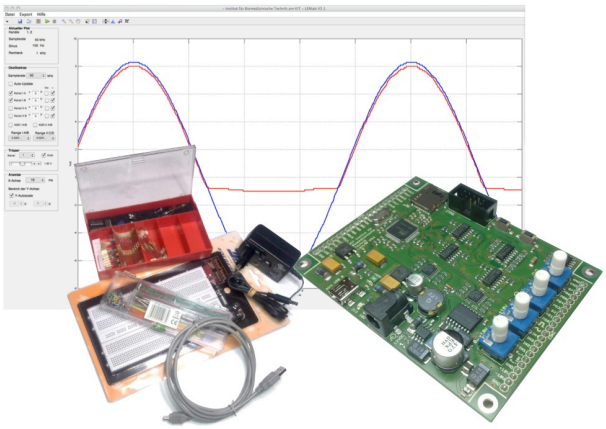
Karlsruher Institut für Technologie

Workshop

Elektrotechnik und Informationstechnik

Kurs 4

Digitale Signalverarbeitung



Gruppe 62

Vorname	Nachname	Matrikel-Nr.	u-Account	E-Mail
Benedikt Johannes	Biehrer	2273070	uqnqk	uqnqk@student.kit.edu
Anselm	Scherr	2281170	unnkn	unnkn@student.kit.edu
Marius Jens	Kleinfeld	2281487	ufkuy	ufkuy@student.kit.edu

9. Juli 2021

Inhaltsverzeichnis

1 Aufgabe 1

Code in der main.c Datei für den Moving-Average-Filter:

```
#include "driverlib/sysctl.h"
#include "driverlib/adc.h"
#include "driverlib/gpio.h"
#include "driverlib/timer.h"

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


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


// globale Variablen
// hier die benÃ¶tigten globalen Variablen, wie den Ringbuffer ↩
   einfuegen

uint32_t ringbuffer[BUFFER_SIZE];
uint32_t insertIndex = 0;
uint32_t tempSum = 0;


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
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
```

```
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);

    // konfiguriere GPIO
    GPIOPinTypeADC(GPIO_PORTE_BASE,GPIO_PIN_2);
    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);

    // konfiguriere Timer
    TimerConfigure(TIMER0_BASE,TIMER_CFG_PERIODIC);
    TimerLoadSet(TIMER0_BASE, TIMER_A, period - 1);
    TimerControlTrigger(TIMER0_BASE,TIMER_A,true);
    TimerEnable(TIMER0_BASE,TIMER_A);

    // konfiguriere ADC
    ADCClockConfigSet(ADC0_BASE,ADC_CLOCK_RATE_FULL,1);
    ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_TIMER, 0);
    ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_CH1|↩
   ADC_CTL_IE|ADC_CTL_END);
    ADCSequenceEnable(ADC0_BASE, 3);
    ADCIntClear(ADC0_BASE,3);
    ADCIntRegister(ADC0_BASE,3,adcIntHandler);
    ADCIntEnable(ADC0_BASE,3);

}




void adcIntHandler (void){
   uint32_t adcInputValue;
   uint32_t adcInputValueQuad;
   uint32_t output;

   ADCSequenceDataGet(ADC0_BASE,3,&adcInputValue);
   //Aufgabe 1 (MA-Filter) von Benedikt Biehrer

   //Input Quadrieren
   adcInputValueQuad = adcInputValue * adcInputValue;

   //insertIndex liegt auf Ã¤ltesten Wert des Buffers
   //also entfernen wir diesen aus der Summe

   tempSum = tempSum - ringBuffer[insertIndex];
```

```
//neuen Wert einfÃ¼gen
ringBuffer[insertIndex] = adcInputValueQuad;

//neunen Wert in Summe einfÃ¼gen
tempSum = tempSum + adcInputValue;

//Index updaten
if(insertIndex < (BUFFERSIZE - 1) ){
    insertIndex++;
}
else if(insertIndex == (BUFFERSIZE - 1) ){
    insertIndex = 0;
}

//Summe mitteln und in Output speichern

output = tempSum * (1 / BUFFERSIZE);



//LEDs:

if(1 < output && output < 10000){
    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);

}
else if (10000 < output && output < 40000)
{
    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0, 0xFF);
    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1 | GPIO_PIN_2
                  |GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5
                  |GPIO_PIN_6 | GPIO_PIN_7, 0x00);

 }
 else if (40000 < output && output <= 70000)
 {
    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1 | GPIO_PIN_2,
                  0xFF);
    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2 | GPIO_PIN_3
                    | GPIO_PIN_4| GPIO_PIN_5 | GPIO_PIN_6
```

```
                    | GPIO_PIN_7, 0x00);

    }
    else if (70000 < output && output <= 100000)
    {
       GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2 | GPIO_PIN_1,
                    0xFF);
       GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3 | GPIO_PIN_4
                     | GPIO_PIN_5| GPIO_PIN_6 | GPIO_PIN_7,
                      0x00);

    }
    else if (100000 < output && output <= 150000)
    {
       GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3 | GPIO_PIN_2
                    | GPIO_PIN_1, 0xFF);
       GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4 | GPIO_PIN_5
                    | GPIO_PIN_6| GPIO_PIN_7, 0x00);

    }
    else if (150000 < output && output <= 200000)
    {
       GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4 | GPIO_PIN_3
                    | GPIO_PIN_2 | GPIO_PIN_1, 0xFF);
       GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5 | GPIO_PIN_6
                    | GPIO_PIN_7, 0x00);

    }
    else if (200000 < output && output <= 300000)
    {
       GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5 | GPIO_PIN_4
                    | GPIO_PIN_3 | GPIO_PIN_2 | GPIO_PIN_1,
                    0xFF);
       GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6 | GPIO_PIN_7,
                    0x00);

    }
    else if (300000 < output && output <= 400000)
    {
       GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6 | GPIO_PIN_5
                      | GPIO_PIN_4 | GPIO_PIN_3 | GPIO_PIN_2
                      | GPIO_PIN_1, 0xFF);
       GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_7, 0x00);
```

```
    }
    else if (400000 < output)
    {
       GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_7 | GPIO_PIN_6
                    | GPIO_PIN_5 | GPIO_PIN_4 | GPIO_PIN_3
                    | GPIO_PIN_2 | GPIO_PIN_1, 0xFF);
    }


   // am Ende von adcIntHandler, Interrupt-Flag loeschen
   ADCIntClear(ADC0_BASE,3);
}
```

:

2 Aufgabe 2

main.c

```
#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"
#include "driverlib/timer.h"
#include "FIR-Filter-50.h"

// Praeprozessor-Makros
#define SAMPLERATE 44000
#define FIR_FILTER_SIZE 50
// Funktionen-Deklarationen
void adcIntHandler(void);
void setup(void);
// hier nach Bedarf noch weitere Funktionsdeklarationen einfuegen


// global variables
int32_t fir_buffer[FIR_FILTER_SIZE];
uint32_t puls = 0;
uint32_t counter = 0;
uint32_t zeiger  = 0;
// hier nach Bedarf noch weitere globale Variablen einfuege


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
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);

    // aktiviere Gleitkommazahlen-Modul
    FPUEnable();
    FPUStackingEnable();
    FPULazyStackingEnable();
    FPUFlushToZeroModeSet(FPU_FLUSH_TO_ZERO_EN);

    // konfiguriere GPIO
    GPIOPinTypeADC(GPIO_PORTE_BASE,GPIO_PIN_2);
    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);

    // konfiguriere Timer
    TimerConfigure(TIMER0_BASE,TIMER_CFG_PERIODIC);
    TimerLoadSet(TIMER0_BASE, TIMER_A, period - 1);
    TimerControlTrigger(TIMER0_BASE,TIMER_A,true);
    TimerEnable(TIMER0_BASE,TIMER_A);

    // konfiguriere ADC
    ADCClockConfigSet(ADC0_BASE,ADC_CLOCK_RATE_FULL,1);
    ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_TIMER, 0);
    ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_CH1|↩
   ADC_CTL_IE|ADC_CTL_END);
    ADCSequenceEnable(ADC0_BASE, 3);
    ADCIntClear(ADC0_BASE,3);
    ADCIntRegister(ADC0_BASE,3,adcIntHandler);
    ADCIntEnable(ADC0_BASE,3);

}

void adcIntHandler(void){

    uint32_t adcInputValue;

    //Pin PE3 auslesen und wert in adcInputValue speichern.
    ADCSequenceDataGet(ADC0_BASE,3,&adcInputValue);
```

```
 //Aufgabe 2 (FIR-Filter) von Anselm Scherr
 uint32_t i = 0;

 //Erst Ring-Buffer befÃ¼llen.
 if(counter < FIR_FILTER_SIZE - 1){

     //Ring-Buffer mit den Werten von Pin PE3 befÃ¼llen.
     fir_buffer[counter] = adcInputValue * adcInputValue;

     //Counter zÃ¤hlt so lange hoch, bis die grÃ¶Ãe des Filter ↩
Array ereicht wurde.
     counter++;
 //Wenn Buffer voll...
 } else {

     //Ring-Buffer mit den Werten von Pin PE3 befÃ¼llen.
     fir_buffer[counter] = adcInputValue * adcInputValue;
     float y = 0;

     //Hier beginnt die eigentliche FIR-Filterung.
     for(i = 0; i < FIR_FILTER_SIZE; i++) {

         //Summe der aus Matlab generierten Werte, ↩
multipliziert mit dem inkrementierenden Wert des Pins PE3.
         y += (ArrayFIR[i] * fir_buffer[(FIR_FILTER_SIZE - 1) -↩
 i]);

     }

     //Zuweisung des FIR-Filter Ergebnis zu puls.
     puls = y;

     //Counter zurÃ¼cksetzen.
     counter = 0;

     //Ton-zu-Led-converter
        //LautstÃ¤rke auf 100 % stellen!
        //puls = adcInputValue * adcInputValue;


             if(1 <= puls && puls < 10000){
                 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);

                }
                else if (10000 < puls && puls < 40000)
                {
                    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0, 0↩
xFF);
                    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1 | ↩
GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 ↩
| GPIO_PIN_7, 0x00);

                }
                else if (40000 < puls && puls <= 70000)
                {
                    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1 | ↩
GPIO_PIN_2, 0xFF);
                    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2 | ↩
GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7,↩
 0x00);

                }
                else if (70000 < puls && puls <= 100000)
                {
                    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2 | ↩
GPIO_PIN_1, 0xFF);
                    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3 | ↩
GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7, 0x00);

                }
                else if (100000 < puls && puls <= 150000)
                {
                    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3 | ↩
GPIO_PIN_2 | GPIO_PIN_1, 0xFF);
                    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4 | ↩
GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7, 0x00);

                }
                else if (150000 < puls && puls <= 200000)
                {
                    GPIOPinWrite(GPIO_PORTB_BASE,
                    GPIO_PIN_4 | GPIO_PIN_3 | GPIO_PIN_2 | ↩
GPIO_PIN_1, 0xFF);
                    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5 | ↩
GPIO_PIN_6 | GPIO_PIN_7, 0x00);
```

```
                }
                else if (200000 < puls && puls <= 300000)
                {
                    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5 | ↩
   GPIO_PIN_4 | GPIO_PIN_3 | GPIO_PIN_2 | GPIO_PIN_1, 0xFF);
                    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6 | ↩
   GPIO_PIN_7, 0x00);

                }
                else if (300000 < puls && puls <= 400000)
                {
                    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6 | ↩
   GPIO_PIN_5 | GPIO_PIN_4 | GPIO_PIN_3 | GPIO_PIN_2 | GPIO_PIN_1,↩
    0xFF);
                    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_7, 0↩
   x00);

                }
                else if (400000 < puls)
                {
                    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_7 | ↩
   GPIO_PIN_6 | GPIO_PIN_5 | GPIO_PIN_4 | GPIO_PIN_3 | GPIO_PIN_2 ↩
   | GPIO_PIN_1, 0xFF);


                }
    }

   // am Ende von adcIntHandler, Interrupt-Flag loeschen
   ADCIntClear(ADC0_BASE,3);
}
```

:

FIR-Filter-50.h

```
/*
 * Filter Coefficients (C Source) generated by the Filter Design ↩
   and Analysis Tool
 * Generated by MATLAB(R) 9.9 and Signal Processing Toolbox 8.5.
 * Generated on: 18-Jun-2021 15:20:12
 */

/*
 * 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\R2020b\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.
 */


const float FirLength = 50;
float  ArrayFIR[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
                    };
```

:

3 Aufgabe 3

Folgender Code wurde für die Aufgabe 3 programmiert:

```
#include <stdint.h>
#include <stdbool.h>
#include <math.h>
#include <stdio.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 "driverlib/fpu.h"

// Praepozessor-Makros
#define SAMPLERATE 44000

//Funktionen-Deklarationen
void adcIntHandler(void);
void setup(void);
// hier nach Bedarf noch weitere Funktionsdeklarationen einfuegen

// globale Variablen
const float DoublePi = 6.283185308;
uint32_t bufferSample[440];
uint32_t adcInputValue;
int counter;
int i;
uint32_t frequenz;
int n;
int k = 0;
float realpart_dft;
float imagpart_dft;
uint32_t output_dft[440];
// hier nach Bedarf noch weitere globale Variablen einfuegen

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

```
void setup(void)
{ //konfiguriert den Mikrocontroller

    // konfiguriere SystemClock
    SysCtlClockSet(
            SYSCTL_SYSDIV_5 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN
                    | SYSCTL_XTAL_16MHZ);
    uint32_t period = SysCtlClockGet() / SAMPLERATE;

    // aktiviere Peripherie
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);

    // aktiviere Gleitkommazahlen-Modul
    FPUEnable();
    FPUStackingEnable();
    FPULazyStackingEnable();
    FPUFlushToZeroModeSet(FPU_FLUSH_TO_ZERO_EN);

    // konfiguriere GPIO
    GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_2);
    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);

    // konfiguriere Timer
    TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
    TimerLoadSet(TIMER0_BASE, TIMER_A, period - 1);
    TimerControlTrigger(TIMER0_BASE, TIMER_A, true);
    TimerEnable(TIMER0_BASE, TIMER_A);

    // konfiguriere ADC
    ADCClockConfigSet(ADC0_BASE, ADC_CLOCK_RATE_FULL, 1);
    ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_TIMER, 0);
    ADCSequenceStepConfigure(ADC0_BASE, 3, 0,
                             ADC_CTL_CH1 | ADC_CTL_IE | ↩
   ADC_CTL_END);
    ADCSequenceEnable(ADC0_BASE, 3);
    ADCIntClear(ADC0_BASE, 3);
```

```
    ADCIntRegister(ADC0_BASE, 3, adcIntHandler);
    ADCIntEnable(ADC0_BASE, 3);

}

void adcIntHandler(void)
{

    /*
     * Bearbeiter  : Marius Jens Kleinfeld
     *
     * Aufgabe 3.1.: Einfuegen der DFT als Funktion in die ISR (↩
   adcIntHandler)
     *
     */

    ADCSequenceDataGet(ADC0_BASE, 3, &adcInputValue);

    if (counter < 440)
    {
        bufferSample[counter] = adcInputValue;  //bufferSample ↩
   fuellen mit "adcInputValue" - Werten
        counter++;
    }
    else
    {
        counter = 0;

        //Berechnung von Real- und Imaginaerteil; jeweils alle ↩
   Werte als Floats deklariert

        for (k = 0; k < 440; k++)
        {
            imagpart_dft = 0;
            realpart_dft = 0;
            for (n = 0; n < 440; n++)
            {
                imagpart_dft += bufferSample[n] * sinf((DoublePi /↩
    440) * n * k);
                realpart_dft += bufferSample[n] * cosf((DoublePi /↩
    440) * n * k);
            }
            output_dft[k] = (uint32_t)sqrtf((realpart_dft * ↩
   realpart_dft) + (imagpart_dft * imagpart_dft));
```

```
        }

        /*
          *
          * Frequenzaufloesung df = SAMPLERATE / 440 = 100Hz
          *
          * Aufgabe 3.2.: LEDs sollen nach jeweilig definierten ↩
    Frequenzband leuchten
          *
          */

            uint32_t max = 0;
        for(i = 1; i < 440; i++)        //Maximumssuche ueber alle↩
     440 Werte
        {
            if(output_dft[i] > max)
            {
                max = output_dft[i];
                frequenz = (SAMPLERATE / 440) * i;  //Berechnung ↩
    der eigentlichen Frequenz mittels der Frequenzaufloesung
            }
        }

        //LEDs den entsprechenden Frequenzbaendern zuordnen

            if (1 <= frequenz && frequenz <= 500)
            {
                GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0, 0xFF);
                GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1 | ↩
    GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 ↩
    | GPIO_PIN_7, 0x00);

            }
            else if (500 < frequenz && frequenz <= 1000)
            {
                GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, 0xFF);
                GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0 | ↩
    GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 ↩
    | GPIO_PIN_7, 0x00);


            }
            else if (1000 < frequenz && frequenz <= 1500)
            {
```

```
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0xFF);
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0 | ↩
GPIO_PIN_1 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 ↩
| GPIO_PIN_7, 0x00);


        }
        else if (1500 < frequenz && frequenz <= 2000)
        {
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0xFF);
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0 | ↩
GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 ↩
| GPIO_PIN_7, 0x00);

        }
        else if (2000 < frequenz && frequenz <= 2500)
        {
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4, 0xFF);
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0 | ↩
GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_5 | GPIO_PIN_6 ↩
| GPIO_PIN_7, 0x00);


        }
        else if (2500 < frequenz && frequenz <= 3000)
        {
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0xFF);
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0 | ↩
GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_6 ↩
| GPIO_PIN_7, 0x00);


        }
        else if (3000 < frequenz && frequenz <= 3500)
        {
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6, 0xFF);
            GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0 | ↩
GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 ↩
| GPIO_PIN_7, 0x00);


        }
        else if (3500 < frequenz && frequenz <= 4000)
        {
```

```
                GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_7, 0xFF);
                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, 0x00);
            }
        }

    // am Ende von adcIntHandler, Interrupt-Flag loeschen
    ADCIntClear(ADC0_BASE, 3);
}
```

:

3.1 Aufgabe 3.3.

Wenn immer nur die erste LED leuchtet, bedeutet das, dass immer nur Frequenzen im untersten Frequenzband berechnet werden. Hier müsste der Quotient in der Frequenzauflösung entsprechend angepasst werden.

3.2 Aufgabe 3.4.

Der Ausgangspunkt der Aufgabe ist eine Sinus - Funktion, wobei hier lediglich die jeweils periodisch vorkommende, positive Halbwelle betrachtet werden soll. Dies fordert die Multipliktion des Betrages des Sinus mit einem Impulskamm. Aufstellen der Rechteckfunktion:

$$rect_{\pi}(t - \frac{\pi}{2}) = \begin{cases} 1, & 0 \leq t \leq \pi \\ 0, & sonst \end{cases}$$

Mit dieser Rechteckfunktion kann aufgrund von $sin(2\pi f_0 t) > 0$ ohne Leckeffekt gefenstert werden. Zur Berechnung der Fourier-Transformierten der periodischen, postiven Halbwellen muss die Rechteckfunktion mit einem Impulskamm gefaltet werden. Eine Faltung im Zeitbereich entspricht einer Multiplikation Frequenzbereich. Die Fourier-Transformierte der Rechteckfunktion und der Impulsfolge lauten:

$$rect_{\pi}(t - \frac{\pi}{2}) \longrightarrow \pi sinc(\pi f) exp(j\pi^2 f_0)$$

$$\sum_{n=-\infty}^{\infty} = \delta\left(t - \left(2n - \frac{3}{2}\right)\pi\right) \qquad \longrightarrow \frac{1}{\pi} \sum_{k=-\infty}^{\infty} \delta\left(f - \frac{2k - \frac{3}{2}}{\pi}\right)$$

Zusammengefasst ergibt sich somit der Rechteckkamm im Frequenzbereich aus der Multiplikation der Fourier-Transformierten der Rechteckfunktion und der Impulsfunktion zu:

$$X(f) = sinc(\pi f) exp(j\pi^2 f_0) \sum_{k=-\infty}^{\infty} \delta \left(f - \frac{2k - \frac{3}{2}}{\pi} \right)$$

Der Sinus quadriert besteht aus lauter positiven Halbwellen. Die Fourier - Transformierte des quadrierten Sinus lautet:

$$y(t) = sin^2(t) \longrightarrow Y(f) = -\frac{1}{2}\sqrt{\frac{\pi}{2}}\delta(f-2) + \sqrt{\frac{\pi}{2}}\delta(f) - \frac{1}{2}\sqrt{\frac{\pi}{2}}\delta(f+2)$$

Die Multiplikation des quadrierten Sinus und des Rechteckkammes im Zeitbereich ergeben eine Faltung im Frequenzbereich.

$$S(f) = X(f) * Y(f)$$

$$= \underbrace{\left(sinc(\pi f) exp(j\pi^2 f_0) \sum_{k=-\infty}^{\infty} \delta \left(f - \frac{2k - \frac{3}{2}}{\pi} \right) \right)}_{1.\ \text{Term}} * \underbrace{\left(-\frac{1}{2}\sqrt{\frac{\pi}{2}}\delta(f-2) + \sqrt{\frac{\pi}{2}}\delta(f) - \frac{1}{2}\sqrt{\frac{\pi}{2}}\delta(f+2) \right)}_{2.\ \text{Term}}$$

Die Faltung der beiden Terme resultiert in einer Einzelfaltung des Termes 1 mit den einzelnen Subtrahenden bzw. des Summandes des Termes 2. Die entstehenden Integrale können mittels der Ausblendeigenschaft des Diracs vereinfacht werden. Hierbei verschieben sich die Komponenten, die abhängig von f sind um 2 und um -2 und für den Summand in Term 2 fällt lediglich $\delta(f)$ weg.

3.3 Aufgabe 3.5.

Wenn der erste Frequenzkoeffizient ignoriert werden soll, muss die for - Schleife für die Maximussuche bei

$$i = 1$$

starten. In diesem Fall wird der erste Wert in dem Array übersprungen. Die sich daraus ergebende Seqeunz der LEDs sieht, bezüglich der im Programmcode verwendeten LED - Frequenzzuweisung, folgendermaßen aus:

GPIO_PIN4, GPIO_PIN7, GPIO_PIN1, GPIO_PIN5,
GPIO_PIN3, GPIO_PIN2, GPIO_PIN4 .

Die restlichen drei Töne in den letzten vier Sekunden der Audiodatei werden nur sporadisch bzw. der letzte gar nicht angezeigt. Sporadisch am Ende der Tonspur aufleuchtend sind in der richtigen Reihenfolge $GPIO\_PIN6$ und $GPIO\_PIN7$, wobei der letzte Ton nicht mehr dargestellt wird.