Mikroprozessorpraktikum

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

04-01 IWDG

A04-01.1

> Die Grundstruktur des Programms zur Lösung der Aufgabenstellung besteht in einer Endlosschleife. Zur besseren Visualisierung der Programmabarbeitung sollen mehrere Ausgaben auf die USART2 eingebunden werden. Der Fehler wird mit der Taste 2 generiert.
>
> In Einzelteile zerlegt, soll folgender Ablauf programmiert werden. - Vor der Endlosschleife erfolgt eine Konfiguration des Taster 2, der USART2 und des IWDG. - Der IWDG soll dabei auf 5 Sekunden konfiguriert werden. - Noch vor dem Eintritt in die Endlosschleife, soll auf die USART2 die Ausgabe von "\r\nNeustart\r\n" erfolgen. - In der Endlosschleife soll zuerst die Ausgabe von "Schleife\r\n" auf die USART2 gefolgt von einem wait_uSek(500000) erfolgen. Danach wird der Abwärtszähler wieder auf den voreingestellten Wert geladen. - Dann folgt eine Abfrage des Zustands der Taste 2. Wenn die Taste gedrückt ist soll folgender Ablauf realisiert werden. - Ausgabe des Textes "Taste2 gedrückt\r\n" - Aufruf von einem wait_uSek(..). Testen Sie Verzögerungszeiten oberhalb und unterhab des Watchdog Zeitintervalls aus.
>
> Wird die Taste 2 gedrückt, unterbricht der Watchdog die while(1){;} Schleif nach 5 Sekunden und und Startet den Mikrocontroller neu.
> Überprüfen Sie anhand der Ausgaben die Funktionsweise des IWDG und erklären Sie das Verhalten.

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 A03-01.1
extern int i;

// Aufgabe A03-01.2
extern char usart2_rx_buffer[50];
extern char usart2_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();

// Aufgabe A02-01.4
extern void slowMode(void);
extern void fastMode(void);

// Aufgabe A03-01.1
extern void init_usart_2_tx();
```

```
// Aufgabe A03-01.2
extern void usart_2_print();

// Aufgabe A03-01.4
extern void empty_buffers();

// Aufgabe A04-01.1
extern void init_iwdg();

//===========================================================================
#endif
```

Auszug aufgabe.c

```
#include "aufgabe.h"

// 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;
}

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){}
    }
}

void init_usart_2(){
    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;

    // Taktsystem f r 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
}

// Initialisiere den IWDG
void init_iwdg()
{
    // Schreibrechte aktivieren
        IWDG_WriteAccessCmd(IWDG_WriteAccess_Enable);
    // den Vorteiler (4, 8 , 16 ,..., 256) auf 64 setzen
        IWDG_SetPrescaler(IWDG_Prescaler_64);
    // den Wert (0...4095) einstellen ab dem runtergez?hlt wird
        IWDG_SetReload(2500);
    // setzt den Wachdog auf den eingestellten Maximalwert
        IWDG_ReloadCounter();
    // aktiviert dem IWDG
        IWDG_Enable();
    // Das Zeitintervall t berechnet sich folgenderma?en
    // t = (1/32000) * 64 * 2500 = 5 Sekunden
}
```

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 beide Tasten
    //init_taste_1();
    init_taste_2();

    init_usart_2();
```

```
    init_iwdg();

    usart_2_print("\r\nNeustart\r\n");

    while(1)
    {
        usart_2_print("Schleife\r\n");

        wait_uSek(500000);

        IWDG_ReloadCounter();

        int taste_2_gedruckt = led_steuerung();

        if(taste_2_gedruckt == 1) {
            usart_2_print("Taste2 gedr?ckt\r\n");

            wait_uSek(4800000);
        }
    }
}
```

Wenn Taste 2 gedrückt wird und die Wartezeit unter 5 Sekunden ist, wie in unserem Fall 3 Sekunden, läuft die Schleife ohne Probleme weiter. Ist die Wartezeit über 5 Sekunden, wie bei uns 6 Sekunden, und Taste 2 wird gedrückt, wird das Board neu gestartet. Dieses Verhalten kann man leicht über die Ausgabe des Boards beobachten.
Setzen wir die Wartezeit auf 4,8 Sekunden, wird das Board neu gestartet. Der Grund für dieses Verhalten liegt darin, dass der Zähler erst nach der Wartezeit nach der Ausgabe "Schleife" zurückgesetzt wird. Die beiden Wartezeiten addieren sich und sind dann über dem Watchdog-Intervall. Setzt man die innere Wartezeit unter 4,5 Sekunden, wird das Board nicht neugestartet.