NATIONAL UNIVERSITY OF SCIENCE AND TECHNOLOGY POLITEHNICA BUCHAREST
Faculty of Electronics, Telecommunications and Information Technology

MES Project

Digital Guitar Effect Pedal using DSP Microcontroller

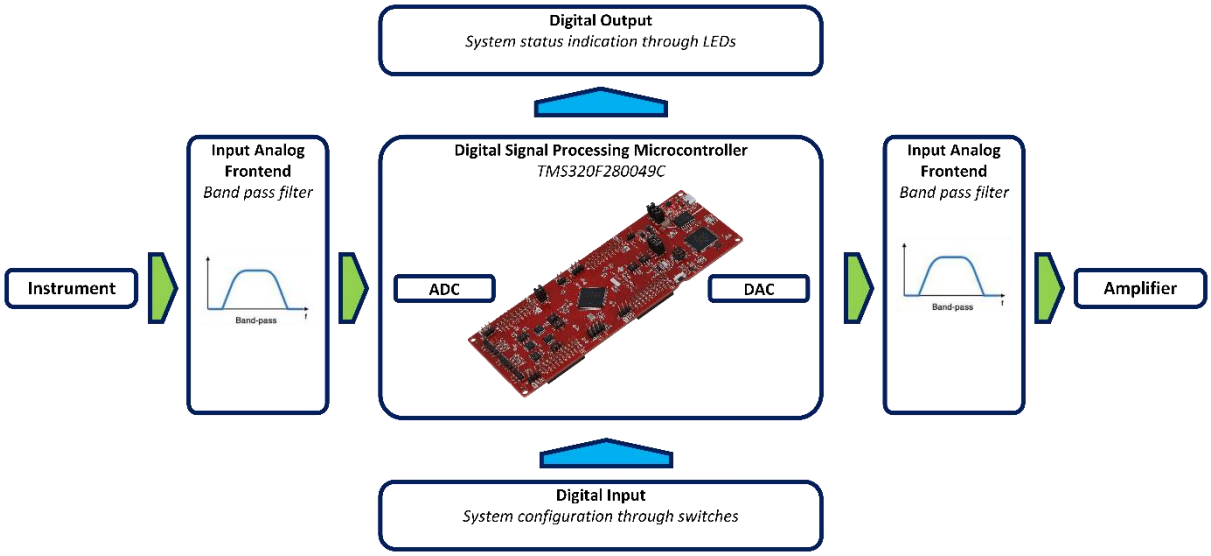
Author: Medinceanu Paul-Cătălin – Advanced Microelectronics

2023-2024

Contents

1. Project description

The objective of this project was to create a digital guitar effect system using a DSP microcontroller. Guitar effects, or guitar pedals, are circuits that modify the sound of the instrument. They can add effects like echo, alter the volume of specific frequencies, or distort the signal. The system developed for this course project incorporates a three-band equalizer and three types of distortion. To adjust the parameters of each effect, a basic interface has been incorporated. Analog frontends are necessary to interface the signal from the instrument to the microcontrollerț's ADC module and, similarly, to interface the signal from the DAC to the amplifier. The system architecture is presented in the figure below.

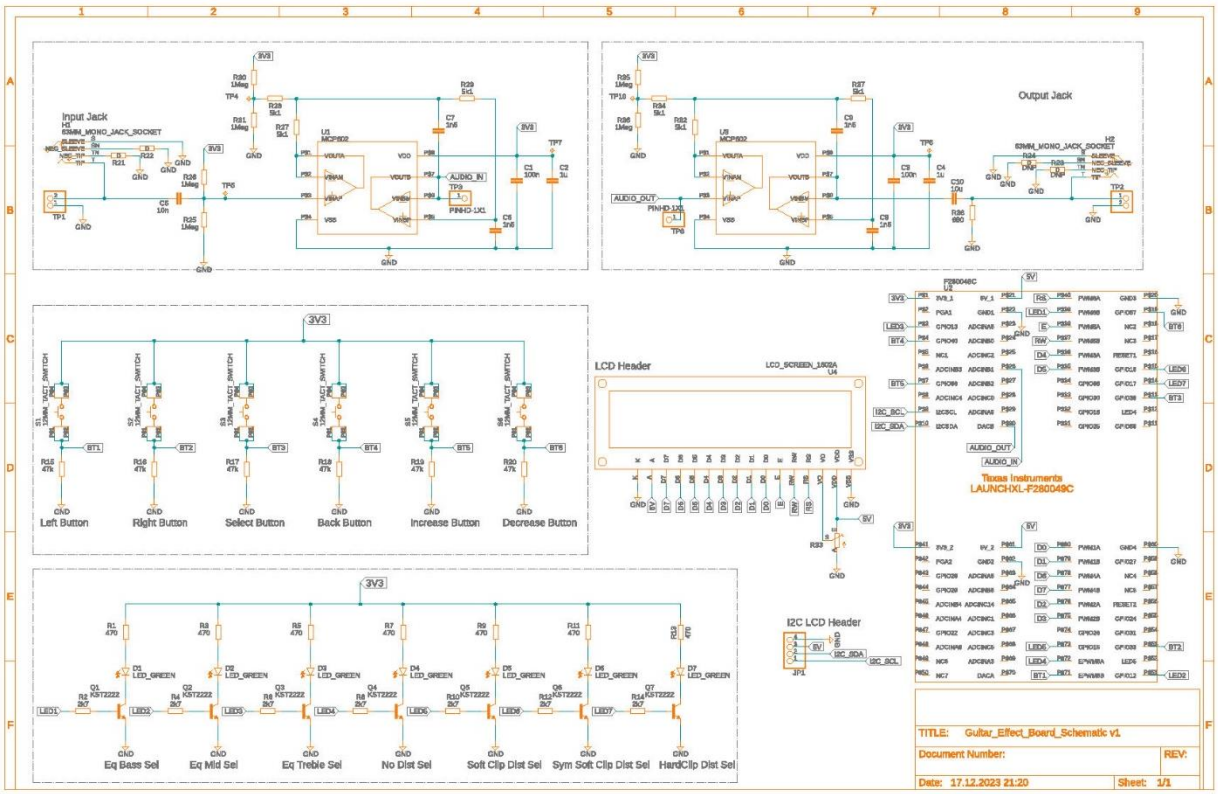


2. Hardware description

The selected microcontroller, F280049C, is part of Texas Instruments' TMS320 family of Digital Signal Processing(DSP) microcontrollers, which are based on their proprietary C2000 CPU architecture. This 32-bit processor stands out due to its floating-point unit and dedicated trigonometric and complex math units. The developed application involves real-time signal processing, and these features played a crucial role in achieving the required processing speed.

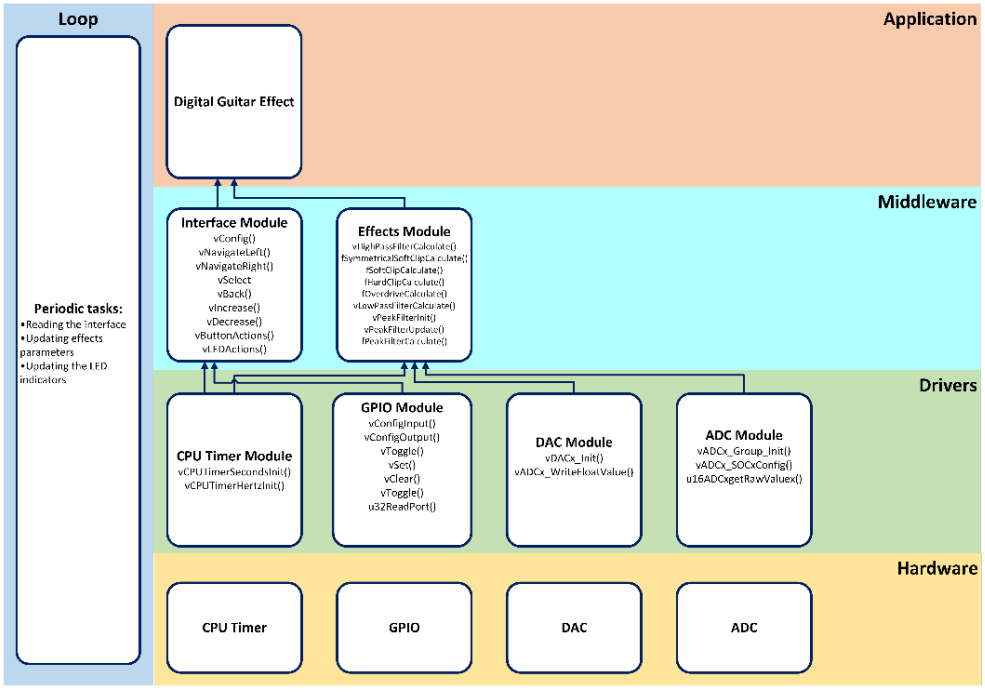
The other relevant peripherals for this project are the 12-bit ADC and 12-bit DAC. To interface the audio signal with these modules, analog frontends are essential. These circuits are implemented using operational amplifiers and are designed to filter out frequencies beyond the audio spectrum and those higher than half of the sampling frequency. These circuits also introduce a common-mode voltage to the AC signal, aligning it with half of the ADC's reference value to fit the varying component of the signal to the input domain.

The system's parameters are adjusted through the physical interface, which is implemented using LED indicators and tactile buttons. The schematic for the whole hardware implementation can be seen below.

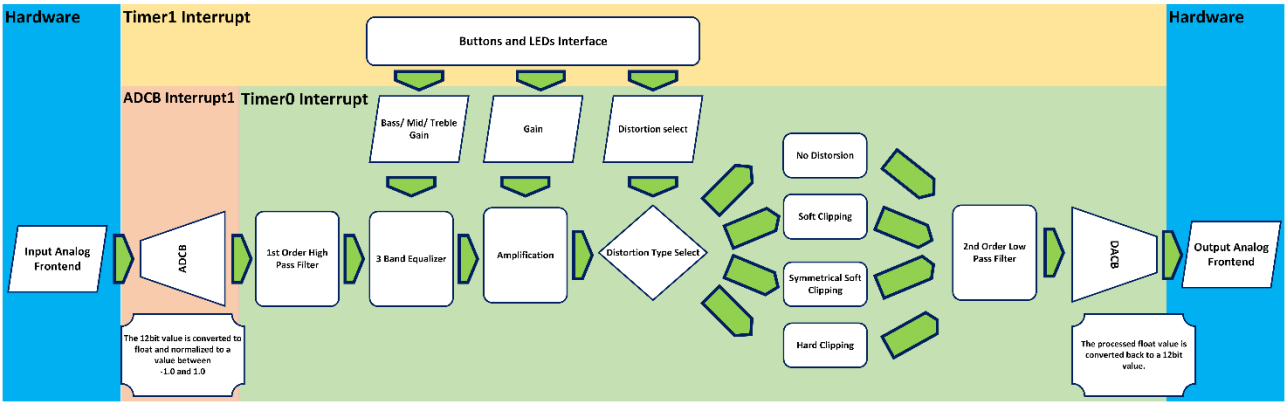


3. Software description

The driver libraries and the fucnctions written for this application are presented in the image below. Their dependencies are marked by the arrows. The chosen programming language is C.



The following diagram presents the signal path, and includes the hardware and software modules.



The input signal is filtered and brought into the ADC's voltage domain by the analog frontends. The ADC conversion is initiated by a timer interupt at the beggining of each sampling period. The ADC end of conversion(EOC) triggers an intterupt during which the conversion result is transformed into a float value normalized between -1.0 and 1.0. This choice was made to simplify the implementation of the effects fucntions. During the same timer interrupt that triggers the ADC, the float value passes through each effect module. The input parameters for these effects are marked at the top of the figure and are set through the interface. The interface can select the path for the distortion stage. The options are: No distortion, Soft Clipping, Symmetrical Soft Clipping and Hard Clipping. At the end of the software effects chain, the float value is converted to an integer value, passed to the DAC module. The continuous signal is sent to another analog interface that filters the signal and removes the DC component for safe interfacimg with the amplifier.

The inputs from the interface and the indicators are read, and respectively updated during another timer interrupt that occurs each 10 milliseconds.

The filters and distortion effects were developed and dimensioned using MATLAB. The filters in particular were initially implemented as continuos transfer functions in the Laplace domain and then converted to the discrete domain using the Tustin transform. To implement them in software, the reverse Z transform was applied, resulting in the discrete-time equations of the filters.

4. Problems and solutions

After testing the system connected to a guitar and an amplifier, I discovered some problems arising from the filters and distortion effects. The peak filters forming the three-band equalizer can become unstable due to their Q factor and bandwidth parameters. Additionally, the distortion functions don't produce the expected sound because they aren't properly configured for the float values obtained after converting the input signal. Both of these issues can be addressed by redesigning the integer-to-float conversion process and reconfiguring the effects.

Another problem is that the produced sound seems "discontinuous." This could be attributed to insufficient runtime for the CPU. In practice, the signal processing is performed on a large number of samples sequentially, and these samples are moved around the memory using the DMA peripheral.

5. Conclusions

Audio signal processing using embedded systems is a field that demands knowledge in both signals and systems, as well as microcontroller architecture. The transition from analog effects to digital effects brings numerous advantages, including the implementation of more efficient functions, the ability to incorportate multiple effects into a system with a smaller footprint, and the configuration of effects with exact vales that do not vary over time due to aging or degradation.

The implementation realized for this project could benefit from more advanced calculation techniques for the filters, utilizing the Direct Acces Memory(DMA) peripheral to apply the processing on an array of values, or employing a high-fidelity audio CODEC instead of the integrated ADCs and DACs.

6. Source code

Driver libraries and headers:

- System parameters:

```
/*
 * SYSPARAM.h
 *
 *  Created on: 20 oct. 2023
 *      Author: Paul
 */


#ifndef SYSPARAM_H_
#define SYSPARAM_H_


/*Defines*/
#define MS_TO_NS 1000000UL
#define SYS_CLK_HZ 100000000UL
#define SYS_PERIOD_NS 10
#define ENABLE 1
#define DISABLE 0
#define UINT32MAX 4294967295
#define BIT12_MAX_VALUE_INT 4095UL
#define BIT12_MAX_VALUE_FLOAT 4095.0
#define UINT16_TO_FLOAT_SLOPE  0.0004884
#define UINT16_TO_FLOAT_OFFSET -1.0
```

```
#define FLOAT_TO_UINT16_SLOPE 2047.5

#define FLOAT_TO_UINT16_OFFSET 2047.5


#define DEBUG


#define SOFTWARE_FILTERS


#define DEBUG_PINS 3

#define DEBUG_PIN1 7

#define DEBUG_PIN2 8

#define DEBUG_PIN3 9


#endif /* SYSPARAM_H_ */
```

- CPU Timer:

```
/*
 * TIMER.h
 *
 *  Created on: 14 oct. 2023
 *      Author: Paul
 */

#ifndef TIMER_H_
#define TIMER_H_

/*Defines*/
#define CPUTIMER0 0
#define CPUTIMER1 1
#define CPUTIMER2 2


/*Structures*/
typedef struct{
    Uint32 u32Period_ms;
    Uint16 u16CPUTimer;
    Uint16 u16CPUTimerInterruptEnable;
}tstCPUTimerSecondsConfig;

typedef struct{
    Uint32 u32Frequency_Hz;
    Uint16 u16CPUTimer;
    Uint16 u16CPUTimerInterruptEnable;
}tstCPUTimerHertzConfig;

/*Structure Initializations*/
//CPUTIMER0
#define TIMER1_SEC_INIT {  300, CPUTIMER1, ENABLE }
#define TIMER1_HEZ_INIT {  96000, CPUTIMER0, ENABLE }
//CPUTIMER1
```

```
#define TIMER2_SEC_INIT {  3000, CPUTIMER1, ENABLE }
#define TIMER2_HEZ_INIT {    1, CPUTIMER1, ENABLE }
//CPUTIMER2
#define TIMER3_SEC_INIT { 100000, CPUTIMER2, DISABLE}
#define TIMER3_HEZ_INIT { 100000, CPUTIMER2, DISABLE}

/*Function prototypes*/
void TIMER_vCPUTimerSecondsInit(tstCPUTimerSecondsConfig CPU);
void TIMER_vCPUTimerHertzInit(tstCPUTimerHertzConfig CPU);
/*Variables*/
extern tstCPUTimerSecondsConfig stCPU1;
extern tstCPUTimerSecondsConfig stCPU2;
extern tstCPUTimerSecondsConfig stCPU3;
extern tstCPUTimerHertzConfig stCPU4;
extern tstCPUTimerHertzConfig stCPU5;
extern tstCPUTimerHertzConfig stCPU6;

#endif /* TIMER_H_ */

/*
 * TIMER.c
 *
 *  Created on: 14 oct. 2023
 *      Author: Paul
 */



/* Project Headers */
#include "F28x_Project.h"

/* System Headerfiles*/
#include <stdlib.h>
#include "SYSPARAM.h"
/* Own Headerfiles */
#include "TIMER.h"
#include "GPIO.h"
//#include "ADC_C.h"
#include "DAC.h"
#include "EFFECTS.h"
#include "INTERFACE.h"

/* Extern Headerfiles */


/* Function Prototypes */
void TIMER_vCPUTimerSecondsInit(tstCPUTimerSecondsConfig CPU);
void TIMER_vCPUTimerHertzInit(tstCPUTimerHertzConfig CPU);
__interrupt void cputimer0_isr();
__interrupt void cputimer1_isr();
__interrupt void cputimer2_isr();

/* Global Variables */
tstCPUTimerSecondsConfig stCPU1 = TIMER1_SEC_INIT;
tstCPUTimerSecondsConfig stCPU2 = TIMER2_SEC_INIT;
tstCPUTimerSecondsConfig stCPU3 = TIMER3_SEC_INIT;
tstCPUTimerHertzConfig stCPU4 = TIMER1_HEZ_INIT;
tstCPUTimerHertzConfig stCPU5 = TIMER2_HEZ_INIT;
```

```
tstCPUTimerHertzConfig stCPU6 = TIMER3_HEZ_INIT;
/*Notes*/
/*
INT1.7 - Timer0(Through PIE)
INT13 - Timer1(Directly connected)
INT14 - Timer2(Directly connected)
*/
void TIMER_vCPUTimerSecondsInit(tstCPUTimerSecondsConfig CPU)
{
    EALLOW;
    Uint64 u64TBPRD;
    Uint16 u16PRESCALER = 0;
    Uint16 u16LPRESCALER = 0;//Low bits of divider
    Uint16 u16HPRESCALER = 0;//High bits of divider
    u64TBPRD = ( CPU.u32Period_ms * MS_TO_NS ) / 2 / SYS_PERIOD_NS;
    while(u64TBPRD > UINT32MAX)
    {
         if(u16PRESCALER == 0)
             u16PRESCALER = 1;
         else
             u16PRESCALER = u16PRESCALER + 1;
         u64TBPRD = ( CPU.u32Period_ms * MS_TO_NS ) / 2 / ( SYS_PERIOD_NS *
u16PRESCALER );
    }

    u16LPRESCALER = u16PRESCALER & 0x00FF;
    u16HPRESCALER = (u16PRESCALER & 0xFF00)>>8;

    switch(CPU.u16CPUTimer)
        {
            case CPUTIMER0:
                CpuTimer0Regs.TCR.bit.TSS = 1; // Stop Timer
                CpuTimer0Regs.PRD.all = u64TBPRD;     // Set the Period
Register
                CpuTimer0Regs.TPR.bit.TDDR = u16LPRESCALER; // Set prescaler
registers to 0
                CpuTimer0Regs.TPRH.bit.TDDRH = u16HPRESCALER ; // Set
prescaler high registers to 0
                CpuTimer0Regs.TCR.bit.TRB = 1; // Reload Timer
                CpuTimer0Regs.TCR.bit.SOFT = 1;
                CpuTimer0Regs.TCR.bit.FREE = 1;
                CpuTimer0Regs.TCR.bit.TIF = 1; // Clear Interrupt Flag
                CpuTimer0Regs.TCR.bit.TIE = CPU.u16CPUTimerInterruptEnable;
// Enable/Disable Timer Interrupts
                if(CPU.u16CPUTimerInterruptEnable)
                {
                    PieCtrlRegs.PIEIER1.bit.INTx7 = 1;
                    PieVectTable.TIMER0_INT = &cputimer0_isr;
                }
                CpuTimer0Regs.TCR.bit.TSS = 0; // Start Timer
            break;

            case CPUTIMER1:
                CpuTimer1Regs.TCR.bit.TSS = 1; // Stop Timer
                CpuTimer1Regs.PRD.all = u64TBPRD;     // Set the Period
Register
                CpuTimer1Regs.TPR.bit.TDDR = u16LPRESCALER; // Set prescaler
registers to 0
```

```
                CpuTimer1Regs.TPRH.bit.TDDRH = u16HPRESCALER; // Set
prescaler high registers to 0
                CpuTimer1Regs.TCR.bit.TRB = 1; // Reload Timer
                CpuTimer1Regs.TCR.bit.SOFT = 1;
                CpuTimer1Regs.TCR.bit.FREE = 1;
                CpuTimer1Regs.TCR.bit.TIF = 1; // Clear Interrupt Flag
                CpuTimer1Regs.TCR.bit.TIE = CPU.u16CPUTimerInterruptEnable;
// Disable Timer Interrupts
                if(CPU.u16CPUTimerInterruptEnable)PieVectTable.TIMER1_INT =
&cputimer1_isr;
                CpuTimer1Regs.TCR.bit.TSS = 0; // Start Timer
            break;

            case CPUTIMER2:
                CpuTimer2Regs.TCR.bit.TSS = 1; // Stop Timer
                CpuTimer2Regs.PRD.all = u64TBPRD;     // Set the Period
Register
                CpuTimer2Regs.TPR.bit.TDDR= u16LPRESCALER; // Set prescaler
registers to 0
                CpuTimer2Regs.TPRH.bit.TDDRH = u16HPRESCALER; // Set
prescaler high registers to 0
                CpuTimer2Regs.TCR.bit.TRB = 1; // Reload Timer
                CpuTimer2Regs.TCR.bit.SOFT = 1;
                CpuTimer2Regs.TCR.bit.FREE = 1;
                CpuTimer2Regs.TCR.bit.TIF = 1; // Clear Interrupt Flag
                CpuTimer2Regs.TCR.bit.TIE = CPU.u16CPUTimerInterruptEnable;
// Disable Timer Interrupts
                if(CPU.u16CPUTimerInterruptEnable)PieVectTable.TIMER2_INT =
&cputimer2_isr;
                CpuTimer2Regs.TCR.bit.TSS = 0; // Start Timer
            break;


        }
    EDIS;
}

void TIMER_vCPUTimerHertzInit(tstCPUTimerHertzConfig CPU)
{
    EALLOW;
    Uint64 u64TBPRD;
    Uint16 u16PRESCALER = 0;
    Uint16 u16LPRESCALER = 0;//Low bits of divider
    Uint16 u16HPRESCALER = 0;//High bits of divider
    u64TBPRD =  SYS_CLK_HZ  / CPU.u32Frequency_Hz;
    while(u64TBPRD > UINT32MAX)
    {
         if(u16PRESCALER == 0)
             u16PRESCALER = 1;
         else
             u16PRESCALER = u16PRESCALER + 1;
         u64TBPRD =  SYS_CLK_HZ  / ( u16PRESCALER * CPU.u32Frequency_Hz );
    }

    u16LPRESCALER = u16PRESCALER & 0x00FF;
    u16HPRESCALER = (u16PRESCALER & 0xFF00)>>8;

    switch(CPU.u16CPUTimer)
        {
```

```
            case CPUTIMER0:
                CpuTimer0Regs.TCR.bit.TSS = 1; // Stop Timer
                CpuTimer0Regs.PRD.all = u64TBPRD;     // Set the Period
Register
                CpuTimer0Regs.TPR.bit.TDDR = u16LPRESCALER; // Set prescaler
registers to 0
                CpuTimer0Regs.TPRH.bit.TDDRH = u16HPRESCALER ; // Set
prescaler high registers to 0
                CpuTimer0Regs.TCR.bit.TRB = 1; // Reload Timer
                CpuTimer0Regs.TCR.bit.SOFT = 1;
                CpuTimer0Regs.TCR.bit.FREE = 1;
                CpuTimer0Regs.TCR.bit.TIF = 1; // Clear Interrupt Flag
                CpuTimer0Regs.TCR.bit.TIE = CPU.u16CPUTimerInterruptEnable;
// Enable/Disable Timer Interrupts
                if(CPU.u16CPUTimerInterruptEnable)
                {
                    PieCtrlRegs.PIEIER1.bit.INTx7 = 1;
                    PieVectTable.TIMER0_INT = &cputimer0_isr;
                }
                CpuTimer0Regs.TCR.bit.TSS = 0; // Start Timer
            break;

            case CPUTIMER1:
                CpuTimer1Regs.TCR.bit.TSS = 1; // Stop Timer
                CpuTimer1Regs.PRD.all = u64TBPRD;     // Set the Period
Register
                CpuTimer1Regs.TPR.bit.TDDR = u16LPRESCALER; // Set prescaler
registers to 0
                CpuTimer1Regs.TPRH.bit.TDDRH = u16HPRESCALER; // Set
prescaler high registers to 0
                CpuTimer1Regs.TCR.bit.TRB = 1; // Reload Timer
                CpuTimer1Regs.TCR.bit.SOFT = 1;
                CpuTimer1Regs.TCR.bit.FREE = 1;
                CpuTimer1Regs.TCR.bit.TIF = 1; // Clear Interrupt Flag
                CpuTimer1Regs.TCR.bit.TIE = CPU.u16CPUTimerInterruptEnable;
// Disable Timer Interrupts
                if(CPU.u16CPUTimerInterruptEnable)PieVectTable.TIMER1_INT =
&cputimer1_isr;
                CpuTimer1Regs.TCR.bit.TSS = 0; // Start Timer
            break;

            case CPUTIMER2:
                CpuTimer2Regs.TCR.bit.TSS = 1; // Stop Timer
                CpuTimer2Regs.PRD.all = u64TBPRD;     // Set the Period
Register
                CpuTimer2Regs.TPR.bit.TDDR= u16LPRESCALER; // Set prescaler
registers to 0
                CpuTimer2Regs.TPRH.bit.TDDRH = u16HPRESCALER; // Set
prescaler high registers to 0
                CpuTimer2Regs.TCR.bit.TRB = 1; // Reload Timer
                CpuTimer2Regs.TCR.bit.SOFT = 1;
                CpuTimer2Regs.TCR.bit.FREE = 1;
                CpuTimer2Regs.TCR.bit.TIF = 1; // Clear Interrupt Flag
                CpuTimer2Regs.TCR.bit.TIE = CPU.u16CPUTimerInterruptEnable;
// Disable Timer Interrupts
                if(CPU.u16CPUTimerInterruptEnable)PieVectTable.TIMER2_INT =
&cputimer2_isr;
                CpuTimer2Regs.TCR.bit.TSS = 0; // Start Timer
            break;
```

```
        }
    EDIS;
}

//Time0 ISR
__interrupt void cputimer0_isr(void)
{

    #ifdef DEBUG
    GPIO_vSet(DEBUG_PIN1);
    #endif

    #ifdef SOFTWARE_FILTERS
    fBuffer2 = EFFECTS_fHighPassFilterCalculate(&stHPF1, fBuffer1) ;
    fBuffer3 = EFFECTS_fPeakFilterCalculate(&stBass, fBuffer2);
    fBuffer4 = EFFECTS_fPeakFilterCalculate(&stMid, fBuffer3);
    fBuffer5 = EFFECTS_fPeakFilterCalculate(&stTreble, fBuffer4);
    fBuffer6 = EFFECTS_fOverdriveCalculate(fOverdriveGain, fBuffer5,
u16DistortionSelect);
    fBuffer7 = EFFECTS_fLowPassFilterButterCalculate(&stLPF1, fBuffer6) *
FLOAT_TO_UINT16_SLOPE + FLOAT_TO_UINT16_OFFSET;
    DAC_vDACBWriteFloatValue(fBuffer7);
    #else
    fBuffer2 = EFFECTS_fPeakFilterCalculate(&stBass, fBuffer1);
    fBuffer3 = EFFECTS_fPeakFilterCalculate(&stMid, fBuffer2);
    fBuffer4 = EFFECTS_fPeakFilterCalculate(&stTreble, fBuffer3);
    fBuffer5 = EFFECTS_fOverdriveCalculate(fOverdriveGain, fBuffer4,
u16DistortionSelect)  * FLOAT_TO_UINT16_SLOPE + FLOAT_TO_UINT16_OFFSET;
    DAC_vDACBWriteFloatValue(fBuffer5);
    #endif

    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1; // Acknowledge Interrupt Group
    CpuTimer0Regs.TCR.bit.TIF = 1; // Clear Interrupt Flag

    #ifdef DEBUG
    GPIO_vClear(DEBUG_PIN1);
    #endif
}

//Timer1 ISR
__interrupt void cputimer1_isr(void)
{

    #ifdef DEBUG
    GPIO_vSet(DEBUG_PIN2);
    #endif

    INTERFACE_vButtonActions();
    INTERFACE_vLEDActions();
    CpuTimer1Regs.TCR.bit.TIF = 1; // Clear Interrupt Flag

    #ifdef DEBUG
    GPIO_vClear(DEBUG_PIN2);
    #endif
}

//Timer2 ISR
```

```
__interrupt void cputimer2_isr(void)
{
   CpuTimer2Regs.TCR.bit.TIF = 1; // Clear Interrupt Flag
}
```

- GPIO:

```
/*
 * GPIO.h
 *
 *  Created on: 14 oct. 2023
 *      Author: Paul
 */

#ifndef GPIO_H_
#define GPIO_H_

/*Defines*/
//#define LED1 23
//#define LED2 34

#define OUTPUT 1
#define INPUT 0

#define QUAL_SYNC 0
#define QUAL_3SPL 1
#define QUAL_6SPL 2
#define QUAL_ASYNC 3

#define PORTA 0
#define PORTB 1
/*Structures*/
/*Function prototypes*/
void GPIO_vConfigInput(const Uint16 u16PinList[], Uint16 u16Pins);
void GPIO_vConfigOutput(const Uint16 u16PinList[], Uint16 u16Pins);
void GPIO_vToggleMultiple(const Uint16 u16PinList[], Uint16 u16Pins);
void GPIO_vSetMultiple(const Uint16 u16PinList[], Uint16 u16Pins);
void GPIO_vClearMultiple(const Uint16 u16PinList[], Uint16 u16Pins);
void GPIO_vToggle(const Uint16 u16Pin);
void GPIO_vSet(const Uint16 u16Pin);
void GPIO_vClear(const Uint16 u16Pin);
void GPIO_vSetPort(const Uint16 u16FirstBit, Uint32 u32Data);
void GPIO_vClearPort(const Uint16 u16FirstBit, Uint32 u32Data);
Uint16 GPIO_u16ReadPin(const Uint16 u16Pin);
Uint16 GPIO_u16ReadPins(const Uint16 u16PinList[], Uint16 u16Pins);
Uint32 GPIO_u32ReadPort(Uint16 u16Port);
void GPIO_vI2CPins();
/*Variables*/
/*
extern const Uint16 u16PinList1[32];
extern const Uint16 u16PinList2[32];
extern const Uint16 u16PinList3[32];
extern const Uint16 u16PinList4[32];
extern const Uint16 u16PinList5[32];
extern const Uint16 u16PinList6[32];
*/


#endif /* GPIO_H_ */
```

```
/*
 * GPIO.c
 *
 *  Created on: 14 oct. 2023
 *      Author: Paul
 */

/* Project Headers */
#include "F28x_Project.h"

/* System Headers*/
#include <stdlib.h>
#include "SYSPARAM.h"
/* Own Headers */
#include "GPIO.h"

/* External Headers */

/* Function Prototypes */
void GPIO_vConfigInput(const Uint16 u16PinList[], Uint16 u16Pins);
void GPIO_vConfigOutput(const Uint16 u16PinList[], Uint16 u16Pins);
void GPIO_vToggleMultiple(const Uint16 u16PinList[], Uint16 u16Pins);
void GPIO_vSetMultiple(const Uint16 u16PinList[], Uint16 u16Pins);
void GPIO_vClearMultiple(const Uint16 u16PinList[], Uint16 u16Pins);
void GPIO_vToggle(const Uint16 u16Pin);
void GPIO_vSet(const Uint16 u16Pin);
void GPIO_vClear(const Uint16 u16Pin);
void GPIO_vSetPort(const Uint16 u16FirstBit, Uint32 u32Data);
void GPIO_vClearPort(const Uint16 u16FirstBit, Uint32 u32Data);
Uint16 GPIO_u16ReadPin(const Uint16 u16Pin);
Uint16 GPIO_u16ReadPins(const Uint16 u16PinList[], Uint16 u16Pins);
Uint32 GPIO_u32ReadPort(Uint16 u16Port);
void  GPIO_vI2CPins();
/* Global Variables */

void GPIO_vI2CPins()
{
    EALLOW;
    /*Set SCL and SDA as outputs*/
    GpioCtrlRegs.GPBDIR.bit.GPIO37 = OUTPUT;//SCL
    GpioCtrlRegs.GPBDIR.bit.GPIO35 = OUTPUT;//SDA
    GpioCtrlRegs.GPBQSEL1.bit.GPIO37 = QUAL_ASYNC;
    GpioCtrlRegs.GPBQSEL1.bit.GPIO35 = QUAL_ASYNC;
    /*Enable Pull-ups*/
    GpioCtrlRegs.GPBPUD.bit.GPIO37 = 0;
    GpioCtrlRegs.GPBPUD.bit.GPIO35 = 0;
    /*MUX to SDA and SCL*/
    /*Higher 2 bits*/
    GpioCtrlRegs.GPBGMUX1.bit.GPIO35 = 0;
    GpioCtrlRegs.GPBMUX1.bit.GPIO35 = 0;
    /*Lower 2 bits*/
    GpioCtrlRegs.GPBGMUX1.bit.GPIO37 = 3;
    GpioCtrlRegs.GPBMUX1.bit.GPIO37 = 3;
    EDIS;
}

void GPIO_vConfigInput(const Uint16 u16PinList[], Uint16 u16Pins)
{
    EALLOW;
```

```
    volatile Uint32 u32MaskA = 0;
    volatile Uint32 u32MaskB = 0;
    volatile Uint16 i = 0;
    for ( i = 0; i < u16Pins; i++)
    {
        if( u16PinList[i] < 32 )
        {
            u32MaskA = u32MaskA | ( 1UL << u16PinList[i] );
        }
        else
            u32MaskB = u32MaskB | ( 1UL << ( u16PinList[i] % 32 ) );
    }
    //Disable pull-up
    GpioCtrlRegs.GPAPUD.all |= u32MaskA;// 0 PU en, 1 PU dis
    GpioCtrlRegs.GPBPUD.all |= u32MaskB;

    //GPIO Direction Input
    GpioCtrlRegs.GPADIR.all &= ~( u32MaskA );
    GpioCtrlRegs.GPBDIR.all &= ~( u32MaskB );


    EDIS;
}

void GPIO_vConfigOutput(const Uint16 u16PinList[], Uint16 u16Pins)
{
    EALLOW;

    volatile Uint32 u32MaskA = 0;
    volatile Uint32 u32MaskB = 0;
    volatile Uint16 i = 0;
    for ( i = 0; i < u16Pins; i++)
    {
        if( u16PinList[i] < 32 )
        {
            u32MaskA = u32MaskA | ( 1UL << u16PinList[i] );
        }
        else
            u32MaskB = u32MaskB | ( 1UL << ( u16PinList[i] % 32 ) );
    }
    //GPIO Direction Output
    GpioCtrlRegs.GPADIR.all |=  u32MaskA;
    GpioCtrlRegs.GPBDIR.all |=  u32MaskB;

    EDIS;
}

void GPIO_vToggleMultiple(const Uint16 u16PinList[], Uint16 u16Pins)
{
    EALLOW;

    volatile Uint32 u32MaskA = 0;
    volatile Uint32 u32MaskB = 0;
    volatile Uint16 i = 0;
    for ( i = 0; i < u16Pins; i++)
    {
        if( u16PinList[i] < 32 )
        {
```

```
            u32MaskA = u32MaskA | ( 1UL << u16PinList[i] );
        }
        else
            u32MaskB = u32MaskB | ( 1UL << ( u16PinList[i] % 32 ) );
    }
    //GPIO Toggle Output
    GpioDataRegs.GPATOGGLE.all |=  u32MaskA;
    GpioDataRegs.GPBTOGGLE.all |=  u32MaskB;

    EDIS;
}

void GPIO_vSetMultiple(const Uint16 u16PinList[], Uint16 u16Pins)
{
    EALLOW;

    volatile Uint32 u32MaskA = 0;
    volatile Uint32 u32MaskB = 0;
    volatile Uint16 i = 0;
    for ( i = 0; i < u16Pins; i++)
    {
        if( u16PinList[i] < 32 )
        {
            u32MaskA = u32MaskA | ( 1UL << u16PinList[i] );
        }
        else
            u32MaskB = u32MaskB | ( 1UL << ( u16PinList[i] % 32 ) );
    }
    //GPIO Toggle Output
    GpioDataRegs.GPASET.all |=  u32MaskA;
    GpioDataRegs.GPBSET.all |=  u32MaskB;

    EDIS;
}

void GPIO_vClearMultiple(const Uint16 u16PinList[], Uint16 u16Pins)
{
    EALLOW;

    volatile Uint32 u32MaskA = 0;
    volatile Uint32 u32MaskB = 0;
    volatile Uint16 i = 0;
    for ( i = 0; i < u16Pins; i++)
    {
        if( u16PinList[i] < 32 )
        {
            u32MaskA = u32MaskA | ( 1UL << u16PinList[i] );
        }
        else
            u32MaskB = u32MaskB | ( 1UL << ( u16PinList[i] % 32 ) );
    }
    //GPIO Toggle Output
    GpioDataRegs.GPACLEAR.all |=  u32MaskA;
    GpioDataRegs.GPBCLEAR.all |=  u32MaskB;

    EDIS;
}

void GPIO_vToggle(const Uint16 u16Pin)
```

```
{
    EALLOW;

    volatile Uint32 u32MaskA = 0;
    volatile Uint32 u32MaskB = 0;
    if( u16Pin < 32 )
    {
        u32MaskA = u32MaskA | ( 1UL << u16Pin );
    }
    else
        u32MaskB = u32MaskB | ( 1UL << ( u16Pin% 32 ) );
    //GPIO Toggle Output
    GpioDataRegs.GPATOGGLE.all |=  u32MaskA;
    GpioDataRegs.GPBTOGGLE.all |=  u32MaskB;

    EDIS;
}

void GPIO_vSet(const Uint16 u16Pin)
{
    EALLOW;

    volatile Uint32 u32MaskA = 0;
    volatile Uint32 u32MaskB = 0;
    if( u16Pin < 32 )
    {
        u32MaskA = u32MaskA | ( 1UL << u16Pin );
    }
    else
        u32MaskB = u32MaskB | ( 1UL << ( u16Pin % 32 ) );
    //GPIO Toggle Output
    GpioDataRegs.GPASET.all |=  u32MaskA;
    GpioDataRegs.GPBSET.all |=  u32MaskB;

    EDIS;
}

void GPIO_vClear(const Uint16 u16Pin)
{
    EALLOW;

    volatile Uint32 u32MaskA = 0;
    volatile Uint32 u32MaskB = 0;
    if( u16Pin < 32 )
    {
        u32MaskA = u32MaskA | ( 1UL << u16Pin );
    }
    else
        u32MaskB = u32MaskB | ( 1UL << ( u16Pin % 32 ) );
    //GPIO Toggle Output
    GpioDataRegs.GPACLEAR.all |=  u32MaskA;
    GpioDataRegs.GPBCLEAR.all |=  u32MaskB;

    EDIS;
}
void GPIO_vSetPort(const Uint16 u16FirstBit, Uint32 u32Data)
{
    EALLOW;
    volatile Uint32 u32Mask = u32Data;
```

```
    if(u16FirstBit != 0)u32Mask = u32Data << ( u16FirstBit % 32 );
    if(u16FirstBit < 32)
    {
        GpioDataRegs.GPASET.all |= u32Mask;
    }
    else
        GpioDataRegs.GPBSET.all |= u32Mask;

    EDIS;
}
void GPIO_vClearPort(const Uint16 u16FirstBit, Uint32 u32Data)
{
    EALLOW;
    volatile Uint32 u32Mask = u32Data;
    if(u16FirstBit != 0)u32Mask = u32Data << ( u16FirstBit % 32 );
    if(u16FirstBit < 32)
    {
        GpioDataRegs.GPACLEAR.all |= u32Mask;
    }
    else
        GpioDataRegs.GPBCLEAR.all |= u32Mask;

    EDIS;
}

Uint16 GPIO_u16ReadPin(const Uint16 u16Pin)
{
    volatile Uint32 u32MaskA = 0;
    volatile Uint32 u32MaskB = 0;
    if( u16Pin < 32 )
    {
        u32MaskA = u32MaskA | ( 1UL << u16Pin );
        if(u32MaskA == ( u32MaskA & GpioDataRegs.GPADAT.all ) )
            return 1;
        else
            return 0;
    }
    else
    {
        u32MaskB = u32MaskB | ( 1UL << ( u16Pin % 32 ) );
        if(u32MaskB == ( u32MaskB & GpioDataRegs.GPBDAT.all ) )
            return 1;
        else
            return 0;
    }

}

Uint16 GPIO_u16ReadPins(const Uint16 u16PinList[], Uint16 u16Pins)
{
    volatile Uint16 i = 0;
    volatile Uint16 u16ReadPins = 0;
    for ( i = 0; i < u16Pins; i++)
    {
        if( GPIO_u16ReadPin(u16PinList[i]) ) u16ReadPins = u16ReadPins | ( 1U
<< i );
    }
    return u16ReadPins;
}
```

```
Uint32 GPIO_u32ReadPort(Uint16 u16Port)
{
    if( u16Port == PORTA)
        return GpioDataRegs.GPADAT.all;
    else
        return GpioDataRegs.GPBDAT.all;
}
```

- DAC:

```
/*
 * DAC.h
 *
 *  Created on: 21 oct. 2023
 *      Author: Paul
 */

#ifndef DAC_H_
#define DAC_H_

/*Defines*/

#define REFERENCE_VDAC  0
#define REFERENCE_VREF  1

/*Which EPWM Period will update the DAC*/
#define EPWM1SYNCPER 0
#define EPWM2SYNCPER 1
#define EPWM3SYNCPER 2
#define EPWM4SYNCPER 3
#define EPWM5SYNCPER 4
#define EPWM6SYNCPER 5
/*Update on SYSCLK or on SYNCSEL*/
#define SYSCLK 0
#define SYNCSELCLK 1

#define GAIN1 0
#define GAIN2 1

#define MAX12BIT 4095U

/*Structures*/
typedef struct{
    Uint16 u16Update;
    Uint16 u16UpdateSrc;
    Uint16 u16Value;
    float fSlope;
    int16 i16Offset;
}tstDACData;

/*Structure Initializations*/
#define DAC1_INIT {SYSCLK, 0, 0, 1, 0}
#define DAC2_INIT {SYSCLK, 0, 0, 1, 0}

/*Function prototypes*/
void DAC_vDACAInit(tstDACData DACData);
void DAC_vDACBInit(tstDACData DACData);
void DAC_vDACAWriteValue(tstDACData DACData);
void DAC_vDACBWriteValue(tstDACData DACData);
```

```
void DAC_vDACAWriteFloatValue(float Value);
void DAC_vDACBWriteFloatValue(float Value);

/*Variables*/
extern tstDACData stDACData1;
extern tstDACData stDACData2;

#endif /* DAC_H_ */

/*
 * DAC.c
 *
 *  Created on: 21 oct. 2023
 *      Author: Paul
 */

#include "F28x_Project.h"

/* System Headefiles*/
#include <stdlib.h>

/* Own Headerfiles */
#include "SYSPARAM.h"
#include "DAC.h"

/* External Headers */

/* Function Prototypes */
void DAC_vDACAInit(tstDACData DACData);
void DAC_vDACBInit(tstDACData DACData);
void DAC_vDACAWriteValue(tstDACData DACData);
void DAC_vDACBWriteValue(tstDACData DACData);
void DAC_vDACAWriteFloatValue(float Value);
void DAC_vDACBWriteFloatValue(float Value);
/* Global Variables */
tstDACData stDACData1 = DAC1_INIT;
tstDACData stDACData2 = DAC2_INIT;

void DAC_vDACAInit(tstDACData DACData)
{
    EALLOW;
    /*DACA*/
    CpuSysRegs.PCLKCR16.bit.DAC_A = ENABLE;
    AnalogSubsysRegs.ANAREFCTL.bit.ANAREFASEL = ADC_INTERNAL;
    AnalogSubsysRegs.ANAREFCTL.bit.ANAREFA2P5SEL = ADC_VREF3P3;
    DacaRegs.DACCTL.bit.LOADMODE = DACData.u16Update;
    DacaRegs.DACCTL.bit.SYNCSEL = DACData.u16UpdateSrc;
    DacaRegs.DACCTL.bit.MODE = GAIN2;
    DacaRegs.DACCTL.bit.DACREFSEL = REFERENCE_VREF;
    DacaRegs.DACOUTEN.bit.DACOUTEN = ENABLE;
    DELAY_US(100);
    EDIS;
}

void DAC_vDACBInit(tstDACData DACData)
{
    EALLOW;
    /*DACB*/
    CpuSysRegs.PCLKCR16.bit.DAC_B = ENABLE;
```

```
    AnalogSubsysRegs.ANAREFCTL.bit.ANAREFBSEL = ADC_INTERNAL;
    AnalogSubsysRegs.ANAREFCTL.bit.ANAREFB2P5SEL = ADC_VREF3P3;
    /*B and C References bonded together!*/
    AnalogSubsysRegs.ANAREFCTL.bit.ANAREFCSEL = ADC_INTERNAL;
    AnalogSubsysRegs.ANAREFCTL.bit.ANAREFC2P5SEL = ADC_VREF3P3;
    DacbRegs.DACCTL.bit.LOADMODE = DACData.u16Update;
    DacbRegs.DACCTL.bit.SYNCSEL = DACData.u16UpdateSrc;
    DacbRegs.DACCTL.bit.MODE = GAIN2;
    DacbRegs.DACCTL.bit.DACREFSEL = REFERENCE_VREF;
    DacbRegs.DACOUTEN.bit.DACOUTEN = ENABLE;
    DELAY_US(100);
    EDIS;
}

void DAC_vDACAWriteValue(tstDACData DACData)
{
    DacaRegs.DACVALS.bit.DACVALS = DACData.u16Value * DACData.fSlope +
DACData.i16Offset;
}

void DAC_vDACBWriteValue(tstDACData DACData)
{
    DacbRegs.DACVALS.bit.DACVALS = DACData.u16Value * DACData.fSlope +
DACData.i16Offset;
}

void DAC_vDACAWriteFloatValue(float Value)
{
    DacaRegs.DACVALS.bit.DACVALS = (Uint16)Value;
}
void DAC_vDACBWriteFloatValue(float Value)
{
    DacbRegs.DACVALS.bit.DACVALS = (Uint16)Value;
}
```

- ADC:

```
/*
 * ADC_B.h
 *
 *  Created on: 21 oct. 2023
 *      Author: Paul
 */

#ifndef ADC_B_H_
#define ADC_B_H_

/*Defines*/
/*ADC Triggers*/
#define SOFTWARE 0
#define TIMER0 1
#define TIMER1 2
#define TIMER2 3
#define GPIO_EXT 4
#define EPWM1SOCA 5
#define EPWM1SOCB 6
#define EPWM2SOCA 7
#define EPWM2SOCB 8
#define EPWM3SOCA 9
#define EPWM3SOCB 10
```

```
#define EPWM4SOCA 11
#define EPWM4SOCB 12
#define EPWM5SOCA 13
#define EPWM5SOCB 14

/*Select End of Conversion*/
#define EOC0 0
#define EOC1 1
#define EOC2 2
#define EOC3 3
/*Structures*/
typedef struct{
    Uint16 u16ADCChannel;//Input Pin
    Uint16 u16AqWindow;//
    Uint16 u16Interrupt;//Interrupt number 1 - 4
    Uint16 u16TriggerSRC;//Trigger Source(EPWM/CPU)
}tstADCBConfig;

/*Structure Initializations*/
#define ADCB1_INIT {1, 47, 1, TIMER0}
#define ADCB2_INIT {4, 14, 1, EPWM4SOCA}
#define ADCB3_INIT {3, 14, 1, TIMER0}
#define ADCB4_INIT {4, 14, 1, EPWM4SOCA}

void ADC_B_vADCB_Group_Init();
void ADC_B_vADCB_SOC0Config(tstADCBConfig ADC);
void ADC_B_vADCB_SOC1Config(tstADCBConfig ADC);
void ADC_B_vADCB_SOC0Disable();
void ADC_B_vADCB_SOC1Disable();
Uint16 ADC_B_u16ADCBgetRawValue0();
Uint16 ADC_B_u16ADCBgetRawValue1();
/*Variables*/
extern tstADCBConfig stADCB1;
extern tstADCBConfig stADCB2;
extern tstADCBConfig stADCB3;
extern tstADCBConfig stADCB4;



#endif /* ADC_B_H_ */

/*
 * ADC_B.c
 *
 *  Created on: 21 oct. 2023
 *      Author: Paul
 */

/* Project Headers */
#include "F28x_Project.h"

/* System Headers*/
#include <stdlib.h>
#include <ADC_B.h>
/* Own Headers */
#include "SYSPARAM.h"
#include "GPIO.h"
#include "EFFECTS.h"
```

```
/* External Headers */

/* Function Prototypes */
void ADC_B_vADCB_Group_Init();
void ADC_B_vADCB_SOC0Config(tstADCBConfig ADC);
void ADC_B_vADCB_SOC1Config(tstADCBConfig ADC);
Uint16 ADC_B_u16ADCBgetRawValue0();
Uint16 ADC_B_u16ADCBgetRawValue1();
__interrupt void adcb1_isr(void);
__interrupt void adcb2_isr(void);
__interrupt void adcb3_isr(void);
__interrupt void adcb4_isr(void);
/* Global Variables */
tstADCBConfig stADCB1 = ADCB1_INIT;
tstADCBConfig stADCB2 = ADCB2_INIT;
tstADCBConfig stADCB3 = ADCB3_INIT;
tstADCBConfig stADCB4 = ADCB4_INIT;

/*Initialize ADC Group*/
void ADC_B_vADCB_Group_Init()
{
    SetVREF(ADC_ADCB, ADC_INTERNAL, ADC_VREF3P3);
    SetVREF(ADC_ADCC, ADC_INTERNAL, ADC_VREF3P3);
    EALLOW;
    AdcbRegs.ADCCTL2.bit.PRESCALE = 0;//Divide sysclk by 1
    AdcbRegs.ADCCTL1.bit.INTPULSEPOS = 1;
    AdcbRegs.ADCCTL1.bit.ADCPWDNZ = 1;
    PieVectTable.ADCB1_INT = &adcb1_isr;
    PieVectTable.ADCB2_INT = &adcb2_isr;
    PieVectTable.ADCB3_INT = &adcb3_isr;
    PieVectTable.ADCB4_INT = &adcb4_isr;
    EDIS;
    DELAY_US(1000);
}
/*Initialize ADC by SOC*/
void ADC_B_vADCB_SOC0Config(tstADCBConfig ADC)
{
        EALLOW;
        AdcbRegs.ADCSOC0CTL.bit.CHSEL = ADC.u16ADCChannel;  //Read pin
        AdcbRegs.ADCSOC0CTL.bit.ACQPS = ADC.u16AqWindow; //Sample window
        AdcbRegs.ADCSOC0CTL.bit.TRIGSEL = ADC.u16TriggerSRC;
        if(ADC.u16Interrupt == 1)
        {
            AdcbRegs.ADCINTSEL1N2.bit.INT1SEL = EOC0; //end of SOC0 will set
INT1 flag
            AdcbRegs.ADCINTSEL1N2.bit.INT1E = ENABLE;   //enable INT1 flag
            AdcbRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //make sure INT1 flag is
cleared
            PieCtrlRegs.PIEIER1.bit.INTx2 = 1; //ADCB1 Interrupt
        }
        if(ADC.u16Interrupt == 2)
        {
            AdcbRegs.ADCINTSEL1N2.bit.INT2SEL = EOC0; //end of SOC0 will set
INT2 flag
            AdcbRegs.ADCINTSEL1N2.bit.INT2E = ENABLE;   //enable INT2 flag
            AdcbRegs.ADCINTFLGCLR.bit.ADCINT2 = 1; //make sure INT2 flag is
cleared
            PieCtrlRegs.PIEIER10.bit.INTx6 = 1; //ADCB2 Interrupt
        }
```

```
        if(ADC.u16Interrupt == 3)
        {
            AdcbRegs.ADCINTSEL3N4.bit.INT3SEL = EOC0; //end of SOC0 will set
INT3 flag
            AdcbRegs.ADCINTSEL3N4.bit.INT3E = ENABLE;   //enable INT3 flag
            AdcbRegs.ADCINTFLGCLR.bit.ADCINT3 = 1; //make sure INT3 flag is
cleared
            PieCtrlRegs.PIEIER10.bit.INTx7 = 1; //ADCB3 Interrupt
        }
        if(ADC.u16Interrupt == 4)
        {
            AdcbRegs.ADCINTSEL3N4.bit.INT4SEL = EOC0; //end of SOC0 will set
INT4 flag
            AdcbRegs.ADCINTSEL3N4.bit.INT4E = ENABLE;   //enable INT4 flag
            AdcbRegs.ADCINTFLGCLR.bit.ADCINT4 = 1; //make sure INT4 flag is
cleared
            PieCtrlRegs.PIEIER10.bit.INTx8 = 1; //ADCB3 Interrupt
        }

        EDIS;
}

/*Disable the Trigger of the SOC*/
void ADC_B_vADCB_SOC0Disable()
{
    EALLOW;
    AdcbRegs.ADCSOC0CTL.bit.TRIGSEL = DISABLE;
    EDIS;
}

/*Initialize ADC by SOC*/
void ADC_B_vADCB_SOC1Config(tstADCBConfig ADC)
{
        EALLOW;
        AdcbRegs.ADCSOC1CTL.bit.CHSEL = ADC.u16ADCChannel;  //SOC will
convert pin A0
        AdcbRegs.ADCSOC1CTL.bit.ACQPS = ADC.u16AqWindow; //sample window is
100 SYSCLK cycles
        AdcbRegs.ADCSOC1CTL.bit.TRIGSEL = ADC.u16TriggerSRC;
        if(ADC.u16Interrupt==1)
        {
            AdcbRegs.ADCINTSEL1N2.bit.INT1SEL = EOC1; //end of SOC will set
INT1 flag
            AdcbRegs.ADCINTSEL1N2.bit.INT1E = ENABLE;  //enable INT1 flag
            AdcbRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //make sure INT1 flag is
cleared
            PieCtrlRegs.PIEIER1.bit.INTx2 = 1; //ADCB1 Interrupt
        }
        if(ADC.u16Interrupt==2)
        {
            AdcbRegs.ADCINTSEL1N2.bit.INT2SEL = EOC1; //end of SOC0 will set
INT1 flag
            AdcbRegs.ADCINTSEL1N2.bit.INT2E = ENABLE;  //enable INT1 flag
            AdcbRegs.ADCINTFLGCLR.bit.ADCINT2 = 1; //make sure INT1 flag is
cleared
            PieCtrlRegs.PIEIER10.bit.INTx6 = 1; //ADCB2 Interrupt
        }
        if(ADC.u16Interrupt==3)
        {
```

```
            AdcbRegs.ADCINTSEL3N4.bit.INT3SEL = EOC1; //end of SOC0 will set
INT1 flag
            AdcbRegs.ADCINTSEL3N4.bit.INT3E = ENABLE;;   //enable INT1 flag
            AdcbRegs.ADCINTFLGCLR.bit.ADCINT3 = 1; //make sure INT1 flag is
cleared
            PieCtrlRegs.PIEIER10.bit.INTx7 = 1; //ADCB3 Interrupt
        }
        if(ADC.u16Interrupt==4)
        {
            AdcbRegs.ADCINTSEL3N4.bit.INT4SEL = EOC1; //end of SOC0 will set
INT1 flag
            AdcbRegs.ADCINTSEL3N4.bit.INT4E = ENABLE;   //enable INT1 flag
            AdcbRegs.ADCINTFLGCLR.bit.ADCINT4 = 1; //make sure INT1 flag is
cleared
            PieCtrlRegs.PIEIER10.bit.INTx8 = 1; //ADCB3 Interrupt
        }

        EDIS;
}

/*Disable the Trigger of the SOC*/
void ADC_B_vADCB_SOC1Disable()
{
    EALLOW;
    AdcbRegs.ADCSOC1CTL.bit.TRIGSEL = DISABLE;
    EDIS;
}

Uint16 ADC_B_u16ADCBgetRawValue0()
{
    return AdcbResultRegs.ADCRESULT0;
}

Uint16 ADC_B_u16ADCBgetRawValue1()
{
    return AdcbResultRegs.ADCRESULT1;
}

//ADCB Interrupt 1
__interrupt void adcb1_isr(void)
{

    #ifdef DEBUG
    GPIO_vSet(DEBUG_PIN3);
    #endif

    fBuffer1 = ( (float)ADC_B_u16ADCBgetRawValue0() ) * UINT16_TO_FLOAT_SLOPE
+ UINT16_TO_FLOAT_OFFSET;
    AdcbRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //clear INT1 flag
    if(1 == AdcbRegs.ADCINTOVF.bit.ADCINT1)
    {
        AdcbRegs.ADCINTOVFCLR.bit.ADCINT1 = 1; //clear INT1 overflow flag
        AdcbRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //clear INT1 flag
    }
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;

    #ifdef DEBUG
    GPIO_vClear(DEBUG_PIN3);
    #endif
```

```
}

//ADCB Interrupt 2
__interrupt void adcb2_isr(void)
{
    AdcbRegs.ADCINTFLGCLR.bit.ADCINT2 = 1; //clear INT1 flag
    if(1 == AdcbRegs.ADCINTOVF.bit.ADCINT2)
    {
        AdcbRegs.ADCINTOVFCLR.bit.ADCINT2 = 1; //clear INT1 overflow flag
        AdcbRegs.ADCINTFLGCLR.bit.ADCINT2 = 1; //clear INT1 flag
    }
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP10;
}

//ADCB Interrupt 3
__interrupt void adcb3_isr(void)
{
    AdcbRegs.ADCINTFLGCLR.bit.ADCINT3 = 1; //clear INT1 flag
    if(1 == AdcbRegs.ADCINTOVF.bit.ADCINT3)
    {
        AdcbRegs.ADCINTOVFCLR.bit.ADCINT3 = 1; //clear INT1 overflow flag
        AdcbRegs.ADCINTFLGCLR.bit.ADCINT3 = 1; //clear INT1 flag
    }
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP10;
}

//ADCB Interrupt 4
__interrupt void adcb4_isr(void)
{

    AdcbRegs.ADCINTFLGCLR.bit.ADCINT4 = 1; //clear INT1 flag
    if(1 == AdcbRegs.ADCINTOVF.bit.ADCINT4)
        {
            AdcbRegs.ADCINTOVFCLR.bit.ADCINT4 = 1; //clear INT1 overflow flag
            AdcbRegs.ADCINTFLGCLR.bit.ADCINT4 = 1; //clear INT1 flag
        }
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP10;
}
```

Middleware:

- Interface:

```
/*
 * INTERFACE.h
 *
 *  Created on: 19 nov. 2023
 *      Author: Paul
 */
/*
Interface Map

LED1 - GPIO11
LED2 - GPIO12
LED3 - GPIO13
LED4 - GPIO14
LED5 - GPIO15
LED6 - GPIO16
LED7 - GPIO17
```

```
BT1 - GPIO32
BT2 - GPIO33
BT3 - GPIO39
BT4 - GPIO40
BT5 - GPIO56
BT6 - GPIO57

 */
#ifndef INTERFACE_H_
#define INTERFACE_H_

/*Defines*/

#define LED_RED 23
#define LED_GREEN 34

#define INPUTS 6
#define BUTTON_LEFT 32
#define BUTTON_RIGHT 33
#define BUTTON_SELECT 39
#define BUTTON_BACK 40
#define BUTTON_INCREASE 56
#define BUTTON_DECREAE 57

#define OUTPUTS 8
#define BASS 11
#define MID 12
#define TREBLE 13
#define NO_DIST 14
#define SOFT_DIST 15
#define SYM_SOFT_DIST 16
#define HARD_DIST 17

#define LED_BASS 0x0001
#define LED_MID 0x0002
#define LED_TREBLE 0x0004
#define LED_NO_DIST 0x0008
#define LED_SOFT_DIST 0x0010
#define LED_SYM_SOFT_DIST 0x0020
#define LED_HARD_DIST 0x0040

/*Structures*/
/*Function prototypes*/
void INTERFACE_vConfig();
void INTERFACE_vNavigateLeft();
void INTERFACE_vNavigateRight();
void INTERFACE_vSelect();
void INTERFACE_vBack();
void INTERFACE_vIncrease();
void INTERFACE_vDecease();
void INTERFACE_vButtonActions();
void INTERFACE_vLEDActions();
/*Variables*/
extern const Uint32 u32MaskButtonLeft;
extern const Uint32 u32MaskButtonRight;
extern const Uint32 u32MaskButtonSelect;
extern const Uint32 u32MaskButtonBack;
extern const Uint32 u32MaskButtonIncrease;
extern const Uint32 u32MaskButtonDecrease;
```

```
const extern Uint32 u32MaskButtons;
extern Uint16 u16InputPinList[];
extern Uint16 u16OutputPinList[];
extern Uint16 u16ButtonPanel;
extern Uint16 u16LEDPanel;
extern Uint16 u16UpdateBass;
extern Uint16 u16UpdateMid;
extern Uint16 u16UpdateTreble;



#endif /* INTERFACE_H_ */

/*
 * INTERFACE.c
 *
 *  Created on: 19 nov. 2023
 *      Author: Paul
 */

/* Project Headers */
#include "F28x_Project.h"

/* System Headerfiles*/
#include <stdlib.h>
#include "SYSPARAM.h"
/* Own Headerfiles */
#include "INTERFACE.h"
#include "GPIO.h"
#include "EFFECTS.h"

/* Extern Headerfiles */

/* Function Prototypes */
void INTERFACE_vConfig();
void INTERFACE_vNavigateLeft();
void INTERFACE_vNavigateRight();
void INTERFACE_vSelect();
void INTERFACE_vBack();
void INTERFACE_vIncrease();
void INTERFACE_vDecease();
void INTERFACE_vButtonActions();
void INTERFACE_vLEDActions();
/* Global Variables */
const Uint32  u32MaskButtonLeft = 1UL << ( BUTTON_LEFT % 32 );
const Uint32 u32MaskButtonRight = 1UL << ( BUTTON_RIGHT % 32 );
const Uint32 u32MaskButtonSelect = 1UL << ( BUTTON_SELECT % 32 );
const Uint32 u32MaskButtonBack = 1UL << ( BUTTON_BACK % 32 );
const Uint32 u32MaskButtonIncrease = 1UL << ( BUTTON_INCREASE % 32 );
const Uint32 u32MaskButtonDecrease = 1UL << ( BUTTON_DECREAE % 32 );
const Uint32 u32MaskButtons = 1UL << ( BUTTON_LEFT % 32 ) | 1UL << (
BUTTON_RIGHT % 32 ) | 1UL << ( BUTTON_SELECT % 32 ) | 1UL << ( BUTTON_BACK %
32 ) | 1UL << ( BUTTON_INCREASE % 32 ) | 1UL << ( BUTTON_DECREAE % 32 );
Uint16 u16InputPinList[] = {BUTTON_LEFT, BUTTON_RIGHT, BUTTON_SELECT,
BUTTON_BACK, BUTTON_INCREASE, BUTTON_DECREAE };
Uint16 u16OutputPinList[] = {BASS, MID, TREBLE, NO_DIST, SOFT_DIST,
SYM_SOFT_DIST, HARD_DIST, LED_RED };
Uint16 u16LEDPanel = NO_DIST;
Uint16 u16ButtonPanel = LED_NO_DIST;
```

```
Uint16 u16UpdateBass = 0;
Uint16 u16UpdateMid = 0;
Uint16 u16UpdateTreble = 0;

void INTERFACE_vConfig()
{
    GPIO_vConfigInput(u16InputPinList, INPUTS);
    GPIO_vConfigOutput(u16OutputPinList, OUTPUTS);
}

void INTERFACE_vNavigateLeft()
{
    if((u16ButtonPanel >= LED_BASS) && (u16ButtonPanel < LED_HARD_DIST))
        u16ButtonPanel = u16ButtonPanel << 1UL;
    else
        u16ButtonPanel = LED_HARD_DIST;
}

void INTERFACE_vNavigateRight()
{
    if((u16ButtonPanel > LED_BASS) && (u16ButtonPanel <= LED_HARD_DIST))
        u16ButtonPanel = u16ButtonPanel >> 1UL;
    else
        u16ButtonPanel = LED_BASS;
}

void INTERFACE_vSelect()
{
    if(u16ButtonPanel == LED_NO_DIST)
        u16DistortionSelect = NO_DISTORTION;
    else if(u16ButtonPanel == LED_SOFT_DIST)
        u16DistortionSelect = SOFT_CLIP;
    else if(u16ButtonPanel == LED_SYM_SOFT_DIST)
        u16DistortionSelect = SYM_SOFT_CLIP;
    else if(u16ButtonPanel == LED_HARD_DIST)
        u16DistortionSelect = HARD_CLIP;
}

void INTERFACE_vBack()
{

}

void INTERFACE_vIncrease()
{
    if( (u16ButtonPanel == LED_NO_DIST) || (u16ButtonPanel == LED_SOFT_DIST)
|| (u16ButtonPanel == LED_SYM_SOFT_DIST) || (u16ButtonPanel ==
LED_HARD_DIST))
    {
        if(fOverdriveGain <= GAIN_MAX)
            fOverdriveGain = fOverdriveGain + GAIN_STEP;
        else fOverdriveGain = GAIN_MAX;
    }
    else if(u16ButtonPanel == LED_BASS)
    {
        if(fBassGain <= GAIN_MAX)
        {
            fBassGain = fBassGain + GAIN_STEP;
            u16UpdateBass = 1;
```

```
        }
        else
        {
            fBassGain = GAIN_MAX;
            u16UpdateBass = 1;
        }
    }
    else if(u16ButtonPanel == LED_MID)
    {
        if(fMidGain <= GAIN_MAX)
        {
            fMidGain = fMidGain + GAIN_STEP;
            u16UpdateMid = 1;
        }
        else
        {
            fMidGain = GAIN_MAX;
            u16UpdateMid = 1;
        }
    }
    else if(u16ButtonPanel == LED_TREBLE)
    {
        if(fTrebleGain <= GAIN_MAX)
        {
            fTrebleGain = fTrebleGain + GAIN_STEP;
            u16UpdateTreble = 1;
        }
        else
        {
            fTrebleGain = GAIN_MAX;
            u16UpdateTreble = 1;
        }
    }
}

void INTERFACE_vDecrease()
{
    if( (u16ButtonPanel == LED_NO_DIST) || (u16ButtonPanel == LED_SOFT_DIST)
|| (u16ButtonPanel == LED_SYM_SOFT_DIST) || (u16ButtonPanel ==
LED_HARD_DIST))
    {
        if(fOverdriveGain >= GAIN_MIN)
            fOverdriveGain = fOverdriveGain - GAIN_STEP;
        else fOverdriveGain = GAIN_MIN;
    }
    else if(u16ButtonPanel == LED_BASS)
    {
        if(fBassGain >= GAIN_MIN)
        {
            fBassGain = fBassGain - GAIN_STEP;
            u16UpdateBass = 1;
        }
        else
        {
            fBassGain = GAIN_MIN;
            u16UpdateBass = 1;
        }
    }
    else if(u16ButtonPanel == LED_MID)
```

```
    {
        if(fMidGain >= GAIN_MIN)
        {
            fMidGain = fMidGain - GAIN_STEP;
            u16UpdateMid = 1;
        }
        else
        {
            fMidGain = GAIN_MIN;
            u16UpdateMid = 1;
        }
    }
    else if(u16ButtonPanel == LED_TREBLE)
    {
        if(fTrebleGain >= GAIN_MIN)
        {
            fTrebleGain = fTrebleGain - GAIN_STEP;
            u16UpdateTreble = 1;
        }
        else
        {
            fTrebleGain = GAIN_MIN;
            u16UpdateTreble = 1;
        }
    }
}

void INTERFACE_vButtonActions()
{
    volatile Uint32 u32Port = GPIO_u32ReadPort(PORTB) & u32MaskButtons;
    if(u32Port != 0)
    {

        if(u32Port & u32MaskButtonLeft)
        {
            GPIO_vClear(LED_RED);
            INTERFACE_vNavigateLeft();
            GPIO_vSet(LED_RED);
        }
        else if( (u32Port & u32MaskButtonRight) == u32MaskButtonRight)
        {
            GPIO_vClear(LED_RED);
            INTERFACE_vNavigateRight();
            GPIO_vSet(LED_RED);
        }
        else if( (u32Port & u32MaskButtonSelect) == u32MaskButtonSelect)
        {
            GPIO_vClear(LED_RED);
            INTERFACE_vSelect();
            GPIO_vSet(LED_RED);
        }
        else if( (u32Port & u32MaskButtonBack) == u32MaskButtonBack)
        {
            GPIO_vClear(LED_RED);
            INTERFACE_vBack();
            GPIO_vSet(LED_RED);
        }
        else if( (u32Port & u32MaskButtonIncrease) == u32MaskButtonIncrease)
        {
```

```
            GPIO_vClear(LED_RED);
            INTERFACE_vIncrease();
            GPIO_vSet(LED_RED);
        }
        else if( (u32Port & u32MaskButtonDecrease) == u32MaskButtonDecrease)
        {
            GPIO_vClear(LED_RED);
            INTERFACE_vDecrease();
            GPIO_vSet(LED_RED);
        }
    }
}

void INTERFACE_vLEDActions()
{
    GPIO_vClear(u16LEDPanel);
    switch(u16ButtonPanel)
    {
        case LED_BASS:
            u16LEDPanel = BASS;
        break;

        case LED_MID:
            u16LEDPanel = MID;
        break;

        case LED_TREBLE:
            u16LEDPanel = TREBLE;
        break;

        case LED_NO_DIST:
            u16LEDPanel = NO_DIST;
        break;

        case LED_SOFT_DIST:
            u16LEDPanel = SOFT_DIST;
        break;

        case LED_SYM_SOFT_DIST:
            u16LEDPanel = SYM_SOFT_DIST;
        break;

        case LED_HARD_DIST:
            u16LEDPanel = HARD_DIST;
        break;
    }
    GPIO_vSet(u16LEDPanel);
}
```

- Effects:

```
/*
 * EFFECTS.H
 *
 *  Created on: 29 oct. 2023
 *      Author: Paul
 */

#ifndef EFFECTS_H_
#define EFFECTS_H_
```

```
/*Defines*/
#define SOFT_CLIP_THRESHOLD 1.0

#define HARD_CLIP_THRESHOLD 0.9

#define SAMPLE_PERIOD 1.0417e-5

#define BASS_CENTER_FREQUENCY 77.5
#define BASS_BANDWIDTH 280.0
#define MID_CENTER_FREQUENCY 1095.0
#define MID_BANDWIDTH 3700.0
#define TREBLE_CENTER_FREQUENCY 7746.0
#define TREBLE_BANDWIDTH 11000.0

#define NO_DISTORTION 0
#define SOFT_CLIP 1
#define SYM_SOFT_CLIP 2
#define HARD_CLIP 3

#define GAIN_DEFAULT 1.0
#define GAIN_MAX 3.0
#define GAIN_MIN 0.1
#define GAIN_STEP 0.1

/*Structures*/
typedef struct{
    float fDataInOld;
    float fDataOutOld;
    float fCoeff1;
    float fCoeff2;
    float fCoeff3;
}tstEffectHPF;

typedef struct{
    float fDataInOld1;
    float fDataInOld2;
    float fDataOutOld1;
    float fDataOutOld2;
    float fCoeff1;
    float fCoeff2;
    float fCoeff3;
    float fCoeff4;
    float fCoeff5;
}tstEffectLPF;

typedef struct{
    float fDataInOld1;
    float fDataInOld2;
    float fDataOutOld1;
    float fDataOutOld2;
    float fCoeff1;//a0
    float fCoeff2;//a1
    float fCoeff3;//a2
    float fCoeff4;//b0
    float fCoeff5;//b1
    float fCoeff6;//b2
    float fGain;
    float fBandwidth;
    float fCenterFrequency;
```

```
}tstEffectPeak;

/*Structure Initializations*/

#define HPF1_INIT {  0, 0, 0.9993 ,0.9993, 0.9987 }//fs = 96k

#define LPF1_INIT { 0, 0, 0, 0, 0.2262, 0.4523, 0.2262, 0.2809, 0.1956 }//fs
= 96k



/*Function prototypes*/
void  EFFECTS_vHighPassFilterUpdate(tstEffectHPF *Filter, float Coeff1, float
Coeff2, float Coeff3);
float EFFECTS_fHighPassFilterCalculate(tstEffectHPF *Filter, float DataIn);

float EFFECTS_fSymmetricalSoftClipCalculate(float DataIn);

float EFFECTS_fSoftClipCalculate(float DataIn);

float EFFECTS_fHardClipCalculate(float DataIn);

float EFFECTS_fOverdriveCalculate(float Gain, float DataIn, Uint16
GainSelect);

void  EFFECTS_vLowPassFilterButterUpdate(tstEffectLPF *Filter, float Coeff1,
float Coeff2, float Coeff3, float Coeff4, float Coeff5);
float EFFECTS_fLowPassFilterButterCalculate(tstEffectLPF *Filter, float
DataIn);

void EFFECTS_vPeakFilterInit(tstEffectPeak *Filter, float fCenterFrequency,
float fBandwidth, float fGain);
void EFFECTS_vPeakFilterUpdate(tstEffectPeak *Filter, float fCenterFrequency,
float fBandwidth, float fGain);
float EFFECTS_fPeakFilterCalculate(tstEffectPeak *Filter, float DataIn);

/*Variables*/
extern tstEffectHPF stHPF1;
extern tstEffectHPF stHPF2;

extern float fOverdriveGain;
extern Uint16 u16DistortionSelect;

extern tstEffectLPF stLPF1;
extern tstEffectLPF stLPF2;

extern tstEffectPeak stBass;
extern float fBassGain;
extern tstEffectPeak stMid;
extern float fMidGain;
extern tstEffectPeak stTreble;
extern float fTrebleGain;

extern float fBuffer1;
extern float fBuffer2;
extern float fBuffer3;
extern float fBuffer4;
extern float fBuffer5;
extern float fBuffer6;
```

```
extern float fBuffer7;
#endif /* EFFECTS_H_ */

/*
 * EFFECTS.c
 *
 *  Created on: 29 oct. 2023
 *      Author: Paul
 */
/* Project Headers */
#include "F28x_Project.h"

/* System Headerfiles*/
#include <stdlib.h>
#include "SYSPARAM.h"
/* Own Headerfiles */
#include "EFFECTS.h"
/* Extern Headerfiles */
/* Function Prototypes */
void  EFFECTS_vHighPassFilterUpdate(tstEffectHPF *Filter, float Coeff1, float
Coeff2, float Coeff3);
float EFFECTS_fHighPassFilterCalculate(tstEffectHPF *Filter, float DataIn);

float EFFECTS_fSymmetricalSoftClipCalculate(float DataIn);

float EFFECTS_fSoftClipCalculate(float DataIn);

float EFFECTS_fHardClipCalculate(float DataIn);

float EFFECTS_fOverdriveCalculate(float Gain, float DataIn, Uint16
GainSelect);

void  EFFECTS_vLowPassFilterButterUpdate(tstEffectLPF *Filter, float Coeff1,
float Coeff2, float Coeff3, float Coeff4, float Coeff5);
float EFFECTS_fLowPassFilterButterCalculate(tstEffectLPF *Filter, float
DataIn);

void EFFECTS_vPeakFilterInit(tstEffectPeak *Filter, float fCenterFrequency,
float fBandwidth, float fGain);
void EFFECTS_vPeakFilterUpdate(tstEffectPeak *Filter, float fCenterFrequency,
float fBandwidth, float fGain);
float EFFECTS_fPeakFilterCalculate(tstEffectPeak *Filter, float DataIn);

/* Global Variables */
tstEffectHPF stHPF1 = HPF1_INIT;
tstEffectLPF stLPF1 = LPF1_INIT;
tstEffectPeak stBass;
float fBassGain = GAIN_DEFAULT;
tstEffectPeak stMid;
float fMidGain = GAIN_DEFAULT;
tstEffectPeak stTreble;
float fTrebleGain = GAIN_DEFAULT;

float fBuffer1 = 0.0;
float fBuffer2 = 0.0;
float fBuffer3 = 0.0;
float fBuffer4 = 0.0;
float fBuffer5 = 0.0;
float fBuffer6 = 0.0;
```

```
float fBuffer7 = 0.0;
float fOverdriveGain = GAIN_DEFAULT;
Uint16 u16DistortionSelect = NO_DISTORTION;
/*High Pass Filter*/
void  EFFECTS_vHighPassFilterUpdate(tstEffectHPF *Filter, float Coeff1, float
Coeff2, float Coeff3)
{
   Filter->fCoeff1 = Coeff1;
   Filter->fCoeff2 = Coeff2;
   Filter->fCoeff3 = Coeff3;
}

float EFFECTS_fHighPassFilterCalculate(tstEffectHPF *Filter, float DataIn)
{
    volatile float DataOut;
    DataOut = (Filter->fCoeff1 * DataIn) - (Filter->fCoeff2 * Filter-
>fDataInOld) + (Filter->fCoeff3 * Filter->fDataOutOld);
    Filter->fDataInOld = DataIn;
    Filter->fDataOutOld = DataOut;
    return DataOut;
}
/*Distortion*/
/*
float EFFECTS_fSymmetricalSoftClipCalculate(float DataIn)
{
    volatile float DataOut;
    volatile float IntermediateData;
    IntermediateData = 2.0 - ( 3.0 * DataIn );
    if( ( 0.667 < DataIn ) && ( DataIn <= 0.9 ) )
        DataOut = 0.8;
    else if( ( 0.333 <= DataIn ) && ( DataIn < 0.667 ) )
        DataOut = ( 3.0 - (IntermediateData * IntermediateData) ) / 3.0;
    else if( ( -0.333 <= DataIn ) && ( DataIn < 0.333 ) )
        DataOut = 2.0 * DataIn;
    else if( ( -0.667 <= DataIn ) && ( DataIn < -0.333 ) )
        DataOut = -( 3.0 - (IntermediateData * IntermediateData) ) / 3.0;
    else if( ( -0.9 <= DataIn ) && ( DataIn < -0.667 ) )
        DataOut = -0.8;

    return DataOut;
}
*/

float EFFECTS_fSymmetricalSoftClipCalculate(float DataIn)
{
    volatile float DataOut = 0.0 ;
    volatile float AbsoluteDataIn = __builtin_fabs(DataIn);
    volatile float SignDataIn = ( DataIn >= 0.0) ? 1.0 : -1.0;
    if( AbsoluteDataIn < 0.333){
        DataOut = 2.0 * DataIn;
    }else if( AbsoluteDataIn >= 0.333 && AbsoluteDataIn < 0.667 ){
        DataOut = SignDataIn * ( 3.0 - ( 2.0  - 3.0 *AbsoluteDataIn ) * ( 2.0
- 3.0 *AbsoluteDataIn ) ) / 3.0;
    }else {
        DataOut = SignDataIn;
    }
    return DataOut;
}
float EFFECTS_fSoftClipCalculate(float DataIn)
```

```
{
    volatile float DataOut;
    if( DataIn  <= -SOFT_CLIP_THRESHOLD )
        DataOut = -0.667;
    else if( ( DataIn >= -SOFT_CLIP_THRESHOLD ) && ( DataIn <=
SOFT_CLIP_THRESHOLD ) )
        DataOut = DataIn - ( ( DataIn * DataIn * DataIn ) / 3 );
    else if( DataIn >= SOFT_CLIP_THRESHOLD)
        DataOut = 0.667;
    return DataOut;
}
float EFFECTS_fHardClipCalculate(float DataIn)
{
    volatile float DataOut;
    if( DataIn  <= -HARD_CLIP_THRESHOLD )
        DataOut = -1.0;
    else if( ( DataIn > -HARD_CLIP_THRESHOLD ) && ( DataIn <
HARD_CLIP_THRESHOLD ) )
        DataOut = DataIn;
    else if( DataIn >= HARD_CLIP_THRESHOLD)
        DataOut = 1.0;
    return DataOut;
}

float EFFECTS_fOverdriveCalculate(float Gain, float DataIn, Uint16
GainSelect)
{
   switch( GainSelect )
   {
       case NO_DISTORTION: return ( Gain * DataIn ) ;
       case SOFT_CLIP: return EFFECTS_fSoftClipCalculate( Gain * DataIn );
       case SYM_SOFT_CLIP: return EFFECTS_fSymmetricalSoftClipCalculate( Gain
* DataIn );
       case HARD_CLIP: return EFFECTS_fHardClipCalculate( Gain * DataIn );
       default: return ( Gain * DataIn ) ;
   }
}
/*Low Pass Filter*/
void  EFFECTS_vLowPassFilterButterUpdate(tstEffectLPF *Filter, float Coeff1,
float Coeff2, float Coeff3, float Coeff4, float Coeff5)
{
    Filter->fCoeff1 = Coeff1;
    Filter->fCoeff2 = Coeff2;
    Filter->fCoeff3 = Coeff3;
    Filter->fCoeff4 = Coeff4;
    Filter->fCoeff5 = Coeff5;
}

float EFFECTS_fLowPassFilterButterCalculate(tstEffectLPF *Filter, float
DataIn)
{
    volatile float DataOut;
    DataOut = (Filter->fCoeff1 * DataIn) + (Filter->fCoeff2 * Filter-
>fDataInOld1) + (Filter->fCoeff3 * Filter->fDataInOld2) - (Filter->fCoeff4 *
Filter->fDataOutOld1) - (Filter->fCoeff5 * Filter->fDataOutOld2);
    Filter->fDataInOld2 = Filter->fDataInOld1;
    Filter->fDataOutOld2 = Filter->fDataOutOld1;
    Filter->fDataInOld1 = DataIn;
    Filter->fDataOutOld1 = DataOut;
```

```
    return DataOut;
}
/*Peak Filter*/
void EFFECTS_vPeakFilterUpdate(tstEffectPeak *Filter, float fCenterFrequency,
float fBandwidth, float fGain)
{
   Filter->fGain = fGain;
   Filter->fBandwidth = fBandwidth;
   Filter->fCenterFrequency = fCenterFrequency;
   volatile float Q = fCenterFrequency / fBandwidth;
   volatile float OmegaCenter = 2 * 3.14 * fCenterFrequency;
   Filter->fCoeff1 = 4.0 + ( 2 * fGain / Q * OmegaCenter * SAMPLE_PERIOD ) +
( OmegaCenter * SAMPLE_PERIOD * OmegaCenter * SAMPLE_PERIOD );//a0
   Filter->fCoeff2 = ( 2 * ( OmegaCenter * SAMPLE_PERIOD * OmegaCenter *
SAMPLE_PERIOD ) ) - 8.0;//a1
   Filter->fCoeff3 = 4.0 - ( 2 * fGain / Q * OmegaCenter * SAMPLE_PERIOD ) +
( OmegaCenter * SAMPLE_PERIOD * OmegaCenter * SAMPLE_PERIOD );//a2
   Filter->fCoeff4 = 4.0 + ( 2 / Q * OmegaCenter * SAMPLE_PERIOD ) + (
OmegaCenter * SAMPLE_PERIOD * OmegaCenter * SAMPLE_PERIOD );//b0
   Filter->fCoeff5 = - ( 2 * ( OmegaCenter * SAMPLE_PERIOD * OmegaCenter *
SAMPLE_PERIOD ) - 8.0 );//b1
   Filter->fCoeff6 = - ( 4.0 - ( 2 / Q * OmegaCenter * SAMPLE_PERIOD ) + (
OmegaCenter * SAMPLE_PERIOD * OmegaCenter * SAMPLE_PERIOD ) );//b2
}
void EFFECTS_vPeakFilterInit(tstEffectPeak *Filter, float fCenterFrequency,
float fBandwidth, float fGain)
{
    Filter->fDataInOld1 = 0.0;
    Filter->fDataInOld2 = 0.0;
    Filter->fDataOutOld1 = 0.0;
    Filter->fDataOutOld2 = 0.0;
    EFFECTS_vPeakFilterUpdate(Filter, fCenterFrequency, fBandwidth, fGain);
}

float EFFECTS_fPeakFilterCalculate(tstEffectPeak *Filter, float DataIn)
{
    volatile float DataOut;
    if(Filter->fGain == 1.0)
        return DataIn;
    else
        {
            DataOut = ( ( Filter->fCoeff1 * DataIn ) + ( Filter->fCoeff2 *
Filter->fDataInOld1 ) + ( Filter->fCoeff3 * Filter->fDataInOld2 ) + ( Filter-
>fCoeff5 * Filter->fDataOutOld1 ) + ( Filter->fCoeff6 * Filter->fDataOutOld2
) ) /Filter->fCoeff4;
            Filter->fDataInOld2 = Filter->fDataInOld1;
            Filter->fDataOutOld2 = Filter->fDataOutOld1;
            Filter->fDataInOld1 = DataIn;
            Filter->fDataOutOld1 = DataOut;
            return DataOut;
        }
}
```

Application

- Main:

```
#include "F28x_Project.h"
#include "SYSPARAM.h"
```

```
#include "GPIO.h"
#include "TIMER.h"
#include "ADC_B.h"
#include "DAC.h"
#include "EFFECTS.h"
#include "INTERFACE.h"
void main(void)

{

    InitSysCtrl();
    //
    // Initialize GPIO:
    // This example function is found in the F2837xS_Gpio.c file and
    // illustrates how to set the GPIO to it's default state.
    //
    InitGpio();

    //
    // Clear all interrupts and initialize PIE vector table:
    // Disable CPU interrupts
    //
    DINT;

    //
    // Initialize the PIE control registers to their default state.
    // The default state is all PIE interrupts disabled and flags
    // are cleared.
    // This function is found in the F2837xS_PieCtrl.c file.
    //
    InitPieCtrl();

    //
    // Disable CPU interrupts and clear all CPU interrupt flags:
    //
    IER = 0x0000;
    IFR = 0x0000;

    //
    // Initialize the PIE vector table with pointers to the shell Interrupt
    // Service Routines (ISR).
    // This will populate the entire table, even if the interrupt
    // is not used in this example.  This is useful for debug purposes.
    // The shell ISR routines are found in F2837xS_DefaultIsr.c.
    // This function is found in F2837xS_PieVect.c.
    //
    InitPieVectTable();

    EALLOW;

    EFFECTS_vPeakFilterInit(&stBass, BASS_CENTER_FREQUENCY, BASS_BANDWIDTH,
fBassGain);

    EFFECTS_vPeakFilterInit(&stMid, MID_CENTER_FREQUENCY, MID_BANDWIDTH,
fMidGain);

    EFFECTS_vPeakFilterInit(&stTreble, TREBLE_CENTER_FREQUENCY,
TREBLE_BANDWIDTH, fTrebleGain);
```

```
    ADC_B_vADCB_Group_Init();

    TIMER_vCPUTimerHertzInit(stCPU4);//ADCB SOC Interrupt

    TIMER_vCPUTimerSecondsInit(stCPU1);//Interface Interrupt

    ADC_B_vADCB_SOC0Config(stADCB1);

    DAC_vDACBInit(stDACData1);

    #ifdef DEBUG
    const Uint16 u16DebugPinList[] = {DEBUG_PIN1, DEBUG_PIN2, DEBUG_PIN3};
    GPIO_vConfigOutput(u16DebugPinList, DEBUG_PINS);
    #endif

    INTERFACE_vConfig();

    IER |= M_INT1; //Enable group 1 interrupts, TIMER0 Interrupt
    IER |= M_INT13;// TIMER1 Interrupt
    EINT;
    ERTM;
    EDIS;
    for(;;)
    {
        if(u16UpdateBass)
            {
                EFFECTS_vPeakFilterUpdate(&stBass, BASS_CENTER_FREQUENCY,
BASS_BANDWIDTH, fBassGain);
                u16UpdateBass = 0;
            }

        if(u16UpdateMid)
            {
                EFFECTS_vPeakFilterUpdate(&stMid, MID_CENTER_FREQUENCY,
MID_BANDWIDTH, fMidGain);
                u16UpdateMid = 0;
            }

        if(u16UpdateTreble)
            {
                EFFECTS_vPeakFilterUpdate(&stTreble, TREBLE_CENTER_FREQUENCY,
BASS_BANDWIDTH, fTrebleGain);
                u16UpdateTreble = 0;
            }
    }
}
```

Obs:

- The project presentations will be on 08.01.2024 for ACES and on 12.01.2024 for AM, during the laboratory sessions.
- You can add chapters or subsections to the documentation, if needed.
- You must prepare a short presentation containing the essential information from your project (no more than 10 minutes presentation + questions).

- You must prepare a functional prototype of you project.
- You must create an archive with your word document (also a pdf version), power point presentation and the source code and send it to george.popescu1012@upb.ro until 06.01.2024.
- The maximum grade for the project will be 50 points (half of the final grade).