MP-P2
Gruppe 4

2021

Timer und Interrupts

14. 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

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: GPIO-Port Konfiguration

2.1 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.

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

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.

2.2 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:

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

2.3 Teilaufgabe D

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

$$p = \frac{16MHz}{f_p} = 244,8 \approx 245 \tag{3}$$

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

$$\text{ILR} = (\frac{16M}{245} \cdot 1) - 1 = 65306 - 1 \tag{4}$$

Wir schreiben also in unseren Code:

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

3 Programm: Binärzähler mit Timer

Aufgabe 3.0

In dieser Aufgabe wird der Binärzähler mit Timer implementiert. Das Binärmuster wird mit Hilfe der LED´s ausgegeben. Jede Zahl soll eine Sekunde lang ausgegeben werden, ehe dann die nächste Zahl angezeigt wird.

Der umgesetzte Code sieht wie folgt aus:

```
/*
Mikroprozessortechnik 1 - Praktikum 2: Timer und Interrupts
Aufgabe 2
Autoren: Emily Antosch, Florian Tietjen
Beschreibung: Implementierung des Timers als Zeitmessung fuer die Darstellung von
    Binaerzahlen mithilfe von LEDs auf dem Evaluation Board.
*/

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

void init();
void timerInit();

int main(void)
{
    // Call to the init function
    init();
    // Call to the timerInit function
    timerInit();
    // Initialization of the variables for the binary number
    unsigned short displayedValue = 0;
    // Infinite loop

    while(1)
    {
        // Calculating the binary number
        GPIO_PORTF_AHB_DATA_R = (displayedValue & 0x02) << 3
        | (displayedValue & 0x01);
        GPIO_PORTN_DATA_R = (displayedValue & 0x0C) >> 2;
        // Incrementing the displayedValue
        displayedValue == 15 ? displayedValue = 0 : displayedValue++;
        //Waiting for the timer to reach timeout
        while ((TIMER0_RIS_R & (1<<0)) == 0);
        // Clearing the timer timeout flag
        TIMER0_ICR_R |= (1<<0);
    }
}
// Function to initialize the registers
void init(){
    // Setting the clock to both registers N and F
    SYSCTL_RCGCGPIO_R = 0x00001020;
    // Waiting for the clock to be ready
    while ((SYSCTL_PRGPIO_R & 0x00001020) == 0);
    // Setting the direction of the pins to output
```

```
45      // and enabling the both registers with the pins
46      GPIO_PORTF_AHB_DEN_R = 0x11;
47      GPIO_PORTF_AHB_DIR_R = 0x11;
48      GPIO_PORTN_DEN_R = 0x03;
49      GPIO_PORTN_DIR_R = 0x03;
50
51  }
52  // Function to initialize the timer
53  void timerInit(){
54      // Setting the clock to the timer
55      SYSCTL_RCGCTIMER_R |= (1<<0);
56      // Waiting for the clock to be ready
57      while((SYSCTL_PRTIMER_R & 0x01) == 0);
58      // Disabling the timer
59      TIMER0_CTL_R &= ~(1<<0);
60      // Setting the timer to be in 32-bit mode
61      TIMER0_CFG_R = 0x00;
62      // Setting the timer to be in periodic mode and match enable
63      TIMER0_TAMR_R = 0x02 | (1<<3);
64      // Setting the load value to be equal to about 1 second
65      TIMER0_TAILR_R = 0x00FFFFFF;
66      // Enabling the timer
67      TIMER0_CTL_R |= (1<<0);
68  }
```

Listing 2: timer.c

Dabei schließen wir das Evaluation Board an, führen das Programm aus und setzen das Board in den Run-Modus. Der Binärzähler beginnt von eins bis 15 hochzuzählen. Die LED´s spiegeln das Muster der Binärzahlen perfekt wieder. Die LED´s leuchten jedes Muster für circa eine Sekunde, ehe sie dann das nächste Binärmuster wiederspiegeln. Um den zeitlichen Abstand zwischen jedem Muster zu überprüfen wird das Oszilloskop an einer der LED´s angeschlossen. Dabei ergibt sich folgendes Bild:

Abbildung 1: Zeitlicher Spannungsverlauf an der LED

Mithilfe der Measurement-Funktion des Oszilloskop wurde die Länge eines High-Signals ermittelt. Dabei ergab die ermittelte Zeit $\Delta T = 1,04s$ was ziemlich genau dem erwartetem Wert aus [2.1] entspricht.

4 Programm: Interrupts und Interrupthandler

Aufgabe 4.0

In der zweiten Aufgabe wird nun ein periodischer Interrupt implementiert. Alle zwei Sekunden soll ein Interrupt auslösen, bei jeder Auslösung soll ein beliebiges LED-Muster angezeigt werden.

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
```

```
56      TIMER2_ICR_R = 0x01;
57      // Enable timer to start an interrupt on timeout flag
58      TIMER2_IMR_R = 0x01;
59      // Enable NVIC to timer 2 to send an interrupt to CPU
60      NVIC_EN0_R |= (1<<23);
61  }
62
63  // Function to handle the interrupt
64  void Timer2A_Handler(){
65      // Create i for delay
66      int i = 0;
67      // Clear timeout flag
68      TIMER2_ICR_R |= 0x01;
69      // Set LED pattern to light up all LEDs
70      GPIO_PORTN_DATA_R = 0x03;
71      GPIO_PORTF_AHB_DATA_R = 0x11;
72      for(i = 0; i < 600000; i++)
73          ;
74      // Clear LEDs to wait for next interrupt
75      GPIO_PORTN_DATA_R = 0x00;
76      GPIO_PORTF_AHB_DATA_R = 0x00;
77  }
```

Listing 3: Timer-Interrupt mithilfe des Timers 2 im 32-bit-mode

Für die Interruptbehandlung nutzen wir das Vorlagenprogramm startup_ccs.c, welches wir im voreingestellten Arbeitsverzeichnis finden und in das Projekt einbinden. Der komplette Code befindet sich im Anhang [A]. Das Programm beinhaltet die Interrupt-Vector-Tabelle. Wir fügen lediglich den Prototypen des Interrupthandlers aus der Main ein:

```
extern void Timer2A_Handler(void);      // Functionprototype
```

Und rufen wir diesen an dem richtigen IVT-Eintrag wieder auf:

```
Timer2A_Handler,                        // Timer 2 subtimer A
```

Um nun zu überprüfen, ob der Interrupt wirklich alle zwei Sekunden ausgelöst wird, messen wir mithilfe des Oszilloskop das Signal an der LED:

Abbildung 2: Zeitlicher Spannungsverlauf an der LED

Da das Aufleuchten der LED zeitlich gesehen ein Teil des nächsten periodischen Interrupts ist, messen wir die Zeit zwischen dem ersten und zweiten Einschalten der LED. Es ergibt sich eine Zeit von $\Delta T = 2,02s$. Also genau die gewünschten zwei Sekunden.

5 Konklusion

Die Implementierung von Timern und Interrupts ist gelungen. Bei den Messungen mit dem Oszilloskop konnte erneut der Umgang geübt und vertieft werden. Die Messungen stellen ein wichtiges Instrument zur Analyse und Bewertungen von Signalen dar. So können auch kleinste Ungenauigkeiten festgestellt und interpretiert werden.
Das zweite Praktikum erwies sich als logische Fortsetzung des ersten Praktikums. Dadurch konnten viele Inhalte aus dem ersten Praktikum wiederverwendet werden. Das Programmieren in CCS wird immer vertrauter.

A Anhang

startup-File

```
//*****************************************************************************
//
// Startup code for use with TI's Code Composer Studio.
//
// Copyright (c) 2011-2014 Texas Instruments Incorporated.  All rights reserved.
// Software License Agreement
//
// Software License Agreement
//
// Texas Instruments (TI) is supplying this software for use solely and
// exclusively on TI's microcontroller products. The software is owned by
// TI and/or its suppliers, and is protected under applicable copyright
// laws. You may not combine this software with "viral" open-source
// software in order to form a larger program.
//
// THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
// NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
// NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
// A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY
// CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
// DAMAGES, FOR ANY REASON WHATSOEVER.
//
//*****************************************************************************

#include <stdint.h>

//*****************************************************************************
//
// Forward declaration of the default fault handlers.
//
//*****************************************************************************
void ResetISR(void);
static void NmiSR(void);
static void FaultISR(void);
static void IntDefaultHandler(void);
extern void Timer2A_Handler(void);

//*****************************************************************************
//
// External declaration for the reset handler that is to be called when the
// processor is started
//
//*****************************************************************************
extern void _c_int00(void);

//*****************************************************************************
//
// Linker variable that marks the top of the stack.
//
//*****************************************************************************
extern uint32_t __STACK_TOP;

//*****************************************************************************
//
// External declarations for the interrupt handlers used by the application.
//
//*****************************************************************************
// To be added by user

//*****************************************************************************
//
// The vector table.  Note that the proper constructs must be placed on this to
// ensure that it ends up at physical address 0x0000.0000 or at the start of
// the program if located at a start address other than 0.
```

```
65 //
66 //*****************************************************************************
67 #pragma DATA_SECTION(g_pfnVectors, ".intvecs")
68 void (* const g_pfnVectors[])(void) =
69 {
70     (void (*)(void))((uint32_t)&__STACK_TOP),
71                                             // The initial stack pointer
72     ResetISR,                               // The reset handler
73     NmiSR,                                  // The NMI handler
74     FaultISR,                               // The hard fault handler
75     IntDefaultHandler,                      // The MPU fault handler
76     IntDefaultHandler,                      // The bus fault handler
77     IntDefaultHandler,                      // The usage fault handler
78     0,                                      // Reserved
79     0,                                      // Reserved
80     0,                                      // Reserved
81     0,                                      // Reserved
82     IntDefaultHandler,                      // SVCall handler
83     IntDefaultHandler,                      // Debug monitor handler
84     0,                                      // Reserved
85     IntDefaultHandler,                      // The PendSV handler
86     IntDefaultHandler,                      // The SysTick handler
87     IntDefaultHandler,                      // GPIO Port A
88     IntDefaultHandler,                      // GPIO Port B
89     IntDefaultHandler,                      // GPIO Port C
90     IntDefaultHandler,                      // GPIO Port D
91     IntDefaultHandler,                      // GPIO Port E
92     IntDefaultHandler,                      // UART0 Rx and Tx
93     IntDefaultHandler,                      // UART1 Rx and Tx
94     IntDefaultHandler,                      // SSI0 Rx and Tx
95     IntDefaultHandler,                      // I2C0 Master and Slave
96     IntDefaultHandler,                      // PWM Fault
97     IntDefaultHandler,                      // PWM Generator 0
98     IntDefaultHandler,                      // PWM Generator 1
99     IntDefaultHandler,                      // PWM Generator 2
100    IntDefaultHandler,                      // Quadrature Encoder 0
101    IntDefaultHandler,                      // ADC Sequence 0
102    IntDefaultHandler,                      // ADC Sequence 1
103    IntDefaultHandler,                      // ADC Sequence 2
104    IntDefaultHandler,                      // ADC Sequence 3
105    IntDefaultHandler,                      // Watchdog timer
106    IntDefaultHandler,                      // Timer 0 subtimer A
107    IntDefaultHandler,                      // Timer 0 subtimer B
108    IntDefaultHandler,                      // Timer 1 subtimer A
109    IntDefaultHandler,                      // Timer 1 subtimer B
110    Timer2A_Handler,                        // Timer 2 subtimer A
111    IntDefaultHandler,                      // Timer 2 subtimer B
112    IntDefaultHandler,                      // Analog Comparator 0
113    IntDefaultHandler,                      // Analog Comparator 1
114    IntDefaultHandler,                      // Analog Comparator 2
115    IntDefaultHandler,                      // System Control (PLL, OSC, BO)
116    IntDefaultHandler,                      // FLASH Control
117    IntDefaultHandler,                      // GPIO Port F
118    IntDefaultHandler,                      // GPIO Port G
119    IntDefaultHandler,                      // GPIO Port H
120    IntDefaultHandler,                      // UART2 Rx and Tx
121    IntDefaultHandler,                      // SSI1 Rx and Tx
122    IntDefaultHandler,                      // Timer 3 subtimer A
123    IntDefaultHandler,                      // Timer 3 subtimer B
124    IntDefaultHandler,                      // I2C1 Master and Slave
125    IntDefaultHandler,                      // CAN0
126    IntDefaultHandler,                      // CAN1
127    IntDefaultHandler,                      // Ethernet
128    IntDefaultHandler,                      // Hibernate
129    IntDefaultHandler,                      // USB0
130    IntDefaultHandler,                      // PWM Generator 3
131    IntDefaultHandler,                      // uDMA Software Transfer
132    IntDefaultHandler,                      // uDMA Error
133    IntDefaultHandler,                      // ADC1 Sequence 0
```

```
134     IntDefaultHandler,                      // ADC1 Sequence 1
135     IntDefaultHandler,                      // ADC1 Sequence 2
136     IntDefaultHandler,                      // ADC1 Sequence 3
137     IntDefaultHandler,                      // External Bus Interface 0
138     IntDefaultHandler,                      // GPIO Port J
139     IntDefaultHandler,                      // GPIO Port K
140     IntDefaultHandler,                      // GPIO Port L
141     IntDefaultHandler,                      // SSI2 Rx and Tx
142     IntDefaultHandler,                      // SSI3 Rx and Tx
143     IntDefaultHandler,                      // UART3 Rx and Tx
144     IntDefaultHandler,                      // UART4 Rx and Tx
145     IntDefaultHandler,                      // UART5 Rx and Tx
146     IntDefaultHandler,                      // UART6 Rx and Tx
147     IntDefaultHandler,                      // UART7 Rx and Tx
148     IntDefaultHandler,                      // I2C2 Master and Slave
149     IntDefaultHandler,                      // I2C3 Master and Slave
150     IntDefaultHandler,                      // Timer 4 subtimer A
151     IntDefaultHandler,                      // Timer 4 subtimer B
152     IntDefaultHandler,                      // Timer 5 subtimer A
153     IntDefaultHandler,                      // Timer 5 subtimer B
154     IntDefaultHandler,                      // FPU
155     0,                                      // Reserved
156     0,                                      // Reserved
157     IntDefaultHandler,                      // I2C4 Master and Slave
158     IntDefaultHandler,                      // I2C5 Master and Slave
159     IntDefaultHandler,                      // GPIO Port M
160     IntDefaultHandler,                      // GPIO Port N
161     0,                                      // Reserved
162     IntDefaultHandler,                      // Tamper
163     IntDefaultHandler,                      // GPIO Port P (Summary or P0)
164     IntDefaultHandler,                      // GPIO Port P1
165     IntDefaultHandler,                      // GPIO Port P2
166     IntDefaultHandler,                      // GPIO Port P3
167     IntDefaultHandler,                      // GPIO Port P4
168     IntDefaultHandler,                      // GPIO Port P5
169     IntDefaultHandler,                      // GPIO Port P6
170     IntDefaultHandler,                      // GPIO Port P7
171     IntDefaultHandler,                      // GPIO Port Q (Summary or Q0)
172     IntDefaultHandler,                      // GPIO Port Q1
173     IntDefaultHandler,                      // GPIO Port Q2
174     IntDefaultHandler,                      // GPIO Port Q3
175     IntDefaultHandler,                      // GPIO Port Q4
176     IntDefaultHandler,                      // GPIO Port Q5
177     IntDefaultHandler,                      // GPIO Port Q6
178     IntDefaultHandler,                      // GPIO Port Q7
179     IntDefaultHandler,                      // GPIO Port R
180     IntDefaultHandler,                      // GPIO Port S
181     IntDefaultHandler,                      // SHA/MD5 0
182     IntDefaultHandler,                      // AES 0
183     IntDefaultHandler,                      // DES3DES 0
184     IntDefaultHandler,                      // LCD Controller 0
185     IntDefaultHandler,                      // Timer 6 subtimer A
186     IntDefaultHandler,                      // Timer 6 subtimer B
187     IntDefaultHandler,                      // Timer 7 subtimer A
188     IntDefaultHandler,                      // Timer 7 subtimer B
189     IntDefaultHandler,                      // I2C6 Master and Slave
190     IntDefaultHandler,                      // I2C7 Master and Slave
191     IntDefaultHandler,                      // HIM Scan Matrix Keyboard 0
192     IntDefaultHandler,                      // One Wire 0
193     IntDefaultHandler,                      // HIM PS/2 0
194     IntDefaultHandler,                      // HIM LED Sequencer 0
195     IntDefaultHandler,                      // HIM Consumer IR 0
196     IntDefaultHandler,                      // I2C8 Master and Slave
197     IntDefaultHandler,                      // I2C9 Master and Slave
198     IntDefaultHandler,                      // GPIO Port T
199     IntDefaultHandler,                      // Fan 1
200     0,                                      // Reserved
201 };
202
```

```
203 //*****************************************************************************
204 //
205 // This is the code that gets called when the processor first starts execution
206 // following a reset event.  Only the absolutely necessary set is performed,
207 // after which the application supplied entry() routine is called.  Any fancy
208 // actions (such as making decisions based on the reset cause register, and
209 // resetting the bits in that register) are left solely in the hands of the
210 // application.
211 //
212 //*****************************************************************************
213 void
214 ResetISR(void)
215 {
216     //
217     // Jump to the CCS C initialization routine.  This will enable the
218     // floating-point unit as well, so that does not need to be done here.
219     //
220     __asm("    .global _c_int00\n"
221           "    b.w     _c_int00");
222 }
223
224 //*****************************************************************************
225 //
226 // This is the code that gets called when the processor receives a NMI.  This
227 // simply enters an infinite loop, preserving the system state for examination
228 // by a debugger.
229 //
230 //*****************************************************************************
231 static void
232 NmiSR(void)
233 {
234     //
235     // Enter an infinite loop.
236     //
237     while(1)
238     {
239     }
240 }
241
242 //*****************************************************************************
243 //
244 // This is the code that gets called when the processor receives a fault
245 // interrupt.  This simply enters an infinite loop, preserving the system state
246 // for examination by a debugger.
247 //
248 //*****************************************************************************
249 static void
250 FaultISR(void)
251 {
252     //
253     // Enter an infinite loop.
254     //
255     while(1)
256     {
257     }
258 }
259
260 //*****************************************************************************
261 //
262 // This is the code that gets called when the processor receives an unexpected
263 // interrupt.  This simply enters an infinite loop, preserving the system state
264 // for examination by a debugger.
265 //
266 //*****************************************************************************
267 static void
268 IntDefaultHandler(void)
269 {
270     //
271     // Go into an infinite loop.
```

```
272     //
273     while(1)
274     {
275     }
276 }
```

Listing 4: startup_ccs.c