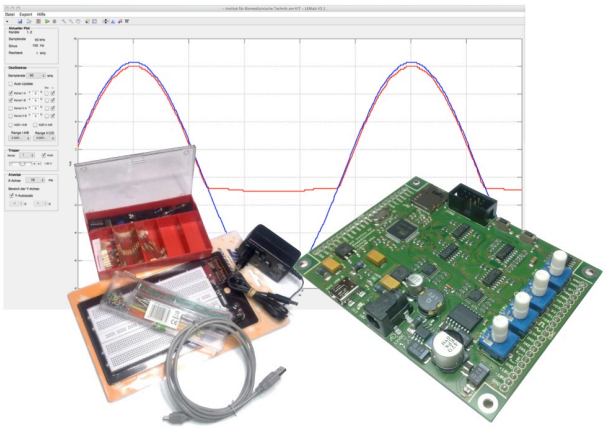
Karlsruher Institut für Technologie

Workshop

Elektrotechnik und Informationstechnik

Kurs 4

Digitale Signalverarbeitung



Gruppe 55

Vorname	Nachname	Matrikel-Nr.	u-Account	E-Mail
Jonathan	Hufgard	2119471	uuqaq	uuqaq@student.kit.edu
Annika	Lang	2158918	ufvev	ufvev@student.kit.edu
Tobias	Schneider	2119039	uyoas	uyoas@student.kit.edu

20. November 2018

Abstract

Um Signale mit einem digitalen Rechner, egal ob Mikrocontroller oder vollwertiger PC, verarbeiten zu können, müssen analoge physikalische Signale in eine für den Rechner verständliche digitale Form gebracht werden. Umgekehrt können digitale Signale eines Rechners auch in analoge zeitkontinuierliche Signale gewandelt werden. Die Schnittstelle sind hierbei die Ein- und Ausgänge des Tiva LaunchPads.

Ziel dieses Kurses ist, die Funktionsweise der Signalverarbeitung eines Mikrocontrollers kennenzulernen und dabei den Konvertierungsvorgang von analogen zeit- und wertkontinuierlichen Signalen in digitale Signale und umgekehrt zu untersuchen. Hierfür werden die auf dem Mikrocontroller des LaunchPads vorinstallierte Software durch ein eigenes Programm ersetzt, welches ermöglicht, das Board auch ohne PC für verschiedenste Anwendungen zu nutzen. Als Startpunkt für weitere eigene Projekte mit dem LaunchPad sollen in diesem Kurs mit Hilfe des Mikrocontrollers am Beispiel von Musik Signale digitalisiert, verarbeitet und visualisiert werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

1.1 Interrupts

Programme können kurzzeitig unterbrochen werden, um andere Programme aufzurufen. Eine Möglichkeit hierfür ist das Interrupt, bei dem im Gegensatz zum Polling nicht ständig abgefragt werden muss, ob Daten anstehen. Wenn zum Beispiel eine Tastatureingabe erfolgt, wird dem Prozessor ein Interrupt Request gesendet. Der unterbricht dann seinen aktuellen Prozess und führt die Eingabe aus.

Interrupts werden vor allem bei zeitkritischen Anwendungen verwendet und müssen zur optimalen Funktionsweise eines Systems unterschiedlich gewichtet werden. So wird gewährleistet, dass die CPU optimal arbeitet. [1]

1.2 Diskretisierung

In $10\,\mathrm{s}$ werden bei einer Abtastfrequenz von $44{,}1\,\mathrm{kHz}$ insgesamt 441000 Werte eines Musiksignals erfasst. Bei einer Auflösung von 16 bit werden dafür 7056000 bit bzw. 882 kB Speicherplatz benötigt.

$$10\,\mathrm{s} \cdot 44{,}1\,\mathrm{kHz} \cdot 16\,\mathrm{bit} = 7\,056\,000\,\mathrm{bit} = 882\,\mathrm{kB}$$

2 Mikrocontrollerprogrammierung

2.1 RGB LED

Im in der Aufgabenstellung gegebenen LED-Blink-Programm wird mit 0x04 das zweite Bit im Register GPIO Data gesetzt und so die blaue LED auf PF2 angesteuert. Dieser Hexadezimalwert kann durch den Binärwert 0b00000100 ersetzt werden.

Um die grüne LED auf PF3 anzusteuern, muss also das dritte Bit gesetzt werden. Dazu wird 0b00001000 = 0x08 in das GPIO Data Register geschrieben.

Zum Ansteuern beider LEDs wird sowohl das zweite als auch das dritte Bit gesetzt. Dazu wird 0b00001100 = 0x0C verwendet.

Für die Ausführung des Programms macht es keinen Unterschied, ob die Werte binär oder hexadezimal eingetragen werden. Bei Binärwerten ist zwar direkt ersichtlich, welche Bits angesteuert werden sollen, im Hexadezimalsystem werden dagegen weniger Zeichen benötigt, weshalb sich gerade große Zahlen übersichtlicher darstellen lassen.

2.2 GPIO-Blinklicht

Mithilfe des TivaWare-Frameworks lässt sich der gegebene Quellcode für das Blinken der blauen LED wie in Quellcode 1 vereinfachen.

Quellcode 1: GPIO-Blinklicht in blau

```
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  // Warteschleife
9  void delay(void)
10 {
11     uint32_t i=50000;
12     while(i) {i--;}
13 }
14
15 int main(void)
16 {
17     // Peripherie Port B aktivieren
18     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
19
20     // setze Pin als Ausgang und Digital
21     GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,GPIO_PIN_2);
22
23     while(1) {
24
25         // Setze Pin auf High Pegel
26         GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_2, 0xFF);
27         delay();
28
29         // Setze Pin auf Low Pegel
30         GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_2, 0x00);
31         delay();
32     }
33 }
```

Für weißes Licht muss neben der bereits verwendeten blauen und grünen LED die rote leuchten. Im Schaltplan ist zu erkennen, dass für die rote LED PF1 verwendet wird. Das Programm in Quellcode 2 steuert PF1 - PF3 an.

Quellcode 2: GPIO-Blinklicht in weiß

```
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  // Warteschleife
9  void delay(void)
10 {
11     uint32_t i=50000;
12     while(i) {i--;}
13 }
14
15 int main(void)
16 {
17     // Peripherie Port B aktivieren
18     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
19
20     // setze Pin als Ausgang und Digital
21     GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1 + ↩
          GPIO_PIN_2 + GPIO_PIN_3);
22
23     while(1) {
24
25         // Setze Pin auf High Pegel
26         GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 + ↩
              GPIO_PIN_2 + GPIO_PIN_3, 0xFF);
27         delay();
28
29         // Setze Pin auf Low Pegel
30         GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 + ↩
              GPIO_PIN_2 + GPIO_PIN_3, 0x00);
31         delay();
32     }
33 }
```

2.3 Externe LED Schaltung

Für die Ansteuerung einer externen LED Schaltung wird Quellcode 3 erstellt.

Quellcode 3: Externe LED Schaltung mit Blinken

```
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
9  // Warteschleife
10 void delay(void)
11 {
12     uint32_t i=50000;
13     while(i) {i--;}
14 }
15
16 int main(void)
17 {
18     uint32_t GPIO_PIN_0TO7 = GPIO_PIN_0 + GPIO_PIN_1 + ↩
          GPIO_PIN_2 + GPIO_PIN_3 + GPIO_PIN_4 + GPIO_PIN_5 + ↩
          GPIO_PIN_6 + GPIO_PIN_7;
19     // Peripherie Port B aktivieren
20     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
21
22     // setze Pin als Ausgang und Digital
23     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_0TO7 );
24
25     while(1) {
26
27         // Setze Pin auf High Pegel
28         GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0TO7, 0xFF);
29         delay();
30
31         // Setze Pin auf Low Pegel
32         GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0TO7, 0x00);
33         delay();
34     }
35 }
```

Damit die LEDs konstant mit halber Intensität leuchten, muss die Dauer der einzelnen Zustände verkürzt werden, sodass statt eines Blinklichtes ein konstantes Licht wahrgenommen wird. In Quellcode 4 wurde die bisherige delay-Funktion so verändert, dass statt von 50000 nur von 100 herunter gezählt wird. Die Verzögerung wird nicht komplett weggelassen, um mögliche Ungenauigkeiten beim Wechsel zwischen den Zuständen zu kompensieren. Für die halbe Intensität der LEDs muss nichts verändert werden, da dafür wie bisher die LED gleich lang an- wie ausgeschaltet sein muss.

Quellcode 4: Externe LED Schaltung mit konstantem Leuchten

```
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
9  // Warteschleife
10 void delay(void)
11 {
12     uint32_t i=100;
13     while(i) {i--;}
14 }
15
16 int main(void)
17 {
18     uint32_t GPIO_PIN_0TO7 = GPIO_PIN_0 + GPIO_PIN_1 + ↩
          GPIO_PIN_2 + GPIO_PIN_3 + GPIO_PIN_4 + GPIO_PIN_5 + ↩
          GPIO_PIN_6 + GPIO_PIN_7;
19     // Peripherie Port B aktivieren
20     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
21
22     // setze Pin als Ausgang und Digital
23     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_0TO7 );
24
25     while(1) {
26
27         // Setze Pin auf High Pegel
28         GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0TO7, 0xFF);
29         delay();
30
31         // Setze Pin auf Low Pegel
```

```
32            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0TO7, 0x00);
33            delay();
34        }
35    }
```

2.4 Warteschleifen und Taktfrequenz

Im Quellcode zur Erzeugung einer System-Clock aus der Aufgabenstellung ist zu sehen, dass als Divisor für die Taktfrequenz 5 verwendet wird. Unter Berücksichtigung des internen Halbierens beträgt die Taktfrequenz des Mikrocontrollers

$$\frac{400\,\text{MHz}}{5 \cdot 2} = 40\,\text{MHz}$$

Die Funktion delay_ms in Quellcode 5 bewirkt eine Verzögerung um den Eingabeparameter waitTime. Um die LEDs für weißes Licht jede halbe Sekunde blinken zu lassen, müssen beide Zustände durch die delay_ms-Funktion je 250 ms gehalten werden.

Quellcode 5: GPIO-Blinklicht in weiß mit einer Frequenz von 2 Hz

```
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 uint32_t systemTime_ms;
13 void InterruptHandlerTimer0A (void)
14 {
15     // Clear the timer interrupt to prevent the interrupt ↩
       function from immediately being called again on exit
16     TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
17     // count up one ms
18     systemTime_ms++;
19 }
20 void clockSetup(void)
21 {
22     uint32_t timerPeriod;
```

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

```
59
60      systemTime_ms = 0;
61      clockSetup();
62      while(1)
63      {
64          // Setze Pin auf High Pegel
65          GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_2, 0xFF);
66          delay_ms(250);
67
68          // Setze Pin auf Low Pegel
69          GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_2, 0x00);
70          delay_ms(250);
71      }
72      return 1;
73  }
```

Durch schrittweises Erhöhen der Frequenz kann mit diesem Programm die Flimmerverschmelzungsfrequenz der Gruppenmitglieder herausgefunden werden. In diesem Versuch wurden die Lichtblitze ab einer waitTime von 11 ms als kontinuierliches Licht wahrgenommen. Das entspricht einer Flimmerverschmelzungsfrequenz von

$$\frac{1}{2 \cdot 11\,\text{ms}}$$

2.5 Taster

Bei der Ausführung des Programms in Quellcode 6 leuchtet die LED solange wie sich zwei Drähte an PC7 und an einem GND-Pin berühren.

Quellcode 6: Schalter durch Berührung zweier Drähte

```
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   int main(void) {
9       // Initialisierung der Peripherie
10      // Peripherie Port F und C aktivieren
11      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
12      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
13      // setze Pin als Ausgang und Digital
14      GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_2 );
```

```
15
16      // Taster als Eingang mit Pull-up Widerstand schalten
17      GPIOPinTypeGPIOInput(GPIO_PORTC_BASE,GPIO_PIN_7);
18      GPIOPadConfigSet(GPIO_PORTC_BASE,GPIO_PIN_7,↩
            GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);
19
20      // Auswertung in Schleife
21      // Taster gedrueckt
22      while(1){
23          if(GPIOPinRead(GPIO_PORTC_BASE,GPIO_PIN_7)==0) {
24              // auf Tastendruck reagieren
25              // Setze Pin auf High Pegel
26              GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0xFF);
27              while(GPIOPinRead(GPIO_PORTC_BASE,GPIO_PIN_7)↩
                    ==0);
28          }
29
30              // auf Öffnen des Tasters reagieren
31          GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0x00);
32
33      }
34  }
```

Das Programm in Quellcode 7 schaltet die LED bei einer Berührung der beiden Drähte an- bzw. aus.

Quellcode 7: Taster durch Berührung zweier Drähte

```
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   int main(void) {
9       // Initialisierung der Peripherie
10      // Peripherie Port F und C aktivieren
11      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
12      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
13      // setze Pin als Ausgang und Digital
14      GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_2 );
15
```

```
16      // Taster als Eingang mit Pull-up Widerstand schalten
17      GPIOPinTypeGPIOInput(GPIO_PORTC_BASE,GPIO_PIN_7);
18      GPIOPadConfigSet(GPIO_PORTC_BASE,GPIO_PIN_7,↩
           GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);
19
20      // Auswertung in Schleife
21      // Taster gedrueckt
22      while(1){
23          if(GPIOPinRead(GPIO_PORTC_BASE,GPIO_PIN_7)==0) {
24              // auf Tastendruck reagieren
25              // Zustand ändern
26              if(GPIOPinRead(GPIO_PORTF_BASE,GPIO_PIN_2)==0){
27                  GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0↩
                       xFF);
28              }
29              else{
30                  GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0↩
                       x00);
31              }
32
33              while(GPIOPinRead(GPIO_PORTC_BASE,GPIO_PIN_7)↩
                   ==0);
34          }
35      }
36  }
```

Quellcode 8 ist so verändert, dass statt einer Berührung der Drähte an PC7 und am GND-Pin der Taster SW1 an PF4 für das An- und Ausschalten verwendet wird.

Quellcode 8: Taster durch Betätigung von SW1

```
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   int main(void) {
9       // Initialisierung der Peripherie
10      // Peripherie Port F und C aktivieren
11      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
12      // setze Pin als Ausgang und Digital
```

```
13      GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_2 );
14
15      // Taster als Eingang mit Pull-up Widerstand schalten
16      GPIOPinTypeGPIOInput(GPIO_PORTF_BASE,GPIO_PIN_4);
17      GPIOPadConfigSet(GPIO_PORTF_BASE,GPIO_PIN_4,↩
           GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);
18
19      // Auswertung in Schleife
20      // Taster gedrueckt
21      while(1){
22          if(GPIOPinRead(GPIO_PORTF_BASE,GPIO_PIN_4)==0) {
23              // auf Tastendruck reagieren
24              // Zustand ändern
25              if(GPIOPinRead(GPIO_PORTF_BASE,GPIO_PIN_2)==0){
26                  GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0↩
                       xFF);
27              }
28              else{
29                  GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0↩
                       x00);
30              }
31
32              while(GPIOPinRead(GPIO_PORTF_BASE,GPIO_PIN_4)↩
                   ==0);
33          }
34      }
35  }
```

2.6 Externer Interrupt

In Quellcode 9 ist ein Programm zu sehen, bei dem eine LED blinkt und bei Betätigen des Tasters SW1 auf die nächste LED in der Schaltung gewechselt wird. Nach der letzten LED blinkt wieder die erste. Während vorher ständig abgefragt wurde, ob der Taster gedrückt ist, wird nun mit einem Interrupt gearbeitet.

Quellcode 9: Externer Interrupt

```
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  int main(void) {
9      // Initialisierung der Peripherie
10     // Peripherie Port F und C aktivieren
11     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
12     // setze Pin als Ausgang und Digital
13     GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_2 );
14
15     // Taster als Eingang mit Pull-up Widerstand schalten
16     GPIOPinTypeGPIOInput(GPIO_PORTF_BASE,GPIO_PIN_4);
17     GPIOPadConfigSet(GPIO_PORTF_BASE,GPIO_PIN_4,↩
          GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);
18
19     // Auswertung in Schleife
20     // Taster gedrueckt
21     while(1){
22         if(GPIOPinRead(GPIO_PORTF_BASE,GPIO_PIN_4)==0) {
23             // auf Tastendruck reagieren
24             // Zustand ändern
25             if(GPIOPinRead(GPIO_PORTF_BASE,GPIO_PIN_2)==0){
26                 GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0↩
                      xFF);
27             }
28             else{
29                 GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0↩
                      x00);
30             }
31
32             while(GPIOPinRead(GPIO_PORTF_BASE,GPIO_PIN_4)↩
                  ==0);
33         }
34     }
35 }
```

2.7 A/D-Umsetzer

Im gegebenen Spannungsteiler werden für R_{ADC} Werte gewählt und die zugehörige Spannung U_{ADC} berechnet.

$$U_{ADC} = 3{,}3\,\mathrm{V} \cdot \frac{R_{ADC}}{1\,\mathrm{k\Omega} + R_{ADC}} \tag{1}$$

Mit dem 12-bit-ADC des Mikrocontrollers soll diese Spannung gemessen werden. Der maximale ADC-Wert 4095 entspricht der maximalen Spannung $3{,}3\,\mathrm{V}$. Hiermit lässt sich für jede Spannung U_{ADC} über

$$Wert_{ADC} = \frac{U_{ADC}}{3{,}3\,\mathrm{V}} \cdot 4095 \tag{2}$$

der erwartete Wert $Wert_{ADC}$ berechnen.

Tabelle 1: Erwartete Werte bei der Widerstandsmessung von R_{ADC}

eingesetzter R_{ADC} in kΩ	1,5	4,7	6,8	10	11,5	14,7	16,8	20
erwartete U_{ADC} in V durch (1)	1,98	2,76	2,88	3,00	3,04	3,09	3,11	3,14
erwarteter $Wert_{ADC}$ durch (2)	2457	3424	3573	3722	3772	3834	3859	3896

2.8 Widerstandsmessung

Mit dem Programm in Quellcode 10 soll zunächst mit dem ADC die Spannung U_{ADC} gemessen werden. Dafür müssen die vom ADC ausgelesenen Werte in Spannungswerte umgerechnet werden. Formel (2) wird dafür folgendermaßen umgeformt.

$$U_{ADC} = \frac{Wert_{ADC}}{4095} \cdot 3{,}3\,\mathrm{V} \tag{3}$$

Durch Kombination der Formeln (1) und (3) kann mit folgender Formel auf den Wert des für R_{ADC} eingesetzten Widerstandes geschlossen werden.

$$R_{ADC} = \frac{1\,\mathrm{k\Omega}}{\frac{4095}{Wert_{ADC}} - 1} \tag{4}$$

Tabelle 2: Gemessene Werte bei der Widerstandsmessung von R_{ADC}

eingesetzter R_{ADC} in kΩ	1,5	4,7	6,8	10	11,5	14,7	16,8	20
gemessene U_{ADC} in V durch (3)	1,99	2,71	2,86	2,99	3,02	3,07	3,10	3,13
gemessener R_{ADC} in kΩ durch (4)	1,5	4,6	6,5	9,6	10,8	13,4	15,5	18,4

Die visuelle Ausgabe erfolgt über 8 LEDs, denen Widerstandsbereiche zugeordnet werden, die in Tabelle 3 angegeben sind. Die LED mit dem passenden Widerstandsbereich beginnt bei erfolgreicher Messung zu blinken.

Tabelle 3: Widerstands- und zugehörige ADC-Bereiche

blinkende LED	Bereich von R_{ADC} in kΩ	Bereich von $Wert_{ADC}$
1	0 bis 2,5	0 bis 2925
2	2,5 bis 5	2925 bis 3412
3	5 bis 7,5	3412 bis 3613
4	7,5 bis 10	3613 bis 3722
5	10 bis 12,5	3722 bis 3791
6	12,5 bis 15	3791 bis 3839
7	15 bis 17,5	3839 bis 3873
8	17,5 bis 20	3873 bis 3900

Quellcode 10: Widerstandsmessung

```
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  #include"driverlib/adc.h"
8  #include"inc/hw_adc.h"
9
10 uint32_t GPIO_PIN_0TO7 = GPIO_PIN_0 + GPIO_PIN_1 + ↩
      GPIO_PIN_2 + GPIO_PIN_3 + GPIO_PIN_4 + GPIO_PIN_5 + ↩
      GPIO_PIN_6 + GPIO_PIN_7;
11 float widerstand = 1;
12 int anzahlLEDs = 8;
13 int bereich = 0;
14 float vorwiderstand = 1000;
15 int pin;
16
17 void main(void){
18     // Peripherie ADC aktivieren
19     SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
20     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
21
22     // Peripherie aktivieren
23     // Peripherie Port B, C und F aktivieren
```

```
24      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
25      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
26      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
27      // setze Pins als Ausgang und Digital
28      GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_4 );
29      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_0TO7 );
30      // Taster als Eingang mit Pull-up Widerstand schalten
31      GPIOPinTypeGPIOInput(GPIO_PORTF_BASE,GPIO_PIN_4);
32      GPIOPadConfigSet(GPIO_PORTF_BASE,GPIO_PIN_4,↩
           GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);
33
34      // Taster mit Pullup
35      GPIOPinTypeGPIOInput(GPIO_PORTC_BASE,GPIO_PIN_7);
36      GPIOPadConfigSet(GPIO_PORTC_BASE, GPIO_PIN_7, ↩
           GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);
37
38      // PIN PE2 ADC Funktion zuweisen
39      GPIOPinTypeADC(GPIO_PORTE_BASE,GPIO_PIN_2);
40      // ADC konfigurieren
41      ADCSequenceConfigure(ADC0_BASE,1,ADC_TRIGGER_PROCESSOR↩
           ,0); // Prozessor als Trigger Quelle
42      ADCSequenceStepConfigure(ADC0_BASE,1,0,ADC_CTL_CH1|↩
           ADC_CTL_IE| ADC_CTL_END); // AI1 abtasten/Interrupt ↩
           erzeugen bei Ende/letzter Schritt
43      ADCSequenceEnable(ADC0_BASE,1); // ADC Sequenz 1 ↩
           aktivieren
44
45      uint32_t ui32ADC0Value;
46      while(1){
47          // Eingang abfragen
48          ADCIntClear(ADC0_BASE,1); // evtl vorhandene ADC ↩
               Interrupts loeschen
49          ADCProcessorTrigger(ADC0_BASE,1); // Konvertierung ↩
               beginnen
50          while(!ADCIntStatus(ADC0_BASE,1,false)); // warten ↩
               bis Konvertierung abgeschlossen
51          ADCSequenceDataGet(ADC0_BASE,1,&ui32ADC0Value); // ↩
               Wert auslesen
52          //Widerstand bestimmen
53          widerstand = vorwiderstand/((4096.0/ui32ADC0Value)↩
               -1);
54          //cout >> widerstand;
```

```
55          //in welchem Bereich Widerstand liegt
56          bereich = widerstand/ (20000/anzahlLEDs);
57          if(bereich <7){
58              pin = 0b1 << bereich;
59          }
60          else{
61              pin = 0b10000000;
62          }
63          //LED Ansteuerung
64          GPIOPinWrite(GPIO_PORTB_BASE,pin, 0xFF);
65          GPIOPinWrite(GPIO_PORTB_BASE,0xFF-pin, 0x00);
66      }
67  }
```

2.9 Lautstärkepegel

In Quellcode 11 ist ein Programm zu sehen, das ein externes Musiksignal über den ADC wandelt, im Mikrocontroller verarbeitet und passend zur Musik LEDs ansteuert. Die Schaltung zur Vorverarbeitung des Audiosignals wird wie in der Aufgabenstellung angegeben aufgebaut. Das vorgegebene Programmgerüst löst mit einer Frequenz von 44 kHz Interrupts aus. In der Interruptroutine werden ADC-Werte ausgelesen, quadriert und in buffer_sample gespeichert. In buffer_sample_sum ist die Summe der letzten 1000 Energiewerte gespeichert, die inkrementell bei jedem neuen Wert angepasst wird. MaximalerWert enthält den größten Wert in buffer_sample seit Start des Programms. Den 8 LEDs werden stufenweise Werte von 0 bis MaximalerWert zugeordnet um die Lautstärke des Musiksignals zu visualisieren.

Quellcode 11: 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;
18 uint32_t MaximalerWert = 0;
19 int32_t  anzahlLEDs = 8;
20 int32_t bereich = 0;
21 // momentaner Pegel
22
23 // Prototypen
24 void ADC_int_handler(void);
25
26 int main(void)
27 {
28     // SystemClock konfigurieren
29     SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|↩
          SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);
30     uint32_t ui32Period = SysCtlClockGet()/FSAMPLE;
31
32     // Peripherie aktivieren
33     SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
34     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
35     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
36     SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
37
38     // GPIO konfigurieren
39     GPIOPinTypeADC(GPIO_PORTE_BASE,GPIO_PIN_2);
40     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);
41
42     //Timer0 konfigurieren
43     TimerConfigure(TIMER0_BASE,TIMER_CFG_PERIODIC);
44     TimerLoadSet(TIMER0_BASE, TIMER_A, ui32Period - 1);
45     TimerControlTrigger(TIMER0_BASE,TIMER_A,true);
46     TimerEnable(TIMER0_BASE,TIMER_A);
47
48     // ADC konfigurieren
49     ADCClockConfigSet(ADC0_BASE,ADC_CLOCK_RATE_FULL,1);
50
51     ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_TIMER, 0)↩
          ;
```

```
52      ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_CH1|↩
           ADC_CTL_IE|ADC_CTL_END);
53      ADCSequenceEnable(ADC0_BASE, 3);
54      ADCIntClear(ADC0_BASE,3);
55      ADCIntRegister(ADC0_BASE,3,ADC_int_handler);
56      ADCIntEnable(ADC0_BASE,3);
57
58      while(1)
59      {
60      }
61  }
62
63  // Interrupt handler
64  void ADC_int_handler(void)
65  {
66      uint32_t ui32ADC0Value;
67      //uint32_t i = 0;
68
69      ADCIntClear(ADC0_BASE, 3);  // delete interrupt flag
70      ADCProcessorTrigger(ADC0_BASE,3); // Konvertierung ↩
           beginnen
71      while(!ADCIntStatus(ADC0_BASE,3,false)); // warten bis ↩
           Konvertierung abgeschlossen
72      ADCSequenceDataGet(ADC0_BASE,3,&ui32ADC0Value); // Wert ↩
           auslesen
73
74      // vorherigen Wert subtrahieren
75      buffer_sample_sum -= buffer_sample[i_sample];
76      // vorherigen Wert mit neuem überschreiben
77      buffer_sample[i_sample] = ui32ADC0Value^2;
78      // neuen Wert addieren
79      buffer_sample_sum += buffer_sample[i_sample];
80      // MaximalerWert anpassen
81      if(MaximalerWert < buffer_sample_sum){
82          MaximalerWert = buffer_sample_sum;
83      }
84
85      if(i_sample < BUFFER_SIZE-1 ){
86          i_sample++;
87      }
88      else{
89          i_sample = 0;
```

```
90      }
91
92      bereich =  (buffer_sample_sum) / (MaximalerWert/↩
           anzahlLEDs);
93      int pin = (1 << bereich)-1;
94      GPIOPinWrite(GPIO_PORTB_BASE,pin, 0xFF);
95      GPIOPinWrite(GPIO_PORTB_BASE,0xFF-pin, 0x00);
96  }
```

2.10 Eigene Verbesserungen

In Quellcode 12 ist die Umsetzung einer Geschwindigkeitsmessung zu sehen, die sich an einer Geschwindigkeitsanzeige aus dem Straßenverkehr orientiert. Mithilfe zweier Lichtschranken wird die Geschwindigkeit eines sich bewegenden Gegenstandes gemessen. Bis zu einer vorgegebenen Maximalgeschwindigkeit v_max leuchten ähnlich zur vorherigen Aufgabe zwischen 0 und 6 LEDs. Überschreitet der Körper die Maximalgeschwindigkeit, so beginnen eine gelbe und eine rote LED zu blinken. Als Grundgerüst wird der Code aus der GPIO-Blinklicht-Aufgabe verwendet. Das ständige Blinken wird jedoch durch Geschwindigkeitsmessung und Ansteuerung der LEDs ersetzt.

Quellcode 12: Geschwindigkeitsmessung

```
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 //Anpassungsvariablen
12 uint32_t delta_s_cm = 10;
13 uint32_t v_max = 20;
14
15 //stores ms since startup
16 uint32_t systemTime_ms;
17 void InterruptHandlerTimer0A (void)
18 {
19     // Clear the timer interrupt to prevent the interrupt ↩
       function from immediately being called again on exit
```

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

```
56 int main(void)
57 {
58     // Peripherie Port B und C aktivieren
59     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
60     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
61
62     // Lichtschranken als Eingang mit Pull-up Widerstand ↩
          schalten
63     GPIOPinTypeGPIOInput(GPIO_PORTC_BASE,GPIO_PIN_6|↩
          GPIO_PIN_7);
64     GPIOPadConfigSet(GPIO_PORTC_BASE,GPIO_PIN_6,↩
          GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);
65     GPIOPadConfigSet(GPIO_PORTC_BASE,GPIO_PIN_7,↩
          GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);
66
67     // setze Pin als Ausgang und Digital
68     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);
69
70     systemTime_ms = 0;
71     clockSetup();
72     while(1)
73     {
74         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, 0x00);
75         //Sobald erste Lichtschranke unterbrochen ist
76         if(GPIOPinRead(GPIO_PORTC_BASE,GPIO_PIN_6)==0){
77             uint32_t start = systemTime_ms;
78             //warten solang zweite Lichtschranke offen ist
79             while(GPIOPinRead(GPIO_PORTC_BASE,GPIO_PIN_7)↩
                  !=0){
80             }
81             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, 0xFF);
82             uint32_t ende = systemTime_ms;
83             //Werte
84             uint32_t delta_t_ms = ende - start;
85
86
```

```
87              //speed in Meter pro Sekunde
88              uint32_t speed_cm_s = (delta_s_cm) / (delta_t_ms↩
                    / 1000.0);
89
90              //Ansteuerung
91              if(speed_cm_s > v_max){
92                  GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0|↩
                       GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3|↩
                       GPIO_PIN_4|GPIO_PIN_5, 0x00);
93                  uint32_t b = 1;
94                  for(b; b<=5; b++) {
95
96                          // Setze Pin auf High Pegel
97                          GPIOPinWrite(GPIO_PORTB_BASE, ↩
                             GPIO_PIN_6|GPIO_PIN_7, 0xFF);
98                          delay_ms(100);
99
100                         // Setze Pin auf Low Pegel
101                         GPIOPinWrite(GPIO_PORTB_BASE, ↩
                             GPIO_PIN_6|GPIO_PIN_7, 0x00);
102                         delay_ms(100);
103                     }
104
105             }
106             else{
107                 uint32_t bereich = speed_cm_s / (v_max / ↩
                       6.0);
108                 uint32_t pin = (0b1 << bereich) - 1;
109
110                 GPIOPinWrite(GPIO_PORTB_BASE, 0xFF - pin, 0↩
                       x00);
111                 GPIOPinWrite(GPIO_PORTB_BASE, pin, 0xFF);
112                 delay_ms(2000);
113             }
114         }
115     }
116 }
```

Literaturverzeichnis

[1] https://www.elektronik-kompendium.de/sites/com/0610151.htm, Abrufdatum: 8. November 2018.