1 Protokoll – Übung 3

HÖHERE TECHNISCHE BUNDESLEHRANSTALT HOLLABRUNN

Höhere Abteilung für Elektronik – Technische Informatik

Klasse/ Jahrgang: 5BHEL	Übungsbetreuer: Dipl. Ing. Josef Reisinger
Übungsnummer: 3	Übungstitel: CM3 Peripheral Library
Datum der Vorführung: -	Gruppe: Paul Raffer, Stefan Grubmüller
Datum der Abgabe: 11.04.2021	Unterschrift:

Beurteilungskriterien

Programm:	**Punkte**
Programm Demonstration	
Erklärung Programmfunktionalität	
Protokoll:	**Punkte**
Pflichtenheft (Beschreibung Aufgabenstellung)	
Beschreibung SW Design (Flussdiagramm, Blockschaltbild,..)	
Dokumentation Programmcode	
Testplan (Beschreibung Testfälle)	
Kommentare / Bemerkungen	
Summe Punkte	

Note: _____

CM3 Peripheral Library

Inhaltsverzeichnis

2 Aufgabenstellung

Übung#3- CM3 Peripheral Library

Einführung:

Die Grundidee der Übung besteht darin Demoprogramme für die Peripheral Library des Mikrocontrollers STM32F10X zu realisieren, die reale Anwendungen simulieren.
Das Demoprogramm soll auf dem HTL eigen Mikrocontrollersystem auf Basis des Cortex M3 Mikrocontroller lauffähig sein. Der Zugriff auf die Peripherieeinheiten (GPIO, ADC, Timer, UART,….) des Mikrocontrollers ausschließlich über die **Standard Peripherial Library** der Entwicklungsumgebung Keil µVision erfolgen.
Zustandsänderungen im System sollen über UART protokolliert werden und können mit einem Terminalprogramm angesehen werden. Über dieses **Logging** sollte der Funktionsbeweis geführt werden.

Abgabe:

Abzugeben ist ein Protokoll (*.pdf bzw. *.doc) welches die **Aufgabestellung**. Die Aufgabe erläutert eine reale Anwendung bzw. wie diese auf dem ARM- Minimalsytem simuliert wird. Desweitern ein vollständiges **Blockschaltbild** zur Aufgabenstellung sowie den dokumentierten Source Code. Anhand des Blockschaltbilds soll die Funktionsweise des Systems erläutert werden, sodass man in der Lage ist den Source Code zu verstehen. Schließlich soll das Programm einen Funktionsnachweis in Form von **Screenshots** mit entsprechenden Erläuterungen enthalten die nachweisen das Aufgabenstellung erfüllt ist.

Die Abgabe erfolgt über die entsprechende MS-Teams Gruppe. Alle Files (Quelldateien, ausführbare Dateien, Dokumente) sind in ein Archiv Datei (*.ZIP) zu geben, welches das bei der entsprechenden MS-Teams Aufgabe hochzuladen ist.

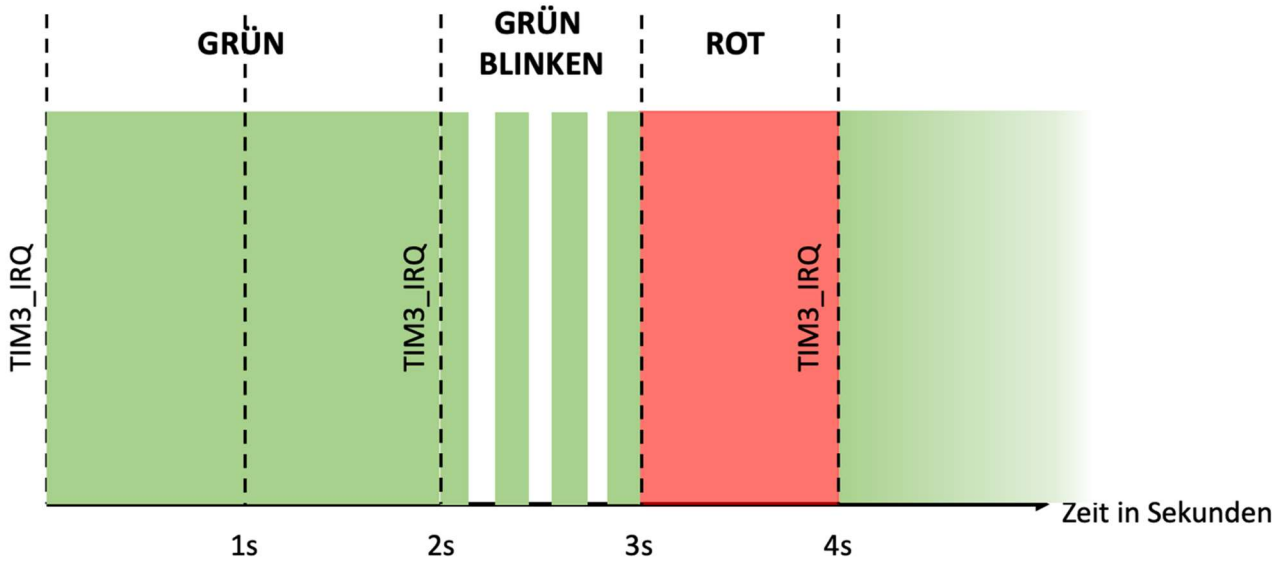
Verfügbare Hardwareeinheiten (ARM Minimalsystem):

- LED / Schalter – LED Schalterplatine (LED0-LED7, SW0-SW7)
- LED‘s Europlatine (LED Array)
- LED / Taster DIL Adapter:
  (DIL_LED1, DIL_LED2, DIL_LED3, DILTaster_1, DILTaster_2, DILTaster_3)
- Potentiometer LED / Schalterplatine, Poti DIL-Adapter
- Externe Interrupts
- ADC: Single Conversion Mode, Continous Conversion Mode, Scan Mode - Pollling/Interrupt/DMA Mode)
- Timer mit Interrupt (Timer1-Tmer4, SysTick, RTC)
- Input Capture Einheit für Timer (Interrupt)
- Timer Output Compare Einheit
- I2C Interface (keine Hardware verfübar)
- SPI Interface (keine Hardware verfügbar)
- UART#1/UART#2 (V24 Modul) - Polling/Interrupt/DMA Betrieb
- LCD Anzeige (Ansteuerung über HTL eigene ARMV10_STD.LIB)
- LFU
- NE555
- DS18B20 – (One Wire Temperatursensor)
- Joystick

Thema: Output Compare Einheit Timer 3, LED /Schalter DIL Adapter, UART#2(Polling)

3 Allgemeines

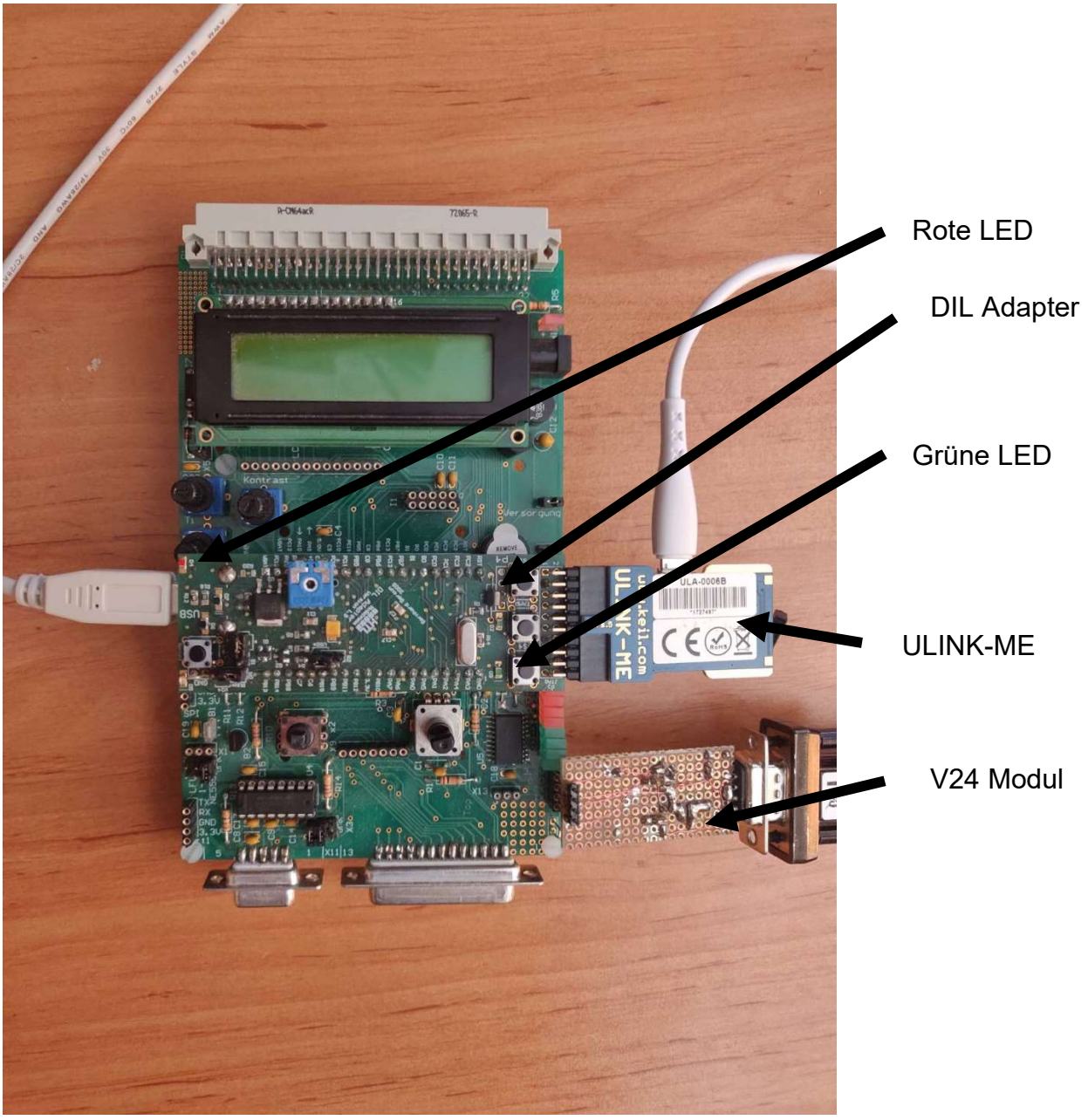
Es ist ein mit Unterprogrammen strukturiertes Programm für den Mikrocontroller ARM Cortex M3 zu schreiben, welches eine Ampel nachbildet. Realisiert wurde dies durch die Verwendung der roten LED (DIL_LED_1) und der grünen LED (DIL_LED_3) des DIL Boards, auf welchem der verwendete Mikrocontroller SMT32F103RB liegt. Für das Timing wurde der Channel 1 des Timers 3 verwendet. Nach dem Überlauf des Zählers (ARR) wird ein Interrupt ausgelöst, welches die Grünphase der Ampel startet bzw. beendet.
Die LEDs werden über die GPIO Ports angesteuert. Die grüne LED leuchtet als erstes. Danach kündigt die grüne LED durch ein Blinken die nächste Rotphase an. Danach leuchtet die rote LED. Alle Zustandsänderungen werden per USART2 (V24 Modul) geloggt. Über ein Terminal kann per USART der Befehl „1“ eingegeben werden. Dieser startet die Ampel. Mit dem Befehl „0“ kann die Ampel gestoppt werden. Es wurde die Standard Peripheral Libraries des Herstellers verwendet.



Auf diesem Bild sieht man das Timing der Ampelphasen. Nach 4 Sekunden wird der Prozess wiederholt.

4 Produktanforderungen

Es ist ein mit Unterprogrammen strukturiertes Programm für den Microprozessor ARM Cortex M3 zur Ampelsteuerung zu schreiben. Dafür wurde der DIL Adapter mit dem Mikroprozessor und den LEDs, ULINK-ME zum Flashen und Debuggen, sowie das V24 Modul für die Kommunikation über UART2 und das Basisboard benötigt.



5 USART

In diesem Projekt wird der USART2 für die Datenkommunikation, das Steuern und Loggen der Ampel verwendet. Es wird eine **Baudrate** von **115200** verwendet.

Log:

- Initialized …zeigt an, dass die Kommunikation über USART funktioniert
- Ampel gestartet …zeigt an, dass die Ampel über das Terminal gestartet wurde
- Ampel gestoppt …zeigt an, dass die Ampel über das Terminal gestoppt wurde
- Ampel gruen …zeigt an, dass die Ampel grün leuchtet
- Ampel blinkt gruen …zeigt an, dass die Ampel grün blinkt
- Ampel rot …zeigt an, dass die Ampel rot leuchtet

Über das Terminal können folgende **Befehle** eingegeben werden:

- 1 …startet die Ampel
- 0 …stoppt die Ampel

6 Timer

In diesem Projekt wird der **Channel 1** des GPIO **Timers 3** als Output Compare verwendet. Für das Auto Reload Register **ARR** wurde einen Wert von 0xFFFF (65535) festgelegt. Die interne Taktfrequenz ist 8MHz, wodurch sich für **TCK_INT** eine Periodendauer von 125ns ergibt. Über folgende Formel wurde des Prescaler **PSC** berechnet:

$$PSC = \frac{Tx}{TCK\_INT * ARR} - 1 = \frac{2}{125 * 10^{-9} * 65535} - 1 = 0x\mathrm{F3}$$

Dadurch ergibt sich ein Überlauf des Zählers nach genau 2 Sekunden! Bei einem Überlauf wird gleichzeitig auch ein Interrupt ausgelöst. Dieses Interrupt wird genutzt, um die Ampel umzuschalten.

7 GPIO

Es werden in diesem Projekt folgende GPIOs benutzt:

DIL_LED_1 ist am Port D2 angeschlossen. Er dient als **rote LED** und ist als GP Output konfiguriert.

DIL_LED_3 ist am Port A8 angeschlossen. Er dient als **grüne LED** und ist als GP Output konfiguriert.

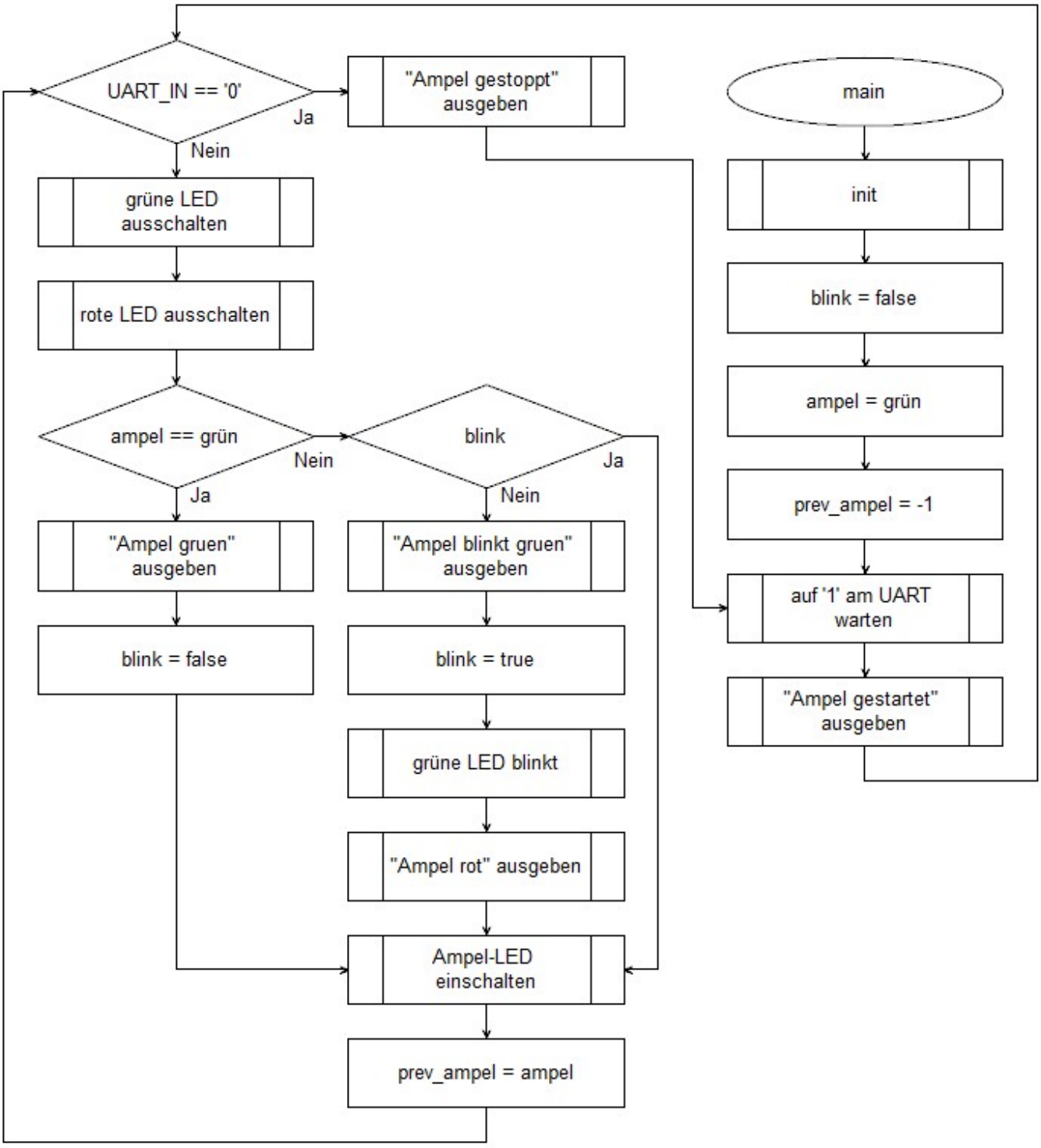
USART2 Tx ist mit dem Port A2 verbunden. Er wird hier als **Tx** Leitung des USART2 benutzt, um die Zustandsänderungen der Ampel zu loggen.

USART2 Rx ist mit dem Port A3 verbunden. Er wird hier als **Rx** Leitung des USART2 im Polling Modus verwendet, um auf Befehle vom Terminal zu warten.

Bemerkung: Die rote LED hängt an einem Port, welcher für das Flashen der Software auf den Mikrocontroller benötigt wird. Das bedeutet, dass man vor dem Flashvorgang den Jumper B2 auf die Position „USB“ setzten muss. Dabei kann die rote LED (D2) nicht verwendet werden. Wird die rote LED benötigt, muss der Jumper B2 auf die Position „rote LED“ gesetzt werden, jedoch kann dann nicht geflasht werden.

8 Software

8.1 Flussdiagramm



8.2 Sourcecode

8.2.1 main.c

```
/*
  main.c
  March 2021
  Paul Raffer, Stefan Grubmueller
  HTBL Hollabrunn 5BHEL

  Funktion:
  Dieses Programm stellt eine rot-gruene Ampel dar.
  Realisiert wurde dies durch die Verwendung der roten LED
  (DIL_LED_1) und der gruenen LED (DIL_LED_3) des DIL Boards,
  auf welchem der verwendete Microcontroller SMT32F103RB liegt.
  Als Taktgeber wurde der Channel 1 des Timer 3 verwendet.
  Die LEDs werden ber die GPIO Ports angesteurt.
  Die gruene LED leutet als erstes. Danach kndigt die gruene
  LED durch ein Blinken die naechste Rotphase an. Danach leuchtet
  die rote LED. Alle Zustandsaenderungen werden per USART2 geloggt.
  Ueber ein Terminal kann man per USART2 Befehle zum Ein/Ausschalten
  der Ampel geben. Es wurde die Standard Periphal Libraries des
  Herstellers verwendet.
*/

//includes
#include "ampel.h"

//ampel = LED
volatile enum led ampel;

//TIMER 3 Channel 1 Interrupt Routine Handler
void TIM3_IRQHandler(void)
{
    TIM3->SR &=~0x00FF;  //delete interrupt pending-bit
    //toggle Ampelstate
    ampel = ampel == green1 ? red : green1;
}

int main()
{
    Init(); //init periphals

    bool blink = false;  //does not blink
    ampel = green1;      //ampel is green
    enum led prev_ampel = -1;//last state not green

    while (1) { //Endlosschleife
        while (!UartInIs(USART2, '1'));  //is '1' received?
        //Ampel startet
        UartPutString(USART2, "\r\nAmpel gestartet\r\n");
        UartPutString(USART2,
"************************************\r\n");

        //cancel when '0'
        while (!UartInIs(USART2, '0')) {
            //if state changed (e.g. red to green)
            if (prev_ampel != ampel) {
                //turn both LEDs off
                set_led(green1, 0);
                set_led(red, 0);
```

```
                //if state is green => green LED on
                if (ampel == green1) {
                    UartPutString(USART2, "Ampel gruen\r\n");
                    blink = false;//shall not blink
                }
                //has not already blinked in this period
                else if (!blink) {
                    //blink
                    UartPutString(USART2, "Ampel blinkt gruen\r\n");
                    blink = true;
                    //green LED shall blink 4 times in intervall 250ms
                    blink_led(green1, 4, 250);
                    //after blinking light is red
                    UartPutString(USART2, "Ampel rot\r\n");
                }
                set_led(ampel, 1);  //red/green LED on
                prev_ampel = ampel; //change state
            }
        }
        //if canceled
        UartPutString(USART2, "\r\nAmpel gestoppt\r\n");
        UartPutString(USART2,
"************************************\r\n");
    }
}
```

8.2.2 ampel.h

```
/*
  ampel.h
  March 2021
  Paul Raffer, Stefan Grubmueller
  HTBL Hollabrunn 5BHEL

  Funktion:
  Dieses Headerfile beinhaltet alle Prototypen und Konstanten,
  welche in main.c und ampel.c benoetigt werden.
*/
/* --------------Define to prevent recursive inclusion -----------------*/
#ifndef __AMPEL_H__
#define __AMPEL_H__

//includes
#include "stm32f10x.h"          //standard library
#include "ARMV10_STD.h"         //wait_ms()
#include "stm32f10x_rcc.h"      //RCC library
#include "stm32f10x_rtc.h"      //RTC library
#include "stm32f10x_gpio.h"     //GPIO library
#include "stm32f10x_usart.h"    //USART communication library
#include "stm32f10x_tim.h"      //Timer library
#include "string.h"
#include "stdbool.h"

//constants
enum led {
    green1,
    red,
};


//configuration for USART2
extern void InitUsart2(void);
//configuration of LEDs
extern void InitGpio(void);
//configuration Timer 3 Channel 1
extern void TIM3_Config(void);
//send string over uart
void UartPutString(USART_TypeDef *USARTx, char *str);
//turn LEDs on/off
void set_led(enum led led, _Bool on);
//let LED blink for set time
void blink_led(enum led led, int count, int time);
//ampel an/aus
bool UartInIs(USART_TypeDef * const usart, char character);
//contains all configuration
void Init();


#endif
```

8.2.3 ampel.c

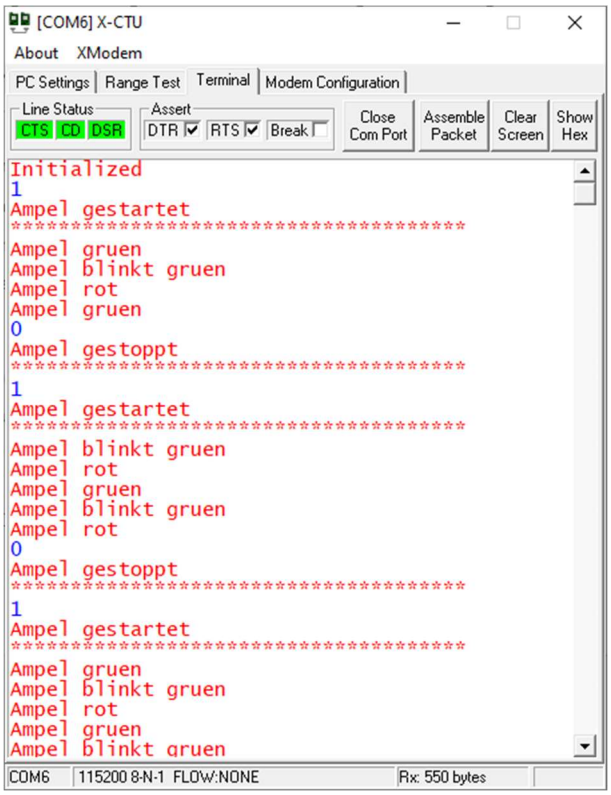
```
/*
    ampel.c
    March 2021
    Paul Raffer, Stefan Grubmueller
    HTBL Hollabrunn 5BHEL

    Funktion:
    Dieses File beinhaltet alle Funktionen, die in main.c
    verwendet werden. Es werden Channel 1 von Timer 3, USART2
    und GPIOs konfiguriert und initialisert.
*/

//Includes
#include "ampel.h"

//InitGpio
//paramter: none
//return: none
//gruene LED: GP-PP PA8
//rote LED: GP-PP PD2
void InitGpio(void)
{
    GPIO_InitTypeDef gpio;
    gpio.GPIO_Mode = GPIO_Mode_Out_PP;
    gpio.GPIO_Speed = GPIO_Speed_50MHz;

    //green LED PA2
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
    gpio.GPIO_Pin = GPIO_Pin_8;
    GPIO_Init(GPIOA, &gpio);

    //red LED PD2
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE);
    gpio.GPIO_Pin = GPIO_Pin_2;
    GPIO_Init(GPIOD, &gpio);
}

//TIM3_Config
//paramter: none
//return: none
//Timer 3 Channel Output Compare
void TIM3_Config(void)
{
    GPIO_InitTypeDef gpio;
    TIM_TimeBaseInitTypeDef TIM_TimeBase_InitStructure;
    TIM_OCInitTypeDef TIM_OCInitStruct;
    NVIC_InitTypeDef nvic;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE); // GPIOA Clock
Enable
    GPIO_StructInit(&gpio);// Create gpio struct and fill it with defaults
    gpio.GPIO_Mode = GPIO_Mode_AF_PP; // PA6(=TIM3_CH1) in AF -Push Pull
    gpio.GPIO_Pin = GPIO_Pin_6;
    GPIO_Init(GPIOA, &gpio);

    //Tx = 2s; T_INT = 125ns, ARR: 0xFFF => PSC: 0x73
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE); // Clock Enable
Timer 3
    TIM_DeInit(TIM3);
    TIM_TimeBase_InitStructure.TIM_ClockDivision = TIM_CKD_DIV1;
```

```
    TIM_TimeBase_InitStructure.TIM_CounterMode = TIM_CounterMode_Up;
    //Auto-Reload Wert (ARR) = Maximaler Zaehlerstand des Upcounters
    TIM_TimeBase_InitStructure.TIM_Period = 0xFFFF;
    //Prescaler (Taktverminderung)
    int Tx = 2;
    double T_INT = 0.000000125;
    TIM_TimeBase_InitStructure.TIM_Prescaler = Tx / (T_INT * 0xFFFF) - 1;
    TIM_TimeBaseInit(TIM3, &TIM_TimeBase_InitStructure);

    /* Compare Value*/
    TIM_OCInitStruct.TIM_OCPolarity = TIM_OCPolarity_High;
    TIM_OC3Init(TIM3,&TIM_OCInitStruct); /* save initialisation */
    TIM_ITConfig (TIM3, TIM_IT_Update,ENABLE); // Timer 3 Update Interrupt
Enable

    // Init NVIC for Timer 3 Update Interrupt
    nvic.NVIC_IRQChannel = TIM3_IRQn;
    nvic.NVIC_IRQChannelCmd = ENABLE;
    nvic.NVIC_IRQChannelPreemptionPriority = 0;
    nvic.NVIC_IRQChannelSubPriority = 0;
    NVIC_Init(&nvic);
    TIM_Cmd(TIM3, ENABLE); //Counter-Enable bit (CEN) Timer 3 setzen
}


///configuration for USART2
//USART2:
//Rx Pin        ....    PA3
//Tx Pin        ....    PA2
//baudrate      ....    115200
//word length   ....    8 bit
//parity bits   ....    none
//stop bit      ....    1
void InitUsart2(void)
{
    GPIO_InitTypeDef gpio;
    USART_ClockInitTypeDef usartclock;
    USART_InitTypeDef usart;

    SystemCoreClockUpdate();
    USART_DeInit(USART2);

    //enable all GPIO and USART clocks needed for USART2
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_AFIO,
ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);

    GPIO_StructInit(&gpio);

    //set PA2 to alternate function push pull (Tx)
    gpio.GPIO_Mode = GPIO_Mode_AF_PP;
    gpio.GPIO_Pin = GPIO_Pin_2;
    gpio.GPIO_Speed= GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &gpio);

    //set PA3 to input floating (Rx)
    gpio.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    gpio.GPIO_Pin = GPIO_Pin_3;
    GPIO_Init(GPIOA, &gpio);

    //init USART2 clock
    USART_ClockStructInit(&usartclock);
```

```
//  memset(&usartclock, 0, sizeof(usartclock));
    usartclock.USART_CPHA = USART_CPHA_2Edge;
    USART_ClockInit(USART2, &usartclock);


    //create usart struct and init USART2 to 115200 baud
    USART_StructInit(&usart);
    usart.USART_BaudRate = 115200;
    usart.USART_WordLength = USART_WordLength_8b;
    usart.USART_StopBits = USART_StopBits_1;
    usart.USART_Parity = USART_Parity_No ;
    usart.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
    usart.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
    USART_Init(USART2, &usart);


    //enable USART2
    USART_Cmd(USART2, ENABLE);
    //sysclock = SystemCoreClock;
}

//send string to USARTx
//parameter:
//USART_TypeDef *USARTx....selected USART interface
//char *str....string to transmit
void UartPutString(USART_TypeDef *USARTx, char *str)
{
    while (*str)
    {
        while (USART_GetFlagStatus(USARTx, USART_FLAG_TXE) == RESET);
        //while (!(USARTx->SR & USART_SR_TXE));
        //uart_put_char(USARTx, *str++);
        USART_SendData(USARTx, *str++);
    }
}

//set_led
//parameter: led ... variable type LED (green1,red)
//           on ... 1 = LED on; 0 = LED off
//return: none
//turns LED on/off
void set_led(enum led led, _Bool on)
{
    GPIO_TypeDef * gpio;
    uint16_t pin;

    switch (led) {
    //green LED = PA8
    case green1:
        gpio = GPIOA;
        pin = GPIO_Pin_8;
        break;
    //red LED = PD2
    case red:
        gpio = GPIOD;
        pin = GPIO_Pin_2;
        break;
    }
    //turn led on
    if (on) {
        GPIO_ResetBits(gpio, pin);
    }
```

```
    //turn led off
    else {
        GPIO_SetBits(gpio, pin);
    }
}

//blink_led
//parameter: led  ...variable type LED (green1,red)
//           count...how often shall it blink(0-x times)
//           time ...time between on LED(0-x ms)
//return: none
//lets LED blink for passed time for x times
void blink_led(enum led led, int count, int time)
{
    for (int i = 0, s = true; i < 2 * count; ++i, s = !s) {
        set_led(led, s);  //turn led on/off
        wait_ms(time);    //wait for passed time
    }
}

//UartInIs
//parameter: usart    ...USart to listen (USARTx)
//           character...charter to listen on('x')
//return:    bool     ...true = character received; false = character not
received
bool UartInIs(USART_TypeDef * const usart, char character)
{
    //was character received?
    return !((USART_GetFlagStatus(usart, USART_FLAG_RXNE) == RESET) ||
USART_ReceiveData(usart) != character);
}

//Init
//inits periphals
void Init(void)
{
    InitUsart2();  //USART2 Init (Log)
    InitGpio();   //GPIOB1 Init (LEDs)
    TIM3_Config();  //TIM4 Output Compare Init (Timer)
    UartPutString(USART2, "Initialized\r\n");

    //turn both LEDs off
    set_led(green1, 0);
    set_led(red, 0);
}
```

9 Funktionsnachweis



Auf diesem Bild sieht man ein Terminal. Als erstes verkündet das Programm mit dem Log „Initialized“, dass die Kommunikation über USART funktioniert. Als nächstes wurde die Ampel mit dem Befehl „1“ über das Terminal gestartet. Das Programm loggt dies mit „Ampel gestartet“. Im Log sieht man auch, dass die erste Ampelphase grün ist und dass die grüne Ampel danach blinkt. Die letzte Ampelphase ist rot. Danach beginnt wieder die erste grüne Phase. Dieser Vorgang wiederholt sich und wird durch die Terminaleingabe „0“ unterbrochen. Dies schaltet die Ampel ab. Im Log wird dies unter „Ampel gestoppt“ verzeichnet. Danach sieht man, dass die Ampel erneut gestartet wurde und dass sich der Vorgang wiederholt.

10 Probleme

- Falsche Konfiguration einiger Peripherieeinheiten
- Fehlerhafte USART Kommunikation
- Fehlerhafte ISR
- Falsche Interpretation der Angabe

11 Erkenntnisse

- Bedienung des Mikroprozessors und den jeweiligen Platinen (V24-Modul)
- Umgang mit Output Compare Timern
- Senden von Daten über UART
- Verwenden von GPIOs

12 Zeitaufwand – Stefan Grubmüller

Tätigkeit	**Aufwand**
Erstellung des Pflichtenhefts	0.5h
Erstellung des Systemdesign (Flussdiagramm bzw. Struktogramm und ev. UI Design)	0.5h
Programmcodierung	9h
Testen der Software	0h
Dokumentation (Protokoll)	2.5h
Gesamt:	**12.5h**

13 Zeitaufwand – Paul Raffer

Tätigkeit	**Aufwand**
Erstellung des Pflichtenhefts	0.5h
Erstellung des Systemdesign (Flussdiagramm bzw. Struktogramm und ev. UI Design)	1h
Programmcodierung	7h
Testen der Software	1h
Dokumentation (Protokoll)	2h
Gesamt:	**11.5h**