USART2, &USART_InitStructure);

    // USART2 freigeben
    USART_Cmd(USART2, ENABLE); // enable USART2
}

void usart_2_print(char* zeichenkette)
{
    int i = 0;
    for(i = 0;i < strlen(zeichenkette);i++)
    {
        USART_SendData(USART2, zeichenkette[i]);
        while (USART_GetFlagStatus(USART2, USART_FLAG_TC) == RESET){}
    }
}

char usart_rx_buffer[50];
char usart_tx_buffer[50];

void init_usart_2() {
    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;

    // Taktsystem für die USART2 freigeben
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);

    // GPIO Port A Taktsystem freigeben
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);

    // USART2 TX an PA2 mit Alternativfunktion Konfigurieren
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    // USART2 TX mit PA2 verbinden
    GPIO_PinAFConfig(GPIOA, GPIO_PinSource2, GPIO_AF_USART2);

    // Datenprotokoll der USART einstellen
    USART_InitStructure.USART_BaudRate = 921600;
    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_Tx | USART_Mode_Rx;
    USART_Init(USART2, &USART_InitStructure);

    // USART2 freigeben
    USART_Cmd(USART2, ENABLE); // enable USART2
}
```

Auszug interrupts.c

```
#include "interrupts.h"
#include "aufgabe.h"

int32_t timer = 0;

void hard_fault_handler_c(unsigned int * hardfault_args);

//============================================================================
void SysTick_Handler(void)
{
    static unsigned long stc_led = 0;
    static unsigned long stc0 = 0;
    static unsigned long stc1 = 0;
    static unsigned long stc2 = 0;
    static unsigned long stc3 = 0;
    stc_led++;
    stc0++;
    stc1++;
    stc2++;
    stc3++;

    //============================================================================
    // DW1000 Timeout
    systickcounter += 1;
    if ( stc0 >= 20 )
        {
            //uwbranging_tick();
            stc0 = 0;
        }



    //============================================================================
    //  CoOS_SysTick_Handler alle 10ms in CoOs arch.c aufrufen
    // nur Einkommentieren wenn CoOS genutzt wird
    if ( stc1 >= 10 )
        {
    //      CoOS_SysTick_Handler();
            stc1 = 0;
        }

    //============================================================================
    // CC3100 alle 50ms Sockets aktualisieren
    if (stc2 >= 50)
        {
            stc2 = 0;
            if ( (IS_CONNECTED(WiFi_Status)) && (IS_IP_ACQUIRED(WiFi_Status)) &&
(!Stop_CC3100_select) && (!mqtt_run) )
            {
            CC3100_select(); // nur aktiv wenn mit AP verbunden
            }
            else
            {
```

```
            _SlNonOsMainLoopTask();
            }
        }
    //=========================================================================
    // SD-Card
    sd_card_SysTick_Handler();

    //=========================================================================
    // MQTT
    MQTT_SysTickHandler();

    //=========================================================================
    // LED laut Aufgabe schalten
    // nach 500mS schalten wir die LED aus
    /*
    if ( stc_led >= 500 )
        {
            LED_GR_OFF;
        }

    // nach weiteren 3000mS schalten wir sie wieder an
    // und setzen den Zähler wieder auf 0
    if ( stc_led >= 3500 )
        {
            LED_GR_ON;
            stc_led = 0;
        }
    */

    if(stc3 >= timer)
        {
            GR_LED_Toggle;
            stc3 = 0;
        }
}
```

main.c

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

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

    // SysTick initialisieren
    // jede ms erfolgt dann der Aufruf
    // des Handlers fuer den Interrupt SysTick_IRQn
    InitSysTick();

    // Initialisiere die grüne LED
    init_leds();

    // Initialisiere PA02 für USART
    init_usart_2();
```

```
    char zeichen;
    int i = 0;
    timer = 1000;

    while(1)
    {
        // Prüfe, ob eine Eingabe vorhanden ist
        if (USART_GetFlagStatus(USART2, USART_FLAG_RXNE) != RESET)
        {
            // Lese das Zeichen
            zeichen = (char) USART_ReceiveData(USART2);
            usart_rx_buffer[i] = zeichen;

            // Prüfe, ob die Eingabe erlaubt ist, also 1, 2 oder 4 ist
            if(zeichen == '1' || zeichen == '2' || zeichen == '4') {
                // Setze das Intervall gemäß der Eingabe
                // und drucke das Ergebnis aus
                int eingabe = zeichen - '0';
                timer = eingabe * 1000;
                sprintf(usart_tx_buffer, "grüne LED im %d Sekundentakt", eingabe);
                usart_2_print(usart_tx_buffer);
            }
            i = (i + 1) % 50;
        }
    }
}
```

A03-01.4

> Über das Terminalprogramm TeraTerm auf dem Arbeitsplatzrechner soll eine Zeichenkette eingegeben und am Ende mit Carriage return (CR, '\r', 0x0D, 13) abgeschlossen werden. Diese Zeichenkette wird automatisch über die USART zum STM32 übertragen. Ein Programm auf dem STM32 soll die Zeichenkette (Zeichen für Zeichen) im Polling empfangen, die Länge der Zeichenkette bestimmen und die Zeichenkette gefolgt von der ermittelten Länge als Zahl über die USART zum PC zurückschicken. Die Länge der Zeichenkette wird dabei mit Hilfe des Steuerzeichens für Carriage return am Ende der Zeichenkette bestimmt.

aufgabe.h

```
#ifndef __aufgabe_h__
#define __aufgabe_h__

//################################################################################
//########## cmsis_lib include
//################################################################################
#include "stm32f4xx.h"
//#include "misc.h"
//#include "stm32f4xx_adc.h"
//#include "stm32f4xx_can.h"
//#include "stm32f4xx_crc.h"
```

```
//#include "stm32f4xx_cryp_aes.h"
//#include "stm32f4xx_cryp_des.h"
//#include "stm32f4xx_cryp_tdes.h"
//#include "stm32f4xx_cryp.h"
//#include "stm32f4xx_dac.h"
//#include "stm32f4xx_dbgmcu.h"
//#include "stm32f4xx_dcmi.h"
//#include "stm32f4xx_dma.h"
//#include "stm32f4xx_exti.h"
//#include "stm32f4xx_flash.h"
//#include "stm32f4xx_fsmc.h"
#include "stm32f4xx_gpio.h"
//#include "stm32f4xx_hash_md5.h"
//#include "stm32f4xx_hash_sha1.h"
//#include "stm32f4xx_hash.h"
//#include "stm32f4xx_i2c.h"
//#include "stm32f4xx_iwdg.h"
//#include "stm32f4xx_pwr.h"
#include "stm32f4xx_rcc.h"
//#include "stm32f4xx_rng.h"
//#include "stm32f4xx_rtc.h"
//#include "stm32f4xx_sdio.h"
//#include "stm32f4xx_spi.h"
//#include "stm32f4xx_syscfg.h"
//#include "stm32f4xx_tim.h"
#include "stm32f4xx_usart.h"
//#include "stm32f4xx_wwdg.h"


//#############################################################################
//########## mpp_lib include
//#############################################################################
//#include "BME280.h"
//=============================================================================
//#include "beeper.h"
//#include "client_ftp.h"
//#include "client_ntp.h"
//#include "global.h"
//#include "i2c.h"
#include "init.h"
#include "interrupts.h"
#include "led.h"
//#include "power.h"
//#include "rtc.h"
#include "taster.h"
//#include "usart.h"
//=============================================================================
//#include "simplelink.h"
//#include "netapp.h"
//#include "CC3100.h"
//#include "CC3100_Board.h"
//=============================================================================
//#include "dw1000_driver.h"
//#include "dw1000_ranging.h"
//#include "DW1000.h"
//=============================================================================
//#include "MQTT.h"
//=============================================================================
```

```
//#include "MPU9250.h"
//=============================================================================
//#include "SDCARD.h"
//=============================================================================
//#include "usbd_cdc_vcp.h"
//#include "usb_conf.h"
//=============================================================================
//#include "CoOS.h"
//#include "mpu9250_driver.h"


//#############################################################################
//########## standart_lib include
//#############################################################################
//#include "stdio.h"
//#include "string.h"
//#include "math.h"


//#############################################################################
//########## Eigene Funktionen, Macros und Variablen
//#############################################################################


//=============================================================================
// Macros
//=============================================================================
#define GR_LED_ON        (GPIO_SetBits(GPIOB, GPIO_Pin_2))
#define GR_LED_OFF       (GPIO_ResetBits(GPIOB, GPIO_Pin_2))
#define GR_LED_Toggle    (GPIO_ToggleBits(GPIOB, GPIO_Pin_2))

#define SEC_IN_USEC       1000000

//=============================================================================
// Variablen
//=============================================================================
// Aufgabe A03-01
extern char usart_rx_buffer[50];
extern char usart_tx_buffer[50];

//=============================================================================
// Funktionen
//=============================================================================
// Aufgabe A01-01
extern void init_leds();

// Aufgabe A01-02
extern void init_taste_1();
extern void init_taste_2();
extern int led_steuerung();

// Aufgabe A02-01
extern void init_PC09();
extern void fastMode();
extern void slowMode();

// Aufgabe A03-01
extern void init_usart_2_tx();
```

```
extern void usart_2_print(char* zeichenkette);
extern void init_usart_2();
extern void empty_buffers();
//==========================================================================
#endif
```

Auszug aufgabe.c

```
#include "aufgabe.h"

// Aufgabe A03-01.1
void init_usart_2_tx() {
    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;

    // Taktsystem für die USART2 freigeben
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);

    // GPIO Port A Taktsystem freigeben
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);

    // USART2 TX an PA2 mit Alternativfunktion Konfigurieren
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    // USART2 TX mit PA2 verbinden
    GPIO_PinAFConfig(GPIOA, GPIO_PinSource2, GPIO_AF_USART2);

    // Datenprotokoll der USART einstellen
    USART_InitStructure.USART_BaudRate = 921600;
    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_Tx;
    USART_Init(USART2, &USART_InitStructure);

    // USART2 freigeben
    USART_Cmd(USART2, ENABLE); // enable USART2
}

void usart_2_print(char* zeichenkette)
{
    int i = 0;
    for(i = 0;i < strlen(zeichenkette);i++)
    {
        USART_SendData(USART2, zeichenkette[i]);
        while (USART_GetFlagStatus(USART2, USART_FLAG_TC) == RESET){}
    }
}

char usart_rx_buffer[50];
```

```
char usart_tx_buffer[50];

void init_usart_2()
{
    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;

    // Taktsystem für die USART2 freigeben
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);

    // GPIO Port A Taktsystem freigeben
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);

    // USART2 TX an PA2 mit Alternativfunktion Konfigurieren
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2 | GPIO_Pin_3;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    // USART2 TX mit PA2 verbinden
    GPIO_PinAFConfig(GPIOA, GPIO_PinSource2, GPIO_AF_USART2);
    GPIO_PinAFConfig(GPIOA, GPIO_PinSource3, GPIO_AF_USART2);

    // Datenprotokoll der USART einstellen
    USART_InitStructure.USART_BaudRate = 921600;
    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_Tx | USART_Mode_Rx;
    USART_Init(USART2, &USART_InitStructure);

    // USART2 freigeben
    USART_Cmd(USART2, ENABLE); // enable USART2
}

// Leere die beiden Buffer für Eingabe und Ausgabe
void empty_buffers()
{
    for (int i = 0; i < 50; i++)
    {
        usart_rx_buffer[i] = '\0';
        usart_tx_buffer[i] = '\0';
    }
}
```

main.c

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

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

```
    // SysTick initialisieren
    // jede ms erfolgt dann der Aufruf
    // des Handlers fuer den Interrupt SysTick_IRQn
    InitSysTick();

    // Initialisiere PA02 für USART
    init_usart_2();

    int i = 0;
    int length = 0;
    char zeichen;

    while (1) {

        // Prüfe, ob eine Eingabe vorliegt
        while (USART_GetFlagStatus(USART2, USART_FLAG_RXNE) != RESET)
        {
            // Lese zuerst die Eingabe, ohne sie in den Buffer zu schreiben
            zeichen = (char) USART_ReceiveData(USART2);

            // Prüfe, ob das Zeichen ein Carriage Return ist
            if (zeichen == '\r')
            {
                // Erstelle den String gemäß Aufgabe und gebe ihn aus
                sprintf(usart_tx_buffer, "%s,%d\n", usart_rx_buffer, length);
                usart_2_print(usart_tx_buffer);

                // Leere beide Buffer und setze die Laufvariablen auf 0
                // für den nächsten Durchlauf
                empty_buffers();
                i = 0;
                length = 0;
            } else {
                // Sonst schreibe das Zeichen in den Buffer
                // und erhöhe die Laufvariablen
                usart_rx_buffer[i] = zeichen;
                length++;
                i = (i + 1) % 50;
            }
        }
    }
}
```