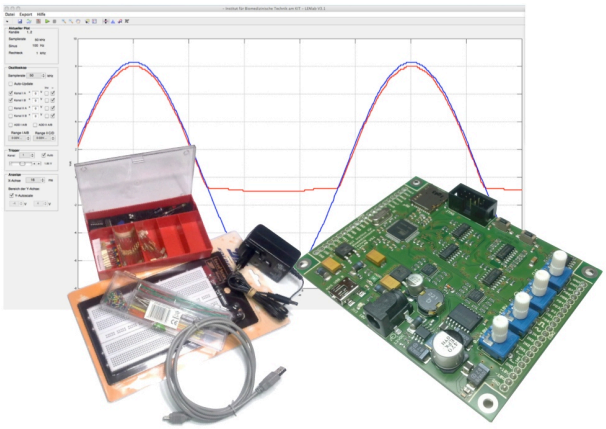
Karlsruher Institut für Technologie

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

17. 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
17 int main(void){
18     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);          //Port F aktivieren
19     GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1);    //Pin 1 bis 3 ↩
           initialisieren
20     GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_2);
21     GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_3);
22
23    while(1)       //unendliche Schleife fuer flimmern der LED in weiss
24    {
25        delay();
26     GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0xFF);
27     GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0xFF);
28     GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0xFF);
29         delay();
30     GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0x00);
31     GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0x00);
32     GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0x00);
33
34
35    }
36     return 0;
37 }
```

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
18 int main(void){
19      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);                    // GPIO Port B ↩
            aktivieren
20      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_0);             // GPIO Port B ↩
            Pin 0-7 aktivieren
21      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_1);
22      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_2);
23      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_3);
24      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_4);
25      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_5);
26      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_6);
27      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_7);
28
29     while(1)
30     {
31         delay();
32      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0, 0xFF);                // GPIO Pins ↩
            auf High stellen
33      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, 0xFF);
34      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0xFF);
35      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0xFF);
36      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4, 0xFF);
37      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0xFF);
38      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6, 0xFF);
39      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_7, 0xFF);
40          delay();
41      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0, 0x00);                // GPIO Pins ↩
            auf Low stellen
42      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, 0x00);
43      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0x00);
44      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0x00);
45      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4, 0x00);
46      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x00);
47      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6, 0x00);
48      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_7, 0x00);
49
50     }
51      return 0;
52 }
```

4.2.2

```
1
2 /**
3  * main.c
```

```
4  */
5  #include <stdint.h>
6  #include <stdbool.h>                                    // definition of type "bool"
7  #include"inc/hw_memmap.h"                              // definition of memory adresses
8  #include"inc/hw_types.h"                               // definition of framework makros
9  #include"driverlib/gpio.h"
10 #include"driverlib/sysctl.h"
11
12 void delay(void)                                       // Blinklicht Delay
13 {
14   uint32_t i=1000;
15   while(i) {i--;}
16  }
17
18 void delayPWM(void)                                    // PWM Delay (Hoher Wert -> Helle LED)
19 {
20   uint32_t i=8000;
21   while(i) {i--;}
22  }
23
24
25 int main(void){
26     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);       // GPIO Port B aktivieren
27     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_0); // GPIO Port B Pin 0-7 aktivieren
28     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_1);
29     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_2);
30     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_3);
31     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_4);
32     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_5);
33     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_6);
34     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_7);
35
36    while(1)
37    {
38        delay();
39
40     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0, 0xFF);   // GPIO Pins 0-7 auf High stellen
41     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, 0xFF);
42     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0xFF);
43     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0xFF);
44     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4, 0xFF);
45     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0xFF);
```

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

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
12 // stores ms since startup
13 volatile uint32_t systemTime_ms = 0;
```

```
14
15
16 void InterruptHandlerTimer0A (void)
17 {
18     // Clear the timer interrupt to prevent the interrupt function from ↩
           immediately being called again on exit
19     TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
20     // count up one ms
21     systemTime_ms++;
22 }
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 1ms)
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
49 void delay_ms(uint32_t waitTime)
50 {
51     uint32_t t = systemTime_ms;
52     while (systemTime_ms - t < waitTime);
53
54 }
55
56 int main(void)
57 {
58 clockSetup();
```

```
59          SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);                // GPIO Port↩
                B aktivieren
60              GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_0);         // GPIO↩
                     Port B Pin 0-7 aktivieren
61              GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_1);
62              GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_2);
63              GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_3);
64              GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_4);
65              GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_5);
66              GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_6);
67              GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_7);
68
69              bool state = 0;
70              //Verbeseserte Variante des Codes der GPIO-Steuerung aus Aufgabe 4
71                  while (1)
72                  {
73                      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0, GPIO_PIN_0*state)↩
                            ;
74                      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, GPIO_PIN_1*state)↩
                            ;
75                      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, GPIO_PIN_2*state)↩
                            ;
76                      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, GPIO_PIN_3*state)↩
                            ;
77                      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4, GPIO_PIN_4*state)↩
                            ;
78                      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, GPIO_PIN_5*state)↩
                            ;
79                      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6, GPIO_PIN_6*state)↩
                            ;
80                      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_7, GPIO_PIN_7*state)↩
                            ;
81
82                      state = !state;
83                      delay_ms(500/2);
84                  }
85              }
```

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
24 int main(void){
25      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);       // GPIO Port B aktivieren
26      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_0); // GPIO Port B Pin 0-7 aktivieren
27      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_1);
28      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_2);
29      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_3);
30      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_4);
31      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_5);
32      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_6);
33      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_7);
34
35      GPIOPinTypeGPIOInput(GPIO_PORTC_BASE,GPIO_PIN_7);  // Schalter input config
```

```
36      GPIOPadConfigSet(GPIO_PORTC_BASE,GPIO_PIN_7,GPIO_STRENGTH_2MA, ↩
            GPIO_PIN_TYPE_STD_WPU);
37
38      while(1)
39      {
40          delay();
41          if(GPIOPinRead(GPIO_PORTC_BASE,GPIO_PIN_7)==0) {
42      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0, 0xFF);              // GPIO Pins ↩
            0-7 auf High stellen
43      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, 0xFF);
44      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0xFF);
45      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0xFF);
46      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4, 0xFF);
47      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0xFF);
48      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6, 0xFF);
49      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_7, 0xFF);
50          }
51          else
52          {
53      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0, 0x00);              // GPIO Pins ↩
            0-7 auf Low stellen
54      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, 0x00);
55      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0x00);
56      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0x00);
57      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4, 0x00);
58      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x00);
59      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6, 0x00);
60      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_7, 0x00);
61          }
62      }
63       return 0;
64  }
```

6.1.2

```
1
2  /**
3   * main.c
4   */
5  #include <stdint.h>
6  #include <stdbool.h>                                               // definition ↩
       of type "bool"
7  #include"inc/hw_memmap.h"                                         // definition ↩
       of memory adresses
8  #include"inc/hw_types.h"                                          // definition ↩
       of framework makros
9  #include"driverlib/gpio.h"
10 #include"driverlib/sysctl.h"
11
```

```
12 void delay(void)
13 {
14   uint32_t i=200000;
15   while(i) {i--;}
16  }
17
18 int main(void){
19
20     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);                    // Port F ↩
           Aktivieren
21     GPIOPinTypeGPIOInput(GPIO_PORTF_BASE,GPIO_PIN_4);               // PF4 als ↩
           Input definieren
22     GPIOPadConfigSet(GPIO_PORTF_BASE,GPIO_PIN_4,GPIO_STRENGTH_2MA,↩
           GPIO_PIN_TYPE_STD_WPU);
23     // Bei der der Unteraufagbe 6.1.2 muesste der der Port B und Pin 7 gewaehlt↩
           werden
24
25     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);                    // GPIO Port B ↩
           aktivieren
26     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_0);             // GPIO Port B ↩
           Pin 0-7 aktivieren
27     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_1);
28     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_2);
29     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_3);
30     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_4);
31     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_5);
32     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_6);
33     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_7);
34
35     bool state = 0;
36
37    while(1)
38    {
39        delay();
40
41        if(GPIOPinRead(GPIO_PORTF_BASE,GPIO_PIN_4)==0)                // Wenn ↩
              Schalter geschlossen
42        {
43           state = !state;                                            // State ↩
                 wird gewechselt
44        }
45
46        GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0, GPIO_PIN_0*state); // GPIO ↩
              Output wird auf jeweiligen State geschaltet
47        GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, GPIO_PIN_1*state);
48        GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, GPIO_PIN_2*state);
49        GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, GPIO_PIN_3*state);
50        GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4, GPIO_PIN_4*state);
51        GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, GPIO_PIN_5*state);
52        GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6, GPIO_PIN_6*state);
```

```
53          GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_7, GPIO_PIN_7*state);
54      }
55       return 0;
56  }
```

6.1.3

Der Code ist der selbe wie bei Aufgabenteil 6.1.2 nur, dass der Port B 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

7.1.2

7.2 Widerstandsmessung

8 Lautstärkepegel

8.1 Lautstärkepegel

8.2 Eigene Verbesserungen