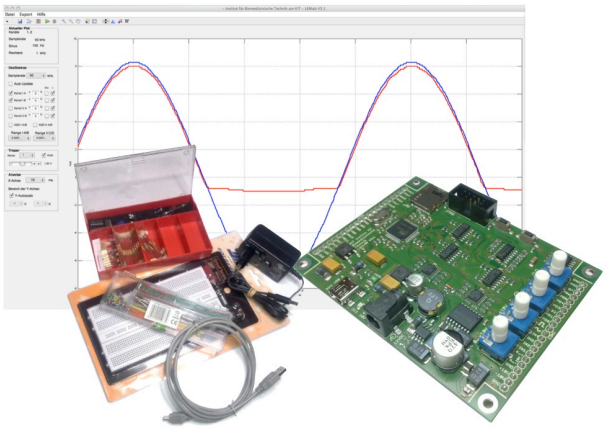
Karlsruher Institut für Technologie

Workshop

Elektrotechnik und Informationstechnik

Kurs 4

Digitale Signalverarbeitung



Gruppe 7

Vorname	Nachname	Matrikel-Nr.	u-Account	E-Mail
Yekta	Ahmadi Simab	2476855	unkho	unkho@student.kit.edu
Pratham	Gupta	2395144	ufuxl	ufuxl@student.kit.edu
Annan	Gonzalez	2374289	uexpj	uexpj@student.kit.edu

15. Mai 2024

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abstract

Im Rahmen des Workshops haben wir uns mit der Verarbeitung von Audiosignalen und die in der Signale und Systeme Vorlesung behandelten Transformationen auseinandergesetzt.
Hierzu wurden das Mikrokontrollerboard Launchpad von Texas Instruments und die von TI bereitgestellten IDE und Compiler benutzt.
In der ersten Aufgabe befassen wir uns mit der Lautstärkenmessung von Audiosignalen und visualisieren diese mithilfe entsprechender Skalierung und acht LEDs.
In Aufgabe zwei experimientieren wir mit der Schnittstelle zweier Softwares, wo wir einen Tiefpass aus dem Matlab Filterdesigner in CSS importieren und die Audiosignale nach Frequenz filtern.

1 Aufgabe 1: Lautstärkemessung mittels Moving-Average-Filter

1.1 Quellcode

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/adc.h"
#include "driverlib/gpio.h"
#include "driverlib/timer.h"
#include "math.h"

// Praeprozessor-Makros
#define BUFFER_SIZE 1000
#define SAMPLERATE 44000


// Funktionen-Deklarationen
void adcIntHandler(void);
void setup(void);

// globale Variablen
int32_t bufferSample [ BUFFER_SIZE ] = { 0 } ; // Ringpuffer ↩
    Deklaration
int lastMean = 0;
int currentMean = 0;
int bufferIndex = 0;
int MAX_ADC_VALUE =  4905;
int maxADCValue = 0;
int i = 0;
#define LED_COUNT 8

void main(void){ // nicht veraendern!! Bitte Code in adcIntHandler ↩
    einfuegen
    setup();
    while(1){}
}

void setup(void){// konfiguriert den MiKrocontroller

    // konfiguriere System-Clock
    SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|↩
    SYSCTL_XTAL_16MHZ);
    uint32_t period = SysCtlClockGet()/SAMPLERATE;

    // aktiviere Peripherie
```

```
42      SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
43      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
44      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
45      SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
46
47      // konfiguriere GPIO
48      GPIOPinTypeADC(GPIO_PORTE_BASE,GPIO_PIN_2);
49      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);
50
51      // konfiguriere Timer
52      TimerConfigure(TIMER0_BASE,TIMER_CFG_PERIODIC);
53      TimerLoadSet(TIMER0_BASE, TIMER_A, period - 1);
54      TimerControlTrigger(TIMER0_BASE,TIMER_A,true);
55      TimerEnable(TIMER0_BASE,TIMER_A);
56
57      // konfiguriere ADC
58      ADCClockConfigSet(ADC0_BASE,ADC_CLOCK_RATE_FULL,1);
59      ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_TIMER, 0);
60      ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_CH1|ADC_CTL_IE|↩
        ADC_CTL_END);
61      ADCSequenceEnable(ADC0_BASE, 3);
62      ADCIntClear(ADC0_BASE,3);
63      ADCIntRegister(ADC0_BASE,3,adcIntHandler);
64      ADCIntEnable(ADC0_BASE,3);
65
66  }
67
68  int getLEDvalue(int mean) {
69      if (mean >= MAX_ADC_VALUE) {
70          return 0xFF;
71      }
72
73      // Apply logarithmic mapping to distribute LED count more evenly
74      double logScaledMean = log10(1 + (double)mean / MAX_ADC_VALUE * 40)↩
        / log10(10) * LED_COUNT; //40 is the logarithmic sensitivity factor↩
        , we played with it and this one resulted in exactly 4 LEDs on at ↩
        50% and all 8 at 100% in the online tone generator
75
76      // Calculate LED count based on logarithmic scale
77      int ledCount = (int)logScaledMean;
78
79      // Ensure LED count is within the valid range
80      if (ledCount > LED_COUNT) {
81          ledCount = LED_COUNT;
82      } else if (ledCount < 0) {
83          ledCount = 0;
84      }
85
86      // Create LED bitmask using bitwise right shift
```

```
87      int ledMask = (0xFF >> (LED_COUNT - ledCount));
88
89      return ledMask;
90  }
91
92  #define BUFFER_SIZE 1000
93
94  void adcIntHandler(void) {
95      uint32_t adcInputValue;
96      ADCSequenceDataGet(ADC0_BASE, 3, &adcInputValue);
97
98      // Update the buffer
99      currentMean = lastMean + (adcInputValue * adcInputValue - ↩
        bufferSample[bufferIndex]);
100     lastMean = currentMean;
101     bufferSample[bufferIndex] = adcInputValue * adcInputValue;
102     bufferIndex++;
103     bufferIndex %= BUFFER_SIZE - 1;
104
105     // Determine the maximum ADC value reached in the current buffer ↩
        iteration
106     for (i = 0; i < BUFFER_SIZE; i++) {
107         if (bufferSample[i] > maxADCValue) {
108             maxADCValue = bufferSample[i];
109         }
110     }
111     MAX_ADC_VALUE = maxADCValue;  // Update MAX_ADC_VALUE
112
113     // Update the LEDs with the logarithmically scaled intensity
114     int intensity = getLEDvalue(currentMean / BUFFER_SIZE);
115     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, ↩
        intensity);
116
117     // Remove Interrupt-Flag
118     ADCIntClear(ADC0_BASE, 3);
119 }
```

Quellcode 1: Aufgabe 1: LautstÃ¤rkemessung mittels Moving-Average-Filter

1.2 Erläuterung

In dieser Aufgabe haben wir wie in der Aufgabenstellung verlangt, die Werte vom AD-Wandler ausgewertet und mit 8 LEDs die Lautstärke angezeigt. In der Funktion adcIntHandler werden zunächst die Werte des ADC in einem Array (Ringpuffer) der Länge 1000 gespeichert und der Mittelwert wird berechnet, danach erfolgt das gleiche für die nächste 1000 Werte. Die aus dem Ringpuffer gewonnenen Mittelwerte werden in eine andere Funktion eingegeben "getLEDvalue". Die Funktion getLEDvalue skaliert die Werte logarithmisch mithilfe der Totalen Anzahl an

LEDs, MAX_ADC_VALUE und einen Empfindlichkeitsfaktor das wir durch mehrere Versuche optimiert haben. Der Output von getLEDvalue funktioniert mit dem bitwise right shift Operator, um die Anzahl der leuchtende LEDs herauszugeben.

Bemerkenswert in dieser Aufgabe war die Benutzung von MAX_ADC_VALUE. Unsere erste Annahme war, dass MAX_ADC_VALUE nach der Dokumentation für unseren AD-Wandler $2^{12} = 4906 - 1$ entspricht. Allerdings haben wir mit diesem maximalen Wert keine gute Ergebnisse erzielt. Die Erklärung dafür ist, dass wir als Inputsignal ein Audiosignal von einem Audiokabel mit maximal 2V Amplitude benutzen, und das theoretische Maximum unseres Boards 3,3V ist. Als Lösung haben wir den maximalen Wert nach jeder vollständigen Ringpuffer-Iteration aktualisiert.

2 Aufgabe 2: Digitale FIR-Filterung

2.1 Filter Headerdatei

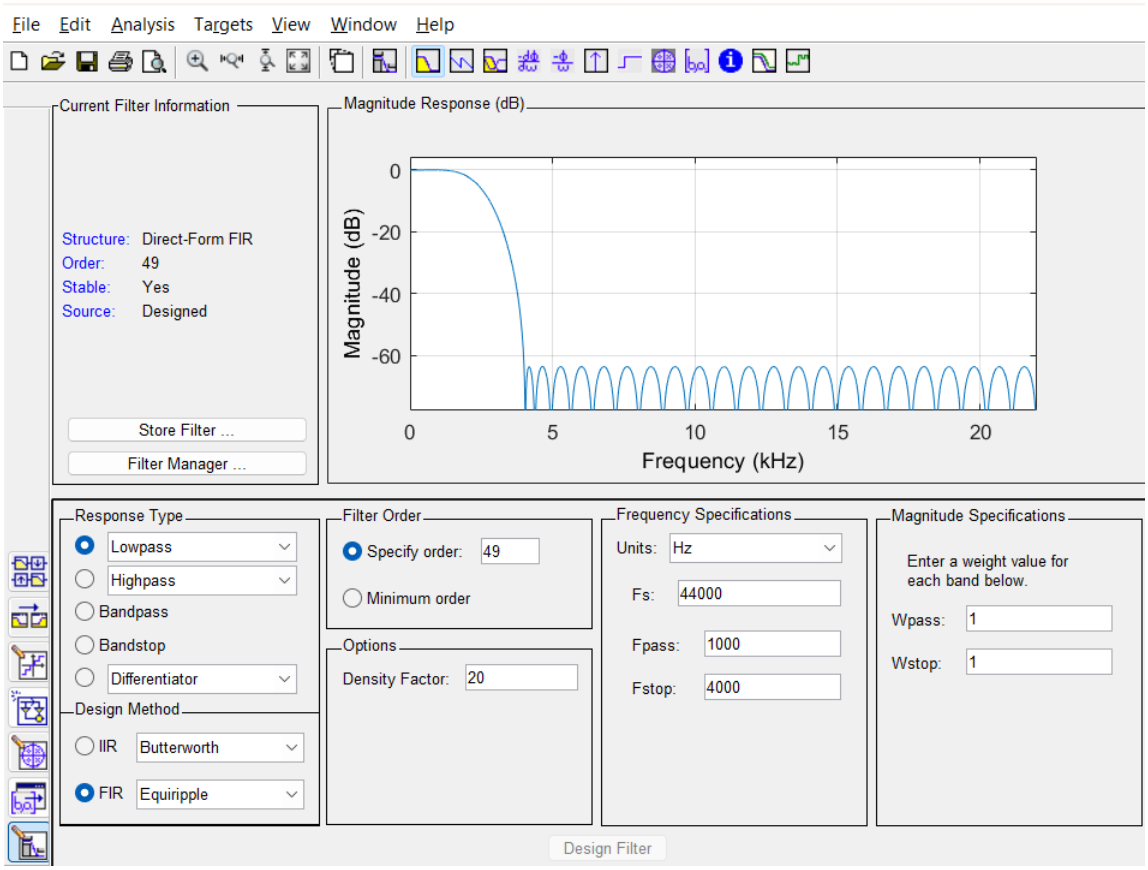


Abbildung 1: Matlab FIR Tiefpass-Filter erstellt mit Matlab Filter Designer.

```
/*
 * Filter Coefficients (C Source) generated by the Filter Design and Analysis Tool
 * Generated by MATLAB(R) 23.2 and Signal Processing Toolbox 23.2.
 * Generated on: 26-Apr-2024 10:34:07
 */

/*
 * Discrete-Time FIR Filter (real)
 * -------------------------------
 * Filter Structure  : Direct-Form FIR
 * Filter Length     : 50
 * Stable            : Yes
 * Linear Phase      : Yes (Type 2)
 */

/* General type conversion for MATLAB generated C-code  */
```

```
#include <stdint.h>
/*
 * Expected path to tmwtypes.h
 * C:\Program Files\MATLAB\R2023b\extern\include\tmwtypes.h
 */
/*
 * Warning - Filter coefficients were truncated to fit specified data type.
 *   The resulting response may not match generated theoretical response.
 *   Use the Filter Design & Analysis Tool to design accurate
 *   single-precision filter coefficients.
 */
extern const int numLength = 50;
extern const float num[50] = {
  0.0007326174527, 0.001079884474, 0.001643151743, 0.002142431913, 0.002394242678,
   0.002187608974, 0.001325643039,-0.0003195040626,-0.002750207437, -0.00579530606,
    -0.0090823723, -0.01204310358, -0.01395735424, -0.01403498556, -0.01152775344,
  -0.005856912117, 0.003264341038,  0.01573598757,   0.0310233999,  0.04816968739,
   0.06587304175,  0.08262299746,  0.09687805921,   0.1072587892,  0.1127275676,
    0.1127275676,   0.1072587892,  0.09687805921,  0.08262299746,  0.06587304175,
   0.04816968739,   0.0310233999,  0.01573598757, 0.003264341038,-0.005856912117,
   -0.01152775344, -0.01403498556, -0.01395735424, -0.01204310358, -0.0090823723,
   -0.00579530606,-0.002750207437,-0.0003195040626, 0.001325643039, 0.002187608974,
   0.002394242678, 0.002142431913, 0.001643151743, 0.001079884474,0.0007326174527
};
```

Quellcode 2: Aufgabe 2: Digitale FIR-Filterung. Headerdatei Importiert aus Matlab

2.2 Quellcode

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/adc.h"
#include "driverlib/gpio.h"
#include "driverlib/fpu.h"
```

```
9  #include "driverlib/timer.h"
10 #include "math.h"
11
12 // hier noch Ihren Filterheader einbinden
13 #include "TiefpassAufgabe2.h"
14
15 // Praeprozessor-Makros
16 #define SAMPLERATE 44000
17 #define FILTERORDER 49
18 int MAX_ADC_VALUE =  4905;
19 int maxADCValue = 0;
20 int i = 0;
21 #define LED_COUNT 8
22
23 // Funktionen-Deklarationen
24 void adcIntHandler(void);
25 void setup(void);
26 // hier nach Bedarf noch weitere Funktionsdeklarationen einfuegen
27
28 // global variables
29 int32_t bufferSample[FILTERORDER];
30 int32_t sampleIndex = 0;
31 float output = 0.0f;
32 // hier nach Bedarf noch weitere globale Variablen einfuegen
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 System-Clock
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  //Uses output to decide which LEDs turn on
80
81  int getLEDvalue(int mean) {
82      if (mean >= MAX_ADC_VALUE) {
83          return 0xFF;
84      }
85
86      // Apply logarithmic mapping to distribute LED count more evenly
87      double logScaledMean = log10(1 + (double)mean / MAX_ADC_VALUE * 66)↩
        / log10(10) * LED_COUNT; //66 is the sensitivity factor of our ↩
        logarithmic scale, we played with it and this one resulted in a ↩
        total turnoff of the LEDs at exactly 4kHz
88
89      // Calculate LED count based on logarithmic scale
90      int ledCount = (int)logScaledMean;
91
92      // Ensure LED count is within the valid range
93      if (ledCount > LED_COUNT) {
94          ledCount = LED_COUNT;
95      } else if (ledCount < 0) {
96          ledCount = 0;
97      }
98
99      // Create LED bitmask using bitwise right shift
100     int ledMask = (0xFF >> (LED_COUNT - ledCount));
101
```

```
102     return ledMask;
103 }
104
105
106 void adcIntHandler(void){
107     /*
108      *Function that saves the inputs and when the buffer reaches 50 values, convolves the input signal with the impulse from the filter.
109      *To do the convolution, according to the formula in the Aufgabenstellung
110      *it sums the multiplication of the filtercoefficients from the matlab filter and the input values saved in the buffer
111      */
112
113   uint32_t adcInputValue;
114   ADCSequenceDataGet(ADC0_BASE,3,&adcInputValue);
115
116   bufferSample[sampleIndex] = adcInputValue*adcInputValue;
117
118   sampleIndex++;
119
120   if(sampleIndex==FILTERORDER){
121       int32_t i; //index
122
123       //Sum all multiplications to do the convolution
124       for(i=0; i<=FILTERORDER;i++){
125           output += bufferSample[FILTERORDER-i]*num[i];
126       }
127
128       // Determine the maximum ADC value reached in the current buffer iteration
129           for (i = 0; i < FILTERORDER; i++) {
130               if (bufferSample[i] > maxADCValue) {
131                   maxADCValue = bufferSample[i];
132               }
133           }
134           MAX_ADC_VALUE = maxADCValue;  // Update MAX_ADC_VALUE
135
136     // update the LEDs
137     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, getLEDvalue((int)output/FILTERORDER));
138     sampleIndex = 0;
139     output = 0;
140
141   }
142
143   // am Ende von adcIntHandler, Interrupt-Flag loeschen
144    ADCIntClear(ADC0_BASE,3);
145 }
```

Quellcode 3: Aufgabe 2: Digitale FIR-Filterung

Die Vorgehensweise in dieser Aufgabe war sehr ähnlich wie in Aufgabe 1. Ein Unterschied war die Implementierung der Frequenzabhängigkeit, die wir durch eine Faltung als Multiplikation der Werte mit den Koeffizienten (siehe Code:2) eines Matlab Filters (siehe Abbildung:1) gemacht haben.

2.3 Erläuterung

Wie bereits in Aufgabenteil 2.1 erklärt, haben wir für diese Aufgabe die Faltungsformel der Aufgabenstellung benutzt.

$$y[n] = s[n] * g[n] = \sum_{l=-\infty}^{\infty} s[n-l]g[l] \tag{1}$$

In der getLEDvalue Funktion wurde der Empfindlichkeitsfaktor auf 66 gestellt, da mit diesem Wert alle LEDs bei genau 4kHz ausgeschaltet sind.

3 Aufgabe 3: Diskrete Fourier-Transformation

In dieser Aufgabe wird ein Programm konzipiert, dass das dominierende Frequenzband von $0Hz$ bis $4kHz$ mit unseren 8 LEDs visualisiert. Für die Verarbeitung des Signales wird eine DFT (Diskrete Fourier Transformation) benutzt, sowie eine Maximumssuche, deren Funktionsweise in den nächsten Aufgabenteilen erklärt wird.

3.1 Frequenzauflösung

In diesem Aufgabenteil soll die Frage: "Welche Frequenzauflösung der diskreten Abtastwerte S[k] ergeben sich mit der gegebenen Abtastrate und Fensterbreite?"geantwortet werden.

Die Frequenzauflösung Δf der DFT wird mit folgender Formel berechnet:

$$\Delta f = \frac{f_A}{N} \tag{2}$$

Das bedeutet in unserem Fall mit $f_A = 44kHz$ und $N = 440$:

$$\Delta f = \frac{4000}{40} = 100Hz \tag{3}$$

3.2 Maximumssuche

In dieser Aufgabe wird nun die Maximumssuche programmiert, sodass das dominierende Frequenzband gefunden und über die LEDs angezeigt wird. Wir müssen den Frequenzbereich von 0 bis $4kHz$ darstellen. Unter Betrachtung der Funktionsweise einer diskreten Fourier Transformation erkennt man, dass die Frequenzbänder mit einem Abstand gleich der Frequenzauflösung $\Delta f = 100Hz$ liegen. Für uns bedeutet das konkret in dem gesuchten Bereich, dass wir K = 40 Frequenzbänder benötigen: $K = \frac{f_{max}}{\Delta f} = 40$. Wie in der Aufgabenstellung beschrieben wurde sie Stack Size deutlich erhöht.
Nach dem Quelltext ist eine Erklärung der Funktionen zu finden:

```
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 #define N 440 // Abgetastete Werte
16 #define K 40 // DFT DurchlÃ¤ufe
17
18 //Funktionen-Deklarationen
19 void adcIntHandler(void);
20 void setup(void);
21 // hier nach Bedarf noch weitere Funktionsdeklarationen einfuegen
22
23 // globale Variablen
24 const float DoublePi = 6.283185308;
25 int32_t bufferSample[440];
26 int32_t sampleIndex = 0;
27 int32_t k = 0;
28 int32_t i = 0;
29 float DFTSample[K]; //Array for the transformed values (Important we ↩
       are revising 40 hence k)
30
31
32 // hier nach Bedarf noch weitere globale Variablen einfuegen
33
34 void main(void){ // nicht veraendern!! Bitte Code in adcIntHandler ↩
       einfuegen
35     setup();
36     while(1){}
37 }
38
39 void setup(void){//konfiguriert den Mikrocontroller
40
41     // konfiguriere SystemClock
42     SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|↩
       SYSCTL_XTAL_16MHZ);
43     uint32_t period = SysCtlClockGet()/SAMPLERATE;
44
45     // aktiviere Peripherie
46     SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
47     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
48     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
49     SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
50
51     // aktiviere Gleitkommazahlen-Modul
52     FPUEnable();
53     FPUStackingEnable();
54     FPULazyStackingEnable();
55     FPUFlushToZeroModeSet(FPU_FLUSH_TO_ZERO_EN);
56
57     // konfiguriere GPIO
58     GPIOPinTypeADC(GPIO_PORTE_BASE,GPIO_PIN_2);
59     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);
```

```
60
61      // konfiguriere Timer
62      TimerConfigure(TIMER0_BASE,TIMER_CFG_PERIODIC);
63      TimerLoadSet(TIMER0_BASE, TIMER_A, period - 1);
64      TimerControlTrigger(TIMER0_BASE,TIMER_A,true);
65      TimerEnable(TIMER0_BASE,TIMER_A);
66
67      // konfiguriere ADC
68      ADCClockConfigSet(ADC0_BASE,ADC_CLOCK_RATE_FULL,1);
69      ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_TIMER, 0);
70      ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_CH1|ADC_CTL_IE|↩
    ADC_CTL_END);
71      ADCSequenceEnable(ADC0_BASE, 3);
72      ADCIntClear(ADC0_BASE,3);
73      ADCIntRegister(ADC0_BASE,3,adcIntHandler);
74      ADCIntEnable(ADC0_BASE,3);
75
76  }
77
78  void CalculateDFT(void) {
79      float realValue;
80      float imgValue;
81
82      for (k = 0; k < K; k++) {
83          //Calculate the transformed values until the frequency band 40 ↩
    (K)
84          realValue = 0;
85          imgValue = 0;
86          for (i = 0; i < N; i++) {
87              realValue += bufferSample[i] * cosf(-DoublePi * i * k / N);
88              imgValue += bufferSample[i] * sinf(-DoublePi * i * k / N);
89          }
90          DFTSample[k] = sqrtf(realValue * realValue + imgValue * ↩
    imgValue); //Save the absolute values in the array
91      }
92  }
93
94  int32_t getMaxrange(void) {
95      //Finds the maximum in the desired frequency band. The returned ↩
    value is the shift used as input for the function getLEDvalue
96      float currentMax = 0;
97      int indexMax = 0;
98      int i;
99      CalculateDFT();
100     for (i = 1; i < K; i++) {
101         if (DFTSample[i] > currentMax) { //updates maximum value
102             currentMax = DFTSample[i];
103             indexMax = i;
104         }
105     }
```

```
106     int shift = indexMax / 5;
107     return shift;
108 }
109
110
111 int getLEDvalue(int Shift) {
112     //Shifts according to the maximum and returns the int value for it ↩
        (0 for all that need to be off and 1 for the one that should be on)
113     return 0b00000001 << Shift;
114 }
115
116 void adcIntHandler(void){
117
118     //Get values and save in the ringbuffer
119     uint32_t adcInputValue;
120     ADCSequenceDataGet(ADC0_BASE,3,&adcInputValue);
121     bufferSample[sampleIndex] = adcInputValue;
122     sampleIndex++;
123     sampleIndex %= N - 1;
124
125     if (sampleIndex % 40 == 0) { //Display t+he values with the LEDs
126             int32_t shift = getMaxrange();
127             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, getLEDvalue(shift));
128         }
129
130
131
132
133    ADCIntClear(ADC0_BASE,3);
134 }
```

Quellcode 4: Aufgabe 4: Maximumsuche

Zusätzlich zu $adcInthandler(void)$ wurden andere Funktionen eingeführt.

$getLEDvalue(intShift)$ funktioniert ähnlich wie in den Aufgaben 1 und 2, in diesem Fall ist der Bitwise Shift nach links und gibt aus, welches der LED eine 1 (an) hat und welche aus sind.

Der Shift, der als input von $getLEDvalue$ benutzt wird, wird mit der Funktion $getMaxrange(void)$ berechnet. in dieser Funktion wird das Maximum in den entsprechenden Frequenzbänder berechnet wie in der Aufgabenstellung verlangt.

Die Funktion $CalculateDFT(void)$ ist wo die DFT berechnet wird, hier werden der reele Anteil und imaginärer Anteil getrennt berechnet und anschließend zusammenadiert. Uns ist aufgefallen, dass Wurzelziehen keine große Aufwirkungen auf das angezeigte Ergebnis hat. Wir vermuten die Wurzel wegzulassen könnte Arbeitsspeicher frei machen aber wir haben die

im Quelltext stehen lassen.

3.3 Programmtest

Da wir in unserem Code direkt implementiert haben, dass die LED Werte nicht immer aktualisiert werden, sondern nur dann wenn wir 38 im Ringpuffer erreichen und weil unsere Berechnung des Maximums jedes mal neu aufgerufen wird in form einer Funktion, leuchten die LEDs bei Änderung der Frequenz, also es leuchtet nicht nur eine Photodiode.

3.4 Theoretische Fourier-Transformation des Signals

Um die Fouriertransformation durchzuführen wird die Sinusfunktion $x(t) = sin(2\pi f_0 t)$ mit einer Rechteckfunktion $r(t)$ mit der Breite $\frac{T_0}{2}$ multipliziert. Eine Multiplikation im Zeitbereich entspricht einer Faltung im Frequenzbereich.

$$X_1(f) = F\left\{sin(2\pi f_0 t)\right\} * F\left\{r_{T_0/2}(t - \tfrac{T_0}{4})\right\}$$

$$X_1(f) = [\tfrac{j}{2}(\delta(f + f_0) - \delta(f - f_0))] * [e^{-j2\pi f \frac{T_0}{4}} \cdot \tfrac{T_0}{2} \cdot sinc(f \cdot \tfrac{T_0}{2})]$$

$$X_1(f) = \tfrac{j}{2}[e^{-j2\pi(f+f_0)\frac{T_0}{4}} \cdot \tfrac{T_0}{2} \cdot sinc((f + f_0)\tfrac{T_0}{2}) - e^{-j2\pi(f-f_0)\frac{T_0}{4}} \cdot \tfrac{T_0}{2} \cdot sinc((f - f_0) \cdot \tfrac{T_0}{2})]$$

$$X_1(f) = \tfrac{T_0}{2}\tfrac{j}{2}[e^{-j2\pi(f+f_0)\frac{T_0}{4}} \cdot sinc((f + f_0)\tfrac{T_0}{4}) - e^{-j2\pi(f-f_0)\frac{T_0}{4}} \cdot sinc((f - f_0) \cdot \tfrac{T_0}{2})]$$

Um sicher zu sein, dass die Funktion periodisch ist, wird im Zeitbereich eine Multiplikation mit der Fourier-Transformierten durchgeführt von:

$$x_2(t) = \sum_{n=-\infty}^{\infty} \delta(t - nT_0)$$

$$X_2(f) = F\left\{x_2(t)\right\} = \tfrac{1}{T_0} \sum_{k=-\infty}^{\infty} \delta(f - \tfrac{k}{T_0}) = \tfrac{1}{T_0} \sum_{k=-\infty}^{\infty} \delta(f - kf_0)$$

Also erhalten wir die Fourier-Transformierte $S(f)$:

$$S(f) = X_1(f) \cdot X_2(f)$$

$$S(f) = \tfrac{T_0}{2}\tfrac{j}{2}[e^{-j2\pi(f+f_0)\frac{T_0}{4}} \cdot sinc((f + f_0)\tfrac{T_0}{4}) - e^{-j2\pi(f-f_0)\frac{T_0}{4}} \cdot sinc((f - f_0) \cdot \tfrac{T_0}{2})] \cdot \sum_{k=-\infty}^{\infty} \delta(t - kT_0)$$

3.5 Maximumssuche mit $S[0]$ ignoriert - LED Sequenz

Die gefundene Sequenz war 4,6,2,8,2,4,6,8,6,8. Nach Analyse mit einem Online Tool (Siehe Quelle: [1]) haben wir gefunden, dass eine ziemlich gute genauigkeit vorhanden ist, die

Werte, die ausweichen machen das nur um 1 LED. Diese Ungenauigkeiten könnten an unser gewähltes Fenster oder das Inputsignal vom Computer liegen.

4 Bonusaufgabe: Frequenzsuche

Wir wissen dass unsere LEDs Frequenzbereiche von 500Hz haben (1. LED 0-500Hz). Wir wissen auch nach Aufgabe 3.3, dass unser Code und Aufbau ziemlich genau die Frequenzen abbilden aber manchmal ein LED zu hoch anzeigt. Mit dem wissen können wir nun die Secret Frequenz Testen.

Bei uns leuchten die LEDs 3 und 4, was ein Bereich zwischen $1000Hz - 1500Hz$ für LED 3 und $1500Hz - 2000Hz$ für LED 4 entspricht. Mit dem Vorwissen von Aufgabenteil 3.3 können wir Annehmen, dass der eigentliche Wert des Signals niedrigere Frequenzen hat, etwa um die **500Hz-1500Hz**.

Literaturverzeichnis

[1] `https://www.checkhearing.org/audiospectrum.php`, Abrufdatum: 05.05.2024.

[2] `http://www.starkerstart.uni-frankfurt.de/43759138/FB09-Musikwissenschaften-Richtiges-Zitieren.pdf`, Abrufdatum: 30. November 2016.

[3] `https://de.wikibooks.org/wiki/LaTeX-Kompendium:_FÃijr_Mathematiker`, Abrufdatum: 16.04.2021

[4] Atmel Corporation. 32-bit ATMEL AVR Microcontroller AT32UC3B0256. `http://www.atmel.com/devices/at32uc3b0256.aspx`, Abrufdatum: 15. Oktober 2013.

[5] I. N. Bronštejn, K. A. Semendjajew, G. Musiol und H. Mühlig (Hrsg.). *Taschenbuch der Mathematik*. Verlag Harri Deutsch, Frankfurt am Main, 8. Auflage, 2012.

[6] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. In: *Transactions of the ASME–Journal of Basic Engineering*, Bd. 82 (D), S. 35–45, 1960.