Mikroprozessorpraktikum

Konstantin Bork & Kean Seng Liew, Gruppe A, HWP8

08-02 PWM

A08-02.1

> Steuern Sie den Signalgeber über den PWM-Ausgang eines Timers an. Entwickeln Sie dazu ein Programm bei dem unterschiedliche Töne über Tastatureingaben (Terminalprogramm) angesteuert werden können.

main.c

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

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

    // USART2 initialisieren
    init_usart_2_irq();

    while(1)
    {

    }
}
```

Auszug aufgabe.h

```
// Aufgabe A08-02.1
void init_pwm(int);
```

Auszug aufgabe.c

```
void init_signalgeber()
{
    // Taktsystem für Port GPIOB freigeben
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);

    // Struktur anlegen
    GPIO_InitTypeDef GPIO_InitStructure;

    // Struktur mit Konfiguration laden
```

```
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_DOWN;

    // Konfiguration aus der Struktur in Register laden
    GPIO_Init(GPIOB, &GPIO_InitStructure);
    GPIO_PinAFConfig(GPIOB, GPIO_PinSource8, GPIO_AF_TIM10);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM10, ENABLE);
}

void init_pwm(int faktor)
{
    // Initialisiere den Signalgeber
    init_signalgeber();

    // benötigte Variablen
    // gewünschte Frequenz in Hz mit Kammerton a als Basis
    uint16_t frequenz_in_Hz = 440 * pow(1.06, faktor * 1.0);
    uint16_t PWM_Periode = 42;              // Periodendauer
    uint16_t PSC_Prescaler = 84;            // Prescaler
    uint16_t PWM_Tastverhaeltnis_OC1 = 5;  // Tastverhältnis TIM10C1 in %

    // Pulsbreite berechnen
    uint16_t PWM_Pulsbreite_OC1 = 0;

    // SystemCoreClock = 168000000 Hz
    // CK_INT = SystemCoreClock / 2 = 84000000 Hz
    // CK_CNT = 1000000 Hz entspricht einer Periodendauer 0.000001s
    // PSC_Prescaler = CK_INT / CK_CNT = 84000000Hz / 1000000Hz = 84
    PSC_Prescaler = 84;

    // PWM_Periode einstellbar bei einem
    // PSC_Prescaler von 84:    2   -> 0.000002s
    //                      65535   -> 0.065535s
    // PWM_Periode = PWM_Periodendauer / CK_CNT_Periodendauer
    PWM_Periode = (uint16_t) (2.0f*1000000.0f/frequenz_in_Hz);

    // PWM_Tastverhaeltnis_OC1 (1...100 in %)
    // PWM_Tastverhaeltnis_OC2 (1...100 in %)
    PWM_Pulsbreite_OC1 = (uint16_t) ( PWM_Periode * PWM_Tastverhaeltnis_OC1 / 100);

    // t_CK_CNT = 1/100000MHz = 0.00001s
    // t_Period = t_CK_CNT * TIM_Period = 0.00001s * 2000 = 0.02s
    // ARR = t_Period / t_CK_CNT = 0.02s / 0.00001s = 2000
```

```
// t_Duty_Cycle = 10 % = 0.1
// CCRx = (uint16_t) ( ARR * 0.1f)

// Anlegen der Struktur
TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;

//Konfigurationsdaten in die Struktur schreiben
TIM_TimeBaseStructure.TIM_Prescaler = PSC_Prescaler -1;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseStructure.TIM_Period = PWM_Periode - 1; //ARR
TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;

// Konfigurationsdaten in die Register schreiben
TIM_TimeBaseInit(TIM10, &TIM_TimeBaseStructure);

// Anlegen der Struktur für den Output Channel
TIM_OCInitTypeDef  TIM_OCInitStructure;

// Konfigurationsdaten in die Strukrur
// für den Output Channel eintragen
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;

TIM_OCInitStructure.TIM_Pulse = PWM_Pulsbreite_OC1; //CCR

// Konfigurationsdaten in die Register schreiben
TIM_OC1Init(TIM10, &TIM_OCInitStructure);   // OutputChannel 1

// Preload Register freigeben
TIM_OC1PreloadConfig(TIM10, TIM_OCPreload_Enable);

// Preload Register freigeben
TIM_ARRPreloadConfig(TIM10, ENABLE);

// Taktsignal für den Timer freigeben
TIM_Cmd(TIM10, ENABLE);

// ab jetzt gibt der Signalgeber einen Ton aus
// für eine halbe Sekunde einen Ton aus
wait_uSek(500000);

// jetzt muss der Port auf Low geschaltet werden
// um den Stromverbrauch zu minimieren
// Damit ist die Tonausgabe beendet.
GPIO_InitTypeDef GPIO_InitStructure;
```

```
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP ;
    GPIO_Init(GPIOB, &GPIO_InitStructure);

    GPIO_ResetBits(GPIOB, GPIO_Pin_8);
}
```

Auszug interrupts.c

```
void USART2_IRQHandler(void)
{
    char zeichen;

    if (USART_GetITStatus(USART2, USART_IT_RXNE) != RESET)
    {
        zeichen = (char) USART_ReceiveData(USART2);
        int zahl = zeichen - '0';
        if (zahl >= 0 && zahl <= 9) {
            init_pwm(zahl);
        }
    }
}
```

A08.02-2

> Servo-Motoren in Modellbau- und Robotik Anwendungen werden meist mittels PWM angesteuert. Programmieren Sie eine Lösung mit der Sie einen Modellbau-Servo über die Portleitung PB08 an der rechten Buchsen Leiste des 3D-SRLD Boards ansteuern können. Die Pulsbreite soll durch Tasten auf der Tastatur über das Terminalprogramm steuerbar sein. Details zur Ansteuerung sind hier verfügbar. Schließen Sie zur Kontrolle der Funktionsfähigkeit einen Servo-Motor an das 3D-SRLD Board an.

main.c

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

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

```
    // USART2  initialisieren
    init_usart_2();
    init_signalgeber();
    init_pwm_servo(200);

    int i = 0;

    while(1)
    {
        char zeichen;

        // lese Zeichen bis der Wagenrücklauf eingegeben wird
        if (USART_GetFlagStatus(USART2, USART_FLAG_RXNE) != RESET){
            zeichen = (char) USART_ReceiveData(USART2);
            if (zeichen == '\r'){
                // erstelle Zahl aus der Eingabe
                int zahl = atoi(usart2_rx_buffer);

                // nur 0 bis 270 sind erlaubt
                // aufgrund des verwendeten Motors
                if (zahl >= 0 && zahl <= 270) {
                    init_pwm_servo(zahl);
                }
                i = 0;
                empty_buffers();
            } else {
                usart2_rx_buffer[i] = zeichen;
                i = (i + 1) % 50;
            }
        }
    }
}
```

Auszug aufgabe.h

```
// Aufgabe A08-02.1
void init_signalgeber();
void init_pwm(int);
void init_pwm_servo(int);
```

Auszug aufgabe.c

```
void init_signalgeber()
{
    // Taktsystem für Port GPIOB freigeben
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
```

```
    // Struktur anlegen
    GPIO_InitTypeDef GPIO_InitStructure;

    // Struktur mit Konfiguration laden
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_DOWN;

    // Konfiguration aus der Struktur in Register laden
    GPIO_Init(GPIOB, &GPIO_InitStructure);
    GPIO_PinAFConfig(GPIOB, GPIO_PinSource8, GPIO_AF_TIM10);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM10, ENABLE);
}

// grad ist die übergebene Gradzahl,
// erlaubt sind Ganzzahlen zwischen 0 und 270
void init_pwm_servo(int grad)
{
    // benötigte Variablen
    uint16_t PWM_Periode = 20000; // Periodendauer von 20ms für den Motor
    uint16_t PSC_Prescaler = 168; // Prescaler

    // 800 entspricht 0,8ms für linken Ausschlag, 2400 entspricht 2,4ms
    // für rechten Ausschlag
    uint16_t PWM_Pulsbreite_OC1 = (uint16_t) (800 + 1600*grad/270);

    // Anlegen der Struktur
    TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;

    //Konfigurationsdaten in die Struktur schreiben
    TIM_TimeBaseStructure.TIM_Prescaler = PSC_Prescaler - 1;
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_TimeBaseStructure.TIM_Period = PWM_Periode - 1; //ARR
    TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;

    // Konfigurationsdaten in die Register schreiben
    TIM_TimeBaseInit(TIM10, &TIM_TimeBaseStructure);

    // Anlegen der Struktur für den Output Channel
    TIM_OCInitTypeDef  TIM_OCInitStructure;

    // Konfigurationsdaten in die Struktur
    // für den Output Channel eintragen
```

```
    TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
    TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;

    TIM_OCInitStructure.TIM_Pulse = PWM_Pulsbreite_OC1; //CCR

    // Konfigurationsdaten in die Register schreiben
    TIM_OC1Init(TIM10, &TIM_OCInitStructure);   // OutputChannel 1

    // Preload Register freigeben
    TIM_OC1PreloadConfig(TIM10, TIM_OCPreload_Enable);

    // Preload Register freigeben
    TIM_ARRPreloadConfig(TIM10, ENABLE);

    // Taktsignal für den Timer freigeben
    TIM_Cmd(TIM10, ENABLE);
}
```