MP-P2
Gruppe 4

2021

Timer und Interrupts

15. Dezember 2021

Florian Tietjen 2519584
Emily Antosch 2519935

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

1 Einführung

Im dritten Praktikum wollen wir uns mit dem A/D-Wandler des TivaWare-Boards auseinandersetzen. Dabei wollen wir sowohl herausfinden, wie man mit externen Peripheriegeräten arbeiten kann, als auch das interne A/D-Modul effektiv nutzen. Darüberhinaus interessiert uns auch der PWM-Modus des Timers als mögliche Dimmung einer LED basierend auf dem analogen Signal eines Sensors (in unserem Beispiel verwenden wir einen Joystick).

2 Aufgabe 1: Externe D/A-Wandler mit Treppenverfahren

In unserer ersten Aufgabe benutzen wir das in der Vorlesung behandelte Treppenverfahren, um den Spannungswert eines Komparators zu ermitteln und diesen mittels BCD-Code auf einem Display auszugeben. Wir möchten zudem dann die Richtigkeit unseres Ergebnisses überprüfen, indem wir die verschiedenen Spannungen auf dem Oszilloskop anzeigen lassen. Die Triggerspannung erhalten wir dabei von Pin $PL(2)$.

Wir verwenden dabei folgenden Code:

Listing 1: stepfnc.c zur Approximation der Spannung U_E mit dem Treppenverfahren

3 Aufgabe 3: Dimmen einer LED mithilfe des internen A/D-Wandlers

In der nächsten Aufgabe wollen wir uns mit dem internen A/D-Wandler befassen. Mit diesem messen wir die Ausgangsspannung eines Joysticks in der Y-Achse. Einen Pin des Ports M verbinden wir dann mit einer LED, um diese mit einem der Timer im PWM-Modus zu dimmen. Eine Bewegung führt daher zu einer Erhöhung oder Verringerung der Helligkeit der LED. Dafür verwenden wir den folgenden Code:

```
#include<stdio.h>
#include<stdint.h>
#include "inc/tm4c1294ncpdt.h"

void init_port(void);
void init_timer(void);
void init_adc(void);
unsigned int read_adc();

int main(void){

    int adc_value = 0; // ADC value variable
    unsigned short int minmax = 0; // If 0 then min, if 1 then max
    init_port();
    init_timer();
    init_adc();

    while (1)
    {
        ADC0_PSSI_R = 0x0001; // enable ADC0 SS0
        adc_value = read_adc(); // Read ADC value
        printf("Spannung: \%d\n",adc_value); // Print ADC value

        if(!(GPIO_PORTM_DATA_R &= (1<<1))){
            // If the button is not pressed
            TIMER2_TAMATCHR_R = 8000 + (2000 - adc_value) * 4 * 0.95; // Set the
    match value for PWM from 95\% to 5\%
        }
        else{
            // If the button is pressed
            minmax = !minmax;
```

```
            TIMER2_TAMATCHR_R = 8000 + (2 - (minmax ? 0x04 : 0x00)) * 4000 * 0.95;
    // Set the match value for PWM for either min or max
            while(GPIO_PORTM_DATA_R &= (1<<1)); // Wait until the button is released
        }
    }
}

// Initialize the port
void init_port(void){
    // Enable clock for port M and Port E
    SYSCTL_RCGCGPIO_R |= (1 << 4) | (1 << 11);
    // Ready?
    while(!(SYSCTL_PRGPIO_R & ((1<<4)|(1<<11))));
    // Enable clock for ADC0
    SYSCTL_RCGCADC_R |= 0x01;
    // Ready?
    while(!(SYSCTL_PRADC_R & 0x01));
    // Port E setup
    GPIO_PORTE_AHB_AFSEL_R |= 0x01;
    GPIO_PORTE_AHB_DEN_R &= ~0x01;
    GPIO_PORTE_AHB_AMSEL_R |= 0x01;
    // Port M setup
    GPIO_PORTM_DEN_R = 0x03;
    GPIO_PORTM_DIR_R = 0x01;
    GPIO_PORTM_DATA_R &= ~(1<<0);
    GPIO_PORTM_AFSEL_R = 0x01;
    GPIO_PORTM_PCTL_R = 0x03;
    GPIO_PORTM_PUR_R = 0x02;
}
// Initialize the timer
void init_timer(void){
    // Enable clock for timer 2
    SYSCTL_RCGCTIMER_R |= (1<<2);
    // Ready?
    while (!(SYSCTL_PRTIMER_R & (1<<2)));
    // Timer 2 setup
    TIMER2_CTL_R &= ~(1<<0);
    TIMER2_CFG_R = 0x04;
    TIMER2_TAMR_R = (1<<3) | 0x02;
    TIMER2_TAILR_R = 16000-1;
    TIMER2_TAMATCHR_R = 16000/2 - 1;
    GPIO_PORTM_DATA_R |= (1<<0);
    TIMER2_CTL_R |= (1<<0);
}
// Initialize the ADC
void init_adc(void){

    unsigned int waitcycle = 0;
    // Disable ADC0 SS0
    ADC0_ACTSS_R &= ~0x0F;
    // Magic code
    SYSCTL_PLLFREQ0_R |= (1<<23);
    while(!(SYSCTL_PLLSTAT_R & 0x01));
    ADC0_CC_R |= 0x01;
    waitcycle++;
    SYSCTL_PLLFREQ0_R &= ~(1<<23);
    // Set Sequencer 0 to sample channel 0, AIN3
    ADC0_SSMUX0_R |= 0x03;
    ADC0_SSCTL0_R |= 0x02;
    // Enable ADC0 SS0
    ADC0_ACTSS_R |= 0x01;
}
// Read the ADC
unsigned int read_adc(){
    unsigned int result = 0;
    while(ADC0_SSFSTAT0_R & (1<<8));
    result = (unsigned int)ADC0_SSFIFO0_R * 5000 / 4095;
    return result;
```

```
98 }
```

Listing 2: Das Programm dim.c zum Dimmen von einer LED mithilfe eines Joysticks in Y-Richtung

Dabei stellen wir die Initial Load Value des Timer 2, den wir mit unserem Output auf die LED verbinden (PM0), auf 16000. Aus

$$T_0 = \frac{1}{f_0} = \frac{1}{1kHz} = 1ms$$

$$L_{ILR} = T_0 \cdot f_{TIVA} = 1ms \cdot 16MHz = 16000$$

können wir dann also die volle Periodenweite berechnen, die wir als ILR einstellen müssen. Für die Ruhestellung des Joysticks bekommen wir also einen Wert von 8000. Die Pulsweite des Timersignals soll zwischen den beiden Extrema 95% und 5% HIGH pro Periode variieren. Wir rechnen:

$$m = 8000 + (2000 - u_{adc}) \cdot 4 \cdot 0.95$$

Dabei ist m die zu berechnende Match-Value, ab welchem Punkt das Signal auf LOW schaltet, und u_{adc} die Spannung, die wir aus dem internen AD-Wandler bekommen in Milivolt, in einem Bereich von $u_{adc} \in [0mV, 4000mV]$.