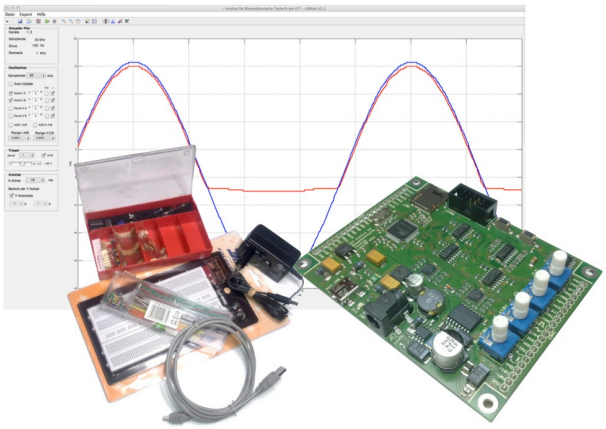
Karlsruher Institut für Technologie

Workshop

Elektrotechnik und Informationstechnik

Kurs 4

Digitale Signalverarbeitung



Gruppe 89

Vorname	Nachname	Matrikel-Nr.	u-Account	E-Mail
Max	Themistocli	2159422	uztld	uztld@student.kit.edu
Jan	Rösler	2096188	uclrb	uclrb@student.kit.edu
David	Ackermann	2121813	uepar	uepar@student.kit.edu

20. November 2018

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Interrupts

Ein Interrupt im Allgemeinen ist eine Unterbrechung eines laufenden Prozesses um zeitkritische Prozesse direkt abarbeiten zu können. Sie sind nicht Bestandteil des laufenden Programmes, sondern werden durch asynchrone, externe Ereignisse ausgelöst und benötigen eine extra Hardware. So wird beispielsweise bei einem Druck einer Taste einer Tastatur ein Interrupt ausgelöst um den jeweiligen Befehl direkt ausführen zu können. Eine Sonderklasse der Interrupts sind die non maskable Interrupts, kurz NMI. Diese sind höhergestellt als normale Interrupts und werden mit höchster Priorität direkt abgearbeitet und können nicht von dem Prozessor abgelehnt werden. Im Falle eines Stromausfalles greifen diese ein und sichern wichtige Daten bevor es zum Absturz kommt. Grundsätzlich Unterscheidet man Interrupts weiter in Hardware, Software, präzise und unpräzise Interrupts. Hardware Interrupts sind hierbei Interrupts die von einer angeschlossenen Hardware an die CPU weitergeleitet werden. Dem entgegen stehen die Software Interrupts, welche wie ein Hardware Interrupt wirken, jedoch von einem Programm aufgerufen werden. Der Ablauf von Interruptzyklen sieht folgender maßen aus: Wenn kein NMI vorliegt wird bei einem Interrupt bis Ende des Befehls gewartet. Nach abarbeiten des Befehls wird der Interruptzyklus der CPU gestartet. Hierbei liest der Datenbus den Interruptvektor und sperrt den maskierten Interrupteingang um ein überschneiden mehrerer Interrupts zu verhindern. Um im Anschluss wieder nahtlos das Programm fortführen zu können speichert die CPU den Befehlszähler im Stack ab. Mit Hilfe des Interruptvektors und einer Interrupttabelle kann der Interruptzeiger bestimmt werden. Anschließend läuft die Interrupt-Service-Routine ab. Hierbei ist zu beachten, dass alle Daten des vorherigen Programms zuvor in den Stack kopiert werden und nach Ablauf des Programms ebenfalls wieder zurück kopiert werden.

2 Diskretisierung

Ein Musiksignal wird mit einer Frequenz von 44,1 kHz abgetastet und hat eine Auflösung von 16 bit. Mit Gleichung 1 benötigt ein 10 s Musiksignal 7,056 Mbit Speicherplatz.

$$44,1\text{kHz} \cdot 10\text{s} \cdot 16\text{bit} = 7,056\text{Mbit} \tag{1}$$

3 Erstes Programm

3.1 Blinkllicht

3.2 RGB LED

3.2.1

Um die Hexadecimalzahl zu einer Binärzahl umzuwandeln wird diese durch 2 geteilt und der Rest notiert bis Null durch Zwei geteilt wird. Der Rest der ersten Division entspricht dem least

significant bit (LSB) und der Rest der letzten Division entspricht dem most significant bit (MSB). Dieser Vorgang wird in Gleichung 2 verdeutlicht.

$$\frac{4}{2} = 2,\ \mathsf{R} = 0; \qquad \frac{2}{2} = 1,\ \mathsf{R} = 0; \qquad \frac{1}{2} = 0,\ \mathsf{R} = 1 \qquad \rightarrow 0\mathrm{x}04 = 0\mathsf{b}100 \tag{2}$$

Port F hat eine Basis-Adresse von $0\mathrm{x}4002.5000$ und das Direction Register hat eine Offset Adresse von $0\mathrm{x}400$. Das setzen eines bits in dem Direction Register konfiguriert den jeweiligen Pin als einen Output. Die Addition der Basis- und Offset-Adressen führt zu einer Adresse bei der nun die Pins für Port F konfiguriert werden können. Durch das beschreiben dieses Registers mit $0\mathrm{x}04$, wird Pin PF2, die blaue LED, als Output konfiguriert.

3.2.2

Sodass die grüne LED blinkt, muss Port PF3 als Output konfiguriert werden. Wie zuvor müssen die Offset- und Basis-Adresse addiert werden und darauf hin Pin PF3 als output definiert werden. PF3 entspricht bit Nummer 3 und kann mit dem Wert $0\mathrm{b}00001000$ beschrieben werden um dies zu bewirken. Der equivalente Hexadezimalwert wäre $0\mathrm{x}08$.

3.2.3

Zurzeit wird nur die LED an PF2 zum blinken gebracht, indem der Inhalt von Register GPIO DATA auf $0\mathrm{x}04$ gesetzt wird. Um die LED an PF3 zum blinken zu bringen muss, wie in der letzten Aufgabe gezeigt, der Inhalt des GPIO DATA Registers auf $0\mathrm{x}08$ gesetzt werden. Sodass beide gleichzeitig an gehen, müssen die oben genannten Inhalte addiert werden. Dies würde einem Wert von $0\mathrm{x}0\mathrm{C}$ oder $0\mathrm{b}00001100$ entsprechen. Der Hexadezimalcode ist hier kürzer, aber bei dem Binären Code kann man sofort sehen welche bits gesetzt sind und man kann diese sofort ändern ohne Berechnungen durchführen zu müssen, wenn man ein zusätzliches bit setzen bzw. löschen will.

4 Verwenden des TivaWare-Framework

4.1 GPIO-Blinklicht

```
1  #include <stdint.h>
2  #include <stdbool.h> // definition of type "bool"
3  #include"inc/hw_memmap.h" // definition of memory adresses
4  #include"inc/hw_types.h" // definition of framework makros
5  #include"driverlib/gpio.h"
6  #include"driverlib/sysctl.h"
7
8  // Delay-Funktion
9
10 void delay(void)
```

```
11 {
12   uint32_t i=80000;
13   while(i) {i--;}
14  }
15
16 int main(void){
17     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);          //Port F aktivieren
18     GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1);    //Pin 1 bis 3 ↩
           initialisieren
19     GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_2);
20     GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_3);
21
22    while(1)       //unendliche Schleife fuer flimmern der LED in weiss
23    {
24        delay();
25     GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0xFF);
26     GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0xFF);
27     GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0xFF);
28         delay();
29     GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0x00);
30     GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0x00);
31     GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0x00);
32    }
33     return 0;
34 }
```

4.2 Externe LED Schaltung

4.2.1

```
1  /**
2   * main.c
3   */
4  #include <stdint.h>
5  #include <stdbool.h>                                          // definition ↩
       of type "bool"
6  #include"inc/hw_memmap.h"                                    // definition ↩
       of memory adresses
7 #include"inc/hw_types.h"                                      // definition ↩
      of framework makros
8  #include"driverlib/gpio.h"
9  #include"driverlib/sysctl.h"
10
11 void delay(void)
12 {
13   uint32_t i=100000;
14   while(i) {i--;}
15  }
16
```

```
17 int main(void){
18     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);                    // GPIO Port B ↩
           aktivieren
19     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_0);             // GPIO Port B ↩
           Pin 0-7 aktivieren
20     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_1);
21     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_2);
22     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_3);
23     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_4);
24     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_5);
25     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_6);
26     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_7);
27
28    while(1)
29    {
30        delay();
31     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0, 0xFF);                // GPIO Pins ↩
           auf High stellen
32     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, 0xFF);
33     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0xFF);
34     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0xFF);
35     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4, 0xFF);
36     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0xFF);
37     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6, 0xFF);
38     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_7, 0xFF);
39         delay();
40     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0, 0x00);                // GPIO Pins ↩
           auf Low stellen
41     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, 0x00);
42     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0x00);
43     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0x00);
44     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4, 0x00);
45     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x00);
46     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6, 0x00);
47     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_7, 0x00);
48
49    }
50     return 0;
51 }
```

4.2.2

```
1 /**
2  * main.c
3  */
4 #include <stdint.h>
5 #include <stdbool.h>                                                 // definition ↩
      of type "bool"
```

```
6  #include"inc/hw_memmap.h"                                    // definition ↩
       of memory adresses
7  #include"inc/hw_types.h"                                     // definition ↩
       of framework makros
8  #include"driverlib/gpio.h"
9  #include"driverlib/sysctl.h"
10
11 void delay(void)                                             // Blinklicht ↩
       Delay
12 {
13   uint32_t i=1000;
14   while(i) {i--;}
15  }
16
17 void delayPWM(void)                                          // PWM Delay (↩
       Hoher Wert -> Helle LED)
18 {
19   uint32_t i=8000;
20   while(i) {i--;}
21  }
22
23 int main(void){
24     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);             // GPIO Port B ↩
           aktivieren
25     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_0);      // GPIO Port B ↩
           Pin 0-7 aktivieren
26     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_1);
27     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_2);
28     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_3);
29     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_4);
30     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_5);
31     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_6);
32     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_7);
33
34    while(1)
35    {
36        delay();
37
38     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0, 0xFF);         // GPIO Pins ↩
           0-7 auf High stellen
39     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, 0xFF);
40     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0xFF);
41     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0xFF);
42     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4, 0xFF);
43     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0xFF);
44     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6, 0xFF);
45     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_7, 0xFF);
46
47         delayPWM();
48
```

```
49      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0, 0x00);            // GPIO Pins ↩
            0-7 auf Low stellen
50      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, 0x00);
51      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0x00);
52      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0x00);
53      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4, 0x00);
54      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x00);
55      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6, 0x00);
56      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_7, 0x00);
57
58    }
59     return 0;
60 }
```

5 Warteschleifen und Taktfrequenz

5.1

Der Mikrokontroller besitzt einen 16 MHz Quarz, welcher mit Hilfe der PLL 400 MHz erreicht. Durch den Divisor Div_5 wird die gegebene Frequenz noch durch 5 dividiert wird. Somit ergibt sich mit der internen Halbierung eine Frequenz von 40 MHz. Die Berechnung ist in Gleichung 3 gezeigt.

$$\frac{400\,\text{MHz}}{2 \times 5} = 40\,\text{MHz} \tag{3}$$

5.2

```
1  #include <stdint.h>
2  #include <stdbool.h>
3  #include "inc/hw_ints.h"
4  #include "inc/hw_memmap.h"
5  #include "inc/hw_types.h"
6  #include "driverlib/gpio.h"
7  #include "driverlib/sysctl.h"
8  #include "driverlib/timer.h"
9  #include "driverlib/interrupt.h"
10
11 // stores ms since startup
12 volatile uint32_t systemTime_ms = 0;
13
14 void InterruptHandlerTimer0A (void)
15 {
16     // Clear the timer interrupt to prevent the interrupt function from ↩
           immediately being called again on exit
17     TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
```

```
18      // count up one ms
19      systemTime_ms++;
20  }
21
22  void clockSetup(void)
23  {
24      uint32_t timerPeriod;
25      //Configure clock
26      SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|↩
            SYSCTL_OSC_MAIN);
27      //enable peripheral for timer
28      SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
29      //configure timer as 32 bit timer in periodic mode
30      TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
31      //set timerPeriod to number of periods needed to generate a timeout with a ↩
            frequency of 1kHz (every 1ms)
32      timerPeriod = (SysCtlClockGet()/1000);
33      //set TIMER-0-A to generate a timeout after timerPeriod-1 cycles
34      TimerLoadSet(TIMER0_BASE, TIMER_A, timerPeriod-1);
35      //Register the function InterruptHandlerTimer0A to be called when an ↩
            interrupt from TIMER-0-A occurs
36      TimerIntRegister(TIMER0_BASE, TIMER_A, &(InterruptHandlerTimer0A));
37      //Enable the interrupt for TIMER-0-A
38      IntEnable(INT_TIMER0A);
39      //generate an interrupt, when TIMER-0-A sends a timeout
40      TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
41      //master interrupt enable for all interrupts
42      IntMasterEnable();
43      //Enable the timer to start counting
44      TimerEnable(TIMER0_BASE, TIMER_A);
45  }
46
47  void delay_ms(uint32_t waitTime)
48  {
49      uint32_t t = systemTime_ms;
50      while (systemTime_ms - t < waitTime);
51  }
52
53  int main(void)
54  {
55  clockSetup();
56          SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);                    // GPIO Port↩
                B aktivieren
57              GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_0);          // GPIO↩
                    Port B Pin 0-7 aktivieren
58              GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_1);
59              GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_2);
60              GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_3);
61              GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_4);
62              GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_5);
```

```
63              GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_6);
64              GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_7);
65
66              bool state = 0;
67              //Verbeseserte Variante des Codes der GPIO-Steuerung aus Aufgabe 4
68                  while (1)
69                  {
70                      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0, GPIO_PIN_0*state)↩
                            ;
71                      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, GPIO_PIN_1*state)↩
                            ;
72                      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, GPIO_PIN_2*state)↩
                            ;
73                      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, GPIO_PIN_3*state)↩
                            ;
74                      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4, GPIO_PIN_4*state)↩
                            ;
75                      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, GPIO_PIN_5*state)↩
                            ;
76                      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6, GPIO_PIN_6*state)↩
                            ;
77                      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_7, GPIO_PIN_7*state)↩
                            ;
78
79                      state = !state;
80                      delay_ms(500/2);
81                  }
82              }
```

5.3

Tabelle 1: Flimmerverschmelzungsfrequenz

Name	Frequenz [Hz]
David	50
Jan	47,6
Max	45

6 Digitale Eingänge

6.1 Taster

6.1.1

```
1  /**
2   * main.c
3   */
4  #include <stdint.h>
5  #include <stdbool.h>                                    // definition of type "bool"
6  #include"inc/hw_memmap.h"                              // definition of memory adresses
7  #include"inc/hw_types.h"                               // definition of framework makros
8  #include"driverlib/gpio.h"
9  #include"driverlib/sysctl.h"
10
11 void delay(void)                                       // Blinklicht Delay
12 {
13   uint32_t i=500000;
14   while(i) {i--;}
15  }
16
17 void delayPWM(void)                                    // PWM Delay (Hoher Wert -> Helle LED)
18 {
19   uint32_t i=10000;
20   while(i) {i--;}
21  }
22
23 int main(void){
24      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);      // GPIO Port B aktivieren
25      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_0);   // GPIO Port B Pin 0-7 aktivieren
26      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_1);
27      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_2);
28      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_3);
29      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_4);
30      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_5);
31      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_6);
32      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_7);
33
34      GPIOPinTypeGPIOInput(GPIO_PORTC_BASE,GPIO_PIN_7);     // Schalter input config
35      GPIOPadConfigSet(GPIO_PORTC_BASE,GPIO_PIN_7,GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);
36
37     while(1)
38     {
39         delay();
40         if(GPIOPinRead(GPIO_PORTC_BASE,GPIO_PIN_7)==0) {
```

```
41      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0, 0xFF);            // GPIO Pins ↩
            0-7 auf High stellen
42      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, 0xFF);
43      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0xFF);
44      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0xFF);
45      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4, 0xFF);
46      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0xFF);
47      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6, 0xFF);
48      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_7, 0xFF);
49         }
50         else
51         {
52      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0, 0x00);            // GPIO Pins ↩
            0-7 auf Low stellen
53      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, 0x00);
54      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0x00);
55      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0x00);
56      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4, 0x00);
57      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x00);
58      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6, 0x00);
59      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_7, 0x00);
60         }
61     }
62      return 0;
63 }
```

6.1.2

```
1  /**
2   * main.c
3   */
4  #include <stdint.h>
5  #include <stdbool.h>                                              // definition ↩
       of type "bool"
6  #include"inc/hw_memmap.h"                                        // definition ↩
       of memory adresses
7  #include"inc/hw_types.h"                                          // definition ↩
       of framework makros
8  #include"driverlib/gpio.h"
9  #include"driverlib/sysctl.h"
10
11 void delay(void)
12 {
13   uint32_t i=200000;
14   while(i) {i--;}
15  }
16
17 int main(void){
```

```
18      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);                    // Port F ↩
            Aktivieren
19      GPIOPinTypeGPIOInput(GPIO_PORTF_BASE,GPIO_PIN_4);               // PF4 als ↩
            Input definieren
20      GPIOPadConfigSet(GPIO_PORTF_BASE,GPIO_PIN_4,GPIO_STRENGTH_2MA,↩
            GPIO_PIN_TYPE_STD_WPU);
21      // Bei der der Unteraufagbe 6.1.2 muesste der der Port C und Pin 7 gewaehlt↩
             werden
22
23      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);                    // GPIO Port B ↩
            aktivieren
24      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_0);             // GPIO Port B ↩
            Pin 0-7 aktivieren
25      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_1);
26      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_2);
27      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_3);
28      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_4);
29      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_5);
30      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_6);
31      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_7);
32
33      bool state = 0;
34
35     while(1)
36     {
37         delay();
38
39         if(GPIOPinRead(GPIO_PORTF_BASE,GPIO_PIN_4)==0)                  // Wenn ↩
               Schalter geschlossen
40         {
41            state = !state;                                              // State ↩
                  wird gewechselt
42         }
43
44         GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0, GPIO_PIN_0*state); // GPIO ↩
               Output wird auf jeweiligen State geschaltet
45         GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, GPIO_PIN_1*state);
46         GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, GPIO_PIN_2*state);
47         GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, GPIO_PIN_3*state);
48         GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4, GPIO_PIN_4*state);
49         GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, GPIO_PIN_5*state);
50         GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6, GPIO_PIN_6*state);
51         GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_7, GPIO_PIN_7*state);
52     }
53      return 0;
54 }
```

6.1.3

Der Code ist der selbe wie bei Aufgabenteil 6.1.2 nur, dass der Port C und Pin 7 gewählt werden müssten, statt Pin 4 vom Port F.

6.2 Externer Interrupt

```
1  #include <stdint.h>
2  #include <stdbool.h>
3  #include "inc/hw_ints.h"
4  #include "inc/hw_memmap.h"
5  #include "inc/hw_types.h"
6  #include "driverlib/gpio.h"
7  #include "driverlib/sysctl.h"
8  #include "driverlib/timer.h"
9  #include "driverlib/interrupt.h"
10 // Addresse zum ansteuern der jeweiligen LED
11 volatile uint32_t LED = 1;
12
13 //Delay-Funktion fuer Blinken
14 void delay(void)
15 {
16   uint32_t i=80000;
17   while(i) {i--;}
18 }
19
20 // Interruptroutine
21
22 void ex_int_handler(void) {
23
24 //Schalte vorherige LED aus
25
26 GPIOPinWrite(GPIO_PORTB_BASE, LED, 0x00);
27
28 // Interrupt-Flag loeschen
29
30 GPIOIntClear(GPIO_PORTF_BASE, GPIO_PIN_4);
31
32 GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE,GPIO_PIN_4);
33
34 LED = LED<<1;
35 // Von 8. LED zurueck auf 1. springen
36 if ( LED >= 1<<8)
37 {
38 LED = 1;
39 }
40 }
41
42 void main (void){
```

```
43      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);                    // GPIO Port B ↩
            aktivieren
44      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_0);             // GPIO Port B ↩
            Pin 0-7 aktivieren
45      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_1);
46      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_2);
47      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_3);
48      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_4);
49      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_5);
50      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_6);
51      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_7);
52
53      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);                    // Port F ↩
            Aktivieren
54      GPIOPinTypeGPIOInput(GPIO_PORTF_BASE,GPIO_PIN_4);               // PF4 als ↩
            Input definieren
55      GPIOPadConfigSet(GPIO_PORTF_BASE,GPIO_PIN_4,GPIO_STRENGTH_2MA,↩
            GPIO_PIN_TYPE_STD_WPU);
56
57      // Pin mit Interrupt
58      GPIOIntDisable(GPIO_PORTF_BASE,GPIO_PIN_4);
59      GPIOIntClear(GPIO_PORTF_BASE, GPIO_PIN_4);
60      GPIOIntTypeSet(GPIO_PORTF_BASE, GPIO_PIN_4, GPIO_FALLING_EDGE);
61      GPIOIntRegister(GPIO_PORTF_BASE, ex_int_handler);
62      GPIOIntEnable(GPIO_PORTF_BASE,GPIO_PIN_4);
63
64  while(1) {
65      delay();
66      GPIOPinWrite(GPIO_PORTB_BASE, LED, 0xFF);
67      delay();
68      GPIOPinWrite(GPIO_PORTB_BASE, LED, 0x00);
69    }
70  }
```

7 Analog-Digital-Umsetzer

7.1 A/D-Umsetzer

7.1.1

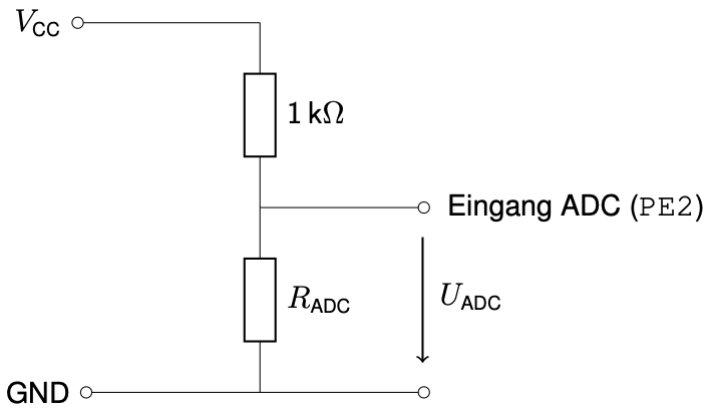


Abbildung 1: Schaltbild des Spannungsteilers zur Spannungsmessung

Die Formel zur Berechnung von U_{ADC} aus Abbildung 1 ist in Gleichung 4 gegeben.

$$U_{ADC} = V_{CC} \cdot \frac{R_{ADC}}{1000 + R_{ADC}} \tag{4}$$

Die Tabelle 2 zeigt die Widerstandswerte R_{ADC}, sowie die Spannungsabfälle U_{ADC} über diese Widerstandswerte und die A/D Werte.

Tabelle 2: Widerstandswerte und dazugehörige Spannungen

R_{ADC} [Ω]	U_{ADC} [V]	A/D Wert
100	0,3	372
150	0,43	534
220	0,60	738
680	1,34	1658
1000	1,65	2048
2200	2,27	2815
10 000	3,00	3724
15 000	3,09	3840

7.1.2

In **??** wird der ADC Wert berechnet, wobei V_{ref} 3,3 V beträgt.

$$\text{A/D Wert} = \left(2^{12} - 1\right) \cdot \frac{U_{\text{ADC}}}{V_{\text{ref}}} \tag{5}$$

Die berechneten Werte sind in Tabelle 2 zu finden.

7.2 Widerstandsmessung

Tabelle 3 zeigt die Widerstandsbereiche und die dazugehörigen A/D-Wertebereiche an.

Tabelle 3: Widerstandsbereiche und A/D Werte

$R_{\text{ADC, min}}$ [Ω]	$R_{\text{ADC max}}$ [Ω]	A/D Minimalwert	A/D Maximalwert
0	100	0	372
101	150	373	534
151	220	535	738
221	680	739	1658
681	1000	1659	2048
1001	2200	2049	2815
2201	10 000	2816	3723
10 001	15 000	3724	3839

```
1  /**
2   * main.c
3   */
4  #include<stdint.h>
5  #include <stdbool.h> // definition of type "bool"
6  #include"inc/hw_memmap.h" // definition of memory adresses
7  #include"inc/hw_types.h" // definition of framework makros
8  #include"driverlib/gpio.h"
9  #include"driverlib/sysctl.h"
10 #include<stdint.h>
11 #include<stdbool.h>
12 #include"inc/hw_ints.h"
13 #include"driverlib/timer.h"
14 #include"driverlib/interrupt.h"
15 #include "driverlib/debug.h"
16 #include "driverlib/sysctl.h"
17 #include "driverlib/adc.h"
18
19
20 //speichert ms seit dem Start
21 uint32_t systemTime_ms;
```

```
22
23 void InterruptHandlerTimer0A (void)
24 {
25 // Loeschen den Timer-Interrupt, um zu verhindern, dass die Interrupt-Funktion ↩
       beim Beenden sofort erneut aufgerufen wird
26 TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
27 // eine ms hoch zaehlen
28     systemTime_ms++;
29 }
30
31 void clockSetup(void)
32 {
33 uint32_t timerPeriod;
34 //Konfiguriert clock
35 SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN↩
       );
36 //Aktiviert peripheral fuer timer
37 SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
38 //konfiguriert timer als 32 bit timer in periodisch mode
39 TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
40 //setze Zeitperiode auf Anzahl der Perioden, die zum Erzeugen eines
41 //Timeouts mit einer Frequenz von 1 kHz (alle 1 ms) benoetigt werden
42 timerPeriod = (SysCtlClockGet()/1000);
43 // setze TIMER-0-A ein, um nach timePeriod-1-Zyklen ein Timeout zu generieren
44 TimerLoadSet(TIMER0_BASE, TIMER_A, timerPeriod-1);
45 //Registriert die Funktion InterruptHandlerTimer0A, die aufgerufen werden soll,
46 //wenn ein Interrupt von TIMER-0-A auftritt
47 TimerIntRegister(TIMER0_BASE, TIMER_A, &(InterruptHandlerTimer0A)) ;
48 //Aktiviert the interrupt fuer TIMER-0-A
49 IntEnable(INT_TIMER0A);
50 //erzeugt einen Interrupt , wenn TIMER-0-A ein Timeout sendet
51 TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
52 //master interrupt aktiviert fuer alle interrupts
53 IntMasterEnable();
54 //Aktiviert den Timer, um mit dem Zaehlen zu beginnen
55 TimerEnable(TIMER0_BASE, TIMER_A);
56
57 }
58
59 // Es erhaelt als Eingabeparameter die Verzoegerung in ms  und eine ↩
       Verzoegerung um genau diese Zeit bewirkt.
60
61 void delay_ms(uint32_t waitTime) {
62 //Variable systemTime_ms zaehlt eine Millisekunde hoch und durch die while ↩
       schleife vergleichen wir der Eingabe Parameter waitTime mit systemTime_ms
63 // Durch Eingabe der Zahl 500 zaehlt diese funktion bis zu 500 ms und wird ↩
       jedoch um diese Zeit verzoegert.
64     systemTime_ms=0;
65     while( systemTime_ms < waitTime);
66 }
```

```
67      void delay(void)
68      {
69        uint32_t i=80000;
70        while(i) {i--;}
71       }
72
73  int main(void){
74
75      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);                    //Set Up
76      SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);                     // Peripherie ↩
            aktivieren
77      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
78
79      GPIOPinTypeADC(GPIO_PORTE_BASE,GPIO_PIN_2);                     // PIN PE2 ADC ↩
            Funktion zuweisen
80
81      ADCSequenceConfigure(ADC0_BASE,1,ADC_TRIGGER_PROCESSOR,0);    // Prozessor ↩
            als Trigger Quelle
82      ADCSequenceStepConfigure(ADC0_BASE,1,0,ADC_CTL_CH1|ADC_CTL_IE|ADC_CTL_END);↩
             // AI1 abtasten/Interrupt erzeugen bei Ende/letzter Schritt
83      ADCSequenceEnable(ADC0_BASE,1);                                 // ADC Sequenz ↩
            1 aktivieren
84
85      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_0);             // GPIO Port B ↩
            Pin 0-7 aktivieren
86      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_1);
87      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_2);
88      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_3);
89      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_4);
90      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_5);
91      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_6);
92      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_7);
93      GPIOPinTypeGPIOInput(GPIO_PORTE_BASE, GPIO_PIN_4);
94
95      bool state = 0;
96      uint32_t ui32ADC0Value;
97
98      int R0 = 372;                       //ADC Schwellenwerte
99      int R1 = 534;
100     int R2 = 739;
101     int R3 = 1658;
102     int R4 = 2048;
103     int R5 = 2816;
104     int R6 = 3724;
105     int R7 = 3840;
106
107    while(1)
108    {
109
110        ADCIntClear(ADC0_BASE,1);                                     // evtl ↩
```

```
        vorhandene ADC Interrupts loeschen
111     ADCProcessorTrigger(ADC0_BASE,1);                                   // ↩
        Konvertierung beginnen
112
113     while(!ADCIntStatus(ADC0_BASE,1,false))                             // warten ↩
        bis Konvertierung abgeschlossen
114         {
115         }
116     ADCSequenceDataGet(ADC0_BASE,1,&ui32ADC0Value);
117
118     if((R1>ui32ADC0Value)&&(ui32ADC0Value>=R0))
119     {
120         GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0, GPIO_PIN_0*0xFF);
121         delay();
122         GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0, GPIO_PIN_0*0x00);
123         delay();
124     }
125
126     else if((R2>ui32ADC0Value)&&(ui32ADC0Value>=R1))
127         {
128         GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, GPIO_PIN_1*0xFF);
129         delay();
130         GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, GPIO_PIN_1*0x00);
131         delay();
132         }
133
134     else if((R3>ui32ADC0Value)&&(ui32ADC0Value>=R2))
135         {
136         GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, GPIO_PIN_2*0xFF);
137         delay();
138         GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, GPIO_PIN_2*0x00);
139         delay();
140         }
141     else if((R4>ui32ADC0Value)&&(ui32ADC0Value>=R3))
142         {
143         GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, GPIO_PIN_3*0xFF);
144         delay();
145         GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, GPIO_PIN_3*0x00);
146         delay();
147         }
148     else if((R5>ui32ADC0Value)&&(ui32ADC0Value>=R4))
149         {
150         GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4, GPIO_PIN_4*0xFF);
151         delay();
152         GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4, GPIO_PIN_4*0x00);
153         delay();
154         }
155     else if((R6>ui32ADC0Value)&&(ui32ADC0Value>=R5))
156         {
157         GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, GPIO_PIN_5*0xFF);
```

```
158             delay();
159             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, GPIO_PIN_5*0x00);
160             delay();
161             }
162
163         else if((R7>ui32ADC0Value)&&(ui32ADC0Value>=R6))
164            {
165             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6, GPIO_PIN_6*0xFF);
166             delay();
167             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6, GPIO_PIN_6*0x00);
168             delay();
169            }
170         else if(ui32ADC0Value>R7)
171         {
172             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_7, GPIO_PIN_7*0xFF);
173             delay();
174             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_7, GPIO_PIN_7*0x00);
175             delay();
176         }
177     }
178 }
```

8 Lautstärkepegel

8.1 Lautstärkepegel

```
1  #include <stdint.h>
2  #include <stdbool.h>
3  #include "inc/hw_memmap.h"
4  #include "inc/hw_types.h"
5  #include "driverlib/sysctl.h"
6  #include "driverlib/adc.h"
7  #include "driverlib/gpio.h"
8  #include "driverlib/timer.h"
9
10 // Makros
11 #define FSAMPLE  44000
12 #define BUFFER_SIZE 1000
13
14 // globale Variable
15 int32_t buffer_sample[BUFFER_SIZE];      //Quadratische Signale
16 uint32_t i_sample = 0;
17 int32_t buffer_sample_sum = 0;           // momentaner Pegel
18 uint32_t index = 0;
19 uint32_t index_zuvor = 0;
20 // Prototypen
21 void ADC_int_handler(void);
22
```

```
23 int main(void)
24 {
25     // SystemClock konfigurieren
26     SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|↩
           SYSCTL_XTAL_16MHZ);
27     uint32_t ui32Period = SysCtlClockGet()/FSAMPLE;
28
29     // Peripherie aktivieren
30     SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
31     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
32     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
33     SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
34
35     // GPIO konfigurieren
36     GPIOPinTypeADC(GPIO_PORTE_BASE,GPIO_PIN_2);
37     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE,GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|↩
           GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7);
38
39     //Timer0 konfigurieren
40     TimerConfigure(TIMER0_BASE,TIMER_CFG_PERIODIC);
41     TimerLoadSet(TIMER0_BASE, TIMER_A, ui32Period - 1);
42     TimerControlTrigger(TIMER0_BASE,TIMER_A,true);
43     TimerEnable(TIMER0_BASE,TIMER_A);
44
45     // ADC konfigurieren
46     ADCClockConfigSet(ADC0_BASE,ADC_CLOCK_RATE_FULL,1);
47
48     ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_TIMER, 0);
49     ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_CH1|ADC_CTL_IE|↩
           ADC_CTL_END);
50     ADCSequenceEnable(ADC0_BASE, 3);
51     ADCIntClear(ADC0_BASE,3);
52     ADCIntRegister(ADC0_BASE,3,ADC_int_handler);
53     ADCIntEnable(ADC0_BASE,3);
54
55     while(1)
56     {
57     }
58 }
59 // Quadrierungsfunktion einer Variable x
60 int square(uint32_t x)                                          // ↩
       Quadrierfunktion
61 {
62    return x * x;
63 }
64 // Interrupt handler
65 void ADC_int_handler(void)
66 {
67     ADCIntClear(ADC0_BASE, 3);  // delete interrupt flag
```

```
68      ADCProcessorTrigger(ADC0_BASE,3);                                   // ↩
            Konvertierung beginnen
69
70      while(!ADCIntStatus(ADC0_BASE,3,false))                             // warten bis ↩
            Konvertierung abgeschlossen
71      {
72      }
73      uint32_t NewADCValue = 0;
74      ADCSequenceDataGet(ADC0_BASE,3,&NewADCValue);                       // Wert ↩
            einziehen
75      NewADCValue = square(NewADCValue);                                  //Wert ↩
            quadrieren
76
77      buffer_sample[index] = NewADCValue;                                 //↩
            speichert Werte im Vektor
78      buffer_sample_sum += NewADCValue - buffer_sample[index_zuvor];      //Summe↩
             der gemessenen Werte berechnen
79      index_zuvor= index;
80      index++;
81
82      if (index == BUFFER_SIZE)
83          {
84          index = 0;
85          }
86      //Grenzwerte bestimmt nach Test
87
88      if(buffer_sample_sum < 50000)
89              {
90              GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0| GPIO_PIN_1|GPIO_PIN_2|↩
                    GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6| GPIO_PIN_7, 0↩
                    b00000000);
91              }
92      if(buffer_sample_sum > 50000)
93          {
94          GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0| GPIO_PIN_1|GPIO_PIN_2|↩
                GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6| GPIO_PIN_7, 0b00000001↩
                );
95          }
96      if(buffer_sample_sum > 100000)
97          {
98          GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0| GPIO_PIN_1|GPIO_PIN_2|↩
                GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6| GPIO_PIN_7, 0b00000011↩
                );
99          }
100     if(buffer_sample_sum > 150000)
101         {
102         GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0| GPIO_PIN_1|GPIO_PIN_2|↩
                GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6| GPIO_PIN_7, 0b00000111↩
                );
103         }
```

```
104     if(buffer_sample_sum > 200000)
105         {
106         GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0| GPIO_PIN_1|GPIO_PIN_2|↩
                GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6| GPIO_PIN_7, 0b00001111↩
                );
107         }
108     if(buffer_sample_sum > 250000)
109         {
110         GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0| GPIO_PIN_1|GPIO_PIN_2|↩
                GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6| GPIO_PIN_7, 0b00011111↩
                );
111         }
112     if(buffer_sample_sum > 300000)
113         {
114         GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0| GPIO_PIN_1|GPIO_PIN_2|↩
                GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6| GPIO_PIN_7, 0b00111111↩
                );
115         }
116     if(buffer_sample_sum > 350000)
117         {
118         GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0| GPIO_PIN_1|GPIO_PIN_2|↩
                GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6| GPIO_PIN_7, 0b01111111↩
                );
119         }
120     if(buffer_sample_sum > 400000)
121         {
122         GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0| GPIO_PIN_1|GPIO_PIN_2|↩
                GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6| GPIO_PIN_7, 0b11111111↩
                );
123         }
124     }
```

8.2 Eigene Verbesserungen

Unser Vorschlag zu einer Verbesserung des bisherigen Codes besteht daraus den Spannungspegel einer Solarzelle auf Knopfdruck zu messen und mittels der LEDs wiederzugeben. Um dies umzusetzen, wird zuerst die Spannung mit dem Pin PE2 gemessen und der Code von der vorherigen Teilaufgabe benutzt. Es wird eine Zener-Diode verwendet um das Board zu schützen. Bei maximaler Beleuchtung gingen die LEDs aus. Unsere Idee war, dass die gemessene Spannung ein Overflow erzeugt, also wurde ein 1 MΩ Widerstand zwischen der Solarzelle und dem Pin eingebaut. Nach dieser Veränderung blieben die LEDs an bei maximaler beleuchtung. Bei geringer Spannung schlagen die LEDs schnell aus, also mussten die A/D Grenzen verändert werden. Nach einigen Versuchen wurde der maximale Pegel auf einen A/D-Wert von 2400000 gesetzt und für die anderen LEDs wurden in Werte in 300000 Schritten reduziert. Es wäre aber eine Energieverschwendung konstant den Spannungspegel zu messen. Deshalb wird zusätzlich ein Knopf aktiviert, welcher beim drücken den Spannungspegel

wiedergibt.

```
1  #include <stdint.h>
2  #include <stdbool.h>
3  #include "inc/hw_memmap.h"
4  #include "inc/hw_types.h"
5  #include "driverlib/sysctl.h"
6  #include "driverlib/adc.h"
7  #include "driverlib/gpio.h"
8  #include "driverlib/timer.h"
9
10 // Makros
11 #define FSAMPLE  44000
12 #define BUFFER_SIZE 1000
13
14 // globale Variable
15 int32_t buffer_sample[BUFFER_SIZE];      //Quadratische Signale
16 uint32_t i_sample = 0;
17 int32_t buffer_sample_sum = 0;           // momentaner Pegel
18 uint32_t index = 0;
19 uint32_t index_zuvor = 0;
20 // Prototypen
21 void ADC_int_handler(void);
22
23 int main(void)
24 {
25     // SystemClock konfigurieren
26     SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|↩
           SYSCTL_XTAL_16MHZ);
27     uint32_t ui32Period = SysCtlClockGet()/FSAMPLE;
28
29     // Peripherie aktivieren
30     SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
31     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
32     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
33     SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
34
35     // GPIO konfigurieren
36     GPIOPinTypeADC(GPIO_PORTE_BASE,GPIO_PIN_2);
37     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE,GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|↩
           GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7);
38
39     //Timer0 konfigurieren
40     TimerConfigure(TIMER0_BASE,TIMER_CFG_PERIODIC);
41     TimerLoadSet(TIMER0_BASE, TIMER_A, ui32Period - 1);
42     TimerControlTrigger(TIMER0_BASE,TIMER_A,true);
43     TimerEnable(TIMER0_BASE,TIMER_A);
44
45     // ADC konfigurieren
46     ADCClockConfigSet(ADC0_BASE,ADC_CLOCK_RATE_FULL,1);
47
```

```
48      ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_TIMER, 0);
49      ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_CH1|ADC_CTL_IE|↩
            ADC_CTL_END);
50      ADCSequenceEnable(ADC0_BASE, 3);
51      ADCIntClear(ADC0_BASE,3);
52      ADCIntRegister(ADC0_BASE,3,ADC_int_handler);
53      ADCIntEnable(ADC0_BASE,3);
54
55      //Schalter Config
56      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);                    // Port F ↩
            Aktivieren
57      GPIOPinTypeGPIOInput(GPIO_PORTF_BASE,GPIO_PIN_4);               // PF4 als ↩
            Input definieren
58      GPIOPadConfigSet(GPIO_PORTF_BASE,GPIO_PIN_4,GPIO_STRENGTH_2MA,↩
            GPIO_PIN_TYPE_STD_WPU);
59
60      while(1)
61      {
62      }
63  }
64  // Quadrierungsfunktion einer Variable x
65  int square(uint32_t x)                                              // ↩
        Quadrierfunktion
66  {
67     return x * x;
68  }
69  // Interrupt handler
70  void ADC_int_handler(void)
71  {
72      ADCIntClear(ADC0_BASE, 3);  // delete interrupt flag
73      ADCProcessorTrigger(ADC0_BASE,3);                               // ↩
            Konvertierung beginnen
74
75      while(!ADCIntStatus(ADC0_BASE,3,false))                         // warten bis ↩
            Konvertierung abgeschlossen
76      {
77      }
78      uint32_t NewADCValue = 0;
79      ADCSequenceDataGet(ADC0_BASE,3,&NewADCValue);                   // Wert ↩
            einziehen
80      NewADCValue = square(NewADCValue);                              //Wert ↩
            quadrieren
81
82      buffer_sample[index] = NewADCValue;                                 //↩
            speichert Werte im Vektor
83      buffer_sample_sum += NewADCValue - buffer_sample[index_zuvor] ;     //Summe↩
             der gemessenen Werte berechnen
84      index_zuvor= index;
85      index++ ;
86
```

```
87      if (index == BUFFER_SIZE )
88          {
89          index = 0;
90          }
91      //Grenzwerte bestimmt nach Test
92      if(GPIOPinRead(GPIO_PORTF_BASE,GPIO_PIN_4)==0) {
93
94      if(buffer_sample_sum < 300000)
95              {
96              GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0| GPIO_PIN_1|GPIO_PIN_2|↩
                    GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6| GPIO_PIN_7, 0↩
                    b00000000);
97              }
98      if(buffer_sample_sum > 300000)
99          {
100         GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0| GPIO_PIN_1|GPIO_PIN_2|↩
                GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6| GPIO_PIN_7, 0b00000001↩
                );
101         }
102     if(buffer_sample_sum > 600000)
103         {
104         GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0| GPIO_PIN_1|GPIO_PIN_2|↩
                GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6| GPIO_PIN_7, 0b00000011↩
                );
105         }
106     if(buffer_sample_sum > 900000)
107         {
108         GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0| GPIO_PIN_1|GPIO_PIN_2|↩
                GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6| GPIO_PIN_7, 0b00000111↩
                );
109         }
110     if(buffer_sample_sum > 1200000)
111         {
112         GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0| GPIO_PIN_1|GPIO_PIN_2|↩
                GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6| GPIO_PIN_7, 0b00001111↩
                );
113         }
114     if(buffer_sample_sum > 1500000)
115         {
116         GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0| GPIO_PIN_1|GPIO_PIN_2|↩
                GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6| GPIO_PIN_7, 0b00011111↩
                );
117         }
118     if(buffer_sample_sum > 1800000)
119         {
120         GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0| GPIO_PIN_1|GPIO_PIN_2|↩
                GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6| GPIO_PIN_7, 0b00111111↩
                );
121         }
122     if(buffer_sample_sum > 2100000)
```

```
123             {
124             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0| GPIO_PIN_1|GPIO_PIN_2|↩
                    GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6| GPIO_PIN_7, 0b01111111↩
                    );
125             }
126         if(buffer_sample_sum > 2400000)
127             {
128             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0| GPIO_PIN_1|GPIO_PIN_2|↩
                    GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6| GPIO_PIN_7, 0b11111111↩
                    );
129             }
130         }
131         else
132         {
133             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0| GPIO_PIN_1|GPIO_PIN_2|↩
                    GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6| GPIO_PIN_7, 0b00000000↩
                    );
134         }
135 }
```