Mikroprozessorpraktikum

Konstantin Bork & Kean Seng Liew, Gruppe A, HWP8

08-01 Timer

A08-01.1

> Konfigurieren Sie einen Timer und setzen Sie folgende Funktionalitäten softwaretechnisch um: - der Timer soll jede Sekunde einen Interrupt auslösen - in der ISR soll bei jedem Interrupt eine Variable inkrementiert werden - in der ISR soll der Wert dieser Variablen auf die USART2 ausgegeben werden

Auszug interrupts.c

```
void TIM7_IRQHandler(void)
{
    static int zahl = 0;
    if(TIM_GetITStatus(TIM7, TIM_IT_Update) != RESET)
    {
        TIM_ClearITPendingBit(TIM7, TIM_IT_Update);
        char data[50] = {0};
        sprintf(data, "%d\r\n", ++zahl);
        usart_2_print(data);
    }
}
```

aufgabe.h

```
// Aufgabe A08-01.1
void init_tim_7_irq();
```

Auszug aufgabe.c

```
void init_tim_7_irq()
{
    // Konfiguration der Interruptcontrollers
    NVIC_InitTypeDef NVIC_InitStructure;
    NVIC_InitStructure.NVIC_IRQChannel = TIM7_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

    // Taktsystem für TIM7 Freigeben
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM7, ENABLE);
```

```
    // Struktur anlegen
    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;

    // TIM7 in der Struktur konfigurieren
    TIM_TimeBaseStructure.TIM_Prescaler = 8400 - 1; // 100us = 8400 * 1/84000000Hz
    TIM_TimeBaseStructure.TIM_Period = 10000 - 1;   // 1s = 10000 * 100µs
    TIM_TimeBaseStructure.TIM_ClockDivision = 0;
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;

    // TIM7 Register aus der Struktur Schreiben
    TIM_TimeBaseInit(TIM7, &TIM_TimeBaseStructure);

    // TIM7 Interrupt erlauben
    TIM_ITConfig(TIM7, TIM_IT_Update, ENABLE);

    // TIM 7 Freigeben (Takt auf Counter schalten)
    TIM_Cmd(TIM7, ENABLE);
}
```

main.c

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

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

    // Start der RTC  falls diese noch
    // nicht initialisiert war wird
    // die RTC mit der LSE-Taktquelle aktiviert
    start_RTC();

    // Initialisiere die USART2 und den Timer inkl. Interrupts
    init_usart_2_irq();
    init_tim_7_irq();

    while(1)
    {

    }
}
```

A08-01.2

> Realisieren Sie auf Basis der Lösung von A08-01.1 einen Timer zur Zeitmessung. - Wird die Taste1 gedrückt - startet die Zeitmessung. - Wird die Taste2 gedrückt - stoppt die Zeitmessung. - Die gemessene Zeit soll mit einer Auflösung in hundertstel Sekunden auf die USART2 ausgegeben werden.

Auszug aufgabe.c

```
void init_tim_7_irq()
{
    // Konfiguration der Interruptcontrollers
    NVIC_InitTypeDef NVIC_InitStructure;
    NVIC_InitStructure.NVIC_IRQChannel = TIM7_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

    // Taktsystem für TIM7 Freigeben
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM7, ENABLE);

    // Struktur anlegen
    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;

    // TIM7 in der Struktur konfigurieren
    TIM_TimeBaseStructure.TIM_Prescaler = 8400 - 1; // 100us = 8400 * 1/84000000Hz
    TIM_TimeBaseStructure.TIM_Period = 100 - 1;   // 10ms = 100 * 100us für eine
einfachere Verarbeitung später
    TIM_TimeBaseStructure.TIM_ClockDivision = 0;
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;

    // TIM7 Register aus der Struktur Schreiben
    TIM_TimeBaseInit(TIM7, &TIM_TimeBaseStructure);

    // TIM7 Interrupt erlauben
    TIM_ITConfig(TIM7, TIM_IT_Update, ENABLE);

    // TIM 7 Freigeben (Takt auf Counter schalten)
    TIM_Cmd(TIM7, ENABLE);
}
```

Auszug interrupts.c

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

uint32_t aktuelle_zeit = 0; // Zähler für die aktuelle Zeit, ein Wert von 100
entspricht 1s

//=============================================================================
void EXTI9_5_IRQHandler(void)
{
    static uint32_t startzeit = 0;

    //===== Taster2
    if (EXTI_GetITStatus(EXTI_Line5) == SET)
    {
        EXTI_ClearFlag(EXTI_Line5);
        EXTI_ClearITPendingBit(EXTI_Line5);
        // Code
        uint32_t dauer = aktuelle_zeit - startzeit;
```

```
        char data[50] = {0};
        sprintf(data, "%.2f Sekunden\r\n", dauer / 100.0);
        usart_2_print(data);
        startzeit = 0;
    }
    //===== Taster 1
    if (EXTI_GetITStatus(EXTI_Line8) == SET)
    {
        EXTI_ClearFlag(EXTI_Line8);
        EXTI_ClearITPendingBit(EXTI_Line8);
        // Code
        startzeit = aktuelle_zeit;
    }
}

//============================================================================
void TIM7_IRQHandler(void)
{
    if (TIM_GetITStatus(TIM7, TIM_IT_Update) != RESET) {
        TIM_ClearITPendingBit(TIM7, TIM_IT_Update);
        aktuelle_zeit++;
    }
}
```

main.c

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

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

    // Start der RTC  falls diese noch
    // nicht initialisiert war wird
    // die RTC mit der LSE-Taktquelle aktiviert
    start_RTC();

    // Tasten initialisieren
    init_taste_1_irq();
    init_taste_2_irq();
    init_nvic();

    // Initialisiere die USART2 und den Timer inkl. Interrupts
    init_usart_2_irq();
    init_tim_7_irq();

    while(1)
    {

    }
}
```

A08-01.3

> Setzen Sie einen Timer zum Zählen von Ereignissen ein. - der Timer soll die Taster Betätigungen von Taste 1 an PC8 zählen - an der Taste 1 PC8 sollte der Timer 3 Channel 3 genutzt werden - die Anzahl der Tastenbetätigungen soll im 2 Sekundentakt (Endlosschleife mit waituSek(..)) auf die USART2 ausgegeben werden - nach zehn Tastenbetätigungen soll ein Interrupt ausgelöst werden - in der ISR soll eine Kontrollausgabe auf die USART2 erfolgen

Auszug interrupts.c

```
void TIM3_IRQHandler(void)
{
    if(TIM_GetITStatus(TIM3, TIM_IT_Update) == SET)
    {
        TIM_ClearITPendingBit(TIM3, TIM_IT_Update);
        usart_2_print("Es wurden wieder 10 Tastendruecke registriert!\r\n");
    }
}
```

Auszug aufgabe.h

```
// Aufgabe A08-01.3
void init_tim_3_irq();
```

Auszug aufgabe.c

```
void init_tim_3()
{
    GPIO_DeInit(GPIOC);

    // Taktsystem für den Port C Freigeben
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);

    // Struktur anlegen
    GPIO_InitTypeDef GPIO_InitStructure;

    // Struktur Initialisieren
    GPIO_StructInit(&GPIO_InitStructure);

    // Portleitung in der Struktur Konfigurieren
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;

    // Werte aus der Struktur in die Register schreiben
    GPIO_Init(GPIOC, &GPIO_InitStructure);
```

```
    // Alternativfunktion der Portleitung Freigeben
    GPIO_PinAFConfig(GPIOC, GPIO_PinSource8, GPIO_AF_TIM3);

    // Taktsystem für Timer TIM3 Freigeben
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);

    // Erkennen steigender Flanke an TI1
    TIM3 -> CCMR1 |= TIM_CCMR1_CC1S_0;
    TIM3 -> CR2 |= TIM_CR2_TI1S;

    // Polarität
    TIM3 -> CCER |= TIM_CCER_CC1P;

    // Slave Mode, external Clock Mode1, TI1FP1 Signalquelle
    TIM3 -> SMCR |= TIM_SMCR_SMS + TIM_SMCR_TS_2 + TIM_SMCR_ETF_0; //TIM_SMCR_TS_0
;
}

void init_tim_3_irq()
{
    init_tim_3();

    // Konfiguration der Interruptcontrollers
    NVIC_InitTypeDef NVIC_InitStructure;
    NVIC_InitStructure.NVIC_IRQChannel = TIM3_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

    // Struktur anlegen
    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;

    // TIM3 in der Struktur konfigurieren
    TIM_TimeBaseStructure.TIM_Prescaler = 1;
    TIM_TimeBaseStructure.TIM_Period = 10 - 1; // Grenzwert Tastenbetätigungen laut
Aufgabe
    TIM_TimeBaseStructure.TIM_ClockDivision = 0;
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;

    // TIM3 Register aus der Struktur Schreiben
    TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);

    // TIM3 Interrupt erlauben
    TIM_ITConfig(TIM3, TIM_IT_Update, ENABLE);

    // Counter auf 0 setzen
    TIM_SetCounter (TIM3, 0x00);

    // Timer TIM3 Freigeben
    TIM_Cmd(TIM3, ENABLE);
}
```

main.c

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

```
#include "aufgabe.h"

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

    // Initialisiere die USART2 und die Timer inkl. Interrupts
    init_usart_2_irq();
    init_tim_7_irq();
    init_tim_3_irq();

    while(1)
    {
        char data[50] = {0};
        uint32_t taste_1_druecke = TIM_GetCounter(TIM3);
        sprintf(data, "Aktuelle Tastendruecke: %d\r\n", taste_1_druecke);
        usart_2_print(data);
        wait_mSek(2000);
    }
}
```

A08-01.4

> Entwickeln Sie einen Reaktionszeit-Tester. Berücksichtigen Sie bitte die folgenden Hinweise. - Die Messreihe soll mit dem Empfang des Zeichen "s" über die USART2 gestartet werden - Die Messung soll 10-mal hintereinander ausgeführt werden - Durch Zufallszahl soll jeweils mit einer Verzögerung zwischen 2...10 Sekunden die grüne LED eingeschaltet werden - Die Zeit zwischen Einschalten der LED und der Betätigung einer Taste soll gemessen werden - Über alle Messungen soll der Minimal-, Maximal- und Mittelwert ermittelt werden und am Ende des Zyklus auf der USART2 ausgegeben werden.

main.c

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

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

    // LED und Tasten initialisieren
    init_leds();
    init_taste_1_irq();
    init_taste_2_irq();

    // USART2 und Timer initialisieren
    init_usart_2_irq();
    init_nvic();
```

```
    init_tim_7_irq();

    while(1)
    {

    }
}
```

Auszug aufgabe.h

```
// Aufgabe A08-01.4
void reaktionszeit_tester();
```

Auszug aufgabe.c

```
// Wir missbrauchen diese Funktion, da sie Betätigungen beider Tasten registriert
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;
}

// Generiere die Zufallszahl mittels dem gegebenen Code
uint32_t generiere_zufallszahl()
{
    uint32_t zahl=0;

    RCC_AHB2PeriphClockCmd(RCC_AHB2Periph_RNG, ENABLE);
    RNG_Cmd(ENABLE);

    while(RNG_GetFlagStatus(RNG_FLAG_DRDY)== RESET);
    zahl = RNG_GetRandomNumber();

    RNG_Cmd(DISABLE);
    RCC_AHB2PeriphClockCmd(RCC_AHB2Periph_RNG, DISABLE);

    zahl %= 10;

    // Wenn der Modulo 0 ist, gebe 10 aus
```

```
    if (zahl == 0) {
        return 10;

    // Wenn der Module 1 ist, geben wir 2 aus, damit wir mindestens 2 Sekunden
Wartezeit haben
    } else if (zahl == 1) {
        return 2;
    } else {

    // Ansonsten geben wir die gegebene Zahl aus
        return zahl;
    }
}

void reaktionszeit_tester()
{
    // Initialisiere Variablen für die Minimal-, Maximal- und totale Dauer
    uint32_t min = 0;
    uint32_t max = 0;
    uint32_t total = 0;

    // Führe 10 Tests durch
    int i;
    for(i = 0; i < 10; i++)
    {
        // Schalte die LED aus, warte eine gewisse, zufällige Zeit, setze den Timer
und
        // messe die Zeit, bis eine Taste gedrückt wurde
        GR_LED_OFF;
        uint32_t wartezeit = generiere_zufallszahl();
        wait_mSek(wartezeit * 1000);
        LED_GR_ON;

        // Die Zeitmessung muss blockierend sein, damit wir nicht in Konflikt mit
anderen
        // Programmteilen kommen und die Zeitmessung zuverlässig ist
        TIM_SetCounter(TIM7, 0);
        uint32_t dauer = TIM_GetCounter(TIM7);

        while(led_steuerung() == 0) {}

        uint32_t stop_zeit = TIM_GetCounter(TIM7);
        dauer = stop_zeit - dauer;

        // Im ersten Durchlauf müssen alle Werte mit der Dauer initialisiert werden
        if(i == 0) {
            min = dauer;
            max = dauer;
            total = dauer;
        } else {
            // Prüfe, ob eine kürzere Dauer vorliegt
            if(dauer < min)
            {
                min = dauer;
            }

            // Prüfe, ob eine längere Dauer vorliegt
            if(dauer > max)
```

```
            {
                max = dauer;
            }
            total += dauer;
        }
    }

    uint32_t mitte = total / 10; // Berechne den Durchschnitt mit der totalen Dauer

    // Gebe alle ermittelten Werte auf Hundertstel Sekunden genau in der Konsole
aus
    sprintf(usart2_tx_buffer, "Minimalwert: %.2f Sekunden\r\n", min / 100.0);
    usart_2_print(usart2_tx_buffer);

    sprintf(usart2_tx_buffer, "Maximalwert: %.2f Sekunden\r\n", max / 100.0);
    usart_2_print(usart2_tx_buffer);

    sprintf(usart2_tx_buffer, "Mittelwert: %.2f Sekunden\r\n", mitte / 100.0);
    usart_2_print(usart2_tx_buffer);
}
```

Auszug interrupts.c

```
// Warte auf das Zeichen 's' in der Eingabe
void USART2_IRQHandler(void)
{
    char zeichen;

    if (USART_GetITStatus(USART2, USART_IT_RXNE) != RESET)
    {
        zeichen = (char) USART_ReceiveData(USART2);
        USART_ClearITPendingBit(USART2, USART_IT_RXNE);
        if (zeichen == 's')
        {
            reaktionszeit_tester();
        }
    }
}
```