|  |
| --- |
| 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 2](#_Toc535484335)

[2. Hardware description 2](#_Toc535484336)

[3. Software description 3](#_Toc535484337)

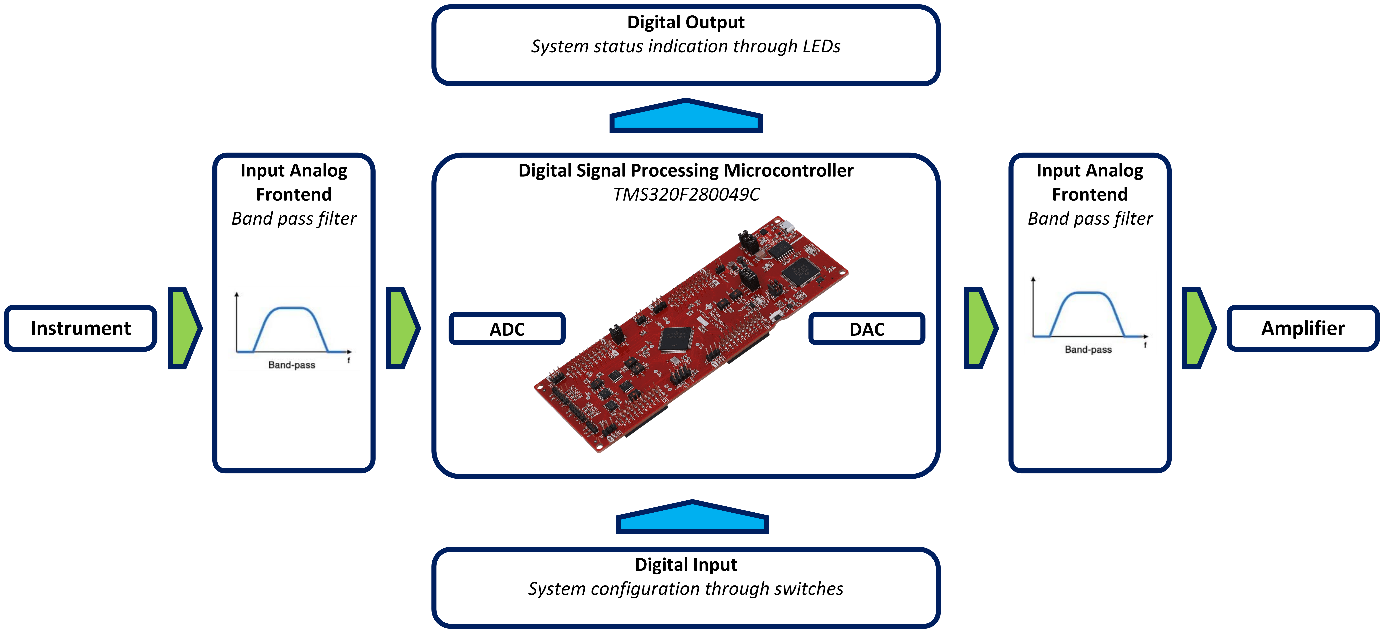
[4. Problems and solutions (optional) 4](#_Toc535484338)

[5. Conclusions 5](#_Toc535484339)

[6. Source code 5](#_Toc535484340)

# 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.*

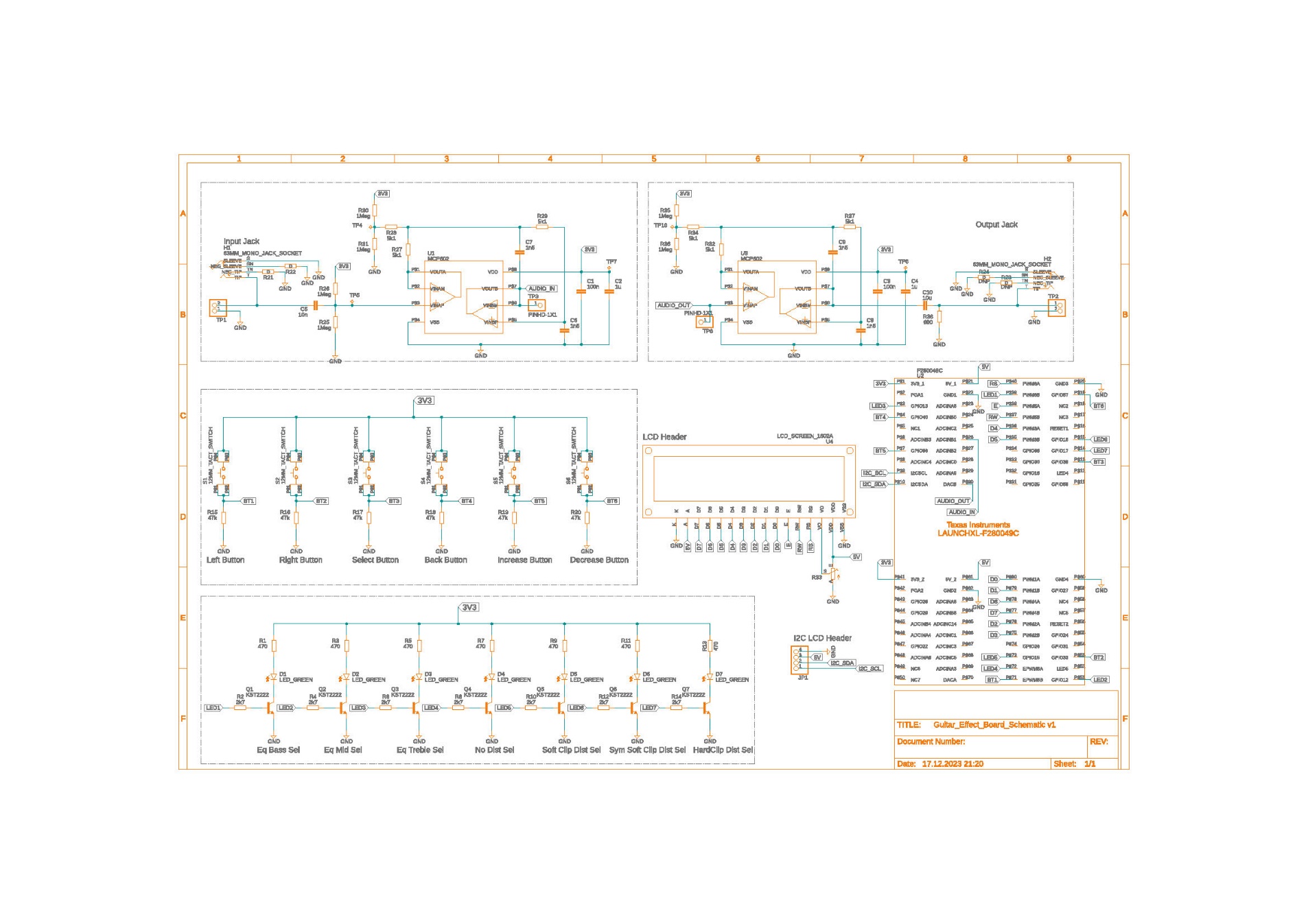
**

# 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.*

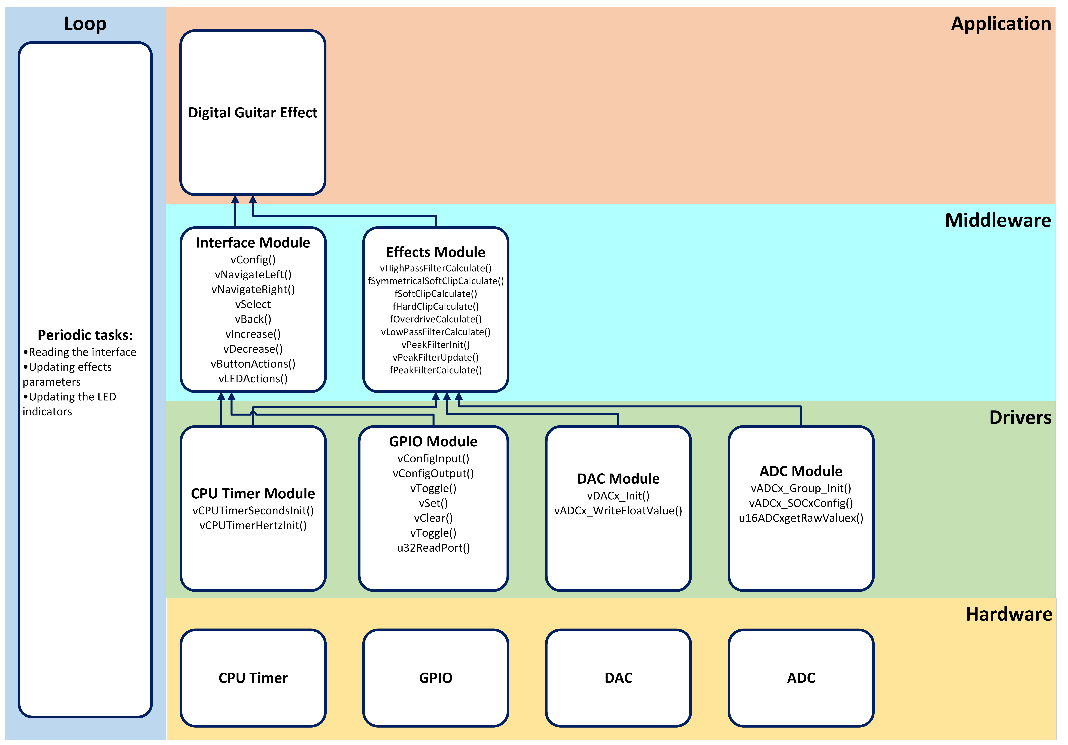
*The other relevant peripherals for this project are the 12-bit ADC and 12-bit DAC. To interface the audio signal with these modules, analog frontends are essential. These circuits are implemented using operational amplifiers and are designed to filter out frequencies beyond the audio spectrum and those 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.*

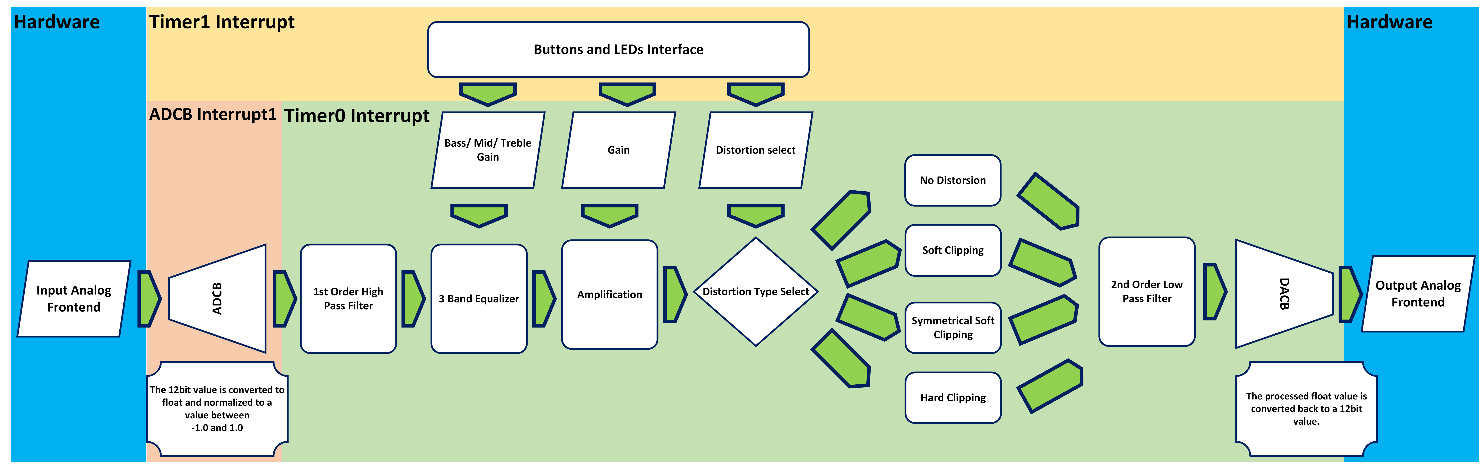
**

# 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.*

# 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.*

# 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.*

# 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](mailto:george.popescu1012@upb.ro) until 06.01.2024.
* The maximum grade for the project will be 50 points (half of the final grade).