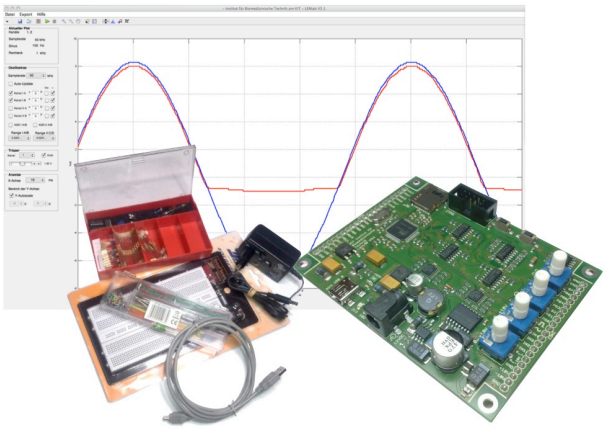
Karlsruher Institut für Technologie

Workshop

Elektrotechnik und Informationstechnik

Kurs 4

Workshop Signale und Systeme



Gruppe 22

Vorname	Nachname	Matrikel-Nr.	u-Account	E-Mail
Abraham	Gassama	2285843	uqetv	uqetv@student.kit.edu
Daniel	Wörner	2207790	utugh	utugh@student.kit.edu

6. Juli 2021

Abstract

Die Signaltheorie spielt in der heutigen Industrie 4.0 eine sehr wichtige Rolle wie z.B. bei der Automatisierungstechnik. Ziel dieses Projekt ist: die analoge Welt zur digitalen zu verbinden. Dies passiert mithilfe Implementierung unterschiedlicher Methoden auf einenm Mikrocontroller und durch Benutzung dessen AD-Wandlers. In den unterschiedlichen Aufgaben werden verschieden Verfahren genutzt, um Musik-Signale auszuwerten mit Hilfe von digitaler Filterung, Analyse der Signalenergie und einer Frequenz-Analyse mithilfe der Fourier-Transformation. Insbesondere wird dabei vorgestellt, wie existierende Frameworks und Bibliotheken eingesetzt werden können, um am Beispiel von Musik Signalen einfache Konzepte der Signalverarbeitung zu implementieren. In der ersten Aufgabe wird die Energie des Eingangssignals berechnet und damit über 6 verschieden LED die Lautstärke der Musik visualisiert. In der zweiten Aufgabe handelt es sich um eine digitale FIR-Filterung und in der dritten um eine diskrete Fourier-Transformation.

Inhaltsverzeichnis

Tabelle 1: Aufgabenverteilung

Aufgabe	**bearbeitet von**
Aufgabe 1	
Recherche	Abraham Gassama
Programmierung	A. Gassama, D. Wörner
Hardwareaufbau	Daniel Wörner
Aufgabe 2	
Recherche	Abraham Gassama
Programmierung	Daniel Wörner
Aufgabe 3	
Recherche	A. Gassama, D. Wörner
Berechnung Fouriertransformation	Abraham Gassama
Beantworten der weiteren Fragen	A. Gassama, D. Wörner
Programmierung	Daniel Wörner
Bonusaufgabe	Daniel Wörner
Dokumentation	Abraham Gassama

1 Erste Aufgabe

```
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 // Praeprozessor-Makros
11 #define BUFFER_SIZE 1000
12 #define SAMPLERATE 44000
13
14
15 // Funktionen-Deklarationen
16 void adcIntHandler(void);
17 void setup(void);
18
19 // globale Variablen
20 // hier die benötigten globalen Variablen, wie den ↩
      Ringbuffer einfuegen
21 float ringbuffer[1000];
22 int counter = 0;
23
24 void main(void){ // nicht veraendern!! Bitte Code in ↩
      adcIntHandler einfuegen
25     setup();
26     while(1){
27 }
28 }
29
30 void setup(void){// konfiguriert den MiKrocontroller
31
32     // konfiguriere System-Clock
33     SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|↩
          SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);
34     uint32_t period = SysCtlClockGet()/SAMPLERATE;
35
36     // aktiviere Peripherie
37     SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
```

```
38      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
39      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
40      SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
41
42      // konfiguriere GPIO
43      GPIOPinTypeADC(GPIO_PORTE_BASE,GPIO_PIN_2);
44      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);
45
46      // konfiguriere Timer
47      TimerConfigure(TIMER0_BASE,TIMER_CFG_PERIODIC);
48      TimerLoadSet(TIMER0_BASE, TIMER_A, period - 1);
49      TimerControlTrigger(TIMER0_BASE,TIMER_A,true);
50      TimerEnable(TIMER0_BASE,TIMER_A);
51
52      // konfiguriere ADC
53      ADCClockConfigSet(ADC0_BASE,ADC_CLOCK_RATE_FULL,1);
54      ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_TIMER, 0)↩
           ;
55      ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_CH1|↩
           ADC_CTL_IE|ADC_CTL_END);
56      ADCSequenceEnable(ADC0_BASE, 3);
57      ADCIntClear(ADC0_BASE,3);
58      ADCIntRegister(ADC0_BASE,3,adcIntHandler);
59      ADCIntEnable(ADC0_BASE,3);
60
61  }
62
63
64  void adcIntHandler (void){
65
66      //Counter reset wenn der Ringbuffer einmal komplett mit ↩
           neuen Messwerten beschrieben wurde
67      if (counter == 1000) {
68          counter = 0;
69          return;
70      }
71
72      //Ringbuffer mit neuem quadriertem Messwert vom ADC ↩
           füllen
73      uint32_t adcInputValue = 0.0;
```

```
74    float inputEnergy = 0.0;
75    float average = 0.0;
76    ADCSequenceDataGet(ADC0_BASE,3,&adcInputValue);
77
78    inputEnergy = adcInputValue^2;
79
80    ringbuffer[counter] = inputEnergy;
81
82    float sum = 0.0;
83
84    //Wenn der Ringbuffer voll ist, den average der 1000 ↩
         Werte berechnen
85    if (counter == 999) {
86
87      int i;
88      for (i = 0; i <= 999; i++) {
89          sum = ringbuffer[i] + sum;
90
91      }
92      average = sum / 1000;
93
94
95      //Ab hier werden die GPIOs angesteuert, die den ↩
           Energiegehalt des Signals repräsentieren
96      if (average > 0.0) {
97          GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0, ↩
              GPIO_PIN_0);
98      }
99      else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0, 0);
100
101     if (average >= 10.0) {
102             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, ↩
                GPIO_PIN_1);
103         }
104     else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, 0);
105
106     if (average >= 21.0) {
107                 GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, ↩
                    GPIO_PIN_2);
108             }
109     else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0);
110
```

```
111     if (average >= 36.5) {
112                 GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, ↩
                       GPIO_PIN_3);
113             }
114     else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0);
115
116     if (average >= 52.0) {
117                 GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4, ↩
                       GPIO_PIN_4);
118             }
119     else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4, 0);
120
121     if (average >= 77.5) {
122                 GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, ↩
                       GPIO_PIN_5);
123             }
124     else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0);
125
126     if (average >= 93.0) {
127                 GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6, ↩
                       GPIO_PIN_6);
128             }
129     else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6, 0);
130
131     if (average >= 108.5) {
132                 GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_7, ↩
                       GPIO_PIN_7);
133             }
134     else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_7, 0);
135
136   }
137
138   counter++;
139
140   // am Ende von adcIntHandler, Interrupt-Flag loeschen
141   ADCIntClear(ADC0_BASE,3);
142 }
```

2 Zweite Aufgabe

```
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/fpu.h"
9  #include "driverlib/timer.h"
10
11
12 // hier noch Ihren Filterheader einbinden
13 #include "FIR_Filter.h"
14
15
16 // Praeprozessor-Makros
17 #define SAMPLERATE 44000
18 #define FILTERORDER 50
19
20 // Funktionen-Deklarationen
21 void adcIntHandler(void);
22 void setup(void);
23 // hier nach Bedarf noch weitere Funktionsdeklarationen ↩
      einfuegen
24
25 // global variables
26 int32_t bufferSample[FILTERORDER];
27 int32_t sampleIndex = 0;
28 // hier nach Bedarf noch weitere globale Variablen einfuegen
29 float ringbuffer[50];
30 int counter = 0;
31
32 void main(void) // nicht veraendern!! Bitte Code in ↩
      adcIntHandler einfuegen
33 {
34     setup();
35     while(1){}
36 }
37
38 void setup(void){// konfiguriert den Mikrocontroller
```

```
39
40      // konfiguriere System-Clock
41      SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|↩
           SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);
42      uint32_t period = SysCtlClockGet()/SAMPLERATE;
43
44      // aktiviere Peripherie
45      SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
46      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
47      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
48      SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
49
50      // aktiviere Gleitkommazahlen-Modul
51      FPUEnable();
52      FPUStackingEnable();
53      FPULazyStackingEnable();
54      FPUFlushToZeroModeSet(FPU_FLUSH_TO_ZERO_EN);
55
56      // konfiguriere GPIO
57      GPIOPinTypeADC(GPIO_PORTE_BASE,GPIO_PIN_2);
58      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);
59
60      // konfiguriere Timer
61      TimerConfigure(TIMER0_BASE,TIMER_CFG_PERIODIC);
62      TimerLoadSet(TIMER0_BASE, TIMER_A, period - 1);
63      TimerControlTrigger(TIMER0_BASE,TIMER_A,true);
64      TimerEnable(TIMER0_BASE,TIMER_A);
65
66      // konfiguriere ADC
67      ADCClockConfigSet(ADC0_BASE,ADC_CLOCK_RATE_FULL,1);
68      ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_TIMER, 0)↩
           ;
69      ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_CH1|↩
           ADC_CTL_IE|ADC_CTL_END);
70      ADCSequenceEnable(ADC0_BASE, 3);
71      ADCIntClear(ADC0_BASE,3);
72      ADCIntRegister(ADC0_BASE,3,adcIntHandler);
73      ADCIntEnable(ADC0_BASE,3);
74
75  }
```

```
76
77
78
79
80 void adcIntHandler(void){
81
82     //Rinbuffer mit 50 neuen Messwerten füllen
83     uint32_t adcInputValue = 0.0;
84     uint32_t currentOutput = 0;
85     ADCSequenceDataGet(ADC0_BASE,3,&adcInputValue);
86
87     ringbuffer[counter] = adcInputValue;
88     counter++;
89
90     //Wenn Ringbuffer 50 neue Werte hat, Tiefpassfilterung ↩
          mit Filterkoeffizienten aus FIR_Filter.h anwenden um ↩
          eine frequenzabhängige
91     //Lautstärkemessung zu realisieren.
92     if (counter == 50) {
93         counter = 0;
94
95         int i;
96         for (i = 0; i <= 49; i++) {
97                 currentOutput = ( B[i] * ringbuffer[49 - i] ↩
                    ) + currentOutput;
98         }
99
100        //current Ouput repräsentiert die Laustärke des ↩
             Signals, die im folgenden über die LEDs angezeigt↩
              werden
101             if (currentOutput > 0.0) {
102                 GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0, ↩
                       GPIO_PIN_0);
103             }
104             else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0, ↩
                    0);
105
106             if (currentOutput >= 70.0) {
107                     GPIOPinWrite(GPIO_PORTB_BASE, ↩
                          GPIO_PIN_1, GPIO_PIN_1);
108                 }
```

```
109             else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, ↩
                    0);
110
111             if (currentOutput >= 150.0) {
112                         GPIOPinWrite(GPIO_PORTB_BASE, ↩
                               GPIO_PIN_2, GPIO_PIN_2);
113                 }
114             else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, ↩
                    0);
115
116             if (currentOutput >= 250.0) {
117                         GPIOPinWrite(GPIO_PORTB_BASE, ↩
                               GPIO_PIN_3, GPIO_PIN_3);
118                 }
119             else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, ↩
                    0);
120
121             if (currentOutput >= 600.0) {
122                         GPIOPinWrite(GPIO_PORTB_BASE, ↩
                               GPIO_PIN_4, GPIO_PIN_4);
123                 }
124             else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4, ↩
                    0);
125
126             if (currentOutput >= 900.0) {
127                         GPIOPinWrite(GPIO_PORTB_BASE, ↩
                               GPIO_PIN_5, GPIO_PIN_5);
128                 }
129             else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, ↩
                    0);
130
131             if (currentOutput >= 1300.0) {
132                         GPIOPinWrite(GPIO_PORTB_BASE, ↩
                               GPIO_PIN_6, GPIO_PIN_6);
133                 }
134             else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6, ↩
                    0);
135
136             if (currentOutput >= 1600.0) {
137                         GPIOPinWrite(GPIO_PORTB_BASE, ↩
                               GPIO_PIN_7, GPIO_PIN_7);
138                 }
```

```
139             else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_7, ↩
                   0);
140
141     }
142    // am Ende von adcIntHandler, Interrupt-Flag loeschen
143    ADCIntClear(ADC0_BASE,3);
144 }
```

2.1 Filterkoeffizienten

```
1  /*
2   * Filter Coefficients (C Source) generated by the Filter ↩
       Design and Analysis Tool
3   * Generated by MATLAB(R) 9.10 and Signal Processing Toolbox↩
       8.6.
4   * Generated on: 22-Jun-2021 18:41:21
5   */
6
7  /*
8   * Discrete-Time FIR Filter (real)
9   * -------------------------------
10  * Filter Structure  : Direct-Form FIR
11  * Filter Length     : 50
12  * Stable            : Yes
13  * Linear Phase      : Yes (Type 2)
14  */
15
16 /* General type conversion for MATLAB generated C-code  */
17 #include <stdint.h>
18 /*
19  * Expected path to tmwtypes.h
20  * C:\Program Files\MATLAB\R2021a\extern\include\tmwtypes.h
21  */
22 /*
23  * Warning - Filter coefficients were truncated to fit ↩
       specified data type.
24  *   The resulting response may not match generated ↩
       theoretical response.
25  *   Use the Filter Design & Analysis Tool to design ↩
       accurate
26  *   single-precision filter coefficients.
```

```
27  */
28 const int BL = 50;
29 const float B[50] = {
30    0.001359080663, 0.001382027287, 0.001775203506, ↩
        0.001932066982, 0.001685087918,
31   0.0008858409128,-0.0005529233022,-0.002624704503,-0.005188337062,-0.007

32    -0.01050630584, -0.01228532381, -0.01269126032, ↩
        -0.01113319863,-0.007119699381,
33   -0.000345002074, 0.009238117374,  0.02137267962,   ↩
       0.0354857333,  0.05072050169,
34      0.0660058856,   0.0801608488,  0.09201967716,   ↩
          0.1005629897,    0.105035387,
35       0.105035387,   0.1005629897,  0.09201967716,   ↩
           0.0801608488,   0.0660058856,
36     0.05072050169,   0.0354857333,  0.02137267962, ↩
         0.009238117374,-0.000345002074,
37   -0.007119699381, -0.01113319863, -0.01269126032, ↩
       -0.01228532381, -0.01050630584,
38   -0.007958900183,-0.005188337062,-0.002624704503,-0.0005529233022,0.0008

39    0.001685087918, 0.001932066982, 0.001775203506, ↩
        0.001382027287, 0.001359080663
40 };
```

3 Dritte Aufgabe

3.1 Implementierung Aufgabenteile 1 und 2

```
1  #include <stdint.h>
2  #include <stdbool.h>
3  #include <math.h>
4  #include <stdio.h>
5  #include "inc/hw_memmap.h"
6  #include "inc/hw_types.h"
7  #include "driverlib/sysctl.h"
8  #include "driverlib/adc.h"
9  #include "driverlib/gpio.h"
10 #include "driverlib/timer.h"
11 #include "driverlib/fpu.h"
12
13 // Praepozessor-Makros
14 #define SAMPLERATE 44000
15
16 //Funktionen-Deklarationen
17 void adcIntHandler(void);
18 void setup(void);
19 // hier nach Bedarf noch weitere Funktionsdeklarationen ↩
      einfuegen
20
21
22
23 // globale Variablen
24 const float DoublePi = 6.283185308;
25 int32_t bufferSample[440];
26 // hier nach Bedarf noch weitere globale Variablen einfuegen
27 int counter = 0;
28 float frequencyComponents[440];
29 double real = 0;
30 double img = 0;
31 int maxFrequency;
32
33
34 void main(void){ // nicht veraendern!! Bitte Code in ↩
      adcIntHandler einfuegen
35
36     setup();
```

```
37      while(1){}
38  }
39
40  void setup(void){//konfiguriert den Mikrocontroller
41
42      // konfiguriere SystemClock
43      SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|↩
            SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);
44      uint32_t period = SysCtlClockGet()/SAMPLERATE;
45
46      // aktiviere Peripherie
47      SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
48      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
49      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
50      SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
51
52      // aktiviere Gleitkommazahlen-Modul
53      FPUEnable();
54      FPUStackingEnable();
55      FPULazyStackingEnable();
56      FPUFlushToZeroModeSet(FPU_FLUSH_TO_ZERO_EN);
57
58      // konfiguriere GPIO
59      GPIOPinTypeADC(GPIO_PORTE_BASE,GPIO_PIN_2);
60      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);
61
62      // konfiguriere Timer
63      TimerConfigure(TIMER0_BASE,TIMER_CFG_PERIODIC);
64      TimerLoadSet(TIMER0_BASE, TIMER_A, period - 1);
65      TimerControlTrigger(TIMER0_BASE,TIMER_A,true);
66      TimerEnable(TIMER0_BASE,TIMER_A);
67
68      // konfiguriere ADC
69      ADCClockConfigSet(ADC0_BASE,ADC_CLOCK_RATE_FULL,1);
70      ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_TIMER, 0)↩
            ;
71      ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_CH1|↩
            ADC_CTL_IE|ADC_CTL_END);
72      ADCSequenceEnable(ADC0_BASE, 3);
73      ADCIntClear(ADC0_BASE,3);
```

```
74      ADCIntRegister(ADC0_BASE,3,adcIntHandler);
75      ADCIntEnable(ADC0_BASE,3);
76
77  }
78
79
80  void adcIntHandler(void){
81
82      //Bei jedem Aufruf einen neuen Messwert in den ↩
           bufferSample schreiben
83      uint32_t adcInputValue;
84      ADCSequenceDataGet(ADC0_BASE,3,&adcInputValue);
85      bufferSample[counter] = adcInputValue;
86
87      counter++;
88
89      //Bei 440 neuen Messwerten wird die DFT berechnet und ↩
           die Fourierkoeffizienten in frequencyComponents[] ↩
           abgelegt
90      if (counter >= 440) {
91
92          int l;
93          int k;
94          int n;
95          //Schleife für einzelnes Frequenzelement
96          for (k = 0; k <= 39; k++) {
97
98                  real = 0;
99                  img = 0;
100                 //Schleife für einzelner Summand in ↩
                       Frequenzelement
101                 for (n = 0; n <= 439; n++) {
102                     real += cosf((DoublePi * n * k) / 440) *↩
                            bufferSample[n];
103                     img += sinf((DoublePi * n * k) / 440) * ↩
                           bufferSample[n];
104
105                     frequencyComponents[k] = real * real + ↩
                           img * img;
106                 }
107
108         }
```

```
109
110         //Ermitteln des betragsmäßig größten ↩
               Fourierkoeffizienten,
111         //sodass maxFrequency das dominierende Frequenzband ↩
               widerspiegelt
112         maxFrequency = 0;
113         for(l = 0; l < 39; l++){
114             if(frequencyComponents[l] > frequencyComponents[↩
                   maxFrequency]) {
115                 maxFrequency = l;
116             }
117         }
118
119         //Berechnung des Frequenzbandes über die ↩
               Frequenzauflösung
120         maxFrequency = maxFrequency * 100;
121
122
123         //Abbilden der Frequenzbereiche auf 8 LEDs. Linear ↩
               verteilt mit Bereichen von 500Hz.
124         if(maxFrequency >= 0 && maxFrequency < 500) {
125             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0, ↩
                   GPIO_PIN_0);
126         }
127         else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0, 0);
128
129         if(maxFrequency >= 500 && maxFrequency < 1000) {
130             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, ↩
                   GPIO_PIN_1);
131         }
132         else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, 0);
133
134         if(maxFrequency >= 1000 && maxFrequency < 1500) {
135             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, ↩
                   GPIO_PIN_2);
136         }
137         else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0);
138
139         if(maxFrequency >= 1500 && maxFrequency < 2000) {
140             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, ↩
                   GPIO_PIN_3);
141         }
```

```
142         else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0);
143
144         if(maxFrequency >= 2000 && maxFrequency < 2500) {
145             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4, ↩
                    GPIO_PIN_4);
146         }
147         else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4, 0);
148
149         if(maxFrequency >= 2500 && maxFrequency < 3000) {
150             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, ↩
                    GPIO_PIN_5);
151         }
152         else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0);
153
154         if(maxFrequency >= 3000 && maxFrequency < 3500) {
155             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6, ↩
                    GPIO_PIN_6);
156         }
157         else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6, 0);
158
159         if(maxFrequency >= 3500 && maxFrequency <= 4000) {
160             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_7, ↩
                    GPIO_PIN_7);
161         }
162         else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_7, 0);
163
164         counter = 0;
165     }
166
167     // am Ende von adcIntHandler, Interrupt-Flag loeschen
168     ADCIntClear(ADC0_BASE,3);
169 }
```

3.2 Aufgabenteil 3

Woran kann es liegen, dass nur die erste LED leuchtet?
Antwort: Da wir unter unseren Messwerten nur positive Werte haben, bekommen wir bei der DFT wie in der Aufgabenstellung beschrieben, aus der Sinus-Funktion ohne untere Halbwelle ein Frequenzspektrum, welches in verschiedene Frequenzbereiche hineinleckt. Und besonders groß wird bei kleinen Frequenzen. Wodurch die Maximumsuche in unserer Implementierung diese fälschlicherweise als dominierendes Freuquenzband erkennt und über die erste LED anzeigt.

3.3 Aufgabenteil 4

Berechnung der Fouriertransformation:

$$s(t) = \sin(2\pi f_0 t) \cdot r_{\frac{1}{2f_0}}\left(t - \frac{1}{4f_0}\right) * \sum_{n=-\infty}^{\infty} \delta\left(t - n\frac{1}{f_0}\right)$$

$$S(f) = \left(\frac{i}{2}\left(\delta(f+f_0) - \delta(f-f_0)\right) * \frac{1}{2f_0} si\left(\pi f \frac{1}{2f_0}\right) e^{i\frac{1}{4f_0}f} \right).$$

$$\hookrightarrow f_0 \sum_{k=-\infty}^{\infty} \delta(f - f_0 k)$$

$$S(f) = \frac{i}{2}\left(\frac{1}{2f_0} si\left(\frac{1}{2f_0}\pi(f+f_0)\right) e^{i\frac{1}{4f_0}(f+f_0)} - \frac{1}{2f_0} si\left(\frac{1}{2f_0}\pi(f-f_0)\right) e^{i\frac{1}{4f_0}(f-f_0}\right.$$

$$\hookrightarrow f_0 \sum_{k=-\infty}^{\infty} \delta(f - f_0 k)$$

$$S(f) = \frac{i}{4}\left(si\left(\frac{1}{2f_0}\pi(f+f_0)\right) e^{i\frac{1}{4f_0}(f+f_0)} - si\left(\frac{1}{2f_0}\pi(f-f_0)\right) e^{i\frac{1}{4f_0}(f-f_0)}\right).$$

$$\hookrightarrow \sum_{k=-\infty}^{\infty} \delta(f - f_0 k)$$

$$S(f) = \frac{i}{4} \sum_{k=-\infty}^{\infty} si\left(\frac{1}{2f_0}\pi(f_0 k + f_0)\right) e^{i\frac{1}{4f_0}(f_0 k + f)} - si\left(\frac{1}{2f_0}\pi(f - f_0)\right) e^{i\frac{1}{4f_0}(f_0 k - f}$$

$$S(f) = \frac{i}{4} \sum_{k=-\infty}^{\infty} si\left(\frac{1}{2f_0}\pi(k+1)f_0\right) e^{i\frac{1}{4f_0}(k+1)f_0} - si\left(\frac{1}{2f_0}\pi(k-1)f_0\right).$$

$$\hookrightarrow e^{i\frac{1}{4f_0}(k-1)f_0}$$

$$S(f) = \frac{i}{4} \sum_{k=-\infty}^{\infty} si\left(\frac{\pi}{2}(k+1)\right) e^{i\frac{1}{4}(k+1)} - si\left(\frac{\pi}{2}(k-1)\right) e^{i\frac{1}{4}(k-1)}$$

Abbildung 1: Fourier Berechnung

3.4 Aufgabenteil 5

Die Sequenz der LEDs, die beim Abspielen der $DFT\_test.mp3$ Datei aufleuchtet lautet:

$3, 7, 0, 4, 2, 1, 2, 5, 3, 7$

Implementierung nach Aufgabenteil 3.5:

```
1  #include <stdint.h>
2  #include <stdbool.h>
3  #include <math.h>
4  #include <stdio.h>
5  #include "inc/hw_memmap.h"
6  #include "inc/hw_types.h"
7  #include "driverlib/sysctl.h"
8  #include "driverlib/adc.h"
9  #include "driverlib/gpio.h"
10 #include "driverlib/timer.h"
11 #include "driverlib/fpu.h"
12
13 // Praepozessor-Makros
14 #define SAMPLERATE 44000
15
16 //Funktionen-Deklarationen
17 void adcIntHandler(void);
18 void setup(void);
19 // hier nach Bedarf noch weitere Funktionsdeklarationen ↩
      einfuegen
20
21
22
23 // globale Variablen
24 const float DoublePi = 6.283185308;
25 int32_t bufferSample[440];
26 // hier nach Bedarf noch weitere globale Variablen einfuegen
27 int counter = 0;
28 float frequencyComponents[440];
29 double real = 0;
30 double img = 0;
31 int maxFrequency;
32
33
34 void main(void){ // nicht veraendern!! Bitte Code in ↩
```

```
    adcIntHandler einfuegen
35
36      setup();
37      while(1){}
38  }
39
40  void setup(void){//konfiguriert den Mikrocontroller
41
42      // konfiguriere SystemClock
43      SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|↩
            SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);
44      uint32_t period = SysCtlClockGet()/SAMPLERATE;
45
46      // aktiviere Peripherie
47      SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
48      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
49      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
50      SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
51
52      // aktiviere Gleitkommazahlen-Modul
53      FPUEnable();
54      FPUStackingEnable();
55      FPULazyStackingEnable();
56      FPUFlushToZeroModeSet(FPU_FLUSH_TO_ZERO_EN);
57
58      // konfiguriere GPIO
59      GPIOPinTypeADC(GPIO_PORTE_BASE,GPIO_PIN_2);
60      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);
61
62      // konfiguriere Timer
63      TimerConfigure(TIMER0_BASE,TIMER_CFG_PERIODIC);
64      TimerLoadSet(TIMER0_BASE, TIMER_A, period - 1);
65      TimerControlTrigger(TIMER0_BASE,TIMER_A,true);
66      TimerEnable(TIMER0_BASE,TIMER_A);
67
68      // konfiguriere ADC
69      ADCClockConfigSet(ADC0_BASE,ADC_CLOCK_RATE_FULL,1);
70      ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_TIMER, 0)↩
            ;
71      ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_CH1|↩
```

```
        ADC_CTL_IE|ADC_CTL_END);
72      ADCSequenceEnable(ADC0_BASE, 3);
73      ADCIntClear(ADC0_BASE,3);
74      ADCIntRegister(ADC0_BASE,3,adcIntHandler);
75      ADCIntEnable(ADC0_BASE,3);
76
77  }
78
79
80  void adcIntHandler(void){
81
82      //Bei jedem Aufruf einen neuen Messwert in den ↩
          bufferSample schreiben
83      uint32_t adcInputValue;
84      ADCSequenceDataGet(ADC0_BASE,3,&adcInputValue);
85      bufferSample[counter] = adcInputValue;
86
87      counter++;
88
89      //Bei 440 neuen Messwerten wird die DFT berechnet und ↩
          die Fourierkoeffizienten in frequencyComponents[] ↩
          abgelegt
90      if (counter >= 440) {
91
92          int l;
93          int k;
94          int n;
95          //Schleife für einzelnes Frequenzelement
96          for (k = 0; k <= 39; k++) {
97
98                  real = 0;
99                  img = 0;
100                 //Schleife für einzelner Summand in ↩
                      Frequenzelement
101                 for (n = 0; n <= 439; n++) {
102                     real += cosf((DoublePi * n * k) / 440) *↩
                            bufferSample[n];
103                     img += sinf((DoublePi * n * k) / 440) * ↩
                          bufferSample[n];
104
105                     frequencyComponents[k] = real * real + ↩
                          img * img;
```

```
106                 }
107
108         }
109
110         //Ermitteln des betragsmäßig größten ↩
               Fourierkoeffizienten,
111         //sodass maxFrequency das dominierende Frequenzband ↩
               widerspiegelt
112         maxFrequency = 1;
113         for(l = 1; l < 39; l++){
114             if(frequencyComponents[l] > frequencyComponents[↩
                   maxFrequency]) {
115                 maxFrequency = l;
116             }
117         }
118
119         //Berechnung des Frequenzbandes über die ↩
               Frequenzauflösung
120         maxFrequency = maxFrequency * 100;
121
122
123         //Abbilden der Frequenzbereiche auf 8 LEDs. Linear ↩
               verteilt mit Bereichen von 500Hz.
124         if(maxFrequency >= 0 && maxFrequency < 500) {
125             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0, ↩
                   GPIO_PIN_0);
126         }
127         else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0, 0);
128
129         if(maxFrequency >= 500 && maxFrequency < 1000) {
130             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, ↩
                   GPIO_PIN_1);
131         }
132         else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, 0);
133
134         if(maxFrequency >= 1000 && maxFrequency < 1500) {
135             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, ↩
                   GPIO_PIN_2);
136         }
137         else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0);
138
139         if(maxFrequency >= 1500 && maxFrequency < 2000) {
```

```
140             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, ↩
                    GPIO_PIN_3);
141         }
142         else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0);
143
144         if(maxFrequency >= 2000 && maxFrequency < 2500) {
145             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4, ↩
                    GPIO_PIN_4);
146         }
147         else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4, 0);
148
149         if(maxFrequency >= 2500 && maxFrequency < 3000) {
150             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, ↩
                    GPIO_PIN_5);
151         }
152         else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0);
153
154         if(maxFrequency >= 3000 && maxFrequency < 3500) {
155             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6, ↩
                    GPIO_PIN_6);
156         }
157         else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6, 0);
158
159         if(maxFrequency >= 3500 && maxFrequency <= 4000) {
160             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_7, ↩
                    GPIO_PIN_7);
161         }
162         else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_7, 0);
163
164         counter = 0;
165     }
166
167     // am Ende von adcIntHandler, Interrupt-Flag loeschen
168     ADCIntClear(ADC0_BASE,3);
169 }
```

4 Bonusaufgabe

Um über die DFT auf die Signalfrequenzen zu kommen, muss man diese mit den Fourierkoeffizienten und der Frequenzauflösung berechnen. Da die Frequenzauflösung bisher 100Hz war, kann die gesuchte Frequenz auch nur auf 100Hz genau bestimmt werden. Um diese Genauigkeit zu erhöhen, muss die Frequenzauflösung verändert werden. Was in unserem Fall nur über die Anzahl der Abtastwerte möglich ist, da wir die Abtastfrequend nicht erhöhen können. Wir haben die Abtastwerte Schritt für Schritt erhöht, bis die Stack Size Probleme gemacht hat. Sodass wir eine maximale Frequenzauflösung von 20Hz erreicht haben. Damit haben wir wieder die DFT berechnet und entsprechend zur Visualisierung die LEDs angesteuert. Das Ergebnis daraus war, dass die Frequenz des $Secret\_Sine.wav$ Signals zwischen 940Hz und 960Hz liegen muss.

Im Folgenden die Implementierung der DFT zur Frequenzanalyse der Bonusaufgabe:

```
1  #include <stdint.h>
2  #include <stdbool.h>
3  #include <math.h>
4  #include <stdio.h>
5  #include "inc/hw_memmap.h"
6  #include "inc/hw_types.h"
7  #include "driverlib/sysctl.h"
8  #include "driverlib/adc.h"
9  #include "driverlib/gpio.h"
10 #include "driverlib/timer.h"
11 #include "driverlib/fpu.h"
12
13 // Praepozessor-Makros
14 #define SAMPLERATE 44000
15
16 //Funktionen-Deklarationen
17 void adcIntHandler(void);
18 void setup(void);
19 // hier nach Bedarf noch weitere Funktionsdeklarationen ↩
      einfuegen
20
21
22
23 // globale Variablen
24 const float DoublePi = 6.283185308;
25 int32_t bufferSample[2200];
26 // hier nach Bedarf noch weitere globale Variablen einfuegen
```

```
27 int counter = 0;
28 float frequencyComponents[2200];
29 double real = 0;
30 double img = 0;
31 int maxFrequency;
32
33
34 void main(void){ // nicht veraendern!! Bitte Code in ↩
      adcIntHandler einfuegen
35
36     setup();
37     while(1){}
38 }
39
40 void setup(void){//konfiguriert den Mikrocontroller
41
42     // konfiguriere SystemClock
43     SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|↩
          SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);
44     uint32_t period = SysCtlClockGet()/SAMPLERATE;
45
46     // aktiviere Peripherie
47     SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
48     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
49     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
50     SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
51
52     // aktiviere Gleitkommazahlen-Modul
53     FPUEnable();
54     FPUStackingEnable();
55     FPULazyStackingEnable();
56     FPUFlushToZeroModeSet(FPU_FLUSH_TO_ZERO_EN);
57
58     // konfiguriere GPIO
59     GPIOPinTypeADC(GPIO_PORTE_BASE,GPIO_PIN_2);
60     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);
61
62     // konfiguriere Timer
63     TimerConfigure(TIMER0_BASE,TIMER_CFG_PERIODIC);
64     TimerLoadSet(TIMER0_BASE, TIMER_A, period - 1);
```

```
65      TimerControlTrigger(TIMER0_BASE,TIMER_A,true);
66      TimerEnable(TIMER0_BASE,TIMER_A);
67
68      // konfiguriere ADC
69      ADCClockConfigSet(ADC0_BASE,ADC_CLOCK_RATE_FULL,1);
70      ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_TIMER, 0)↩
          ;
71      ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_CH1|↩
          ADC_CTL_IE|ADC_CTL_END);
72      ADCSequenceEnable(ADC0_BASE, 3);
73      ADCIntClear(ADC0_BASE,3);
74      ADCIntRegister(ADC0_BASE,3,adcIntHandler);
75      ADCIntEnable(ADC0_BASE,3);
76
77  }
78
79
80  void adcIntHandler(void){
81
82      //Bei jedem Aufruf einen neuen Messwert in den ↩
          bufferSample schreiben
83      uint32_t adcInputValue;
84      ADCSequenceDataGet(ADC0_BASE,3,&adcInputValue);
85      bufferSample[counter] = adcInputValue;
86
87      counter++;
88
89      //Bei 2200 neuen Messwerten wird die DFT berechnet und ↩
          die Fourierkoeffizienten in frequencyComponents[] ↩
          abgelegt
90      if (counter >= 2200) {
91
92          int l;
93          int k;
94          int n;
95          //Schleife für einzelnes Frequenzelement
96          for (k = 30; k <= 50; k++) {
97
98                  real = 0;
99                  img = 0;
100                 //Schleife für einzelner Summand in ↩
                      Frequenzelement
```

```
101                 for (n = 0; n <= 2199; n++) {
102                     real += cosf((DoublePi * n * k) / 2200) ↩
                           * bufferSample[n];
103                     img += sinf((DoublePi * n * k) / 2200) *↩
                            bufferSample[n];
104
105                     frequencyComponents[k] = real * real + ↩
                           img * img;
106                 }
107
108         }
109
110         //Ermitteln des betragsmäßig größten ↩
               Fourierkoeffizienten,
111         //sodass maxFrequency das dominierende Frequenzband ↩
               widerspiegelt
112         maxFrequency = 1;
113         for(l = 30; l < 49; l++){
114             if(frequencyComponents[l] > frequencyComponents[↩
                   maxFrequency]) {
115                 maxFrequency = l;
116             }
117         }
118
119         //Berechnung des Frequenzbandes über die ↩
               Frequenzauflösung
120         maxFrequency = maxFrequency * 20;
121
122
123         //Abbilden der Frequenzbereiche auf die LEDs.
124         if(maxFrequency >= 850 && maxFrequency < 875 ) {
125             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0, ↩
                   GPIO_PIN_0);
126         }
127         else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0, 0);
128
129         if(maxFrequency >= 875 && maxFrequency < 900) {
130             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, ↩
                   GPIO_PIN_1);
131         }
132         else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, 0);
133
```

```
134         if(maxFrequency >= 900 && maxFrequency < 920) {
135             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, ↩
                    GPIO_PIN_2);
136         }
137         else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0);
138
139         if(maxFrequency >= 920 && maxFrequency < 940) {
140             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, ↩
                    GPIO_PIN_3);
141         }
142         else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0);
143
144         if(maxFrequency >= 940 && maxFrequency < 960) {
145             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4, ↩
                    GPIO_PIN_4);
146         }
147         else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4, 0);
148
149         if(maxFrequency >= 960 && maxFrequency < 980) {
150             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, ↩
                    GPIO_PIN_5);
151         }
152         else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0);
153
154         if(maxFrequency >= 980 && maxFrequency <= 1000) {
155             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6, ↩
                    GPIO_PIN_6);
156         }
157         else GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6, 0);
158
159         counter = 0;
160     }
161
162     // am Ende von adcIntHandler, Interrupt-Flag loeschen
163     ADCIntClear(ADC0_BASE,3);
164 }
```

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] https://de.wikipedia.org/wiki/Diskrete_Fourier-Transformation, Abrufdatum: 30. Juni 2021.