MP
Praktikum 2

2021

Timer und Interrupts

25. November 2021

Florian Tietjen 2519584
Emily Antosch 2519935

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

LISTINGS

KAPITEL 1

DIE TEMPERATUR-REGELSTRECKE

1.1 Einführung

Im ersten Praktikum wollen wir uns mit dem Evaluation Board TM4C1294NCPDT vertraut machen, mit dem wir uns auch in den nächsten Praktika beschäftigen wollen. Dabei richten wir über den Projektbrowser und den Projekteigenschaften alle wichtigen Eigenschaften ein, testen erste Konsolenausgaben und lassen LEDs in bestimmten Blinkmustern aufleuchten.

1.2 Vorbereitung

1.2.1 V1.1

Wir wollen uns zunächst über die Begriffe Kennlinienfeld, Arbeitspunkt und stationäres Verhalten klar werden:

Kennlinienfeld Als Kennlinienfeld bezeichnen wir ein Feld, also eine Anreihung, von Kennlinien, also einem Zusammenhang zweier physikalischer Größen in graphischer Darstellung, in einem einzigen Diagramm zur Veranschaulichung der Betriebsmitteleigenschaft unter verschiedenen Umständen

Arbeitspunkt Der Arbeitspunkt beschreibt den Punkt im Kennlinienfeld, in dem ein technisches Gerät, aufgrund der äußeren Einflüsse und der gewählten Umstände arbeitet.

stationäres Verhalten Als stationäres Verhalten beschreiben wir den Zustand, in dem eine Regelstrecke einen festen Zusammenhang zwischen Eingangs- und Ausgangsgröße erreicht hat. Der Endwert des Regelkreises wurde erreicht.

y

Wir wollen nun den Zusammenhang der Kennlinienfelder im Vorfeld bestimmen und das Ergebnis in der Auswertung überprüfen. Aus

$$\vartheta = \frac{1}{c_L \rho_L A} \frac{1}{v} P_{th} \tag{1.1}$$

können wir erkennen, dass der Zusammenhang aus Heizspannung u_{uP} und $u_{y\vartheta}$ linear sein muss. Da alle weiteren Werte in der Gleichung als konstant für unser Kennlinienfeld angenommen werden, bleibt nur noch eine Funktion der Form

$$f(u_{uP}) = u_{y\vartheta} = m \cdot x \tag{1.2}$$

übrig. Da die Spannung auch proportional zur Leistung P_{el} ist, überträgt sich diese Proportionalität auf diesen Zusammenhang.

Der Zusammenhang zwischen Lüfterspannung U_{uL} und $u_{y\vartheta}$ ergibt sich über eine ähnliche Argumentation als linear anti-proportional, da die Lüfterspannung in 1.1 direkt die Luftgeschwindigkeit beeinflusst. Es ergibt sich:

$$f(u_{uL}) = u_{y\vartheta} = m \cdot \frac{1}{x} \tag{1.3}$$

1.2.2 V1.2

Als nächstes wollen wir die Sprungantwort der Temperaturkurve auf einen Heizleisungssprung $P_{el}(t) = \hat{P}_{el}(t)\sigma(t)$ berechnen.

$$P_{el}(t) \circ\!\!-\!\!\bullet P_{el}(s) \tag{1.4}$$

$$P_0 \cdot \sigma(t) \circ\!\!-\!\!\bullet P_0 \cdot \frac{1}{s} \tag{1.5}$$

$$\tag{1.6}$$

Wir setzen nun diese Transformation in die Übertragungsfunktion des Grundlagenteils ein:

$$\vartheta(s) = G_s(s) \cdot P_{el}(s) = K_s \frac{e^{-T_t s}}{1 + T_s s} \cdot P_{el} \tag{1.7}$$

$$\vartheta(s) = K(s) \frac{e^{-T_t s}}{1 + T_t s} \cdot P_0 \cdot \frac{1}{s} \tag{1.8}$$

Wir transformieren diesen Ausdruck nun wieder in den Zeitbereich über

$$\vartheta(t) = K_s \cdot P_0 \cdot \mathcal{L}^{-1}\left\{\frac{1}{s} \cdot e^{-T_t s} \cdot \frac{1}{1 + T_s s}\right\} = K_s \cdot P_0 \cdot \frac{1}{T_s} \cdot \mathcal{L}^{-1}\left\{\frac{1}{s} \cdot e^{-T_t s} \cdot \frac{1}{\frac{1}{T_s} + s}\right\} \tag{1.9}$$

Mithilfe des Verschiebungs- und Integrationssatzes erhalten wir:

$$\vartheta(t) = K_s \cdot P_0 \cdot \frac{1}{T_s} \cdot \int_0^{t - T_t} e^{-\frac{-\tau}{T_s}} d\tau \cdot \sigma(t - T_t) \tag{1.10}$$

Wir bilden nun die Stammfunktion und vereinfachen so weit wie möglich:

$$\vartheta(t) = K_s \cdot P_0 \cdot \frac{1}{T_s} \left[-T_s \cdot e^{-\frac{t}{T_s}}\right]_0^{t - T_t} \sigma(t - T_t) \tag{1.11}$$

1.3 Aufgabe 1: Binärzähler mit Timer

1.3.1 Teilaufgabe A

Für die zu bewältigende Aufgabe müssen die GPIO-Pins der Register N und F konfiguriert werden, da diese direkt mit den LEDs des Evaluation Boards verbunden sind. Dazu verwenden wir folgenden Programmabschnitt, den wir schon aus Praktikum 1 kennen:

```
1  void init() {
2      // Setting the clock to both registers N and F
3      SYSCTL_RCGCGPIO_R = 0x00001020;
4      // Waiting for the clock to be ready
5      while ((SYSCTL_PRGPIO_R & 0x00001020) == 0);
6      // Setting the direction of the pins to output
7      GPIO_PORTF_AHB_DIR_R = 0x11;
8      GPIO_PORTN_DIR_R = 0x03;
9      // Enabling all of the pins
10     GPIO_PORTN_DEN_R = 0x03;
11     GPIO_PORTF_AHB_DEN_R = 0x11;
12 }
```

Listing 1.1: GPIO-Port Konfiguration

1.3.2 Teilaufgabe B

Für das Einstellen der Load Value des Timers im 32-bit Modus wollen wir den Befehl

```
TIMER0_TAILR_R = 0x00FFFFFF;
```

Den einzustellenden Wert errechnen wir aus der Taktfrequenz f des Boards.

$$lV = 1s \cdot f = 1s \cdot 16MHz = 16000000_{10} \tag{1.12}$$

Aus den Vorlesungsfolien des Kapitels 5 zu dem 32-bit-Timer haben wir dann den Zahlenwert für die Timerperiode von einer Sekunde gefunden, welche dort mit $0x00FFFFFF \approx 1,048s$ angegeben ist.

1.3.3 Teilaufgabe C

Mit einem 16-bit-Timer ohne Prescaler kann maximal ein Wert von $T = 65536$ erreicht werden. Dieser entspricht, bei einer Taktfrequenz von $16MHz$, einer Periodendauer von $t = 4,096ms$. Aus $1s >> 4,096ms$ folgt, dass wir einen Prescaler für das Erreichen unseres Zieles brauchen. Daher rechnen wir:

$$T_p = \frac{1s}{2^{16}} = 15,3\mu s \implies \frac{1}{15,3\mu s} = 65,359kHz = f_p \tag{1.13}$$

1.3.4 Teilaufgabe D

Wir wollen nun den direkten Wert für unseren Prescaler berechnen, daher rechnen wir:

$$p = \frac{16MHz}{f_p} = 244,8 \approx 245 \tag{1.14}$$

Um nun die Load Value zu berechnen, nutzen wir eine Formel aus der Vorlesung und schreiben:

$$\text{ILR} = (\frac{16M}{245} \cdot 1) - 1 = 65306 - 1 \tag{1.15}$$

Wir schreiben also in unseren Code:

```
TIMER0_TAPR_R = 245;
TIMER0_ TAILR_R = 65306-1;
```

1.4 Programm: Binärzähler mit Timer

Für das Bearbeiten dieser Aufgabe nutzen wir den folgenden Code:

```
1  /*
2  Mikroprozessortechnik 1 - Praktikum 2: Timer und Interrupts
3  Aufgabe 2
4  Autoren: Emily Antosch, Florian Tietjen
5  Beschreibung: Implementierung des Timers als Zeitmessung fuer die Darstellung von
       Binaerzahlen mithilfe von LEDs auf dem Evaluation Board.
6  */
7
8  #include<stdint.h>
9  #include "inc\tm4c1294ncpdt.h"
10
11 void init();
12 void timerInit();
13
14 int main(void)
15 {
16     // Call to the init function
17     init();
18     // Call to the timerInit function
```

```
19      timerInit();
20      // Initialization of the variables for the binary number
21      unsigned short displayedValue = 0;
22      // Infinite loop
23
24      while(1)
25      {
26          // Calculating the binary number
27          GPIO_PORTF_AHB_DATA_R = (displayedValue & 0x02) << 3 | (displayedValue & 0
        x01);
28          GPIO_PORTN_DATA_R = (displayedValue & 0x0C) >> 2;
29          // Incrementing the displayedValue
30          displayedValue == 15 ? displayedValue = 0 : displayedValue++;
31          //Waiting for the timer to reach timeout
32          while ((TIMER0_RIS_R & (1<<0)) == 0);
33          // Clearing the timer timeout flag
34          TIMER0_ICR_R |= (1<<0);
35      }
36  }
37  // Function to initialize the registers
38  void init(){
39      // Setting the clock to both registers N and F
40      SYSCTL_RCGCGPIO_R = 0x00001020;
41      // Waiting for the clock to be ready
42      while ((SYSCTL_PRGPIO_R & 0x00001020) == 0);
43      // Setting the direction of the pins to output
44      // and enabling the both registers with the pins
45      GPIO_PORTF_AHB_DEN_R = 0x11;
46      GPIO_PORTF_AHB_DIR_R = 0x11;
47      GPIO_PORTN_DEN_R = 0x03;
48      GPIO_PORTN_DIR_R = 0x03;
49
50  }
51  // Function to initialize the timer
52  void timerInit(){
53      // Setting the clock to the timer
54      SYSCTL_RCGCTIMER_R |= (1<<0);
55      // Waiting for the clock to be ready
56      while((SYSCTL_PRTIMER_R & 0x01) == 0);
57      // Disabling the timer
58      TIMER0_CTL_R &= ~(1<<0);
59      // Setting the timer to be in 32-bit mode
60      TIMER0_CFG_R = 0x00;
61      // Setting the timer to be in periodic mode and match enable
62      TIMER0_TAMR_R = 0x02 | (1<<3);
63      // Setting the load value to be equal to about 1 second
64      TIMER0_TAILR_R = 0x00FFFFFF;
65      // Enabling the timer
66      TIMER0_CTL_R |= (1<<0);
67  }
```

Listing 1.2: timer.c

Dabei schließen wir das Evaluation Board an und führen das Programm dann im Debugmodus aus. Auf der Konsole erscheint eine schnell wachsende Liste von dem eingegebnen Namen, der Matrikelnummer und der steigenden Zählervariable cnt. Das zeigt uns zum Einen, wie wir die printf-Funktion in Code Composer Studio zu nutzen haben, um in größeren Programmen, Fehler finden zu können, zum Anderen haben wir das Setup des Boards richtig gemacht und das Kompilieren und Ausführen von Programmen funktioniert.

1.5 Programm: Interrupts und Interrupthandler

Dazu verwenden wir den folgenden Code:

```
/*
Mikroprozessortechnik 1 - Praktikum 2: Timer und Interrupts
Aufgabe 2
Autoren: Emily Antosch, Florian Tietjen
Beschreibung: Implementierung eines Interrupts durch einen Timer auf der NVIC-Ebene.
     Darstellung eines Laufmusters mithilfe der LEDs.
*/

#include<stdio.h>
#include<stdint.h>
#include "inc\tm4c1294ncpdt.h"

// Function prototypes
void init();
void timerInit(unsigned long period);
void Timer2A_Handler();


// main function
int main(void){
    // Call init function
    init();
    // Call timerInit function
    timerInit(0x01EFFFFF);
    // Enable timer
    TIMER2_CTL_R |= 0x00000001;
    // Empty while loop
    while(1)
    {

    }
}
// Function to initialize the GPIO pins
void init(){
    SYSCTL_RCGCGPIO_R |= 0x00001020;
    while((SYSCTL_PRGPIO_R & 0x00001020)==0);
    // Initialization of ports by enabling and setting them to output
    GPIO_PORTF_AHB_DEN_R |= 0x11;
    GPIO_PORTF_AHB_DIR_R |= 0x11;
    GPIO_PORTN_DEN_R |= 0x03;
    GPIO_PORTN_DIR_R |= 0x03;
}
// Function to initialize the timer
void timerInit(unsigned long period){
    // Set clock to timer 2
    SYSCTL_RCGCTIMER_R |= 0x04;
    // Wait for clock to be set up
    while((SYSCTL_PRTIMER_R&0x04)==0);
    //Disable timer 2
    TIMER2_CTL_R &= ~0x01;
    //Set timer to 32-bit mode
    TIMER2_CFG_R = 0x00;
    // Enable periodic mode
    TIMER2_TAMR_R = 0x02;
    // Set load value to parameter period - 1
    TIMER2_TAILR_R = period-1;
    // Clear timeout flag
    TIMER2_ICR_R = 0x01;
    // Enable timer to start an interrupt on timeout flag
    TIMER2_IMR_R = 0x01;
    // Enable NVIC to timer 2 to send an interrupt to CPU
    NVIC_EN0_R |= (1<<23);
}

// Function to handle the interrupt
```

```
65 void Timer2A_Handler(){
66     // Create i for delay
67     int i = 0;
68     // Clear timeout flag
69     TIMER2_ICR_R |= 0x01;
70     // Set LED pattern to light up all LEDs
71     GPIO_PORTN_DATA_R = 0x03;
72     GPIO_PORTF_AHB_DATA_R = 0x11;
73     for(i = 0; i < 600000; i++)
74         ;
75     // Clear LEDs to wait for next interrupt
76     GPIO_PORTN_DATA_R = 0x00;
77     GPIO_PORTF_AHB_DATA_R = 0x00;
78 }
```

Listing 1.3: Timer-Interrupt mithilfe des Timers 2 im 32-bit-mode

Dabei verwenden wir die beiden Register F und N, in welchen die Pins für die LEDs auf dem Board sind. Wir setzen alle Pins auf Ausgang und Enablen diese. Im Nachgang setzen wir die jeweiligen Pins im entsprechenden Blinkmuster auf OFF oder ON mit einer Warteschleife dazwischen.

1.5.1 Ausgabe des Programms

Wir erkennen einen durchgängigen Verlauf der LEDs, bei dem die LEDs der Reihe nach aufleuchten und dann wieder ausgehen, sodass immer nur eine LED zur Zeit eingeschaltet ist. Es entsteht der Effekt eines "laufenden"Lichts, welches sich immer von links nach rechts bewegt.

1.5.2 Gleichzeitige Schaltung beider Eingänge in Port N

Um beide LEDs an Port N gleichzeitig zu schalten müssen wir lediglich folgende Programmzeilen verwenden:

```
1  GPIO_PORTN_DATA_R = 0x03;
2  GPIO_PORTN_DATA_R = 0x00;
```

Listing 1.4: Port N Pins gleichzeitig schalten

Dabei ist die erste Zeile das Schalten beider Pins auf ON, die zweite Zeile schaltet beide auf OFF. Dabei gilt $0x03_{16} = 0011_2$ und $0x00_{16} = 0000_2$, wobei eine 1 auf einem bestimmten Port ein ON-Signal und eine 0 ein OFF-Signal darstellt.