

**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA
CATARINA – CAMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO DE ENGENHARIA ELETRÔNICA**

JOSÉ NICOLAU VARELA

**EQUALIZADOR PARAMÉTRICO DE ÁUDIO DIGITAL COM CINCO
BANDAS, INTERFACE GRÁFICA *TOUCH SCREEN* E OPERAÇÃO EM
TEMPO REAL**

FLORIANÓPOLIS, 2021

**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA
CATARINA – CAMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO DE ENGENHARIA ELETRÔNICA**

JOSÉ NICOLAU VARELA

**EQUALIZADOR PARAMÉTRICO DE ÁUDIO DIGITAL COM CINCO
BANDAS, INTERFACE GRÁFICA *TOUCH SCREEN* E OPERAÇÃO EM
TEMPO REAL**

Trabalho de Conclusão de Curso submetido
ao Instituto Federal de Educação, Ciência e
Tecnologia de Santa Catarina como parte
dos requisitos para obtenção do título de
Engenheiro Eletrônico.

Orientador:
Prof. Fernando Santana Pacheco, Dr. Eng.

FLORIANÓPOLIS, 2021

Ficha de identificação da obra elaborada pelo autor.

Varela, José Nicolau
EQUALIZADOR PARAMÉTRICO DE ÁUDIO DIGITAL COM CINCO BANDAS, INTERFACE GRÁFICA TOUCH SCREEN E OPERAÇÃO EM TEMPO REAL / José Nicolau Varela ; orientação de Fernando Santana Pacheco. - Florianópolis, SC, 2021.
115 p.

Trabalho de Conclusão de Curso (TCC) - Instituto Federal de Santa Catarina, Câmpus Florianópolis. Bacharelado em Engenharia Eletrônica. Departamento Acadêmico de Eletrônica.
Inclui Referências.

1. Equalizadores de áudio. 2. Filtros shelving.
3. Audio digital. 4. Filtros de pico. 5. Equalizadores paramétricos. I. Pacheco, Fernando Santana . II. Instituto Federal de Santa Catarina. Departamento Acadêmico de Eletrônica. III. Título.

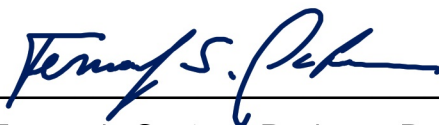
EQUALIZADOR PARAMÉTRICO DE ÁUDIO DIGITAL COM CINCO BANDAS, INTERFACE GRÁFICA *TOUCH SCREEN* E OPERAÇÃO EM TEMPO REAL

JOSÉ NICOLAU VARELA

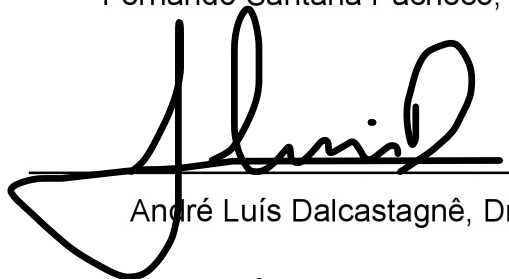
Este trabalho foi julgado adequado para obtenção do título de Engenheiro Eletrônico e aprovado na sua forma final pela banca examinadora do Curso de Engenharia Eletrônica do Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina.

Florianópolis, 13 de maio de 2021.

Banca Examinadora:



Fernando Santana Pacheco, Dr.



André Luís Dalcastagnê, Dr.



Robinson Pizzio, Dr.

Este trabalho é dedicado à minha mãe Maria Maes Varela (*in memoriam*), pilar da
minha busca por conhecimento.

AGRADECIMENTOS

- Para minha esposa Vera, meu filho e nora Eduardo e Kris, que tornaram tudo isso possível, agradeço a paciência, o carinho e o amor.
 - À minha neta Anna por dar mais luz às nossas vidas.
 - À sobrinha Juliana, toda minha gratidão pelo suporte técnico.
- Ao meu orientador, Professor Fernando Santana Pacheco, por toda dedicação, ensinamentos e compreensão.

“Com o avanço das tecnologias e o surgimento dos meios eletrônicos, McLuhan vê o novo retorno da fase oral, visual e auditiva afirmando que a nova interdependência eletrônica recria o mundo à imagem de uma aldeia global”(apud <[medium.com/@jornalismoespm2017.1/mcluhan-a-vida-eletronica-em-uma-aldeia-global-fa7b78ab11b1](https://medium.com/@jornalismoespm/2017.1/mcluhan-a-vida-eletronica-em-uma-aldeia-global-fa7b78ab11b1)>)

RESUMO

O presente trabalho tem como objetivo a implementação em *hardware* de um equalizador paramétrico de áudio digital com cinco bandas de frequência atuando em tempo real, com interface gráfica e de controle em tela *touch screen*. Foi implementado no *kit* de desenvolvimento STM32F746G *Discovery*, o qual tem integrado em sua plataforma um *display touch screen*, interface necessária para que o usuário possa ajustar: volume, frequência de corte, frequência central, ganho e largura de banda. O *display* também mostra o nível de cada parâmetro. Este trabalho apresenta uma fundamentação teórica sobre equalizadores, especialmente sobre filtros *shelving* e de pico. Mostra também o processo de cálculo dos coeficientes dos filtros e simulações no *software* Matlab. Por fim, apresenta a implementação em hardware, testes de bancada e comparação de resultados. O *software* foi implementado no kit de desenvolvimento através da IDE (*integrated development environment*) STM32-CubeMX e, para avaliar o sinal elétrico na saída do sistema, foram utilizados os *softwares* de áudio Logic Pro para reproduzir e o Ocenaudio para gravar. Com a aplicação de vários tons de áudio na entrada, foi analisada a resposta em tempo real do sinal na saída e o display gráfico, observando-se uma sincronia entre o sinal visualizado e o sinal auditivo. Todos os parâmetros responderam aos comandos dos botões virtuais e *sliders* de controle sem causar interrupções ou ruídos audíveis na saída do sistema. Como resultado do presente trabalho, pode-se afirmar que é possível implementar um equalizador paramétrico de áudio digital de mais de uma banda com interface gráfica que opere em tempo real, utilizando o STM32F746G *Discovery*.

Palavras-chave: Equalizadores de áudio. Áudio digital. Equalizadores paramétricos. Filtros *shelving*. Filtros de pico.

ABSTRACT

This work aims to implement in hardware a real time parametric digital audio equalizer with five frequency bands, graphical interface and touch screen control. It was implemented in the STM32F746G Discovery development kit, which has integrated in its platform a touch screen display. Through this interface, a user can adjust: volume, cutoff frequency, central frequency, gain and bandwidth. The display also shows the level of each parameter. This work presents a review of audio equalizers, focusing on shelving and peak filters. We also present how to obtain filter coefficients and simulation results using Matlab. We detail the hardware implementation, testing procedures, and obtained results. The software was implemented in the development kit through the IDE (integrated development environment) STM32-CubeMX and to evaluate the electrical signal at the output of the system, audio software Logic Pro was used to reproduce and Ocenaudio to record. With the application of several audio tones at the input, the real-time response of the signal at the output and the graphic display was analyzed, observing synchrony between the visualized signal and the auditory signal. All parameters responded to the commands of the virtual buttons and control sliders without causing interruptions or audible noise at the output of the system. As a result of this work, it can be said that it is possible to implement a parametric digital audio equalizer for more than one band with a graphical interface that operates in real time, using the STM32F746G Discovery.

Keywords: Audio equalizer. Digital audio. Parametric equalizers. Shelving filters. Peak filters.

LISTA DE FIGURAS

Figura 1 –	Pré-amplificador de violão com equalizador.....	13
Figura 2 –	Etapas de gravação e reprodução sinal analógico por meios digitais.....	18
Figura 3 –	Magnitude da resposta em frequência de um filtro passa-baixa.....	19
Figura 4 –	Magnitude da resposta em frequência de filtros passa-faixa e rejeita-faixa	19
Figura 5 –	Magnitude da resposta em frequência de um filtro passa-alta.....	19
Figura 6 –	Magnitude da resposta em frequência de filtros low shelving (em azul e vermelho) e um passa-baixa (em amarelo).....	20
Figura 7 –	Amplitudes da resposta em frequência de um filtro low shelving em duas configurações.....	21
Figura 8 –	Amplitudes da resposta em frequência de um filtro high shelf em duas configurações.....	21
Figura 9 –	Filtro de Pico com $f_0=400$ Hz, $G=18$ e $Q=2,4$	22
Figura 10 –	Topologia do filtro de forma direta transposta do tipo 2.....	24
Figura 11 –	Sistema de equalização controlado em tempo real.....	26
Figura 12 –	Equalizador shelving com frequência de corte em 12 kHz no high shelving (1), 80 Hz no low shelving (2) e ambos com ganho de 15 dB.....	27
Figura 13 –	Arranjo das bandas de um equalizador gráfico.....	28
Figura 14 –	Arranjo das bandas de um equalizador paramétrico.....	29
Figura 15 –	Equalizador dentro do contexto do áudio.....	29
Figura 16 –	Captura de tela do equalizador do software Logic Pro.....	30
Figura 17 –	Captura de tela do equalizador do software Pro Tools.....	31
Figura 18 –	Captura de tela do equalizador weiss EQ.....	31
Figura 19 –	Captura de tela do equalizador Manley Massive Passive EQ.....	31
Figura 20 –	Etapas de desenvolvimento	35
Figura 21 –	Filtro low shelving com $f_c = 225$ Hz e ganhos de 12 e -12 dB	36
Figura 22 –	Filtro High shelving com $f_c = 13000$ Hz e ganhos de 12 e -12 dB.....	38

Figura 23 –	Filtros de pico com diferentes valores de frequência central.....	39
Figura 24 –	Filtros dispostos em série.....	40
Figura 25 –	Filtros dispostos em série com ganhos de - 12 a 12 dB.....	41
Figura 26 –	STM32F746G <i>DISCOVERY</i> , vistas superior e inferior.....	42
Figura 27 –	Organização do programa em diagrama de blocos.....	43
Figura 28 –	Fluxograma do processamento de sinal dentro da tarefa de Áudio...	46
Figura 29 –	emWin GUIBuilder V5.44.....	48
Figura 30 –	Implementação das <i>flags</i> ao ganho do filtro <i>peak 2</i>	49
Figura 31 –	Fluxograma da tarefa da interface gráfica.....	50
Figura 32 –	Painel frontal da interface gráfica.....	50
Figura 33 –	Display gráfico para monitoração do Sinal.....	51
Figura 34 –	Bancada de testes	52
Figura 35 –	Configuração de testes	53
Figura 36 –	Script de testes.....	54
Figura 37 –	Resposta do low shelving equalizado e simulado, com $G = 12$ dB e $f_c = 800$ Hz.....	55
Figura 38 –	Resposta do low shelving equalizado e simulado com $G = -12$ dB e $f_c = 800$ Hz.....	55
Figura 39 –	Resposta dos filtros de pico do equalizador e simulado, com $Q = 20$, $G = 12$, $f_{0\text{ peak } 1} = 3000$ Hz, $f_{0\text{ peak } 2} = 6000$ Hz e $f_{0\text{ peak } 3} = 9000$ Hz.....	56
Figura 40 –	Resposta dos filtros de pico do equalizador e simulado, com $Q = 20$, $G = -12$, $f_{0\text{ peak } 1} = 3000$ Hz, $f_{0\text{ peak } 2} = 6000$ Hz e $f_{0\text{ peak } 3} = 9000$ Hz.....	56
Figura 41 –	Resposta do filtro high shelving do equalizador e simulado, com $G = -12$ dB e $f_c = 14000$ Hz.....	57
Figura 42 –	Resposta do filtro high shelving do equalizador e simulado, com $G = 12$ dB e $f_c = 14000$ Hz.....	57
Figura 43 –	$G = 12$ dB para todos os filtros.....	58
Figura 44 –	Ganho $G = -12$ todos os filtros.....	58

LISTA DE TABELAS

Tabela 1 – Coeficientes para filtro low shelving (incremento $V_0 = 10^{G/20}$).....	25
Tabela 2 – Coeficientes para filtro low shelving (corte $V_0 = 10^{-G/20}$)	25
Tabela 3 – Coeficientes para filtro high shelving (incremento $V_0 = 10^{G/20}$).....	25
Tabela 4 – Coeficientes para filtro high shelving (corte $V_0 = 10^{-G/20}$).....	25
Tabela 5 – Coeficientes para filtro de pico (incremento $V_0 = 10^{G/20}$).....	25
Tabela 6 – Coeficientes para filtro de pico (corte $V_0 = 10^{-G/20}$).....	26
Tabela 7 – Parâmetros aplicados para a implementação dos filtros.....	36
Tabela 8 – Parâmetros dos filtros (simulação).....	40
Tabela 9 – Valores utilizados para os testes.....	55

LISTA DE ABREVIATURAS E SIGLAS

ADC – *Analog-to-Digital Converter*
ARM – *Advanced RISC Machine*
CMSIS – *Cortex Microcontroller software Interface Standard*
CD – *Compact Disc*
CODEC – *Coder Decoder*
DAC – *Digital-to- Analog Converter*
dB – *Decibel*
DMA – *Direct Memory Access*
DSP – *Digital Signal Processing*
ECG – *Eletrocardiograma*
EQ – *Equalizador*
FFT – *Fast Fourier Transform*
FPGA – *Field-Programmable Gate Array*
GUI – *Graphical User Interface*
HAL – *Hardware Abstraction Layer*
Hz – *Hertz*
IDE – *Integrated Development Environment*
IIR – *Infinite Impulse Response*
LCD-TFT – *Liquid Crystal Display - Thin Film Transistor*
LTDC – *LCD-TFT Display Controller*
RTOS – *Real Time Operating System*
REC – *Recording*
SAI – *Serial Audio Interface*
SNR – *Signal-to-Noise Ratio*
USB – *Universal Serial Bus*

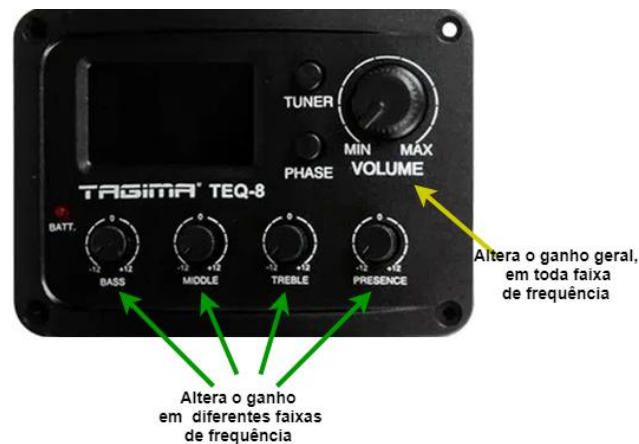
SUMÁRIO

1. INTRODUÇÃO	13
1.1 Definição do Problema	14
1.2 Justificativa	15
1.3 Objetivo Geral	16
1.4 Objetivos Específicos	16
2. REVISÃO DA LITERATURA	17
2.1 Fundamentos Básicos de Áudio Digital	17
2.2 Filtros	18
2.2.1 Filtros Passa-Baixa, Passa-Alta, Passa-Faixa, Rejeita Faixa	18
2.2.2 Filtros Shelving e filtros de Pico	20
2.3 Filtro Paramétrico	22
2.3.1 Design do Filtro Paramétrico	22
2.3.2 Topologia do filtro Paramétrico	24
2.4 Tipos de Equalizadores	26
2.4.1 Equalizador Shelving	27
2.4.2 Equalizador Gráfico	27
2.4.3 Equalizador Paramétrico	28
2.4.4 Equalizador na Produção de Áudio	29
2.4.5 Equipamentos equalizadores	30
2.5 Trabalhos Correlatos	32
3. METODOLOGIA	34
4. DESENVOLVIMENTO	36
4.1 Projeto e Simulação	36
4.1.1 Requisitos Iniciais	36
4.1.2 Determinação dos Coeficientes	37
4.1.3 Simulação dos Filtros	37
4.1.3.1 Simulação dos Filtro Low Shelving e High Shelving	38
4.1.3.2 Simulação dos Filtro de Pico	39
4.1.3.3 Filtros em Cascata	40
4.2 Implementação do Equalizador	41
4.2.1 Estruturação do Software	43
4.2.2 Bloco Principal	43
4.2.3 Bloco Áudio	47
4.2.4 Bloco Interface Gráfica	47
5. TESTES E RESULTADOS	52
6. CONSIDERAÇÕES FINAIS	60

1. INTRODUÇÃO

Pode-se afirmar que a música faz parte da vida das pessoas, mas cada uma a avalia de uma forma, ou seja, a percepção musical é subjetiva. Desse modo, é importante prover meios de adaptar a experiência de escutar música a cada ouvinte. Uma das formas mais simples de modificar o áudio é através da alteração de ganho, popularmente conhecido como “volume”. Porém, em certas situações modificar o ganho de toda faixa de frequência, (*volume*) não é suficiente. Por exemplo, para destacar o som das cordas agudas em um violão, só alterar o volume em um pré-amplificador, com mostrado na Figura 1, não trará a percepção desejada. Assim, há a necessidade de se ajustar o ganho em diferentes faixas de frequência, algo que é função de um equalizador de áudio.

Figura 1 – Pré-amplificador de violão com equalizador



Fonte: Adaptado de TAGIMA (2021).

Os sons, em geral, são compostos por diferentes frequências. Para que a vibração sonora seja captada pelo ouvido humano é necessário que sua frequência esteja, aproximadamente, entre 20 Hz e 20 kHz.

Observando-se a história da tecnologia de áudio, nota-se que a conversão do sinal sonoro mecânico em um sinal elétrico permitiu o desenvolvimento de diferentes equipamentos que alteram as características de amplitude, frequência e fase. Uma das ações de processamento, muito empregada, é a criteriosa alteração da amplitude de faixas de frequências específicas (ZÖLZER, 2008). Essa ação se chama

equalização, que, segundo Zölzer (2008), é um dos métodos mais importantes para o processamento de sinais de áudio. Equalizadores foram desenvolvidos, inicialmente, para atender aos sistemas de telefonia, expandindo-se posteriormente, para sistemas de gravação e reprodução de música. O reforço ou atenuação das amplitudes em faixas de frequências específicas pode servir para corrigir distorções de sistemas de transmissão (por exemplo, um canal de comunicação que atenua as altas frequências), o que justifica o nome equalizar, ou seja, deixar igual ao que era antes. Pode também ser usado para melhorar a sensação sonora, como, por exemplo, na equalização dos graves, médios e agudos de uma música (SILVA, 2007).

Por muito tempo o processamento de áudio foi realizado somente em sistemas analógicos. A evolução dos sistemas eletroeletrônicos e a consequente digitalização permitiu que aparelhos de áudio como gravadores, amplificadores, instrumentos musicais e pedais de efeitos passassem a utilizar a tecnologia de processamento digital de sinais (DSP, do inglês – *Digital Signal Processing*). Deve-se destacar que muitos equipamentos analógicos continuam no mercado por conta do excelente resultado que a eletrônica analógica proporciona em sistemas que operem em tempo real. De qualquer modo, os processadores digitais de sinais trouxeram mais flexibilidade aos equipamentos de áudio (HANNA, 1994). Uma vez que o sinal seja digitalizado, o processador pode ser programado para executar as mais diversas funções, como eliminar ruídos, adicionar efeitos, comprimir e descomprimir o sinal, filtrar, entre outras aplicações (BODANESE, 2008).

Um equalizador pode ser implementado por filtros de sinais analógicos ou digitais. Neste trabalho será projetado um equalizador paramétrico digital de áudio com 5 bandas de atuação utilizando filtros digitais implementados por *software* para operarem em tempo real. O acesso aos parâmetros dos filtros será através de uma interface gráfica em tela *touch screen* para interação com o usuário. A implementação da aplicação se fará no kit de desenvolvimento STM32F746G *Discovery da* STMicroelectronics.

1.1 Definição do Problema

A reprodução de áudio em qualquer equipamento pode exigir correções tanto para compensar a baixa qualidade ou a acústica do ambiente. Nem sempre o sistema de som possui um equalizador, ou quando o tem, muitas vezes é limitado em

número de bandas e controle de acesso aos parâmetros. O equalizador digital desenvolvido no presente projeto pode ser mais uma solução viável dentro das tecnologias empregadas em áudio. Para isto, devem ser respeitados os requisitos que um sistema de som necessita para obter um bom desempenho. Segundo Hanna (1994), em uma implementação digital, erros e instabilidades podem ocorrer quando os parâmetros do filtro são atualizados. Geralmente, um equipamento de áudio precisa operar em tempo real, portanto, deve-se priorizar o processamento do sinal de áudio entre as tarefas a serem executadas, evitando, assim, interferências como interrupção e ruído na saída do sistema.

O desafio deste projeto é desenvolver um equalizador digital que opere em tempo real, utilizando filtros digitais implementados por *software* juntamente com a interface gráfica, sem que nenhum deles seja prejudicado. Desse modo, alicerçados nesses argumentos, este trabalho tentará responder as seguintes questões: é possível implementar um equalizador paramétrico de áudio digital com mais de uma banda e interface gráfica que opere em tempo real? Pretende-se disponibilizar cinco bandas de frequência com o ajuste e monitoramento individual da frequência de interesse, o ganho e a largura de banda. Qual a ordem e tipo de filtro que deve ser empregado? Como o sinal deve ser processado? Como deve ser configurada a interface gráfica? Como devem ser compartilhadas as tarefas respeitando suas prioridades? Qual o tempo necessário de processamento do sinal para que a resposta na saída não seja interrompida? Estas são as principais indagações que o presente trabalho responderá através de pesquisas, testes e implementação de um equalizador de áudio digital em *hardware* dedicado.

1.2 Justificativa

Um equalizador de áudio é um dos equipamentos mais importantes para tornar a experiência sonora mais agradável aos ouvidos humanos. Seja em um estúdio de gravação ou em um sistema de som para apresentações ao vivo, a equalização é fundamental para obter resposta sonora com qualidade. Ela faz com que um instrumento se destaque ou desapareça, torna as palavras de um verso ou narração mais compreensíveis e pode amenizar ou até eliminar distorções e ruídos indesejáveis provenientes de outras fontes. Muitas vezes, uma boa mistura (em inglês - *mix*) das fontes sonoras é obtida praticamente na equalização.

Um equalizador pode ser implementado como um *software (plugin)* instalado em um computador ou através de um equipamento (*hardware*) analógico robusto, com características específicas e componentes especiais. Este projeto se justifica por apresentar uma solução intermediária, ou seja, um equalizador implementado em *software* em um equipamento (*hardware*) independente, compacto, com qualidade profissional que pode ser utilizado em qualquer sistema de som, e ainda com possibilidade de atualizar o *firmware* como, por exemplo, aumentar o número de bandas de atuação ou a complexidade dos filtros. Outro ponto importante a ser destacado é a utilização da interface gráfica em tela *touch screen* que integrará o sistema, tornando desnecessária a adição de botões, potenciômetros ou outro recurso em *hardware* para ajuste e monitoramento visual dos parâmetros.

1.3 Objetivo Geral

O objetivo geral deste trabalho consiste em projetar e implementar um equalizador paramétrico digital de cinco bandas que opere em tempo real, com entrada e saída em estéreo e interface gráfica para ajustar e visualizar os parâmetros dos filtros bem como o ganho do sinal na saída do equalizador. Será utilizado o *kit* de desenvolvimento STM32F746G *Discovery da* STMicroelectronics, com microprocessador e periféricos integrados na mesma placa.

1.4 Objetivos Específicos

Para alcançar o objetivo geral, foram traçados os seguintes objetivos específicos:

- a) Pesquisar sobre estruturas para equalização de áudio analógicas e digitais.
- b) Projetar e simular os filtros.
- c) Implementar o equalizador no *kit* de desenvolvimento.
- d) Implementar a interface gráfica.
- e) Realizar testes e analisar os resultados.

2. REVISÃO DA LITERATURA

Neste capítulo, será apresentada uma breve revisão sobre fundamentos básicos de áudio digital, processamento de sinais, filtros e equalizadores.

2.1 Fundamentos Básicos de Áudio Digital

Para podermos entender melhor o processamento de áudio digital, temos que diferenciar o sinal analógico do digital e o sinal contínuo do sinal discreto no tempo.

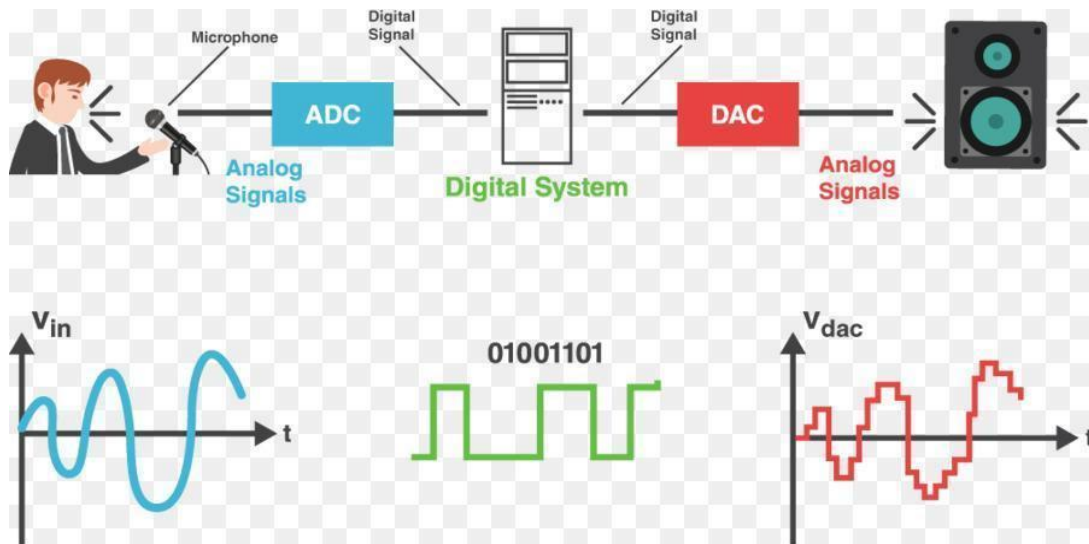
Os termos contínuo e discreto expressam a natureza do sinal no tempo (abscissas), e os termos analógico e digital estão relacionados à amplitude (ordenadas). Um sinal originalmente analógico e em tempo contínuo pode ser processado digitalmente se passar por uma etapa de digitalização, em que dois processos são efetuados: a amostragem e a quantização do sinal (LATHI, 2014).

Em relação à amostragem, o principal parâmetro é a taxa *de* amostragem (f_s), *que* indica quantas amostras do sinal são adquiridas em um segundo. A qualidade do sinal amostrado está diretamente relacionada com a taxa de amostragem. Para evitar perdas de informação na conversão analógica para digital no ADC (do inglês: *Analog-to-Digital Converter*) ou de digital para analógica no DAC (do inglês: *Digital-to-Analog Converter*), deve ser respeitado o teorema de Nyquist que diz que a taxa de amostragem deve ser pelo menos duas vezes a maior frequência esperada (LATHI, 2014). Portanto, no âmbito do áudio profissional, são mais utilizadas as taxas de amostragem de 44,1 kHz e 48 kHz, que garante a integridade do sinal dentro da faixa de frequências do espectro audível do ser humano. Em relação ao processo de quantização, o principal parâmetro é a resolução ou *bit depth*, o tamanho da palavra digital em bits. Quanto maior a palavra, mais precisa será a quantização e melhor será a curva dinâmica, o que significa um som melhor (OWSINSKI, 2 eds. 2010). A resolução de áudio para CD (Compact Disc) é padronizada em 16 *bits*; em sistemas de telefonia, 8 *bits*, e em estúdio de gravação de áudio profissional, geralmente 24 *bits*.

De forma resumida, como mostrado na Figura 2, existem 3 sistemas envolvidos no processamento digital de sinais de áudio. Para a digitalização do sinal analógico são utilizados conversores, um ADC e, para conversão inversa, um DAC. O

sistema que realiza o processamento propriamente dito pode ser um processador de uso geral ou um processador especializado (como um DSP ou FPGA).

Figura 2 – Etapas de gravação e reprodução de sinal analógico por meios digitais



Fonte: PINTEREST (2021).

2.2 Filtros

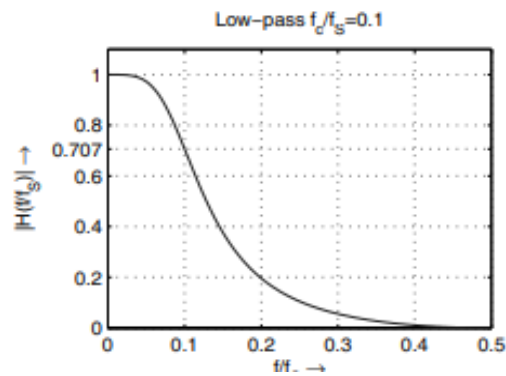
Os filtros, tanto analógicos quanto digitais, em geral, são utilizados para os seguintes propósitos: separação de sinais que tenham sido combinados e restauração de sinais que de alguma forma tenham sido distorcidos (SMITH, 1999).

2.2.1 Filtros Passa-Baixa, Passa-Alta, Passa-Faixa, Rejeita Faixa

Pode-se mencionar aqui quatro tipos de filtros seletores de sinais clássicos que compõem a base para o estudo dos filtros para áudio. Numa visão macro da área de áudio, um filtro passa-baixa (Figura 3) atua na faixa dos sons graves, filtros passa-faixa e rejeita-faixa (Figura 4) atuam predominantemente na faixa dos sons de médias frequências e um passa-alta (Figura 5) atua nas frequências mais altas dentro do espectro do som.

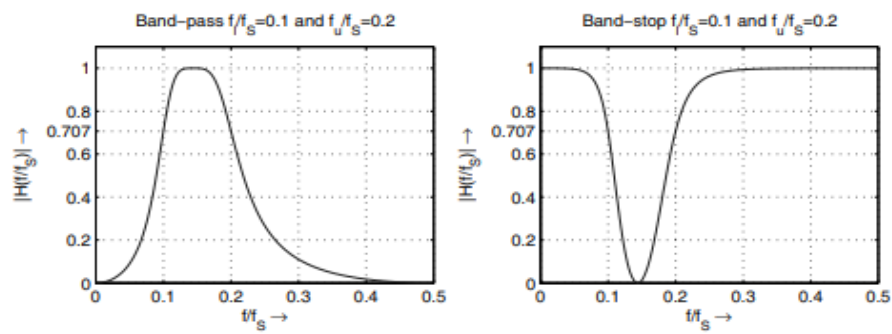
Como exemplo de aplicação para alguns desses filtros, pode-se mencionar a do filtro passa-baixa que é célula base para incremento ou atenuação de graves e o filtro passa-alta, célula base para o incremento ou atenuação de agudos, ambos para o equalizador *shelving* (NOCETI, DALCASTAGNÊ, GONTIJO, 2011).

Figura 3 – Magnitude da resposta em frequência de um filtro passa-baixa



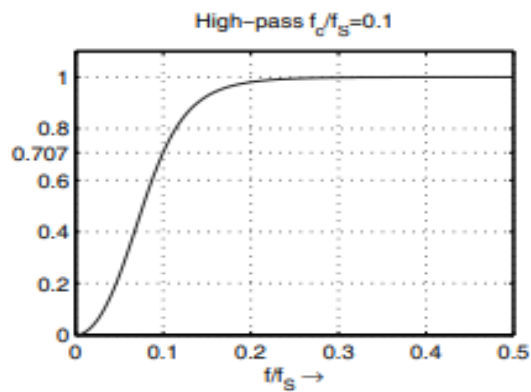
Fonte: Adaptação de ZÖLZER (2008, p. 116).

Figura 4 – Magnitude da resposta em frequência de filtros passa-faixa e rejeita-faixa



Fonte: Adaptação de ZÖLZER (2008, p.116).

Figura 5 – Magnitude da resposta em frequência de um filtro passa-alta

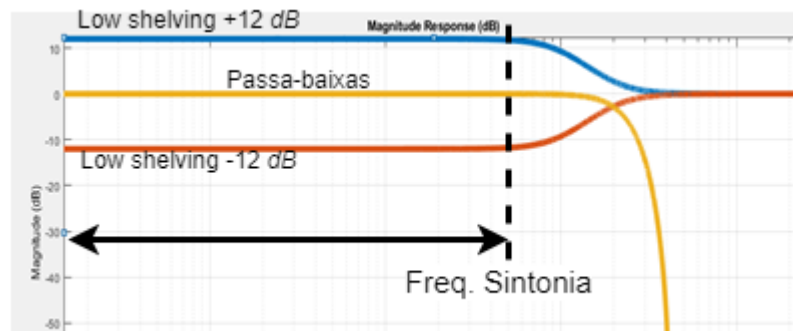


Fonte: Adaptação de ZÖLZER (2008, p. 116).

2.2.2 Filtros *Shelving* e filtros de Pico

Os filtros chamados de *shelf* ou *shelving* se assemelham aos filtros passa-baixa e passa-alta e são denominados de *low shelving* para frequências baixas e *high shelving* para frequências altas. A Figura 6 exemplifica a atuação em baixas frequências de um *low shelving* e de um passa-baixa, tipos de filtros fáceis de serem confundidos. Nota-se que na faixa de zero Hz até a *frequência de sintonia* (frequência a qual o ganho é 1 dB a menos que o ganho máximo), os filtros shelving podem ter incremento ou atenuação em relação a zero dB e pode-se observar estabilidade (CLARK, 2001). Nesta faixa de frequência também está a maior parte da banda passante do passa-baixa. Por outro lado, fora da referida faixa, as amplitudes dos *low shelving* tendem a se estabilizar em ganho unitário (0 dB) enquanto a do passa-baixa atenua continuamente tendendo a zero em valor absoluto. Essa característica dos *shelving* é essencial para equalização de áudio (FOX MEDIA TECH INC., 2021).

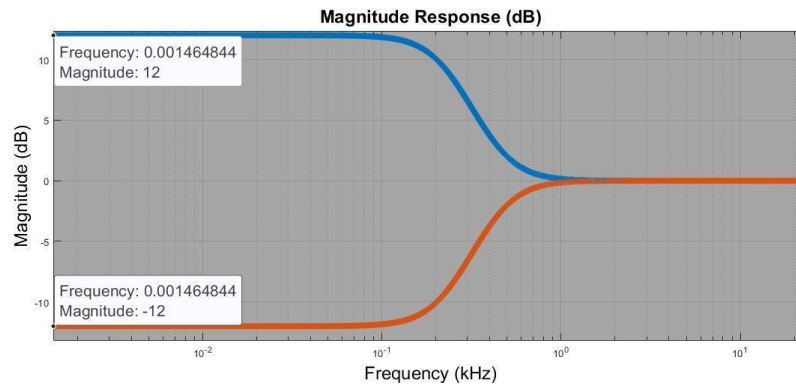
Figura 6 – Magnitude da resposta em frequência de filtros low shelving (em azul e vermelho) e um passa-baixa (em amarelo)



Fonte: AUTOR (2021)

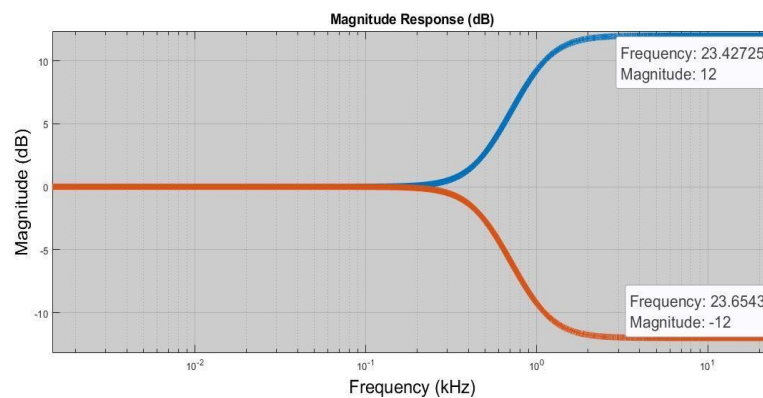
As Figuras 7 e 8 mostram dois filtros *low shelving* e dois filtros *high shelving*, respectivamente. A linha azul indica o aumento do sinal com ganho positivo estabilizando em 12 dB; a linha vermelha indica a atenuação do sinal estabilizando em -12 dB.

Figura 7– Amplitudes da resposta em frequência de um filtro low shelving em duas configurações: em azul, com ganho de 12 dB e, em vermelho, com atenuação de –12 dB.



Fonte: AUTOR (2021)

Figura 8 – Amplitudes da resposta em frequência de um filtro high shelving em duas configurações: em azul, com ganho de 12 dB e, em vermelho, com atenuação de –12 dB.



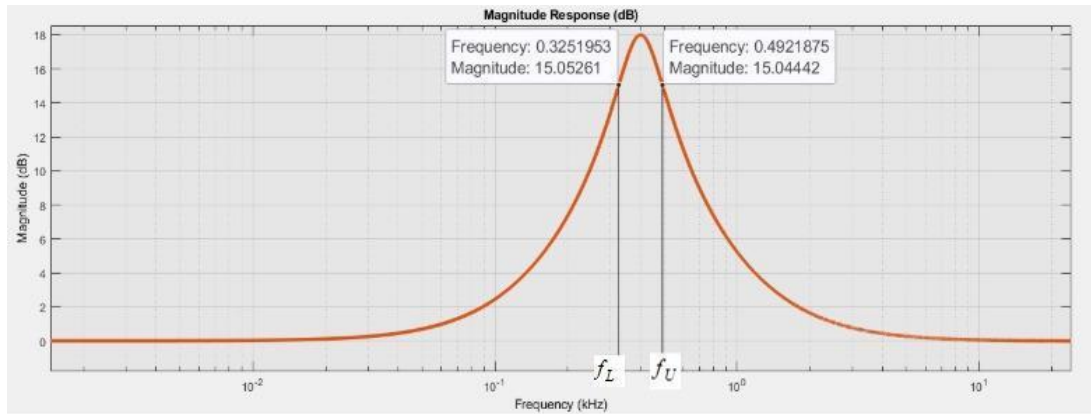
Fonte: AUTOR (2021)

A inclinação (do inglês: *slope*) na resposta de magnitude entre o ganho máximo e zero dB, em muitas vezes é ajustável pelo usuário (CLARK, 2001). Para fins desse trabalho o *slope* será constante e os filtros shelving serão projetados informando-se apenas os seguintes parâmetros: a frequência de corte (f_c), o ganho (G) e a frequência de amostragem (f_s).

O filtro de pico conhecido também como *peak* ou *bell filter* possui três parâmetros que podem ser ajustados pelo usuário: frequência central (f_0), ganho ou atenuação de pico (G) e o fator de qualidade (Q). Este último parâmetro é usado para determinar a largura de banda (BW) através da relação $BW = f_0/Q$, onde BW é a diferença entre a frequência de corte superior (f_U) e a frequência de corte inferior (f_L), em ± 3 dB de amplitude (CLARK, 2001). A Figura 9 mostra um exemplo de filtro de pico com 18 dB de ganho (G) e largura de banda de aproximadamente 167 Hz.

$$BW = f_U - f_L \quad (2.1)$$

Figura 9 – Filtro de Pico com $f_0=400$ Hz, $G=18$ dB e $Q=2,4$



Fonte: AUTOR (2021).

2.3 Filtro Paramétrico

Um filtro paramétrico é assim chamado pelo fato de obter sua resposta através da especificação de três parâmetros: frequência central, largura de banda (ou fator Q) e ganho (incremento ou atenuação). A função de transferência é de segunda ordem e geralmente é referida como biquadrática (HANNA,1994). Segundo Hanna (1994), os equalizadores paramétricos são projetados cascadeando vários estágios de filtros paramétricos. O equalizador descrito neste projeto é implementado no domínio digital como um filtro com resposta ao impulso infinita (*IIR*, do inglês *Infinite Impulse Response*).

2.3.1 Design do Filtro Paramétrico

O projeto dos filtros é apresentado, a seguir, no domínio S , domínio da transformada de Laplace. Estes serão convertido para o domínio Z através da transformação bilinear. Todas as equações desta seção são de Zölzer (2008).

Um filtro passa-baixa *shelving* de segunda ordem pode ser dado pela seguinte função de transferência $H(s)$ no domínio da transformada de Laplace:

$$H(s) = \frac{s^2 + \sqrt{2V_0}s + V_0}{s^2 + \sqrt{2}s + 1} \quad (2.2)$$

onde o parâmetro V_0 determina a amplificação em que a frequência é zero. Alterando o parâmetro V_0 pode-se incrementar ($V_0 > 1$) ou atenuar a amplitude ($V_0 < 1$) (ZÖLZER, 2008).

Para obter a atenuação simétrica da amplitude em relação ao zero com a mesma frequência de corte, basta inverter a equação 2.2, tendo como resultado a equação 2.3.

$$H(s) = \frac{s^2 + \sqrt{2}s + 1}{s^2 + \sqrt{2V_0}s + V_0} \quad (2.3)$$

Um filtro passa-alta *shelving* de segunda ordem pode ser dado pela seguinte função de transferência $H(s)$ no domínio da transformada de Laplace:

$$H(s) = \frac{V_0 s^2 + \sqrt{2V_0}s + 1}{s^2 + \sqrt{2}s + 1} \quad (2.4)$$

Para obter a atenuação simétrica da amplitude em relação ao zero com a mesma frequência de corte, basta inverter a equação 2.4, resultando na equação 2.5.

$$H(s) = \frac{s^2 + \sqrt{2}s + 1}{V_0 s^2 + \sqrt{2V_0}s + 1} \quad (2.5)$$

Outro filtro usado no equalizador paramétrico para cortar ou aumentar qualquer frequência desejada é o filtro de pico. A função de transferência é dada por:

$$H(s) = \frac{s^2 + \frac{V_0}{Q}s + 1}{s^2 + \frac{1}{Q}s + 1} \quad (2.6)$$

Pode ser observado na equação 2.6 que o máximo da amplitude na frequência central é determinado pelo parâmetro V_0 e a largura de banda é fixada pelo fator Q .

O filtro projetado no domínio S com função de transferência $H(s)$ deve ser convertido para o domínio Z por uma transformação adequada para obter a função de transferência $H(z)$. Os pólos e zeros no domínio Z podem ser determinados pela transformação bilinear (2.7) e a variável auxiliar $K = \tan(\pi \frac{f_0}{f_s})$ onde f_s é a frequência de amostragem e f_0 , a frequência de corte para os *shelving* ou frequência central para os filtros de pico (ZÖLZER, 2008).

$$s = \frac{2z - 1}{\frac{1}{f_s}z + 1} \quad (2.7)$$

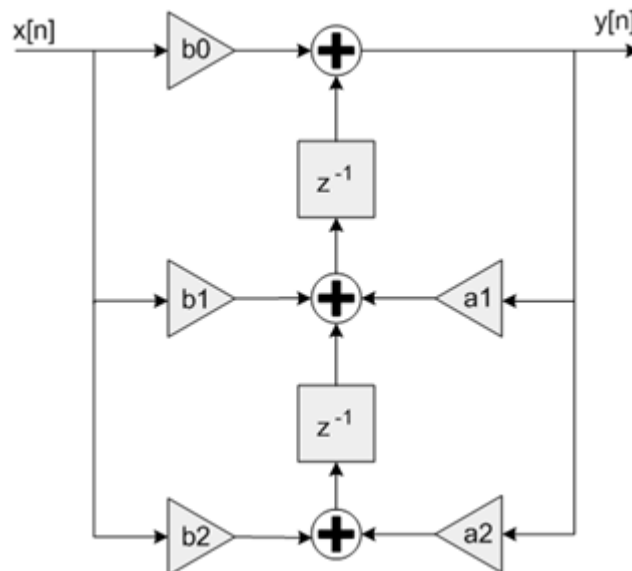
Para projetar os filtros de segunda ordem, deve-se determinar os coeficientes da equação 2.8 (função de transferência $H(z)$).

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \quad (2.8)$$

2.3.2 Topologia do filtro Paramétrico

A topologia do filtro paramétrico que será utilizado neste projeto é a forma direta transposta do tipo 2 (DF2T, do inglês *Transposed Direct Form 2*), mostrada na Figura 10.

Figura 10 – Topologia do filtro de forma direta transposta do tipo 2



Fonte: Adaptado de KEIL CMSIS-DSP (2010)

A equação recursiva para o processamento do sinal é a seguinte:

$$y_i = b_0 x_i + b_1 x_{i-1} + b_2 x_{i-2} - a_1 y_{i-1} - a_2 y_{i-2} \quad (2.9)$$

Note que a_1 e a_2 devem ter o sinal invertido na equação.

As tabelas 1 a 6 apresentam as equações extraídas de Zölzer (2008), para o cálculo dos coeficientes b_0, b_1, b_2, a_1, a_2 para cada filtro, lembrando que $K = \tan(\pi \frac{f_0}{f_s})$. Essas equações serão codificadas para serem utilizadas na simulação e implementação dos filtros.

Tabela 1 – Coeficientes para filtro *low shelving* (incremento $V_0 = 10^{G/20}$)

b_0	b_1	b_2	a_1	a_2
$\frac{1 + \sqrt{2V_0}K + V_0K^2}{1 + \sqrt{2}K + K^2}$	$\frac{2(V_0K^2 - 1)}{1 + \sqrt{2}K + K^2}$	$\frac{1 - \sqrt{2V_0}K + V_0K^2}{1 + \sqrt{2}K + K^2}$	$\frac{2(K^2 - 1)}{1 + \sqrt{2}K + K^2}$	$\frac{1 - \sqrt{2}K + K^2}{1 + \sqrt{2}K + K^2}$

Fonte: Adaptado de ZÖLZER (2008).

Tabela 2 – Coeficientes para filtro *low shelving* (corte $V_0 = 10^{-G/20}$)

b_0	b_1	b_2	a_1	a_2
$\frac{1 + \sqrt{2}K + K^2}{1 + \sqrt{2V_0}K + V_0K^2}$	$\frac{2(K^2 - 1)}{1 + \sqrt{2V_0}K + V_0K^2}$	$\frac{1 - \sqrt{2}K + K^2}{1 + \sqrt{2V_0}K + V_0K^2}$	$\frac{2(V_0K^2 - 1)}{1 + \sqrt{2V_0}K + V_0K^2}$	$\frac{1 - \sqrt{2V_0}K + V_0K^2}{1 + \sqrt{2V_0}K + V_0K^2}$

Fonte: Adaptado de ZÖLZER (2008).

Tabela 3 – Coeficientes para filtro *high shelving* (incremento $V_0 = 10^{G/20}$)

b_0	b_1	b_2	a_1	a_2
$\frac{V_0 + \sqrt{2V_0}K + K^2}{1 + \sqrt{2}K + K^2}$	$\frac{2(K^2 - V_0)}{1 + \sqrt{2}K + K^2}$	$\frac{V_0 - \sqrt{2V_0}K + K^2}{1 + \sqrt{2}K + K^2}$	$\frac{2(K^2 - 1)}{1 + \sqrt{2}K + K^2}$	$\frac{1 - \sqrt{2}K + K^2}{1 + \sqrt{2}K + K^2}$

Fonte: Adaptado de ZÖLZER (2008).

Tabela 4 – Coeficientes para filtro *high shelving* (corte $V_0 = 10^{-G/20}$)

b_0	b_1	b_2	a_1	a_2
$\frac{1 + \sqrt{2}K + K^2}{V_0 + \sqrt{2V_0}K + K^2}$	$\frac{2(K^2 - 1)}{V_0 + \sqrt{2V_0}K + K^2}$	$\frac{1 - \sqrt{2}K + K^2}{V_0 + \sqrt{2V_0}K + K^2}$	$\frac{2(K^2/V_0 - 1)}{1 + \sqrt{2/V_0}K + K^2/V_0}$	$\frac{1 - \sqrt{\frac{2}{V_0}}K + K^2/V_0}{1 + \sqrt{2/V_0}K + K^2/V_0}$

Fonte: Adaptado de ZÖLZER (2008).

Tabela 5 – Coeficientes para filtro de pico (incremento $V_0 = 10^{G/20}$)

b_0	b_1	b_2	a_1	a_2
$\frac{1 + \frac{V_0}{Q}K + K^2}{1 + \frac{1}{Q}K + K^2}$	$\frac{2(K^2 - 1)}{1 + \frac{1}{Q}K + K^2}$	$\frac{1 - \frac{V_0}{Q}K + K^2}{1 + \frac{1}{Q}K + K^2}$	$\frac{2(K^2 - 1)}{1 + \frac{1}{Q}K + K^2}$	$\frac{1 - \frac{1}{Q}K + K^2}{1 + \frac{1}{Q}K + K^2}$

Fonte: Adaptado de ZÖLZER (2008).

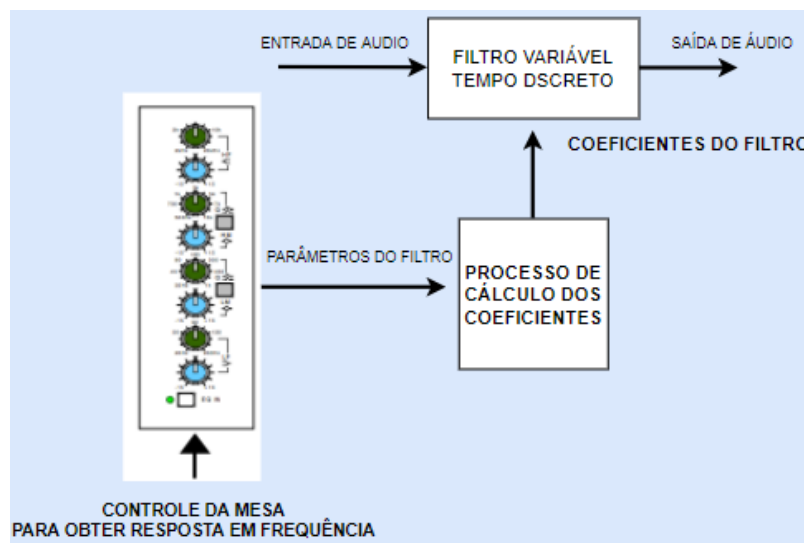
Tabela 6 – Coeficientes para filtro de pico (corte $V_0 = 10^{-G/20}$)

b_0	b_1	b_2	a_1	a_2
$\frac{1 + \frac{1}{Q}K + K^2}{1 + \frac{V_0}{Q}K + K^2}$	$\frac{2(K^2 - 1)}{1 + \frac{V_0}{Q}K + K^2}$	$\frac{1 - \frac{1}{Q}K + K^2}{1 + \frac{V_0}{Q}K + K^2}$	$\frac{2(K^2 - 1)}{1 + \frac{V_0}{Q}K + K^2}$	$\frac{1 - \frac{V_0}{Q}K + K^2}{1 + \frac{V_0}{Q}K + K^2}$

Fonte: Adaptado de ZÖLZER (2008).

Em um sistema de equalização digital, os parâmetros são especificados manipulando os controles da interface de controle do equalizador em uma mesa de som, por exemplo. Isso produz um conjunto de parâmetros de filtro que são utilizados para calcular coeficientes de filtros digitais apropriados. O sinal de áudio tem o seu espectro de frequências alterado pelo filtro variável (CLARK, 2001). A Figura 11 mostra um diagrama dessas etapas.

Figura 11 – Sistema de equalização controlado em tempo real



Fonte: Adaptado de CLARK (2001)

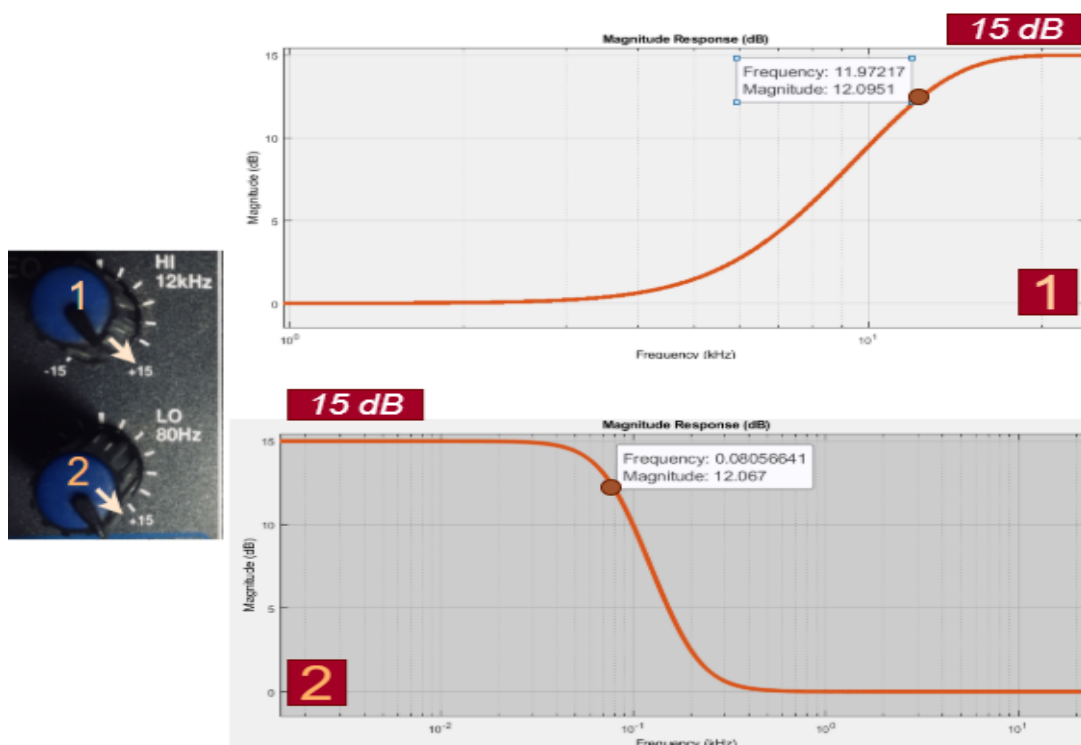
2.4 Tipos de Equalizadores

São muitos os tipos de equalizadores, pode-se citar por exemplo: o *shelving*, o gráfico, o paramétrico, o semiparamétrico, o *sweep* e o paragrafíco (TECLA CENTER, 2021). Nesta seção será abordado sobre o equalizador *shelving*, gráfico e paramétrico.

2.4.1 Equalizador Shelving

Este tipo de equalizador usa somente os filtros *shelving*, geralmente integrado na mesa de áudio, módulo de som de carro ou em equipamentos de som de uso doméstico. A sua regulação se limita a dois ajustes, um para as frequências graves e o outro para as agudas. Sua faixa de atuação é pré-estabelecida pelas frequências de corte dos filtros *low* e *high shelving*. A Figura 12 mostra os *knobs* posicionados em $+15\text{ dB}$ onde a faixa de ganho vai de -15 dB a $+15\text{ dB}$ para cada filtro. Os gráficos da Figura 12 mostram as respostas em frequência nas faixas em que provavelmente o equalizador (posição dos *knobs*) está atuando.

Figura 12 – Equalizador shelving com frequência de corte em 12 kHz no high shelving (1), 80 Hz no low shelving (2) e ambos com ganho de 15 dB



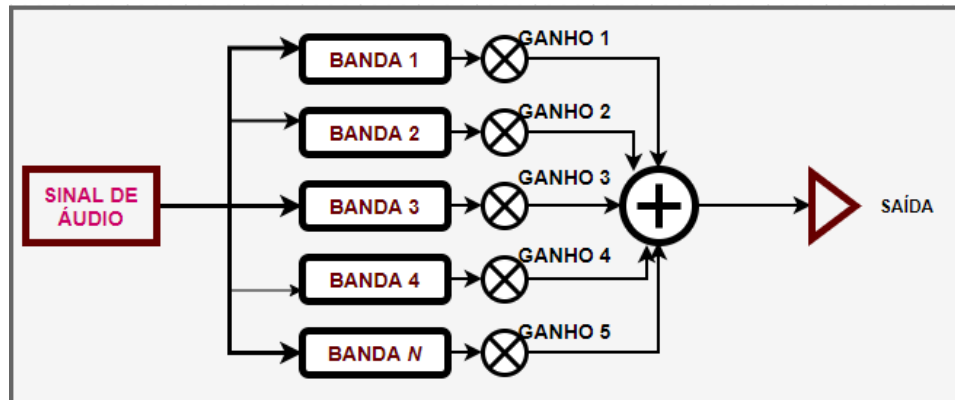
Fonte: AUTOR (2021).

2.4.2 Equalizador Gráfico

Filtros de um equalizador gráfico ajustam o conteúdo espectral (de uma música por exemplo) filtrando os sinais por meio de um banco de filtros paralelos (Figura 13). Cada filtro deixa passar parte do espectro do sinal. São unidos (somados)

novamente após a filtragem e reconstruídos com conteúdo espectral modificado. Antes de somados, o sinal de saída de cada filtro é multiplicado por um ganho ajustável pelo usuário dando ênfase ou atenuando àquela banda. Cada filtro atua dentro de uma largura de banda fixa com frequência central pré-estabelecida (MICROCHIP, 2013).

Figura 13 – Arranjo das bandas de um equalizador gráfico



Fonte: Adaptado de MICROCHIP (2013).

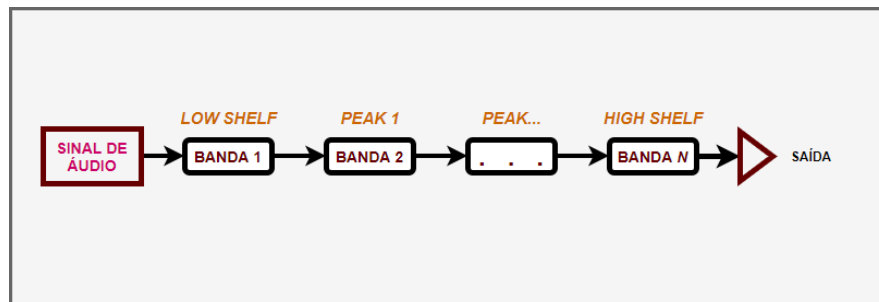
2.4.3 Equalizador Paramétrico

Em muitos equalizadores de áudio, o ajuste ou a exibição dos parâmetros dos filtros não estão disponíveis ao usuário (MICROCHIP, 2013).

A grande vantagem do equalizador paramétrico em relação ao equalizador gráfico está na disponibilização de parâmetros para ajuste da equalização. No equalizador paramétrico, o usuário pode escolher a faixa de frequência, o ganho e o fator de qualidade para cada banda disponível, ao passo que no equalizador gráfico somente o ganho pode ser alterado, pois a frequência e o fator de qualidade de cada banda são pré-fixados (BAUER, 2018).

A Figura 14 mostra a estrutura de um equalizador paramétrico. Segundo MICROCHIP (2013), diferente do equalizador gráfico onde os filtros são colocados em paralelo, no equalizador paramétrico, uma cascata de filtros em série é aplicada ao sinal de áudio. Este tipo de equalizador utiliza os filtros *shelving* para as bandas extremas e filtros de pico (*peak*) para as bandas intermediárias.

Figura 14 – Arranjo das bandas de um equalizador paramétrico



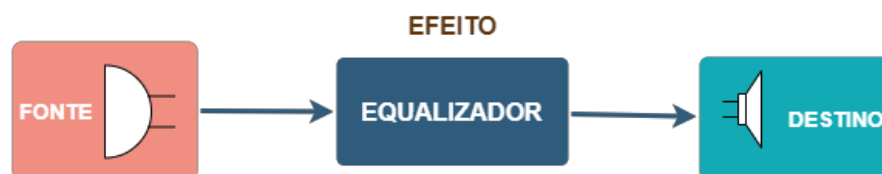
Fonte: Adaptado de MICROCHIP (2013).

Como os filtros são aplicados em cascata, cada um ajusta o sinal em uma determinada faixa limitada de frequências, mas sem levar o sinal a zero absoluto fora dessa faixa de atuação. Isso quer dizer que o filtro deve deixar passar os componentes de frequência fora de sua faixa de atuação com ganho unitário. (MICROCHIP, 2013).

2.4.4 Equalizador na Produção de Áudio

No ambiente da produção de áudio, mais especificamente na gravação e reprodução de música, o equalizador é considerado como um efeito de som e geralmente se encontra num contexto que pode ser representado por três estágios: fonte ou entrada, efeitos e a saída ou destino. A Figura 15 representa o fluxo de sinal mais utilizado no contexto musical.

Figura 15 – Equalizador dentro do contexto do áudio



Fonte: AUTOR (2021).

Há uma grande diversidade de equalizadores no mercado do áudio. Podem ser desde um simples *software plugin*, um equalizador virtual instalado no computador pessoal, até equipamentos *stand-alone* de grande porte usados em sistemas de som profissional. Em alguns casos, o equalizador já faz parte de um sistema maior, com *fonte-efeito-destino* integrados no mesmo equipamento, por exemplo, a mesa de som. A maioria das mesas de som já possuem algum tipo de equalizador, seja gráfico,

paramétrico ou outros já mencionados. Os equalizadores mais procurados por profissionais são aqueles que oferecem mais opções de ajuste e o equalizador paramétrico se enquadra facilmente neste requisito (TECLACENTER, 2021).

2.4.5 Equipamentos Equalizadores

Na maioria das vezes, um equalizador virtual em forma de *plugin* já vem no pacote de instalação dos *softwares* de áudio, o que é uma grande vantagem, pois será otimizado para o próprio *software* de áudio. Outras vantagens podem ser citadas, como por exemplo, a interação digital com as demais partes que integram o sistema de áudio e o limite de equalizadores só depender da capacidade de processamento do computador. São sistemas que apresentam bom custo/benefício principalmente para os *home studios* muito populares nos dias de hoje. Não são muito indicados para som ao vivo com equipamentos de potência. A resposta sonora para este tipo de equalizador não é tão simples de ser avaliada, depende do resto do sistema, como por exemplo a interface de áudio do computador. A Figura 16 mostra a interface do equalizador paramétrico de 8 bandas de atuação de nível profissional do *software Logic Pro da Apple*.

Figura 16 – Captura de tela do equalizador do software Logic Pro



Fonte: AUTOR (2021)

A Figura 17 mostra a interface do equalizador de 7 bandas da *Pro Tools da Avid*, outro conhecido equalizador virtual usado em vários estúdios de grande porte.

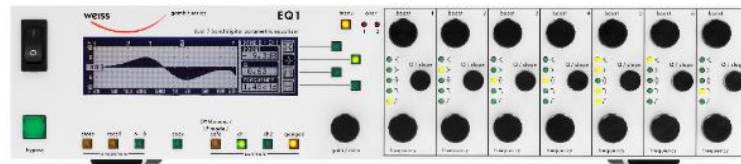
Figura 17 – Captura de tela do equalizador do software Pro Tools



Fonte: AUTOR (2021)

Saindo dos virtuais, o mercado também produz uma infinidade de equalizadores *stand-alone* indicados para qualquer situação, vão desde equalizadores híbridos, tecnologia analógica aliada aos DSPs, como exemplo o da *weiss* mostrado na Figura 18, até equalizadores com filtros inteiramente analógicos como exemplo o *Manley Massive Passive EQ* mostrado na Figura 19.

Figura 18 – Vista frontal do equalizador weiss EQ



Fonte: WEISS (2020).

Figura 19 – Vista frontal do equalizador Manley Massive Passive EQ



Fonte: UNIVERSAL AUDIO (2019).

Por sua versatilidade, os equalizadores paramétricos são os mais procurados por profissionais de áudio, porém são os mais caros por consequência da sua complexidade. Mais uma vez vale ressaltar que o equalizador aqui projetado poderá ser uma excelente alternativa para atender o mercado de áudio.

2.5 Trabalhos Correlatos

A concepção desse projeto foi motivada pela necessidade de desenvolver um protótipo de um equalizador paramétrico digital *stand-alone* em que se fizesse uso de uma interface gráfica para manipulação dos parâmetros em tempo real. Levou-se em consideração as sugestões propostas nas referências aqui pesquisadas.

O trabalho intitulado “Sistema para Equalização Paramétrica de Áudio em Tempo Real” de autoria de Matheus Bauer (BAUER,2018), sugere “*calcular os coeficientes sem atraso para que fosse possível a escolha de qualquer frequência*” (BAUER,2018). Esta referência apresenta um equalizador paramétrico digital de 5 bandas implementado no STM32F746G, o mesmo *kit* utilizado no projeto aqui desenvolvido. Segundo Bauer, os filtros projetados são de oitava ordem, e devido a complexidade dos cálculos para esta ordem de filtro, não é possível definir qualquer frequência no momento de equalizar, a escolha fica limitada aos coeficientes pré-calculados na implementação dos filtros. Por consequência foi necessário também aumentar o espaço de memória *flash* devido a quantidade de dados gerados em decorrência dos coeficientes pré-estabelecidos (BAUER,2018). O maior diferencial entre o presente projeto e o trabalho citado é que apesar dos filtros serem de segunda ordem, é possível escolher qualquer frequência dentro do espectro audível, pois todos os coeficientes são calculados em tempo suficiente (tempo real) para não interferir na performance do equalizador aqui projetado.

O projeto “*Crossover Digital Com Equalização Paramétrica Ajustável*” de autoria de Diogo Tavares, apresenta um *crossover* digital com equalizador paramétrico de uma banda utilizando uma plataforma com a placa STM32F4 *Discovery* integrada com uma placa *Wolfson Pi* e uma interface composta por *display* e botões para acesso aos parâmetros (TAVARES, 2020). Segundo Tavares, os parâmetros do equalizador tais como frequência central, ganho e largura de banda podem ser ajustados por meio de botões. Tavares sugere o aprimoramento de alguns pontos tais como a implementação de mais bandas, colocar o *display* em interrupção e melhorar o incremento dos botões. O presente projeto atende às sugestões do referido trabalho. Em relação ao número de bandas, foi possível implementar cinco bandas com ajuste de frequência e ganho para cada banda e mais o ajuste da largura de banda para os filtros de pico. No que diz respeito ao *display* e controle dos parâmetros, a placa STM32F746G *Discovery* já possui um *display touch screen*

integrado, o que possibilitou implementar uma interface gráfica com controle dos parâmetros em tempo real através da tela sensível ao toque. Foram criadas rotinas específicas com prioridades para evitar possível interferência provocada pelo manuseio da interface gráfica.

3. METODOLOGIA

Optou-se por iniciar este trabalho com testes práticos, implementando alguns exemplos que acompanham o *kit* de desenvolvimento. Assim, conseguimos analisar o comportamento do sistema no que diz respeito aos requisitos estabelecidos para o projeto. A partir do resultado da análise preliminar traçou-se uma rotina sequencial descrita como segue.

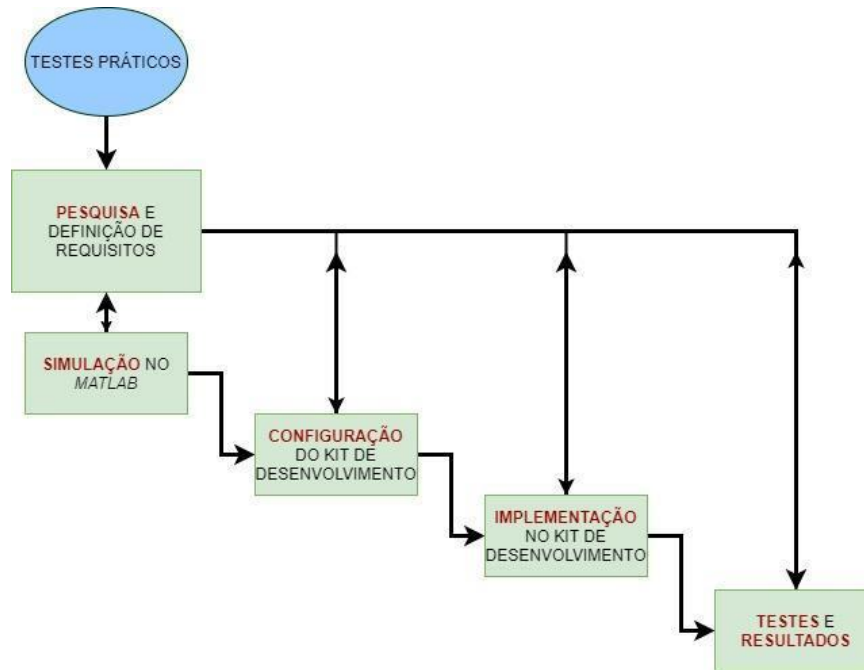
Primeiramente, fez-se a escolha dos tipos de filtro, definição do número de bandas do equalizador, definiu-se a taxa de amostragem, as equações e algoritmos para determinação dos coeficientes. O processo de filtragem poderia ser realizado através de coeficientes pré-estabelecidos ou calculados em tempo real. Neste projeto, optou-se pelo cálculo em tempo real.

Em seguida, simulou-se no Matlab os filtros *low* e *high shelving* e os filtros de pico. Foram utilizadas as funções do Matlab, *dsp.BiquadFilter* para o processamento e a função *fvtool* para a representação gráfica da resposta dos filtros.

Na etapa subsequente, foi realizada a configuração do kit de desenvolvimento. Nesta fase, escolheu-se o *software* de áudio a ser utilizado na fase de testes, o ambiente integrado de desenvolvimento (IDE: *Integrated Development Environment*) e as bibliotecas de *software*. Foram configurados o codec de áudio, a entrada e saída de áudio, e o ajuste do sinal de entrada.

Com o *kit* de desenvolvimento configurado, na fase seguinte implementaram-se os filtros, a interface gráfica e as funções para monitoramento do sinal de áudio no *display* gráfico.

Na última etapa, foram feitos os testes de bancada com o equalizador ligado nos computadores com *software* de áudio e ao osciloscópio. Foram testados os filtros e o *display* gráfico. Esta etapa também foi reservada para a análise dos resultados e considerações finais. O diagrama de blocos da Figura 20 mostra as etapas de desenvolvimento, observa-se que a etapa de pesquisa é constantemente consultada.

Figura 20 – Etapas de desenvolvimento

Fonte: AUTOR (2021).

4. DESENVOLVIMENTO

Neste capítulo será relatado todo o procedimento utilizado para simular, implementar e testar o equalizador paramétrico digital.

4.1 Projeto e Simulação

Para simular o equalizador foi utilizado o *software* Matlab. Os filtros aqui projetados são de segunda ordem do tipo IIR.

4.1.1 Requisitos Iniciais

Nesta fase do projeto foram escolhidos o número de bandas do equalizador, a taxa de amostragem do sinal, a configuração das faixas de atuação e os parâmetros de inicialização dos filtros. Para garantir o bom funcionamento do protótipo, optou-se por disponibilizar cinco bandas de atuação. Duas bandas são de filtros tipo *shelving* e as restantes são compostas por filtros de pico. Considerando que os filtros shelving devam atuar nas frequências extremas e os de pico nas frequências intermediárias, foram escolhidos os valores dos parâmetros para a simulação de cada tipo de filtro. A Tabela 7 mostra as faixas de os valores que serão aplicados na implementação do equalizador. Utilizou-se alguns desses parâmetros nesta seção para a simulação dos filtros. A taxa de amostragem escolhida foi de $f_s = 48 \text{ kHz}$, taxa normalmente utilizada em áudio.

Tabela 7 – Parâmetros aplicados na implementação dos filtros

Parâmetro	Passo	Low shelving	Peak 1	Peak 2	Peak 3	High shelving
Ganho [dB]	1	-12 a 12	-12 a 12	-12 a 12	-12 a 12	-12 a 12
Freq. Inicial [Hz]		225	1450	3500	8800	13 k
Faixas de frequência [Hz]	$\frac{\text{faixa}}{100}$	50 a 1000	250 a 4000	500 a 6500	1600 a 16 k	8000 a 18 k
Passo aproximado por banda [Hz]		9,5	37,5	60	144	100
Fator Q	1		1 a 20	1 a 20	1 a 20	

Fonte: AUTOR (2021).

4.1.2 Determinação dos Coeficientes

Os códigos para calcular os coeficientes foram implementados com base nas equações apresentadas em ZÖLZER (2008), mostradas nas tabelas 1 a 6.

A frequência de interesse, o ganho e a taxa de amostragem compõem os parâmetros de entrada de todos os filtros sendo que para os de pico foi acrescentado o fator de qualidade, parâmetro que ajusta a largura de banda. As funções retornam os coeficientes a_1 , a_2 , b_0 , b_1 e b_2 em duas matrizes, numerador e denominador respectivamente. Para determinar os coeficientes de cada filtro foram criadas funções específicas:

$function [A, B] = lowShelf(f_o, G, f_s)$	Função para cálculo do low shelf.
$function [A, B] = highShelf(f_o, G, f_s)$	Função para cálculo do high shelf.
$function [A, B] = Peak(f_o, G, Q, f_s)$	Função para cálculo do filtro de pico.

Onde f_o é a frequência central para os filtros de pico ou frequência de corte para os *shelving*, G , o ganho, f_s , a frequência de amostragem e Q , o fator de qualidade. Estas funções retornam os coeficientes do filtro, a_1, a_2 em A e b_0, b_1, b_2 em B .

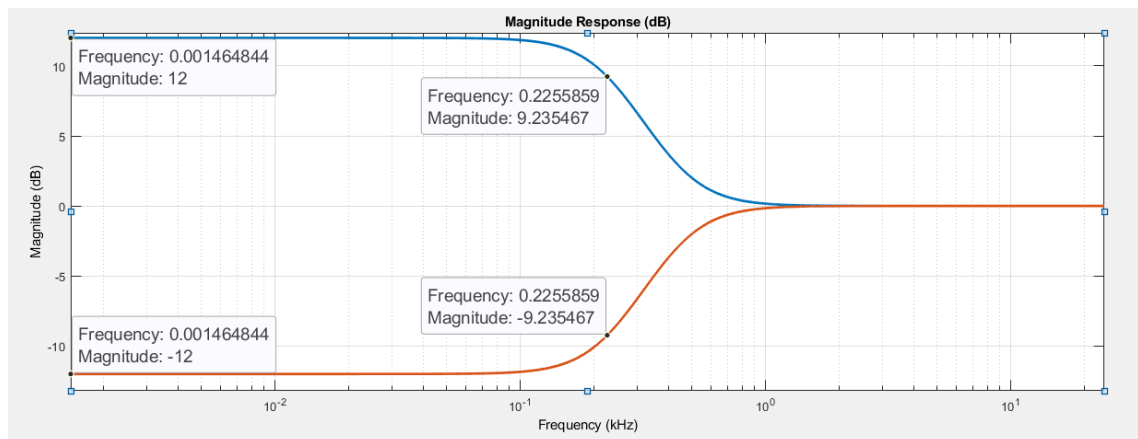
4.1.3 Simulação dos Filtros

Para a simulação dos filtros IIR foram utilizadas as funções *dsp.BiquadFilter* e *fvtool*. A *dsp.BiquadFilter* implementa seções biquadráticas em cascata onde os coeficientes de cada seção estão em uma linha de uma matriz de seções de segunda ordem, com N linhas por 6 colunas (N é o número de seções). Cada linha da matriz contém os coeficientes do numerador e do denominador (considera a normalização $a_0 = 1$) da seção correspondente do filtro. A ferramenta de visualização de filtro, função *fvtool*, foi utilizada para visualizar as amplitudes das respostas em frequência dos filtros.

4.1.3.1 Simulação dos Filtro Low Shelving e High Shelving

A Figura 21 mostra a simulação de filtros *low shelving* com ganhos de 12 e -12 dB e frequência de corte de 225 Hz. Os pontos marcados no gráfico identificam os valores dos ganhos para o nível DC e na frequência de corte.

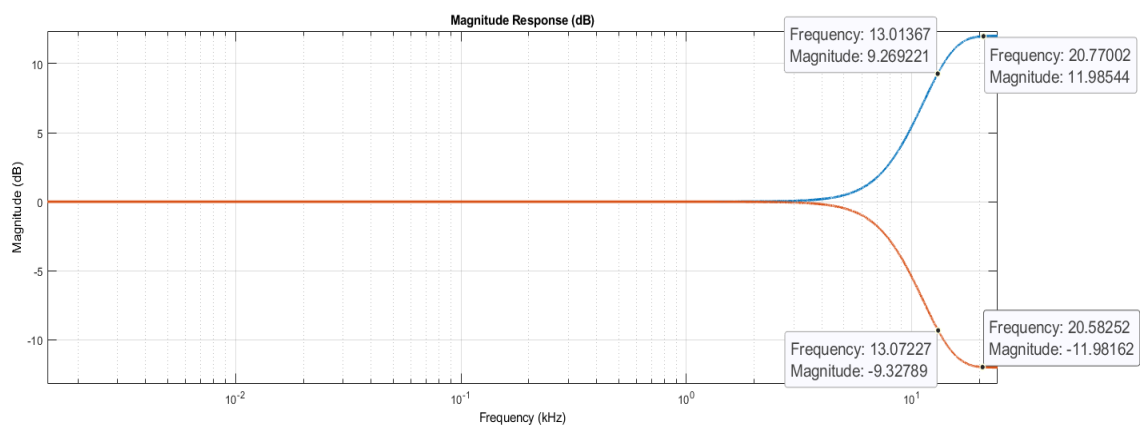
Figura 21 – Filtro low shelving com $f_c = 225$ Hz e ganhos de 12 e -12 dB



Fonte: AUTOR (2021).

A Figura 22 mostra a resposta em frequência de filtros *high shelving*, com frequência de corte foi ajustada em 13 kHz e ganhos em 12 e -12 dB. Os pontos no gráfico confirmam os valores de ganhos na frequência de 20 kHz e na frequência de corte.

Figura 22 – Filtro High shelving com $f_c = 13000$ Hz e ganhos de 12 e -12 dB



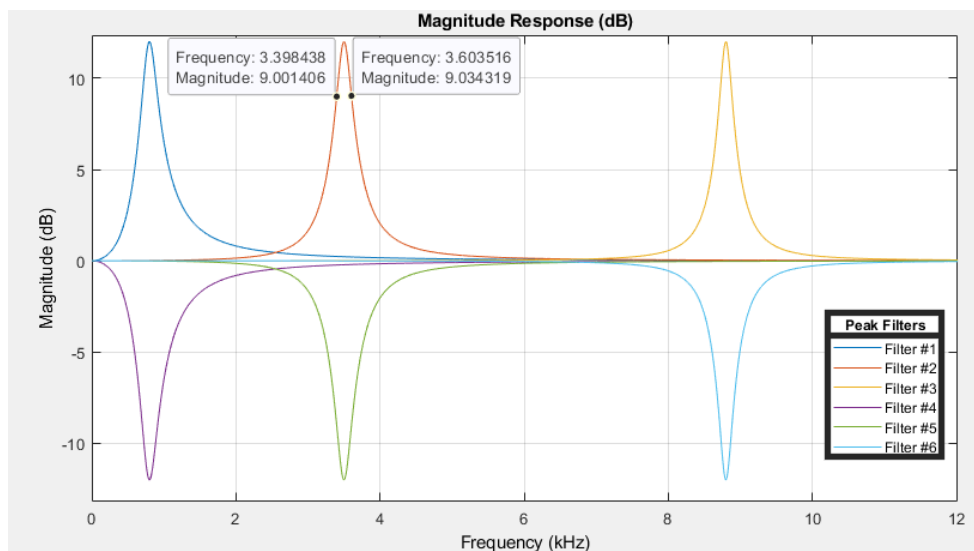
Fonte: AUTOR (2021).

Tanto nos filtros *low shelving* quanto no *high shelving*, pode-se constatar que as amplitudes na banda de interesse ficaram em 12 e -12 dB. Na região de corte a amplitude ficou em torno de -3 dB do módulo da amplitude máxima, o que era esperado.

4.1.3.2 Simulação dos Filtro de Pico

Foram simulados seis filtros de pico com diferentes frequências centrais (1450, 3500, 8800 Hz), ganhos de 12 e -12 dB e fatores de qualidade Q de 7,24; 17,5; 44, com $BW = 200$ Hz para todos os filtros. A Figura 23 mostra a simulação das três bandas do equalizador com dois filtros por banda. Os pontos marcados referem-se a amplitudes das frequências de corte inferior e superior do filtro de frequência central igual a 3500 Hz, com valor de $BW = 3604 - 3398 = 206$ Hz aproximadamente. Para melhor visualização da relação entre os filtros, o eixo das frequências está em escala linear.

Figura 23 – Filtros de pico com diferentes valores de frequência central

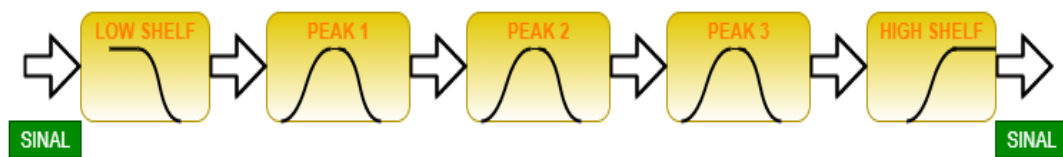


Fonte: AUTOR (2021).

4.1.3.3 Filtros em Cascata

Para exemplificar o funcionamento do equalizador, foi feita uma simulação com os cinco filtros em série. O sinal entra no sistema pelo *low shelving* e sai pelo filtro *high shelving* como mostrado na Figura 24.

Figura 24 – Filtros dispostos em série



Fonte: AUTOR (2021).

Os valores de entrada dos filtros mostrados na Tabela 8 foram escolhidos para a melhor visualização dos gráficos.

Tabela 8 – Parâmetros dos filtros (simulação)

	Low shelf	Peak 1	Peak 2	Peak 3	High shelf
<i>Ganho em dB</i>	-12 a 12	-12 a 12	-12 a 12	-12 a 12	-12 a 12
<i>freq. em Hz</i>	80	800	3000	6000	14000
<i>Fator Q</i>		10	10	10	

Fonte: AUTOR (2021).

Foram simulados 25 filtros com ganho de -12 a 12 dB com passos de 1 dB. Os filtros do equalizador de 5 bandas foram projetados com auxílio da função *dsp.Biquad Filter* do Matlab configurados como mostrado a seguir:

```
meuFiltro01 = dsp.BiquadFilter('SOSMatrix',[B.',[ones(5,1),A.']])
```

```
meuFiltro02 = dsp.BiquadFilter('SOSMatrix',[B.',[ones(5,1),A.']])
```

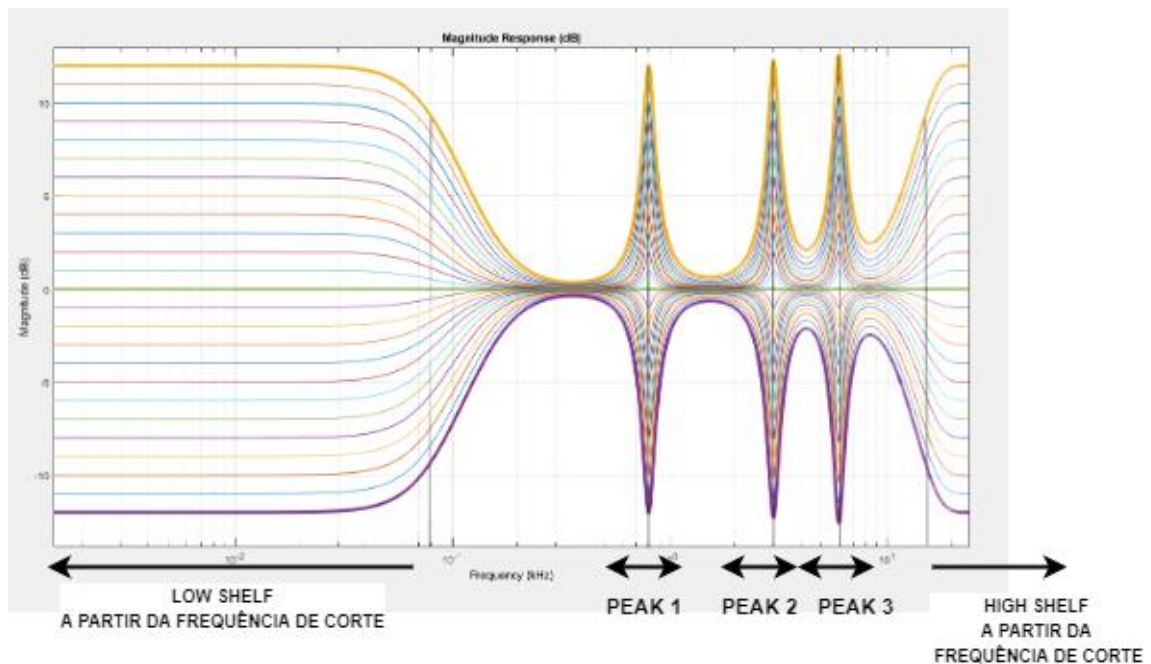
...

```
meuFiltro25 = dsp.BiquadFilter('SOSMatrix',[B.',[ones(5,1),A.']])
```

Onde B e A contêm os coeficientes de cada filtro (neste caso, 25 por filtro).

A Figura 25 mostra as amplitudes das respostas em frequência para os 25 valores de ganho dos filtros de segunda ordem arranjados na mesma sequência mostrada na Figura 24. Como pode ser percebido, cada linha contínua que vai do *low shelving* até o *high shelving* representa uma equalização.

Figura 25 – Filtros dispostos em série com ganhos de -12 a 12 dB



Fonte: AUTOR (2021).

Concluída esta etapa de projeto e simulação, iniciou-se a etapa de implementação no *kit* de desenvolvimento.

4.2 Implementação do Equalizador

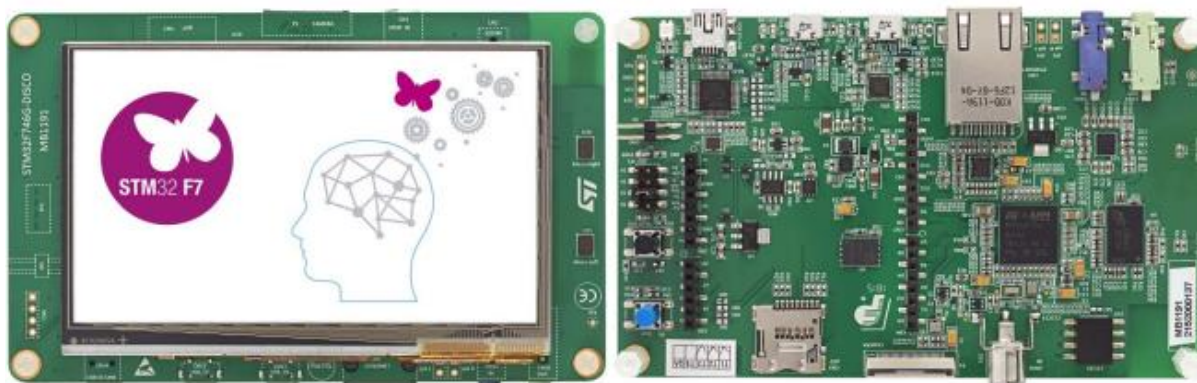
Após a validação das equações no Matlab, foi feita a implementação no kit de desenvolvimento STM32F746G *Discovery* da STMicroelectronics (Figura 26). Faz parte deste kit o microcontrolador STM32F746NGH6 com núcleo ARM Cortex-M7 de 32-bits que opera em frequência máxima de 216 MHz. Há uma gama de recursos na placa que facilita o desenvolvimento das aplicações. Os recursos mais importantes para este trabalho são: Memória *Flash* de 1 Megabyte, *ST-Link* para a depuração,

interface de áudio *SAI* (do inglês *Serial Audio interface*) com *DAC* estéreo que possui entrada e saída de áudio, microfones digitais *ST-MEMS*, tela *LCD-TFT* (*Liquid Crystal Display - Thin Film Transistor*) de 4 por 3 polegadas, *480 x 272 pixels* de resolução, colorida com painel multitoque capacitivo. Junto com a placa é fornecida a biblioteca de *software HAL* (*Hardware Abstraction Layer*) e mais vários exemplos (STMicroelectronics, 2020). O *kit* possui também um codec de áudio WM8994ECS/R da Cirrus Logic, o mesmo usado em *smartphones*, com 24 bits, 4 canais DAC e dois canais ADC, 100 dB de razão sinal-ruído (SNR, do inglês *Signal to Noise Ratio*) durante a reprodução, amplificador classe D e AB estéreo, além de operar com taxa de amostragem de 8 a 96 kHz, dentre outros recursos (CIRRUS LOGIC, 2018).

Além da HAL, foram instaladas as bibliotecas: CMSIS-DSP para o processamento do sinal, BSP que auxilia nas configurações entrada e saída, *STemWin* desenvolvida pela SEGGER para implementação da interface gráfica e a CMSIS RTOS para escalonamento de tarefas.

O ambiente utilizado para implementação e depuração dos códigos foi o STM32 CubeMX-IDE da própria STMicroelectronics, interface similar ao ambiente de programação *Eclipse*, mas com a vantagem de ter integrada uma interface gráfica para a configuração dos pinos e outros recursos de *hardware*.

Figura 26 – STM32F746G DISCOVERY, vistas superior e inferior.



Fonte: STMicroelectronics (2020).

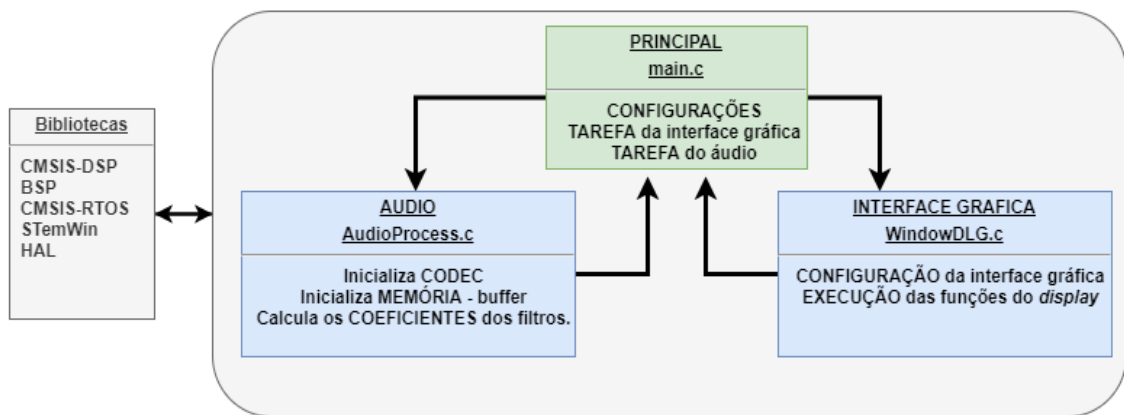
Antes da implementação propriamente dita deste projeto, foram executados alguns exemplos fornecidos pelo fabricante com o propósito de avaliar empiricamente a qualidade do áudio e da interface gráfica. Em seguida, com o auxílio da *STCubeMX*, interface gráfica integrada à IDE, foram configurados: Clock, DMA (do inglês: *Direct*

Memory Access), LTDC (do inglês: *LCD-TFT Display Controller*), interface de áudio, Timer e ADC.

4.2.1 Estruturação do *Software*

Para que o *software* ficasse organizado de forma a facilitar sua codificação e descrição, foi dividido em 3 blocos: PRINCIPAL, ÁUDIO e INTERFACE GRÁFICA. A Figura 27 mostra um diagrama de blocos da organização do programa.

Figura 27– Organização do programa em diagrama de blocos



Fonte: AUTOR (2021)

A seguir será descrito resumidamente o conteúdo e as funcionalidades de cada bloco, sendo que as bibliotecas utilizadas serão mencionadas no decorrer da descrição. Para cada bloco um arquivo foi implementado: *main.c*, *AudioProcess.c* e *WindowDLG.c*, como mostrado na Figura 27.

4.2.2 Bloco Principal

Este bloco inicia com as configurações do sistema em geral, parâmetros de inicialização dos filtros, alocação de memória para os *buffers*, funções de inicialização de tela e *flags*.

O ponto crítico deste projeto está no compartilhamento do tempo entre o processamento dos filtros e a resposta do *display* gráfico, tanto no ajuste dos controles pelo usuário quanto na visualização.

Para que os processos fossem compartilhados sem provocar danos ao som e nem lentidão no manuseio da interface gráfica, foram implementadas duas tarefas para executar rotinas específicas: processamento de áudio e interface gráfica, utilizando, para este propósito, a biblioteca de sistema operacional de tempo real (*Real Time Operating System - RTOS*) CMSIS RTOS.

Para a inicialização das tarefas da interface gráfica e de processamento do áudio, foram usadas as funções destacadas a seguir:

osThreadDef(name, thread, priority, instances, stacks);

As funções foram inicializadas como:

osThreadDef(GUI_Thread, GUIThread, osPriorityRealtime, 0, 512); (tarefa da interface)

osThreadDef(osAudio_Thread, AudioThread, osPriorityRealtime, 0, 512); (tarefa de áudio).

Onde: ***GUIThread*** e ***AudioThread*** são as funções criadas para executar as tarefas da interface de controle e processamento do áudio respectivamente.

Dentro da **tarefa da interface gráfica** (*GUIThread*), estão as chamadas das funções para calcular a FFT (*Fast Fourier Transform*), controlar o toque na tela e atualizar a interface de controle a cada intervalo de tempo. Como as prioridades das tarefas (áudio e interface gráfica) são iguais, foi chamada também a função *osDelay* com 100 ms, permitindo assim que a tarefa de áudio predomine no processo de escalonamento.

Dentro da **tarefa de áudio** (*AudioThread*), foi implementada a parte de processamento de sinais, com chamadas das funções necessárias para calcular os coeficientes, controlar os filtros e copiar dados para serem utilizados na exibição do sinal no display gráfico implementado na interface gráfica.

Para processar o sinal de áudio, fez-se uso da biblioteca de *software* CMSIS-DSP, que possui um extenso conjunto de funções comuns de processamento de sinais para dispositivos baseados no processador Cortex-M (KEIL CMSIS-DSP, 2010).

Levando em consideração os tipos de filtros escolhidos para esse projeto, optou-se pelo uso do *Biquad Cascade IIR*, conjunto de funções que implementam filtros recursivos IIR usando uma estrutura de forma direta tipo 2 transposta de ponto

flutuante com 32 bits. Os filtros são implementados em uma cascata de seções de segunda ordem (como já mostrado na Figura 14).

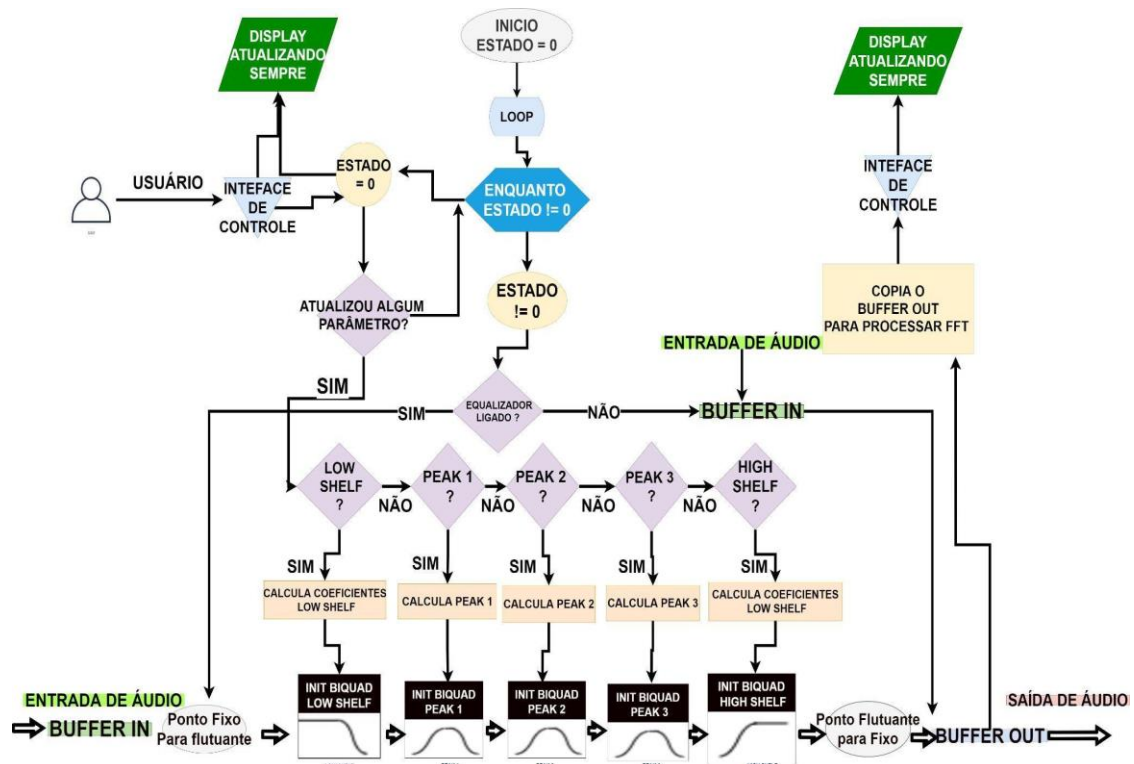
Cada estágio biquad implementa um filtro de segunda ordem que usa as equações recursivas:

$$\begin{aligned}y[n] &= b_0 x[n] + d_1 \\d_1 &= b_1 x[n] + a_1 y[n] + d_2 \\d_2 &= b_2 x[n] + a_2 y[n]\end{aligned}$$

Deve-se observar os coeficientes a_1 e a_2 , dependendo da ferramenta que foi usada para o projeto, há a necessidade de inverter o sinal para a implementação com a CMSIS-DSP. As variáveis de estado d_1 (Z^{-1}) e d_2 (Z^{-2}), são atualizadas após o processamento de cada bloco de dados; os coeficientes permanecem intocados (KEIL CMSIS-DSP, 2010).

Ainda no bloco principal (main.c), em um laço infinito dentro da **tarefa de áudio**, foi implementado o algoritmo para o processamento do sinal cujo diagrama de blocos está representado na Figura 28.

Figura 28 – Fluxograma do processamento de sinal dentro da tarefa de Áudio



Fonte: AUTOR (2021).

Antes de iniciar a descrição do fluxograma, é importante comentar sobre os parâmetros e *flags* utilizados:

a) A *flag ESTADO* indica o estado do *BUFFER IN* (espaço de memória alocado para armazenar o sinal de entrada) que pode ter três valores: 0 (zero) para inicializar, 1 para sinalizar quando o *buffer* chega na metade e 2 para quando o *buffer* estiver cheio.

b) os dados são atualizados a cada 100 ms, tempo suficiente para processá-los. O *buffer* é carregado em duas etapas respeitando as *flags* de *ESTADO* descritas no item (a). Considerando ainda que o tamanho dos *buffers* é 2048 (32 *bits*), o *ESTADO* 0 (zero) fica em laço durante 42 ms dividido em duas etapas (início até a metade e da metade até *buffer* cheio) de 21 ms aproximadamente.

c) foi denominada de interface de controle a tela sensível ao toque, por onde é feita a entrada dos parâmetros pelo usuário, e de display tudo que se refere à visualização; ambos integram a interface gráfica. Dentro da interface gráfica ainda foi implementado um mini *display* que será denominado de display gráfico.

O processamento do sinal ocorre dentro de um laço infinito. Inicia com um laço aguardando o *buffer* de entrada chegar até a metade ou completamente cheio. Enquanto isso, os parâmetros podem ser atualizados através de chamadas de funções para cálculo dos coeficientes e inicialização dos filtros. Importante acrescentar que os parâmetros serão atualizados se houver algum dado novo e somente o filtro referente a esse dado será modificado. Em seguida, quando a *flag ESTADO* é diferente de zero, é feita a verificação se o equalizador está ligado. Se *não estiver*, o *buffer* de entrada é diretamente copiado para o *buffer* de saída com uma cópia em memória para uso no processamento do *display* gráfico. Se o equalizador estiver ligado, o sinal na entrada é convertido de ponto fixo 16 *bits* para ponto flutuante de 32 *bits* e, em seguida, é processado.

O processamento é feito através dos filtros dispostos em série, no sentido das frequências mais graves para as frequências mais agudas, começando com o filtro *low shelving*, passando pelos 3 filtros de pico e finalizando com o *high shelving*. Após o processamento dos filtros, o sinal é reconvertido para ponto fixo 16 *bits*, copiado para a memória e para a saída através do *buffer out*.

4.2.3 Bloco Áudio

Dentro deste bloco estão a inicialização dos dispositivos de áudio, endereçamento e inicialização de memória para os *buffers* e cálculo dos coeficientes. Para a inicialização dos dispositivos (*drivers*) foi utilizada a biblioteca *Audio BSP driver*, biblioteca que facilita a configuração de dispositivos do *kit* de desenvolvimento.

A entrada e saída de áudio foram inicializadas com a função apresentada a seguir:

```
BSP_AUDIO_IN_OUT_INIT (INPUT_DEVICE_INPUT_LINE_1, OUTPUT_DEVICE_HEADPHONE,
FREQUENCY_48K, DEFAULT_AUDIO_IN_BIT_RESOLUTION, DEFAULT_AUDIO_IN_CHANNEL_NBR)
```

Esta função inicializa a gravação e reprodução em paralelo, *line in 1* para a entrada de áudio do dispositivo, *headphone* para a saída, 48 kHz de frequência de amostragem, 16 *bits* de resolução e dois canais de entrada e saída. Os *buffers* foram inicializados na memória SDRAM pela função descrita a seguir:

```
memset((uint16_t*) AUDIO_BUFFER_IN, 0, AUDIO_BLOCK_SIZE*2);
memset((uint16_t*) AUDIO_BUFFER_OUT, 0, AUDIO_BLOCK_SIZE*2);
onde AUDIO_BLOCK_SIZE = 2048 de 32bits.
```

Para o cálculo dos coeficientes dos filtros foram utilizados os mesmos algoritmos da simulação, porém com algumas adaptações para o código de linguagem de programação C.

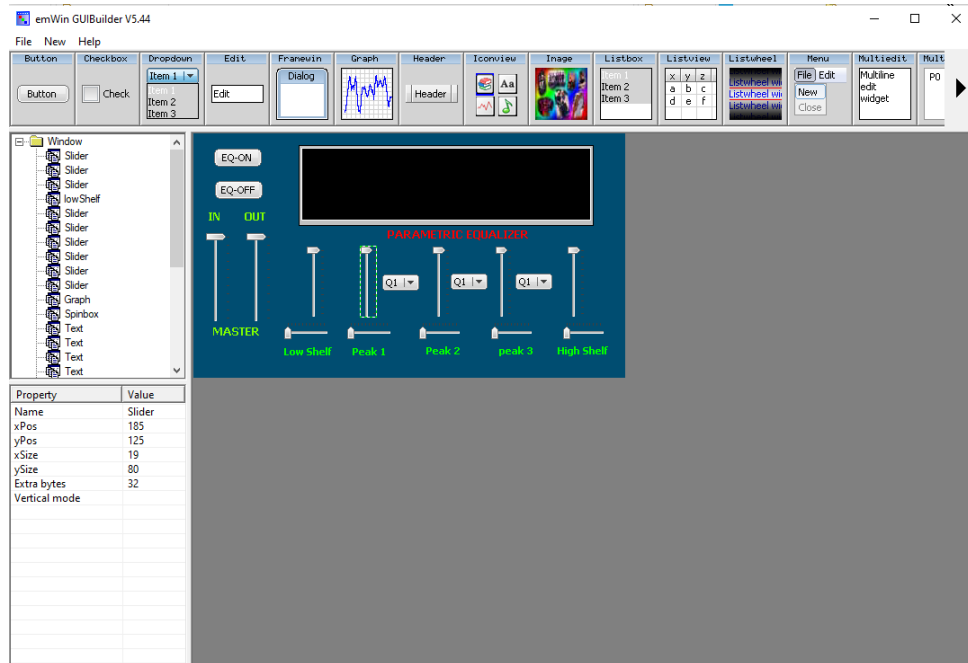
4.2.4 Bloco Interface Gráfica

Para controle e visualização dos parâmetros do sistema foi implementada uma interface gráfica com o auxílio do *software GUIBuilder*, ferramenta gráfica que vem junto à biblioteca STemWin fornecida pela STMicroelectronics em parceria com a SEGGER Microcontrollers GmbH.

Criou-se uma Interface Gráfica do Usuário (GUI, do inglês *Graphical User Interface*) por meio do GUIBuilder (Figura 29), *software* de *design* gráfico que possui um conjunto de *widgets* (ferramentas) para facilitar a criação de botões, controles

deslizantes, *display* gráfico, ou seja, itens que possibilitam a interatividade do usuário com o sistema.

Figura 29 – emWin GUIBuilder V5.44



Fonte: AUTOR (2020)

A interface criada é codificada num arquivo *.c* (*WindowDLG.c*) que interage com o resto do programa pela chamada da função *GUI_CreateDialogBox*. O recurso adicionado na interface gráfica só tem efeito após implementação das ações no *script*. A Figura 30 mostra um exemplo de implementação na GUI. Cada *widget* tem um identificador dentro de uma tabela, que pode reagir a três tipos de notificação: botão clicado, botão liberado e mudança de valor. Neste projeto foi muito utilizada a notificação de mudança de valor para o ajuste dos parâmetros e botão liberado para as *flags*.

Figura 30 – Implementação das *flags* ao ganho do filtro *peak 2*

```

case ID_SLIDER_1: // Notifications sent by 'G_PK2'
    switch(NCode) {
    case WM_NOTIFICATION_CLICKED:
        // USER START (Optionally insert code for reacting on notification message)
        // USER END
        break;
    case WM_NOTIFICATION_RELEASED:
        // USER START (Optionally insert code for reacting on notification message)
        frq = frqPK2;
        gain = gainPK2;
        flag.bandPK2 = 1;
        flag.band = 1;
        // USER END
        break;
    case WM_NOTIFICATION_VALUE_CHANGED:
        // USER START (Optionally insert code for reacting on notification message)
        hItem = WM_GetDlgItem(pMsg->hWin, ID_SLIDER_1);
        gainPK2 = SLIDER_GetValue(hItem);
        gainPK2 *= -1;

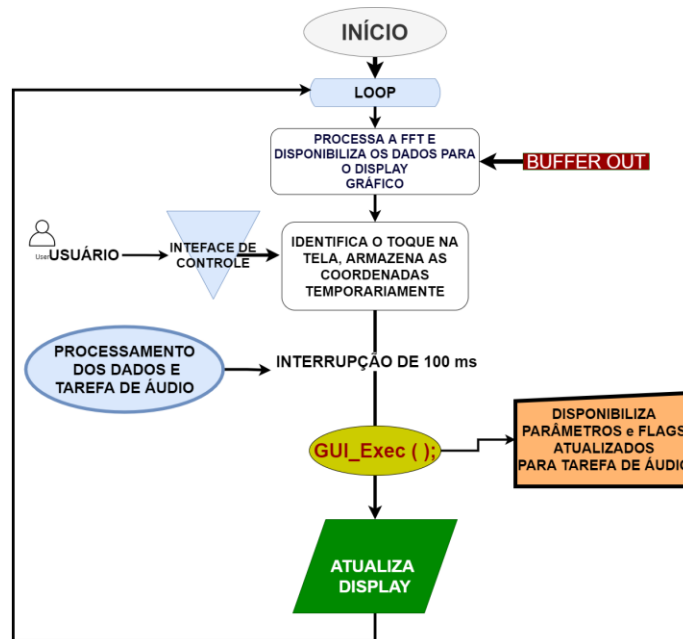
        hItem = WM_GetDlgItem(pMsg->hWin, ID_TEXT_11);
        sprintf(buf, "%d", gainPK2);
        TEXT_SetText(hItem, buf);
        // USER END
    }
}

```

Fonte: AUTOR (2020)

O fluxograma da Figura 31 mostra como a tarefa da interface gráfica é executada. Dentro do laço infinito é feita a chamada de várias funções. A primeira função é para processar a FFT (do inglês: *Fast Fourier Transform*) do sinal de saída dos filtros. Localizada dentro do arquivo *WindowDLG.c*, fornece dados para a representação do sinal no domínio da frequência no *display* gráfico. Após a entrada de dados pela detecção de toque, o fluxo é interrompido por 100 ms pela chamada da função *osDelay* para que os dados adquiridos sejam processados. Por último a função *GUI_exec* executa as ações requisitadas.

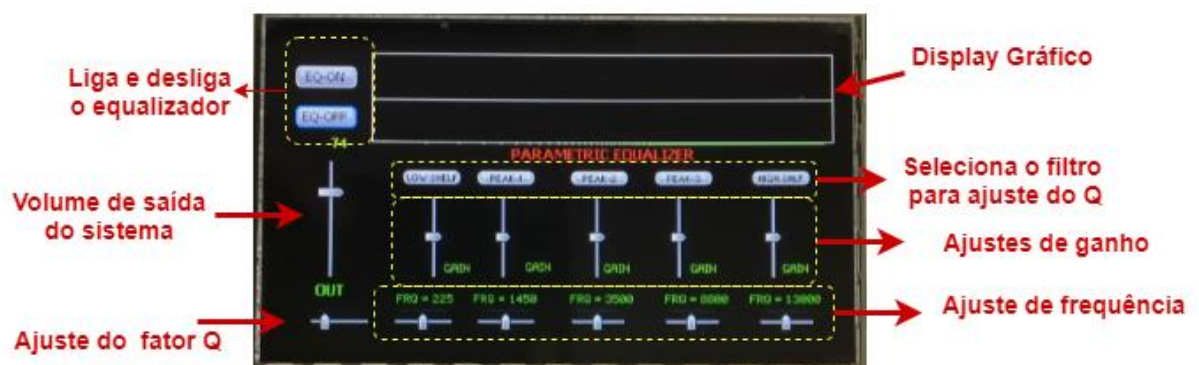
Figura 31 – Fluxograma da tarefa da interface gráfica



Fonte: AUTOR (2021).

A Figura 32 mostra o painel frontal da interface gráfica com as indicações dos controles e a funcionalidade de cada um deles. Os botões liga e desliga ativam ou inativam a filtragem como visto no fluxograma anterior. O volume de saída refere-se ao nível de sinal na saída do sistema em escala linear. O parâmetro Q dos filtros de pico é selecionado por botões individuais (*peak 1*, *peak 2*, *peak 3*) e quantificado pelo *slider* ajuste do fator Q . O ganho e a frequência podem ser ajustados individualmente por *sliders*.

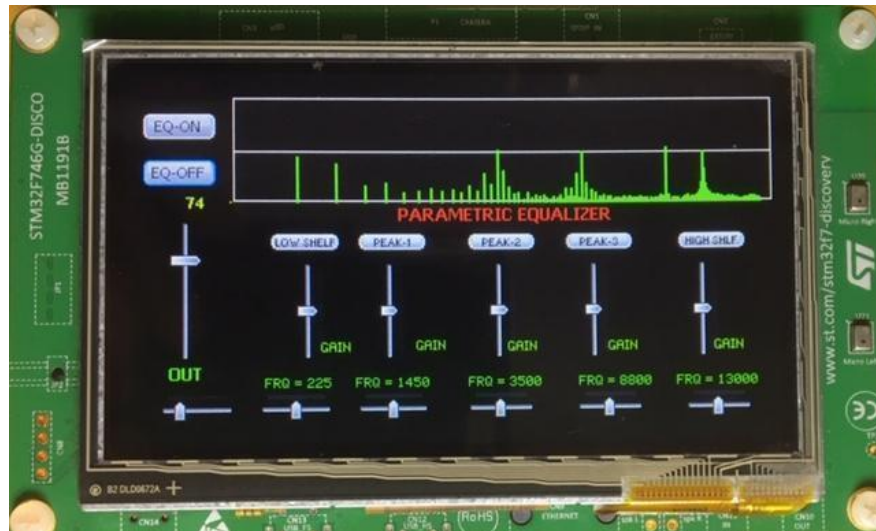
Figura 32 – Painel frontal da interface gráfica



Fonte: AUTOR (2020).

Foi implementado também um *display* gráfico para a visualização do sinal de áudio. A frequência está em escala logarítmica e a amplitude em dB.

Figura 33 – Display gráfico para monitoração do Sinal



Fonte: AUTOR (2021).

Toda parte frontal do equalizador tem dimensões de 13x8 cm. A Figura 33 mostra uma foto real do equalizador, exibindo as frequências de inicialização das bandas do equalizador no display gráfico.

Ao total, foram criados 7 botões, 12 *sliders*, 12 caixas de texto e um display gráfico para a exibição do sinal. A Figura 33 mostra a funcionalidade de cada conjunto de *widgets*.

5. TESTES E RESULTADOS

O equalizador foi submetido a uma bateria de testes com o objetivo de validar seu funcionamento no contexto de uma aplicação real. Para isso, montou-se em uma bancada os equipamentos necessários para tal procedimento, como mostrado na Figura 34.

Figura 34 – Bancada de testes



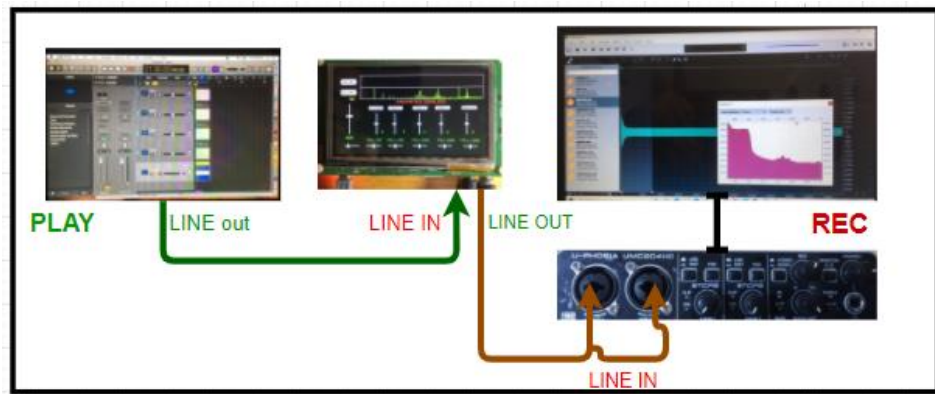
Fonte: AUTOR (2021).

Os materiais utilizados nos testes foram os seguintes:

- Um *notebook* para reproduzir o áudio gerado pelo *software Logic Pro* da *Apple*.
- Um equalizador paramétrico, objeto deste projeto.
- Um outro *notebook* para gravação do sinal pelo software de áudio da *Ocenaudio* da *Ocenaudio Team*.
- Interface de áudio Behringer com 2 canais de entrada.
- Cabos estéreos e conectores.

A figura 35 mostra a configuração da bancada de testes. O sinal elétrico analógico sai do notebook (PLAY) em dois canais, passa pela digitalização e processamento de sinais no equalizador paramétrico e entra em dois canais analógicos na interface de áudio onde é convertido novamente para digital e transmitido pela *USB* para o *notebook* (REC).

Figura 35 – Configuração de testes



Fonte: AUTOR (2021).

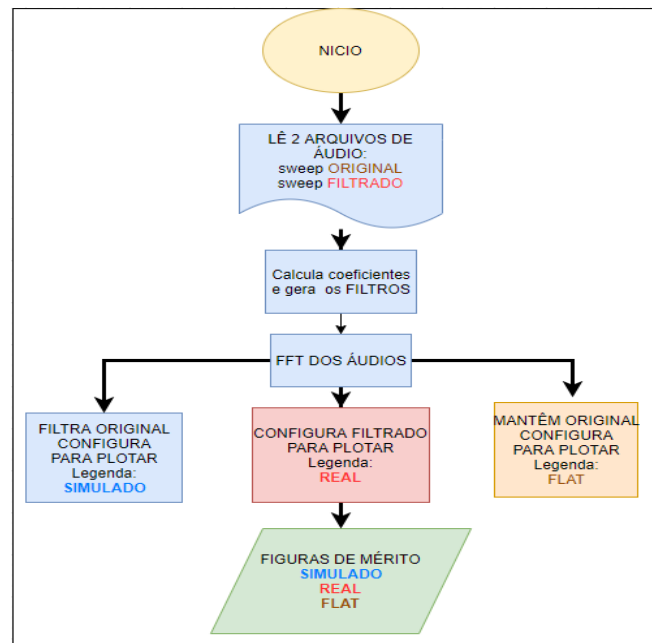
Preliminarmente, foi criado um arquivo de áudio no formato *.wav* com 5 segundos de duração, 48 kHz de amostragem, 16 bits de resolução, contendo uma onda senoidal com frequência variando linearmente de 20 Hz até 20 kHz, características de um sinal conhecido como *sweep*. O *sweep* reproduzido pelo *software* de áudio *Logic Pro* é transmitido eletronicamente para o equalizador paramétrico que processa e repassa para a interface de áudio conectada na *usb* do *notebook* REC, onde é gravado no *Ocenaudio* e salvo para ser utilizado posteriormente.

A validação do presente projeto foi realizada pela comparação de resultados obtidos de simulação e funcionamento real dos filtros e através da observação auditiva.

As figuras de mérito são apresentadas a seguir por meio de gráficos desenhados no *software* *Matlab* a partir dos sinais capturados e gravados em formato *WAV*.

O fluxograma do *script* de testes utilizado está mostrado na Figura 36. O *script* inicia com o carregamento de 2 arquivos de áudio: *sweep* original e *sweep* filtrado, aquele gravado no *Ocenaudio*, resultado do processamento efetuado pelo equalizador. Em seguida, são calculados os coeficientes e gerados os filtros para processar o arquivo original. Para mapear a amplitude em função da frequência foi feita a FFT de cada áudio. Os resultados são apresentados juntos em um gráfico para serem comparados. A legenda *SIMULADO* (azul) refere a simulação do sinal de áudio original, *REAL* (linha vermelha) refere ao sinal do *sweep* processado pelo equalizador, *FLAT* (linha marrom) é o *sweep* original sem equalização.

Figura 36 – Script de testes



Fonte: AUTOR (2021).

Os valores das frequências e do fator de qualidade foram escolhidos por um critério visual, buscou-se a melhor forma de identificar as bandas no gráfico. Para todas as situações usou-se o fator $Q = 20$; a Tabela 8 mostra os valores utilizados nos testes.

Tabela 9 – Valores utilizados para os testes

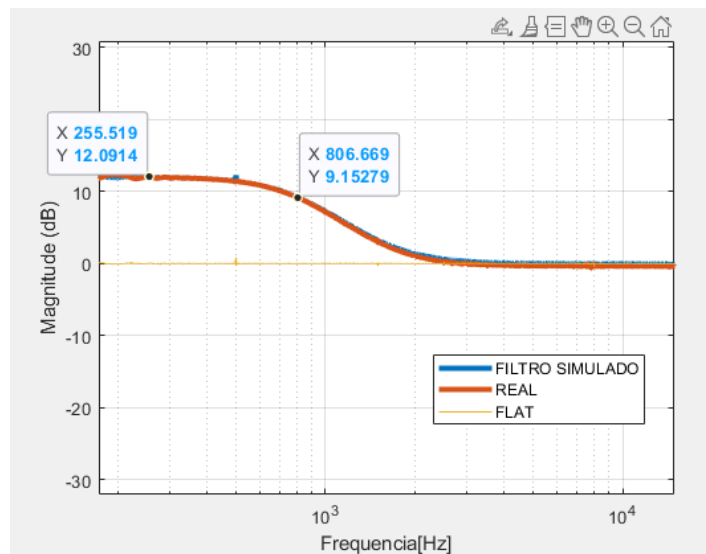
	Low shelving	Peak 1	Peak 2	Peak 3	High shelving
<i>freq.em Hz</i>	800	3000	6000	9000	14000
<i>BW [Hz}</i>		150	300	450	

Fonte: AUTOR (2021).

A seguir serão apresentadas as magnitudes das respostas em frequência dos filtros *low shelving*, *high shelving* e filtros de pico (*peak*). De maneira geral, a resposta do equalizador ficou dentro do esperado, as formas das curvas ficaram semelhantes às apresentadas na literatura, os coeficientes encontrados na simulação e implementação foram os mesmos.

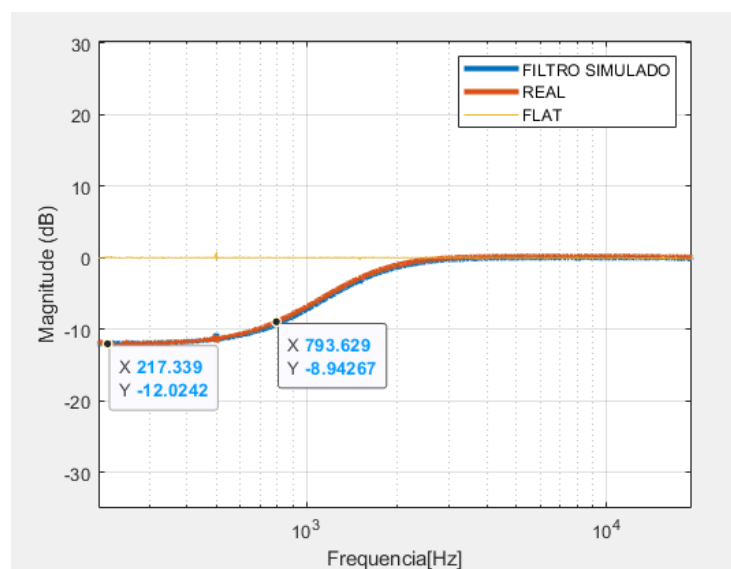
As Figuras 37 e 38 mostram os resultados dos testes do filtro *low shelving* com ganhos de 12 dB e -12 dB, respectivamente. Para melhor visualizar este tipo de filtro, foi aplicada uma frequência de corte de 800 Hz e desenhado em escala logarítmica no eixo das frequências. As curvas simulada e real concordam com os valores esperados na análise teórica.

Figura 37 – Resposta do *low shelving* equalizado e simulado, com $G = 12$ dB e $f_c = 800$ Hz



Fonte: AUTOR (2021).

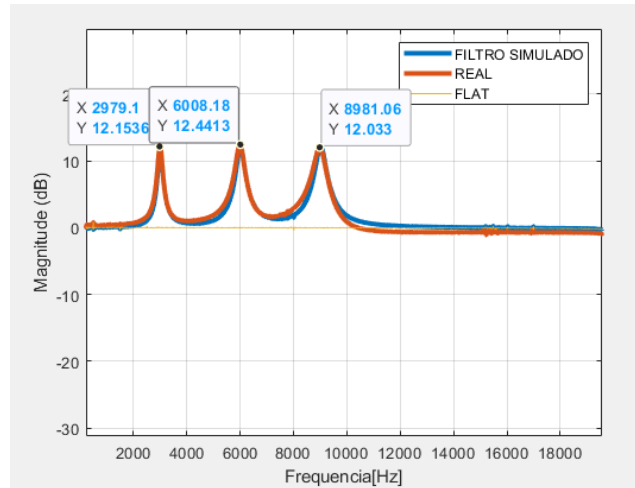
Figura 38 – Resposta do *low shelving* equalizado e simulado com $G = -12$ dB e $f_c = 800$ Hz



Fonte: AUTOR (2021).

Nos filtros de pico apresentados nas Figuras 39 e 40 foram aplicados 12 dB e -12 dB de ganho e frequências centrais de 3000 Hz, 6000 Hz e 9000 Hz. Foi aplicado um fator $Q = 20$ para todos os filtros resultando em largura de banda $BW = 150$ Hz, 300 Hz e 450 Hz para *peak 1*, *peak 2* e *peak 3*, respectivamente.

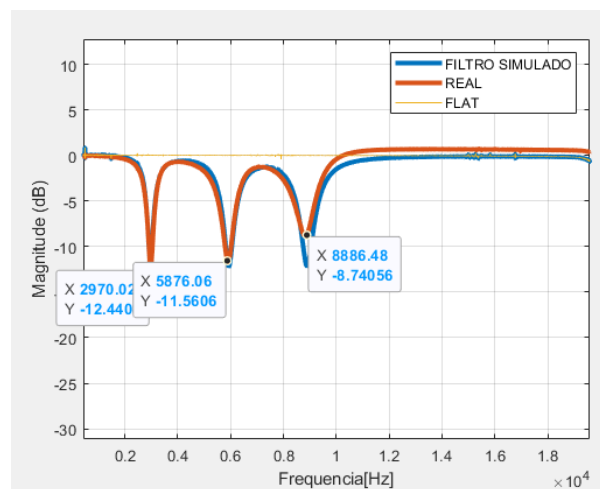
Figura 39 – Respostas dos filtros de pico do equalizador e simulado, com $Q = 20$, $G = 12$, $f_{0 \text{ peak } 1} = 3000$ Hz, $f_{0 \text{ peak } 2} = 6000$ Hz e $f_{0 \text{ peak } 3} = 9000$ Hz



Fonte: AUTOR (2021).

No filtro *peak 3* notou-se uma diferença entre o valor de ganho negativo aplicado e representado no gráfico. Supõe-se que seja a influência do filtro *high shelving* visto adiante.

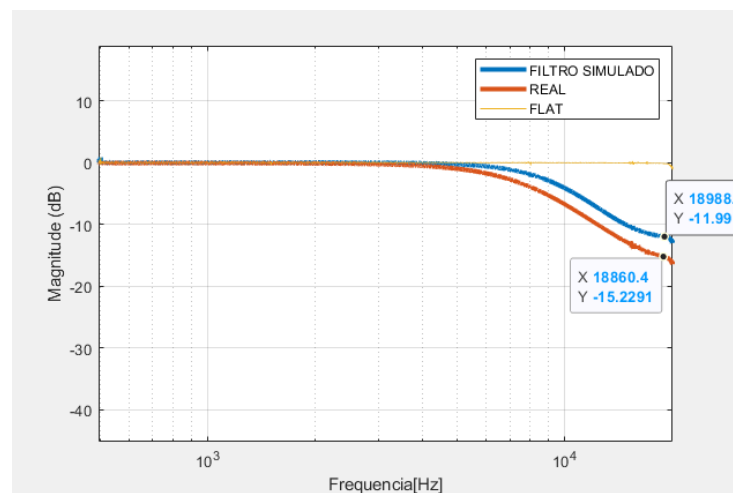
Figura 40 – Respostas dos filtros de pico do equalizador e simulado, com $Q = 20$, $G = -12$, $f_{0 \text{ peak } 1} = 3000$ Hz, $f_{0 \text{ peak } 2} = 6000$ Hz e $f_{0 \text{ peak } 3} = 9000$ Hz



Fonte: AUTOR (2021).

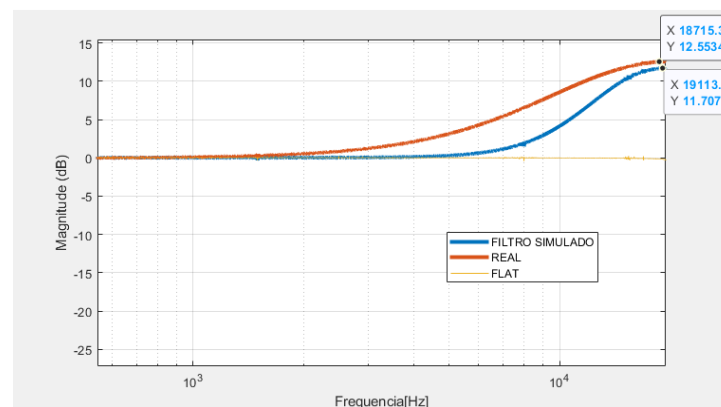
As Figuras 41 e 42 mostram os resultados dos testes do filtro *high shelving* com ganhos de 12 dB e -12 dB, respectivamente. Para melhor visualizar este tipo de filtro, foi aplicada uma frequência de corte de 14 kHz e desenhado em escala logarítmica no eixo das frequências. As curvas dos sinais simulado e real apresentaram uma divergência. A causa não foi identificada já que os cálculos estavam corretos. Deve-se levar em consideração a quantização do DSP que neste projeto está em 16 *bits*. A relação sinal-ruído na entrada do sistema de gravação também pode interferir, os sinais são do mesmo *sweep* mas com gravações diferentes, ou seja, um é o sinal original e o outro é uma cópia equalizada. Este assunto pode ser melhor investigado em trabalhos futuros.

Figura 41 – Resposta do filtro *high shelving* do equalizador e simulado, com $G = -12$ dB e $f_c = 14000$ Hz



Fonte: AUTOR (2021).

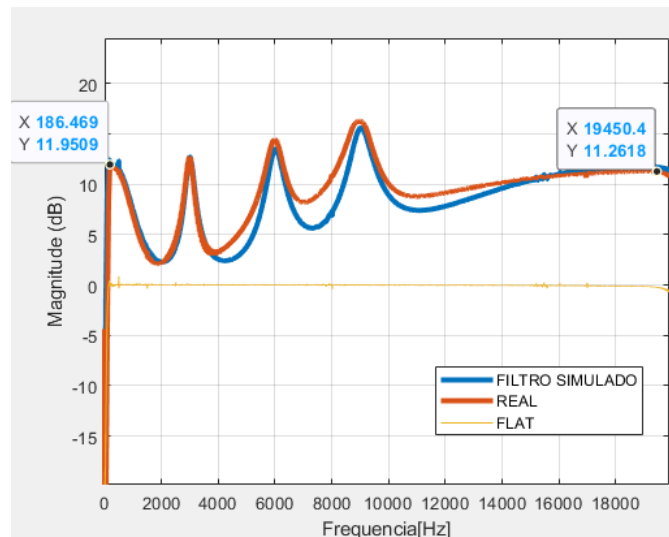
Figura 42 – Resposta do filtro *high shelving* do equalizador e simulado, com $G = 12$ dB e $f_c = 14000$ Hz



Fonte: AUTOR (2021).

A Figura 43 mostra os resultados das 5 bandas na simulação e na saída do equalizador real. A frequência está em escala linear e a amplitude em dB.

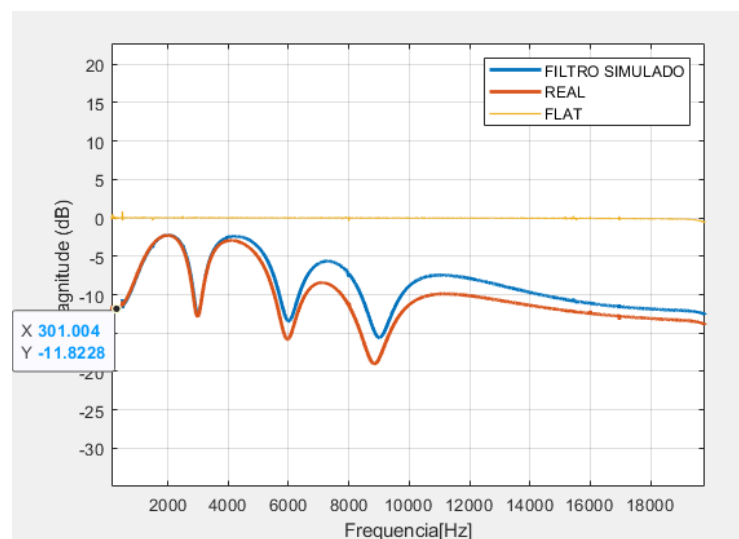
Figura 43 – $G = 12$ dB para todos os filtros



Fonte: AUTOR (2021).

Pode-se notar na figura, a semelhança entre o sinal simulado e o sinal real. Da esquerda para a direita os cumes representam os filtros *low shelf*, *peak 1*, 2 e 3, e, por último, o *high shelving* em forma de platô. O efeito dos filtros em série resultou em valores de ganho das bandas acima do valor estipulado, observou-se que os ganhos dos filtros de pico 2 e 3 foram acrescidos por influência do *high shelving*.

Figura 44 – Ganho $G = -12$ todos os filtros



Fonte: AUTOR (2021).

Na Figura 44 está representado o efeito do equalizador com todas as bandas em -12 dB de ganho. Foi observado mais uma vez a influência do high *shelving* nas bandas próximas.

Por meio da percepção auditiva do sinal de áudio, foi analisada a interferência da interface gráfica no sinal de saída do equalizador. Equipamentos que processam áudio e gráficos intermitentemente, podem apresentar ruídos auditivos indesejáveis se não forem projetados devidamente. Como mencionado anteriormente, isso ocorre porque a tarefa gráfica interrompe a de áudio e vice-versa. O equalizador do presente projeto foi implementado de modo a utilizar o tempo ocioso do processador enquanto os *buffers* de entrada e saída de áudio são processados no periférico. Para uma taxa de amostragem de 48 kHz, o algoritmo recursivo tem 20,83 μ s de tempo disponível para cada amostra e com os processadores hoje disponíveis foi possível implementar um filtro recursivo para a presente aplicação de áudio (ZÖLZER,2008). Na necessidade de perceber o tipo de ruído auditivo que poderia ser provocado pela interface, o *buffer* de áudio foi dimensionado com um tamanho mínimo, suficiente para provocar ruído. Ao manusear a interface gráfica, ia-se observando o áudio na saída. Constatou-se a diminuição de ruído à medida que o *buffer* era aumentado. O procedimento continuou até o ruído sumir por completo com o *buffer* de 2048 amostras, valor utilizado no projeto.

6. CONSIDERAÇÕES FINAIS

Baseado nos resultados apresentados dentro das especificações, pode-se dizer que é possível implementar o equalizador de áudio proposto (um equalizador paramétrico digital de 5 bandas que opere em tempo real), no *hardware* utilizado neste trabalho. As simulações validaram as equações utilizadas para o cálculo dos coeficientes, o *kit* de desenvolvimento utilizado atendeu às necessidades de implementação, os testes apresentaram resultados dentro do esperado, os filtros responderam bem aos comandos da interface gráfica sem apresentar ruídos audíveis, percebeu-se claramente a variação das tonalidades sem distorção, o sistema respondeu ao fator Q dentro da largura de banda como previsto, o tempo de resposta do *display* em relação ao sinal ficou praticamente imperceptível à visão.

É importante considerar aqui, a quantidade de bibliotecas necessárias para o desenvolvimento deste projeto, suas instalações e configurações consumiram um tempo valioso. Pode-se mencionar, por exemplo, a inicialização das entradas de áudio analógicas pela função `BSP_AUDIO_IN_OUT_Init` da biblioteca `BSP`, que não habilitava a entrada de linha e nem aceitava o parâmetro `INPUT_DEVICE_INPUT_LINE_1`. Foi necessário modificar algumas linhas dentro da referida função. Também foi alterado o nível de sinal de saída *default* da placa que vem configurado em 30 dB, nível muito alto para a finalidade deste projeto. Foi alterado para 3 dB (CIRRUS LOGIC, 2018) na configuração do codec (ver *apêndice 1*).

Pode-se considerar que o equalizador aqui projetado, mesmo implementado em *software*, está em um *hardware* que não depende de computador, pode trabalhar *standalone* com entradas e saídas analógicas, fatores que proporcionam maior versatilidade no uso deste equipamento. Os códigos dos arquivos `main.c`, `AudioProcess.c` e `WindowDLG.c` estão listados nos apêndices 2 a 4.

Como sugestão para trabalhos futuros, elencamos: aumentar a ordem dos filtros, usar somente um botão para liga e desliga dos filtros inclusive com um *led* para sinalizar, aperfeiçoar o *display* gráfico, explorar o equalizador de 5 bandas presente no codec (CIRRUS LOGIC, 2018), investigar a diferença nas frequências mais altas, implementar entrada e saída digital para trabalhar com outros equipamentos como um crossover digital, por exemplo.

REFERÊNCIAS

- BAUER, Matheus. **Sistema para Equalização Paramétrica de Áudio em Tempo Real**. 2018. TCC – Curso de Engenharia da Computação, Universidade Tecnológica Federal do Paraná, Pato Branco, PR, 2018. Disponível em: <http://repositorio.roca.utfpr.edu.br/jspui/bitstream/1/12120/1/PB_COENC_2018_2_12.pdf>
- BODANESE, Joao Paulo. **IMPLEMENTAÇÃO DE UM EQUALIZADOR DE ÁUDIO EM DSP**. 2008. 82 f. Monografia (Especialização), Centro Federal de Educação Tecnológica de Santa Catarina (CEFET), Florianópolis, SC, 2008.
- CIRRUS LOGIC, Inc. **WM8994, Multi-channel Audio Hub CODEC for Smartphones**. Rev 4.6, 2018. Disponível em: <www.cirrus.com>.
- CLARK, Robin John. **Investigation into digital audio equaliser systems and the effects of arithmetic and transform errors on performance**. 2001. 289 f. Tese (Doutorado) - Department of Communication And Electronic Engineering Faculty Of Technology, University of Plymouth, 2001. Cap. 2.
- FOX MEDIA TECH Inc. **Audio Shelving EQ: What Are Low Shelf & High Shelf Filters?** 2021. Blog My New Microphone. Disponível em: <<https://mynewmicrophone.com/audio-shelving-eq-what-are-low-shelf-high-shelf-filters/>>. Acesso em: 14 março 2021
- HANNA, C. **Real-Time Control of DSP Parametric Equalizers**. 1994. Disponível em: <[AES13-041_Control_of_DSP_Parametric_EQs.pdf](#) (thatcorp.com)> Acesso em: em 3 abril 2021
- LATHI, B. P. **Sinais e sistemas lineares**. 2. ed. Porto Alegre: Bookman, 2007.
- KEIL CMSIS-DSP. **CMSIS DSP Software Library**. 2020. Disponível em : < CMSIS DSP Software Library (keil.com) > . Acesso em: 17 março 2021.
- MICROCHIP, Technology Inc. **Audio Equalizer Filtering Library Help**, 2013.
- NOCETI FILHO, Sidnei; DALCASTAGNÊ, André Luís; GONTIJO, Walter Antônio. **Filtros e Efeitos de Áudio Analógicos e Digitais**. In: CONGRESSO DE ENGENHARIA DE ÁUDIO, 9., 2011, São Paulo. Anais 9. Congresso de Engenharia de Áudio. São Paulo: Aes, 2011. v. 1, p. 9 - 20.
- OWSINSKI, Bobby. **The Mastering Engineer's Handbook**. 2. ed. Estados Unidos: Bobby Owsinski Media Group, 2007.
- PINTEREST. **An Introduction to Digital Signal Processing, Technical Articles**. 2021. Disponível em < <https://br.pinterest.com/pin/855965472898466120/>> . Acesso em: 26 abril, 2021.

RB-Recording Blogs. **Shelving Filters**. 2020. Disponível em:
<<https://www.recordingblogs.com/wiki/shelving-filter>>. Acesso em: 30 abril, 2021.

REBOOT RECORDING. **EQ Shelving: High Shelf and Low Shelf Filters Explained**. 2021. Disponível em: <<https://rebootrecording.com/high-and-low-shelf/>>. Acesso em 30 março 2021.

SILVA, Marco Aurélio Gonçalves da. **Filtros Digitais Aplicados em Sinais de Áudio**. 2007. 62 f. TCC (Graduação) - Curso de Ciência da Computação, Universidade Federal de Juiz de Fora, Juiz de Fora, 2007.

SMITH, Steven W. **The Scientist and Engineer's Guide to Digital Signal Processing**. S: California Technical Pub, 1997. Disponível em:
< <http://www.dspguide.com/ch14.htm>> Acesso em: 29 abril 2021

STMICROELECTRONICS. **32F746GDISCOVERY board: user manual**. 2017.

TAGIMA. **Dallas EQ**. 2021. Disponível em:
<<https://www.tagima.com.br/detalhe.php?cod=56e6ee625386d#>>. Acesso em: 3 de maio 2021.

TAVARES, Diogo. **Crossover digital com equalização paramétrica ajustável**. 2020. 81 f. TCC (Graduação) - Curso de Engenharia Eletrônica, Departamento Acadêmico de Eletrônica, Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina, Florianópolis, 2020.

TECLACENTER. **Como Usar Equalizadores e Filtros**. 2015. Disponível em:
<<https://www.teclacenter.com.br/blog/como-usar-equalizadores-e-filtros/>> Acesso em: 1 maio 2021.

UNIVERSAL AUDIO. **Manley Massive Passive EQ**. 2019. Disponível em:
<<https://www.uaudio.com/uad-plugins/equalizers/manley-massive-passive.html>> Acesso em: 30 março 2021

ZÖLZER, Udo. **Digital Audio Signal Processing**. 2. ed. Hamburgo: John Wiley & Sons Ltd, 2008. 334 p.

WEISS Engineering. **EQ1 seven-band equalizer**. 2021. Disponível em:
<<https://www.weiss.ch/products/eq1>> Acesso em: 30 março 2021

APÊNDICE 1

Ateração feita na função *BSP_AUDIO_IN_OUT_Init* da biblioteca *BSP* versão *v2.0.2, 2016*, que para habilitar a entrada de linha *INPUT_DEVICE_INPUT_LINE_1*, no arquivo *stm32746g_discovery_audio.c*

```

982~ uint8_t BSP_AUDIO_IN_OUT_Init(uint16_t InputDevice, uint16_t OutputDevice, uint32_t AudioFreq, uint32_t BitRes, uint32_t ChnINbr)
983 {
984     uint8_t ret = AUDIO_ERROR;
985     uint32_t deviceid = 0x00;
986     uint32_t slot_active;
987
988     //if (InputDevice != INPUT_DEVICE_DIGITAL_MICROPHONE_2) /* Only MICROPHONE_2 input supported */
989     //if (InputDevice != INPUT_DEVICE_INPUT_LINE_1) /* Only MICROPHONE_2 input supported */
990     if ((InputDevice != INPUT_DEVICE_INPUT_LINE_1) && (InputDevice != INPUT_DEVICE_DIGITAL_MICROPHONE_2))
991     {
992         ret = AUDIO_ERROR;
993     }
994     else
995     {
996         /* Disable SAI */
997         SAIx_In_DeInit();
998         SAIx_Out_DeInit();
999
1000         /* PLL clock is set depending on the AudioFreq (44.1kHz or 48kHz groups) */
1001         BSP_AUDIO_OUT_ClockConfig(&audio_in_sai, AudioFreq, NULL); /* Clock config is shared between AUDIO IN and OUT */
1002
1003         /* SAI data transfer preparation:
1004         Prepare the Media to be used for the audio transfer from SAI peripheral to memory */
1005         audio_in_sai.Instance = AUDIO_IN_SAIx;
1006         if(HAL_SAI_GetState(&audio_in_sai) == HAL_SAI_STATE_RESET)
1007         {
1008             /* TODO: the SAI MSP: this __weak function can be redefined by the application*/
1009             BSP_AUDIO_IN_MspInit(&audio_in_sai, NULL);
1010         }
1011
1012         /* SAI data transfer preparation:
1013         Prepare the Media to be used for the audio transfer from memory to SAI peripheral */
1014         audio_out_sai.Instance = AUDIO_OUT_SAIx;
1015         if(HAL_SAI_GetState(&audio_out_sai) == HAL_SAI_STATE_RESET)
1016         {
1017             /* TODO: the SAI MSP: this __weak function can be redefined by the application*/
1018             BSP_AUDIO_OUT_MspInit(&audio_out_sai, NULL);
1019         }
1020
1021         /* Configure SAI in master mode :
1022         * - SAI2_block_A in master TX mode
1023         * - SAI2_block_B in slave RX mode synchronous from SAI2_block_A
1024         */
1025         //if (InputDevice == INPUT_DEVICE_DIGITAL_MICROPHONE_2)
1026         if (((InputDevice != INPUT_DEVICE_INPUT_LINE_1) && (InputDevice != INPUT_DEVICE_DIGITAL_MICROPHONE_2))
1027         {
1028             slot_active = CODEC_AUDIOFRAME_SLOT_13;
1029         }
1030         else
1031         {
1032             slot_active = CODEC_AUDIOFRAME_SLOT_02;
1033         }
1034         SAIx_In_Init(SAI_MODEMASTER_TX, slot_active, AudioFreq);
1035
1036         /* WM8994 codec initialization */
1037         deviceid = wm8994_drv.ReadID(AUDIO_I2C_ADDRESS);
1038
1039         if((deviceid) == WM8994_ID)
1040         {
1041             /* Reset the codec Registers */
1042             wm8994_drv.Reset(AUDIO_I2C_ADDRESS);
1043             /* Initialize the audio driver structure */
1044             audio_drv = &wm8994_drv;
1045             ret = AUDIO_OK;
1046         }
1047         else
1048         {
1049             ret = AUDIO_ERROR;
1050         }
1051
1052         if(ret == AUDIO_OK)
1053         {
1054             /* Initialize the codec internal registers */
1055             audio_drv->Init(AUDIO_I2C_ADDRESS, InputDevice | OutputDevice, DEFAULT_AUDIO_IN_VOLUME, AudioFreq);
1056         }
1057         return ret;
1058     }
1059 }
1060
1061 ~~~~~

```

Alteração também no nível de saída da placa de 30 dB para 3 dB (*arquivo wm8994.h*)

```

case INPUT_DEVICE_INPUT_LINE_1 :
/* IN1LN_TO_IN1L, IN1LP_TO_VMID, IN1RN_TO_IN1R, IN1RP_TO_VMID */
counter += CODEC_IO_Write(DeviceAddr, 0x28, 0x0011);

/* Disable mute on IN1L_TO_MIXINL and +30dB on IN1L PGA output */
counter += CODEC_IO_Write(DeviceAddr, 0x29, 0x0020); // 0x0020 setado em 3dB para retornar para 30 dB setar em 0x0035

/* Disable mute on IN1R_TO_MIXINL, Gain = +30dB */
counter += CODEC_IO_Write(DeviceAddr, 0x2A, 0x0020); // 0x0020 setado em 3dB para retornar para 30 dB setar em 0x0035

```

main.c

```

1 /* USER CODE BEGIN Header */
2 /**
3  * *****
4  * @file           : main.c
5  * @brief          : Main program body
6  * *****
7
8  *
9  * *****
10 */
11 /* USER CODE END Header */
12 /* Includes -----*/
13 #include "main.h"
14 #include "cmsis_os.h"
15 #include "fatfs.h"
16 #include "usb_host.h"
17
18 /* Private includes -----*/
19 /* USER CODE BEGIN Includes */
20 #include "arm_math.h"
21 #include <stdlib.h>
22 #include <stdio.h>
23 #include "string.h"
24 #include <math.h>
25
26
27
28
29 /* USER CODE END Includes */
30
31 /* Private typedef -----*/
32 /* USER CODE BEGIN PTD */
33
34 /* USER CODE END PTD */
35
36 /* Private define -----*/
37 /* USER CODE BEGIN PD */
38 /* USER CODE END PD */
39
40 /* Private macro -----*/
41 /* USER CODE BEGIN PM */
42
43 /* USER CODE END PM */
44
45 /* Private variables -----*/
46 ADC_HandleTypeDef hadc3;
47
48 CRC_HandleTypeDef hcrc;
49
50 DCMI_HandleTypeDef hdcmi;
51
52 DMA2D_HandleTypeDef hdma2d;
53
54 ETH_HandleTypeDef heth;
55
56 I2C_HandleTypeDef hi2c1;
57 I2C_HandleTypeDef hi2c3;
58
59 LTDC_HandleTypeDef hltdc;
60
61 QSPI_HandleTypeDef hqspi;
62

```

main.c

```
63 RTC_HandleTypeDef hrtc;
64
65 SAI_HandleTypeDef hsai_BlockA2;
66 SAI_HandleTypeDef hsai_BlockB2;
67 DMA_HandleTypeDef hdma_sai2_a;
68 DMA_HandleTypeDef hdma_sai2_b;
69
70 SD_HandleTypeDef hsd1;
71
72 SPDIFRX_HandleTypeDef hspdif;
73
74 SPI_HandleTypeDef hspi2;
75
76 TIM_HandleTypeDef htim1;
77 //TIM_HandleTypeDef htim2;
78 //TIM_HandleTypeDef htim3;
79 //TIM_HandleTypeDef htim5;
80 //TIM_HandleTypeDef htim8;
81 //TIM_HandleTypeDef htim12;
82
83 UART_HandleTypeDef huart1;
84 UART_HandleTypeDef huart6;
85
86 SDRAM_HandleTypeDef hsdram1;
87
88 osThreadId defaultTaskHandle;
89 /* USER CODE BEGIN PV */
90
91
92 typedef enum
93 {
94     BUFFER_OFFSET_NONE = 0,
95     BUFFER_OFFSET_HALF = 1,
96     BUFFER_OFFSET_FULL = 2,
97 } BUFFER_StateTypeDef;
98
99 volatile uint32_t audio_rec_buffer_state;
100 int j = 0;
101 SAI_HandleTypeDef haudio_in_sai ;
102 Led_TypeDef led1;
103 TS_StateTypeDef state;
104 extern SAI_HandleTypeDef haudio_out_sai;
105
106 ///////////////////////////////////////////////////FILTRO IIR//////////////////////////////////////
107
108 #define AUDIO_BLOCK_SIZE ((uint32_t)2048)
109 #define SIGNAL_SAMPLES AUDIO_BLOCK_SIZE/2
110 #define AUDIO_BUFFER_IN AUDIO_REC_START_ADDR /* In SDRAM */
111 #define AUDIO_BUFFER_OUT (AUDIO_REC_START_ADDR + (AUDIO_BLOCK_SIZE * 2)) /* In SDRAM */
112 #define float_buffer_in (AUDIO_REC_START_ADDR + (AUDIO_BLOCK_SIZE * 4))
113 #define float_buffer_out (AUDIO_REC_START_ADDR + (AUDIO_BLOCK_SIZE * 6))
114 #define AUDIO_buffer_aux (AUDIO_REC_START_ADDR + (AUDIO_BLOCK_SIZE * 8))
115
116 //unsigned char logmsgbuff[128];
117 // #define BUFSIZE 9
118
119 #define numStages_IIR 1
120 #define NUM_TAPS_IIR 5*numStages_IIR
121
122 float32_t pStateIIR0 [2*numStages_IIR];
123 float32_t pStateIIR1 [2*numStages_IIR];
124 float32_t pStateIIR2 [2*numStages_IIR];
```


main.c

```

187 static void MX_SAI2_Init(void);
188 void StartDefaultTask(void const * argument);
189
190 /* USER CODE BEGIN PFP */
191 void Audio_Exec_DMA(void);
192 static void GUIThread(void const * argument);
193 static void AudioThread(void const * argument);
194 extern void audioExec(void);
195 static void PID_X_Exec(void);
196 static void TimerCallback(void const *n);
197 extern WM_HWIN CreateFramewin(void);
198
199 osStatus osDelayus (uint32_t millisec);
200 extern WM_HWIN CreateWindow(void);
201
202
203
204 //static void CopyBuffer(int16_t *pbuffer1, int16_t *pbuffer2, uint16_t BufferSize);
205 void eq_coef_calc(float* pCoeffs,uint16_t f0, int8_t G, float Q, uint16_t fs);
206 void eq_coef_LowShelf(float* pCoeffs,uint16_t f0, int8_t G, uint16_t fs);
207 void eq_coef_HighShelf(float* pCoeffs,uint16_t f0, int8_t G, uint16_t fs);
208 static void Display_DemoDescription(void);
209 void Audio_Init_Test(void);
210 extern void audio_init_with_memory(void);
211 //static void wave_to_buffer (int16_t * playbuffer, uint32_t num_samples);
212 uint8_t CheckForUserInputTP(void);
213 void FilterCoffCalc (int8_t G, uint16_t F0, float Q);
214 //struct timeval GetTimeStamp();
215 uint8_t CheckForUserInput(void);
216
217 void BSP_AUDIO_OUT_HalfTransfer_CallBack(void);
218 void BSP_AUDIO_IN_TransferComplete_CallBack(void);
219 void BSP_AUDIO_IN_HalfTransfer_CallBack(void);
220
221 //////////////////////////////////LIBS FFT
222  DISPLAY////////////////////////////////////
223 extern void MainTask(void);
224 extern void FFT_Data2(void);
225
226 /* USER CODE END PFP */
227 //////////////////////////////////////
228 /* Private user code -----*/
229 /* USER CODE BEGIN 0 */
230 /* USER CODE END 0 */
231
232 /**
233  * @brief The application entry point.
234  * @retval int
235  */
236 int main(void)
237 {
238
239  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
240  HAL_Init();
241
242  /* Configure the system clock */
243  SystemClock_Config();
244
245  /* Initialize all configured peripherals */
246  MX_GPIO_Init();

```

main.c

```
247 MX_DMA_Init();
248 MX_ADC3_Init();
249 MX_CRC_Init();
250 MX_DCMI_Init();
251 MX_DMA2D_Init();
252 MX_ETH_Init();
253 MX_FMC_Init();
254 MX_I2C1_Init();
255 MX_I2C3_Init();
256 MX_LTDC_Init();
257 MX_QUADSPI_Init();
258 MX_RTC_Init();
259 MX_SDMMC1_SD_Init();
260 MX_SPDIFRX_Init();
261 MX_SPI2_Init();
262 MX_TIM1_Init();
263 MX_USART6_UART_Init();
264 MX_FATFS_Init();
265 MX_SAI2_Init();
266
267 /* USER CODE BEGIN 2 */
268
269 Display_DemoDescription();
270 HAL_Delay (4000);
271 BSP_TS_Init(480, 272);
272 BSP_LED_Init(led1);
273 // k_BspInit();
274
275 if (arm_rfft_init_q15(&fft_instance, 256, 0, 1) != ARM_MATH_SUCCESS) {
276     //error
277 }
278 // Initialize audio
279 audio_init_with_memory();
280     pStateIIR0[0] = 0;
281     pStateIIR0[1] = 0;
282     GUI_Init();
283
284 // Init GUI
285     CreateWindow();
286     flag.flagFilt = 0;
287     flag.band = 0;
288
289     WM_MULTIBUF_Enable(1);
290
291 /* USER CODE END 2 */
292
293
294     /* Create GUI task */
295
296 /* USER CODE BEGIN RTOS_THREADS */
297 osThreadDef(GUI_Thread, GUIThread, osPriorityRealtime, 0, 512);
298     /* Start thread GUI */
299     osThreadCreate (osThread(GUI_Thread), NULL);
300 osThreadDef(osAudio_Thread, AudioThread,osPriorityRealtime, 0,512);
301     /* Start thread AUDIO */
302     osThreadCreate (osThread(osAudio_Thread), NULL);
303
304
305 /* add threads, ... */
306 /* USER CODE END RTOS_THREADS */
307
308 /* Start scheduler */
```

main.c

```

309 osKernelStart();
310
311 /* USER CODE BEGIN WHILE */
312 while (1)
313 {
314     /* USER CODE END WHILE */
315
316     /* USER CODE BEGIN 3 */
317
318
319 }
320 /* USER CODE END 3 */
321 }
322
323 /**
324  * @brief System Clock Configuration
325  * @retval None
326  */
327
328 //////////////////////////////////////////////////GUI//////////////////////////////////////
329 static void GUIThread(void const * argument)
330 {
331
332     for(;;)
333     {
334         FFT_Data2();
335         PID_X_Exec();
336         osDelay (110);
337         GUI_Exec();
338     }
339 }
340
341 //////////////////////////////////////////////////fim GUI ////////////////////////////////////
342
343 //////////////////////////////////////////////////AUDIO THREAD//////////////////////////////////////
344 static void AudioThread(void const * argument)
345 {
346
347     BSP_LED_Off(led1);
348     osDelay(100);
349
350 flag.bandLS = 1;
351 FilterCoffCalc(0,225,1);
352 flag.bandPK1 =1;
353 FilterCoffCalc(0,1450,1);
354 flag.bandPK2 =1;
355 FilterCoffCalc(0,3500,1);
356 flag.bandPK3 =1;
357 FilterCoffCalc(0,8800,1);
358 flag.bandHS = 1;
359 FilterCoffCalc(0,13000,1);
360
361     for(;;){
362
363 //Processamento dos Filtros//////////////////////////////////////
364
365         while (audio_rec_buffer_state != BUFFER_OFFSET_HALF)
366         {
367
368
369             if (flag.band == 1)
370
371                 {

```

main.c

```

371
372             FilterCoffCalc(gain, frq, Fact_Q);
373             flag.band = 0;
374
375         }
376     }
377
378     audio_rec_buffer_state = BUFFER_OFFSET_NONE;
379
380     /* Copy recorded 1st half block */
381     if (flag.flagFilt== 0){
382         // No DSP
383         memcpy((uint16_t *) (AUDIO_BUFFER_OUT), (uint16_t *)
384 (AUDIO_BUFFER_IN), AUDIO_BLOCK_SIZE);
385     }
386
387     if (flag.flagFilt==1){
388         // DSP IIR
389
390
391         arm_q15_to_float (AUDIO_BUFFER_IN, float_buffer_in,
392 SIGNAL_SAMPLES);
393         arm_biquad_cascade_df2T_f32(&S0,
394 float_buffer_in, AUDIO_buffer_aux, SIGNAL_SAMPLES);
395         arm_biquad_cascade_df2T_f32(&S1, AUDIO_buffer_aux,
396 float_buffer_out, SIGNAL_SAMPLES);
397         arm_biquad_cascade_df2T_f32(&S2, float_buffer_out,
398 AUDIO_buffer_aux, SIGNAL_SAMPLES);
399         arm_biquad_cascade_df2T_f32(&S3, AUDIO_buffer_aux,
400 float_buffer_out, SIGNAL_SAMPLES);
401         arm_biquad_cascade_df2T_f32(&S4, float_buffer_out,
402 AUDIO_buffer_aux, SIGNAL_SAMPLES);
403         arm_float_to_q15 (AUDIO_buffer_aux, AUDIO_BUFFER_OUT ,
404 SIGNAL_SAMPLES);
405     }
406     //////////////////////////////////OUT //////////////////////////////////
407     memcpy((uint16_t *) dmaBuffer, (uint16_t *) AUDIO_BUFFER_OUT, SIGNAL_SAMPLES);
408     //////////////////////////////////
409     /* Wait end of one block recording */
410     while(audio_rec_buffer_state != BUFFER_OFFSET_FULL)
411     {
412
413         if (flag.band == 1)
414         {
415             FilterCoffCalc(gain, frq, Fact_Q);
416             flag.band = 0;
417         }
418     }
419
420     audio_rec_buffer_state = BUFFER_OFFSET_NONE;
421
422     if (flag.flagFilt==0)
423     {
424         memcpy((uint16_t *) ((AUDIO_BUFFER_OUT) + (AUDIO_BLOCK_SIZE)), (uint16_t *)

```

main.c

```

((AUDIO_BUFFER_IN) + (AUDIO_BLOCK_SIZE)),AUDIO_BLOCK_SIZE);
425     }
426
427     if (flag.flagFilt==1){
428
429
430         arm_q15_to_float((AUDIO_BUFFER_IN + (AUDIO_BLOCK_SIZE)),
float_buffer_in, SIGNAL_SAMPLES);
431         arm_biquad_cascade_df2T_f32(&S0, float_buffer_in,
AUDIO_buffer_aux, SIGNAL_SAMPLES);
432
433         arm_biquad_cascade_df2T_f32(&S1, AUDIO_buffer_aux,
float_buffer_out, SIGNAL_SAMPLES);
434
435         arm_biquad_cascade_df2T_f32(&S2, float_buffer_out,
AUDIO_buffer_aux, SIGNAL_SAMPLES);
436
437         arm_biquad_cascade_df2T_f32(&S3, AUDIO_buffer_aux,
float_buffer_out, SIGNAL_SAMPLES);
438
439         arm_biquad_cascade_df2T_f32(&S4,float_buffer_out,
AUDIO_buffer_aux, SIGNAL_SAMPLES);
440
441         arm_float_to_q15(AUDIO_buffer_aux, AUDIO_BUFFER_OUT +
(AUDIO_BLOCK_SIZE), SIGNAL_SAMPLES);
442
443
444         memcpy((uint16_t *)dmaBuffer,(uint16_t*)(AUDIO_BUFFER_OUT +
(AUDIO_BLOCK_SIZE)),SIGNAL_SAMPLES);
445
446     }
447
448 }
449
450
451 }
452
453
454 ///////////////////////////////////////////////////FIM AUDIO THREAD ///////////////////////////////////
455
456 ///////////////////////////////////////////////////ATIVAR TS ///////////////////////////////////
457 static void PID_X_Exec(void) {
458     TS_StateTypeDef TS_State;
459     static GUI_PID_STATE StatePID;
460     static int IsTouched;
461
462     BSP_TS_GetState(&TS_State);
463     StatePID.Layer = 0;
464     if (TS_State.touchDetected) {
465         IsTouched = 1;
466         StatePID.Pressed = 1;
467         StatePID.x = (int)(TS_State.touchX[0]);
468         StatePID.y = (int)(TS_State.touchY[0]);
469         GUI_PID_StoreState(&StatePID);
470     } else {
471         if (IsTouched == 1) {
472             IsTouched = 0;
473             StatePID.Pressed = 0;
474             GUI_PID_StoreState(&StatePID);
475         }
476     }
477 }

```

main.c

```
478
479 ///////////////////////////////////////////////////FILTRO IIR TESTE //////////////////////////////////////
480
481 void FilterCoffCalc (int8_t G, uint16_t F0, float Q)
482 {
483     // F0 = 725;
484
485     uint16_t Fs = 48000;
486
487     //Q= 20;
488     // G= 12;
489
490     if(flag.bandLS ==1)
491     {
492         eq_coef_LowShelf( coeffLS, F0, G, Fs);
493         coeffLS[3] = -coeffLS[3];
494         coeffLS[4] = -coeffLS[4];
495
496         arm_biquad_cascade_df2T_init_f32(&S0, numStages_IIR,coeffLS, pStateIIR0);
497
498         flag.bandLS = 0;
499     }
500 }
501
502 if(flag.bandPK1 ==1)
503 {
504     // float Q = Fact_Q ;
505     eq_coef_calc(coeffPK1,F0,G, Q, Fs);
506     coeffPK1[3] = -coeffPK1[3];
507     coeffPK1[4] = -coeffPK1[4];
508
509     arm_biquad_cascade_df2T_init_f32(&S1, numStages_IIR,coeffPK1, pStateIIR1);
510
511     flag.bandPK1 =0;
512 }
513
514 if(flag.bandPK2 ==1)
515 {
516     eq_coef_calc(coeffPK2,F0,G, Q, Fs);
517     coeffPK2[3] = -coeffPK2[3];
518     coeffPK2[4] = -coeffPK2[4];
519
520     arm_biquad_cascade_df2T_init_f32(&S2, numStages_IIR,coeffPK2, pStateIIR2);
521
522     flag.bandPK2 =0;
523 }
524
525 if(flag.bandPK3 ==1)
526 {
527
528     eq_coef_calc(coeffPK3,F0,G, Q, Fs);
529     coeffPK3[3] = -coeffPK3[3];
530     coeffPK3[4] = -coeffPK3[4];
531
532     arm_biquad_cascade_df2T_init_f32(&S3, numStages_IIR,coeffPK3, pStateIIR3);
533
534     flag.bandPK3 = 0;
535 }
536
537 if(flag.bandHS ==1)
538 {
539
```

main.c

```

eq_coef_HighShelf( coeffHS, F0, G, Fs);
coeffHS[3] = -coeffHS[3];
coeffHS[4] = -coeffHS[4];

arm_biquad_cascade_df2T_init_f32(&S4, numStages_IIR,coeffHS, pStateIIR4);

flag.bandHS = 0;
}

uint8_t CheckForUserInput(void)
{
    BSP_TS_GetState(&state);
    if(state.touchDetected)
    {
        return 1;
    }
    return 0;
}

//////////////////////////////////////
void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitStruct RCC_ClkInitStruct = {0};
    RCC_PeriphCLKInitTypeDef PeriphClkInitStruct = {0};

    /** Configure LSE Drive Capability */
    HAL_PWR_EnableBkUpAccess();
    /** Configure the main internal regulator output voltage */
    __HAL_RCC_PWR_CLK_ENABLE();
    __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
    /** Initializes the RCC Oscillators according to the specified parameters
     *  in the RCC_OscInitTypeDef structure. */
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_LSI|RCC_OSCILLATORTYPE_HSE;
    RCC_OscInitStruct.HSEState = RCC_HSE_ON;
    RCC_OscInitStruct.LSIState = RCC_LSI_ON;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
    RCC_OscInitStruct.PLL.PLLM = 25;
    RCC_OscInitStruct.PLL.PLLN = 400;
    RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
    RCC_OscInitStruct.PLL.PLLQ = 9;
    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
    {
        Error_Handler();
    }
    /** Activate the Over-Drive mode */
    if (HAL_PWREx_EnableOverDrive() != HAL_OK)
    {
        Error_Handler();
    }
    /** Initializes the CPU, AHB and APB buses clocks */
    RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSTICK

```

main.c

```

602 |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
603 RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
604 RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
605 RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
606 RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
607
608 if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_6) != HAL_OK)
609 {
610     Error_Handler();
611 }
612 PeriphClkInitStruct.PeriphClockSelection = RCC_PERIPHCLK_SPDIFRX|RCC_PERIPHCLK_LTDC
613 |RCC_PERIPHCLK_RTC|RCC_PERIPHCLK_USART1
614 |RCC_PERIPHCLK_USART6|RCC_PERIPHCLK_SAI2
615 |RCC_PERIPHCLK_I2C1|RCC_PERIPHCLK_I2C3
616 |RCC_PERIPHCLK_SDMMC1|RCC_PERIPHCLK_CLK48;
617 PeriphClkInitStruct.PLLI2S.PLLI2SN = 100;
618 PeriphClkInitStruct.PLLI2S.PLLI2SP = RCC_PLLP_DIV2;
619 PeriphClkInitStruct.PLLI2S.PLLI2SR = 2;
620 PeriphClkInitStruct.PLLI2S.PLLI2SQ = 2;
621 PeriphClkInitStruct.PLLSAI.PLLSAIN = 384;
622 PeriphClkInitStruct.PLLSAI.PLLSAIR = 5;
623 PeriphClkInitStruct.PLLSAI.PLLSAIQ = 2;
624 PeriphClkInitStruct.PLLSAI.PLLSAIP = RCC_PLLSAIP_DIV8;
625 PeriphClkInitStruct.PLLI2SDivQ = 1;
626 PeriphClkInitStruct.PLLSAIDivQ = 1;
627 PeriphClkInitStruct.PLLSAIDivR = RCC_PLLSAIDIVR_8;
628 PeriphClkInitStruct.RTCClockSelection = RCC_RTCCCLKSOURCE_LSI;
629 PeriphClkInitStruct.Sai2ClockSelection = RCC_SAI2CLKSOURCE_PLLSAI;
630 PeriphClkInitStruct.Usart1ClockSelection = RCC_USART1CLKSOURCE_PCLK2;
631 PeriphClkInitStruct.Usart6ClockSelection = RCC_USART6CLKSOURCE_PCLK2;
632 PeriphClkInitStruct.I2c1ClockSelection = RCC_I2C1CLKSOURCE_PCLK1;
633 PeriphClkInitStruct.I2c3ClockSelection = RCC_I2C3CLKSOURCE_PCLK1;
634 PeriphClkInitStruct.Clk48ClockSelection = RCC_CLK48SOURCE_PLLSAIP;
635 PeriphClkInitStruct.Sdmmc1ClockSelection = RCC_SDMMC1CLKSOURCE_CLK48;
636 if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInitStruct) != HAL_OK)
637 {
638     Error_Handler();
639 }
640 }
641
642 /**
643  * @brief ADC3 Initialization Function
644  * @param None
645  * @retval None
646  */
647 static void MX_ADC3_Init(void)
648 {
649
650     /* USER CODE BEGIN ADC3_Init 0 */
651
652     /* USER CODE END ADC3_Init 0 */
653
654     ADC_ChannelConfTypeDef sConfig = {0};
655
656     /* USER CODE BEGIN ADC3_Init 1 */
657
658     /* USER CODE END ADC3_Init 1 */
659     /** Configure the global features of the ADC (Clock, Resolution, Data Alignment and
660     number of conversion)
661     */
661     hadc3.Instance = ADC3;
662     hadc3.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;

```

main.c

```

663  hadc3.Init.Resolution = ADC_RESOLUTION_12B;
664  hadc3.Init.ScanConvMode = ADC_SCAN_DISABLE;
665  hadc3.Init.ContinuousConvMode = DISABLE;
666  hadc3.Init.DiscontinuousConvMode = DISABLE;
667  hadc3.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
668  hadc3.Init.ExternalTrigConv = ADC_SOFTWARE_START;
669  hadc3.Init.DataAlign = ADC_DATAALIGN_RIGHT;
670  hadc3.Init.NbrOfConversion = 1;
671  hadc3.Init.DMAContinuousRequests = DISABLE;
672  hadc3.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
673  if (HAL_ADC_Init(&hadc3) != HAL_OK)
674  {
675      Error_Handler();
676  }
677
678
679  /* Compute the prescaler value to have TIM3 counter clock equal to 10 KHz */
680  uwPrescalerValue = (uint32_t) ((SystemCoreClock / 2) / 10000) - 1;
681
682  /* Set TIMx instance */
683  TimHandle.Instance = TIM3;
684
685  /* Initialize TIM3 peripheral as follows:
686      + Period = 500 - 1
687      + Prescaler = ((SystemCoreClock/2)/10000) - 1
688      + ClockDivision = 0
689      + Counter direction = Up
690  */
691  TimHandle.Init.Period = 500 - 1;
692  TimHandle.Init.Prescaler = uwPrescalerValue;
693  TimHandle.Init.ClockDivision = 0;
694  TimHandle.Init.CounterMode = TIM_COUNTERMODE_UP;
695  if (HAL_TIM_Base_Init(&TimHandle) != HAL_OK)
696  {
697      while(1)
698      {
699      }
700  }
701  /** Configure for the selected ADC regular channel its corresponding rank in the
702  sequencer and its sample time.
703  */
704  sConfig.Channel = ADC_CHANNEL_4;
705  sConfig.Rank = ADC_REGULAR_RANK_1;
706  sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
707  if (HAL_ADC_ConfigChannel(&hadc3, &sConfig) != HAL_OK)
708  {
709      Error_Handler();
710  }
711  /* USER CODE BEGIN ADC3_Init 2 */
712  /* USER CODE END ADC3_Init 2 */
713
714 }
715
716 /**
717  * @brief CRC Initialization Function
718  * @param None
719  * @retval None
720  */
721 static void MX_CRC_Init(void)
722 {
723

```

main.c

```

724  /* USER CODE BEGIN CRC_Init 0 */
725
726  /* USER CODE END CRC_Init 0 */
727
728  /* USER CODE BEGIN CRC_Init 1 */
729
730  /* USER CODE END CRC_Init 1 */
731  hcrc.Instance = CRC;
732  hcrc.Init.DefaultPolynomialUse = DEFAULT_POLYNOMIAL_ENABLE;
733  hcrc.Init.DefaultInitValueUse = DEFAULT_INIT_VALUE_ENABLE;
734  hcrc.Init.InputDataInversionMode = CRC_INPUTDATA_INVERSION_NONE;
735  hcrc.Init.OutputDataInversionMode = CRC_OUTPUTDATA_INVERSION_DISABLE;
736  hcrc.InputDataFormat = CRC_INPUTDATA_FORMAT_BYTES;
737  if (HAL_CRC_Init(&hcrc) != HAL_OK)
738  {
739      Error_Handler();
740  }
741  /* USER CODE BEGIN CRC_Init 2 */
742
743  /* USER CODE END CRC_Init 2 */
744
745 }
746
747 /**
748  * @brief DCMI Initialization Function
749  * @param None
750  * @retval None
751  */
752 static void MX_DCMI_Init(void)
753 {
754
755  /* USER CODE BEGIN DCMI_Init 0 */
756
757  /* USER CODE END DCMI_Init 0 */
758
759  /* USER CODE BEGIN DCMI_Init 1 */
760
761  /* USER CODE END DCMI_Init 1 */
762  hdcmi.Instance = DCMI;
763  hdcmi.Init.SynchroMode = DCMI_SYNCHRO_HARDWARE;
764  hdcmi.Init.PCKPolarity = DCMI_PCKPOLARITY_FALLING;
765  hdcmi.Init.VSPolarity = DCMI_VSPOLARITY_LOW;
766  hdcmi.Init.HSPolarity = DCMI_HSPOLARITY_LOW;
767  hdcmi.Init.CaptureRate = DCMI_CR_ALL_FRAME;
768  hdcmi.Init.ExtendedDataMode = DCMI_EXTEND_DATA_8B;
769  hdcmi.Init.JPEGMode = DCMI_JPEG_DISABLE;
770  hdcmi.Init.ByteSelectMode = DCMI_BSM_ALL;
771  hdcmi.Init.ByteSelectStart = DCMI_OEBS_ODD;
772  hdcmi.Init.LineSelectMode = DCMI_LSM_ALL;
773  hdcmi.Init.LineSelectStart = DCMI_OELS_ODD;
774  if (HAL_DCMI_Init(&hdcmi) != HAL_OK)
775  {
776      Error_Handler();
777  }
778  /* USER CODE BEGIN DCMI_Init 2 */
779
780  /* USER CODE END DCMI_Init 2 */
781
782 }
783
784 /**
785  * @brief DMA2D Initialization Function

```

main.c

```
786  * @param None
787  * @retval None
788  */
789 static void MX_DMA2D_Init(void)
790 {
791
792  /* USER CODE BEGIN DMA2D_Init 0 */
793
794  /* USER CODE END DMA2D_Init 0 */
795
796  /* USER CODE BEGIN DMA2D_Init 1 */
797
798  /* USER CODE END DMA2D_Init 1 */
799  hdma2d.Instance = DMA2D;
800  hdma2d.Init.Mode = DMA2D_M2M;
801  hdma2d.Init.ColorMode = DMA2D_OUTPUT_ARGB8888;
802  hdma2d.Init.OutputOffset = 0;
803  hdma2d.LayerCfg[1].InputOffset = 0;
804  hdma2d.LayerCfg[1].InputColorMode = DMA2D_INPUT_ARGB8888;
805  hdma2d.LayerCfg[1].AlphaMode = DMA2D_NO_MODIF_ALPHA;
806  hdma2d.LayerCfg[1].InputAlpha = 0;
807  if (HAL_DMA2D_Init(&hdma2d) != HAL_OK)
808  {
809      Error_Handler();
810  }
811  if (HAL_DMA2D_ConfigLayer(&hdma2d, 1) != HAL_OK)
812  {
813      Error_Handler();
814  }
815  /* USER CODE BEGIN DMA2D_Init 2 */
816
817  /* USER CODE END DMA2D_Init 2 */
818
819 }
820
821 /**
822  * @brief ETH Initialization Function
823  * @param None
824  * @retval None
825  */
826 static void MX_ETH_Init(void)
827 {
828
829  /* USER CODE BEGIN ETH_Init 0 */
830
831  /* USER CODE END ETH_Init 0 */
832
833  /* USER CODE BEGIN ETH_Init 1 */
834
835  /* USER CODE END ETH_Init 1 */
836  heth.Instance = ETH;
837  heth.Init.AutoNegotiation = ETH_AUTONEGOTIATION_ENABLE;
838  heth.Init.Speed = ETH_SPEED_100M;
839  heth.Init.DuplexMode = ETH_MODE_FULLDUPLEX;
840  heth.Init.PhyAddress = LAN8742A_PHY_ADDRESS;
841  heth.Init.MACAddr[0] = 0x00;
842  heth.Init.MACAddr[1] = 0x80;
843  heth.Init.MACAddr[2] = 0xE1;
844  heth.Init.MACAddr[3] = 0x00;
845  heth.Init.MACAddr[4] = 0x00;
846  heth.Init.MACAddr[5] = 0x00;
847  heth.Init.RxMode = ETH_RXPOLLING_MODE;
```

main.c

```
848 heth.Init.ChecksumMode = ETH_CHECKSUM_BY_HARDWARE;
849 heth.Init.MediaInterface = ETH_MEDIA_INTERFACE_RMII;
850
851 /* USER CODE BEGIN MACADDRESS */
852
853 /* USER CODE END MACADDRESS */
854
855 if (HAL_ETH_Init(&heth) != HAL_OK)
856 {
857     Error_Handler();
858 }
859 /* USER CODE BEGIN ETH_Init 2 */
860
861 /* USER CODE END ETH_Init 2 */
862
863 }
864
865 /**
866  * @brief I2C1 Initialization Function
867  * @param None
868  * @retval None
869  */
870 static void MX_I2C1_Init(void)
871 {
872
873     /* USER CODE BEGIN I2C1_Init 0 */
874
875     /* USER CODE END I2C1_Init 0 */
876
877     /* USER CODE BEGIN I2C1_Init 1 */
878
879     /* USER CODE END I2C1_Init 1 */
880     hi2c1.Instance = I2C1;
881     hi2c1.Init.Timing = 0x00C0EAF;
882     hi2c1.Init.OwnAddress1 = 0;
883     hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
884     hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
885     hi2c1.Init.OwnAddress2 = 0;
886     hi2c1.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
887     hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
888     hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
889     if (HAL_I2C_Init(&hi2c1) != HAL_OK)
890     {
891         Error_Handler();
892     }
893     /** Configure Analogue filter
894     */
895     if (HAL_I2CEx_ConfigAnalogFilter(&hi2c1, I2C_ANALOGFILTER_ENABLE) != HAL_OK)
896     {
897         Error_Handler();
898     }
899     /** Configure Digital filter
900     */
901     if (HAL_I2CEx_ConfigDigitalFilter(&hi2c1, 0) != HAL_OK)
902     {
903         Error_Handler();
904     }
905     /* USER CODE BEGIN I2C1_Init 2 */
906
907     /* USER CODE END I2C1_Init 2 */
908
909 }
```

main.c

```
910
911 /**
912  * @brief I2C3 Initialization Function
913  * @param None
914  * @retval None
915  */
916 static void MX_I2C3_Init(void)
917 {
918
919     /* USER CODE BEGIN I2C3_Init 0 */
920
921     /* USER CODE END I2C3_Init 0 */
922
923     /* USER CODE BEGIN I2C3_Init 1 */
924
925     /* USER CODE END I2C3_Init 1 */
926     hi2c3.Instance = I2C3;
927     hi2c3.Init.Timing = 0x00C0EAFF;
928     hi2c3.Init.OwnAddress1 = 0;
929     hi2c3.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
930     hi2c3.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
931     hi2c3.Init.OwnAddress2 = 0;
932     hi2c3.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
933     hi2c3.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
934     hi2c3.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
935     if (HAL_I2C_Init(&hi2c3) != HAL_OK)
936     {
937         Error_Handler();
938     }
939     /** Configure Analogue filter
940     */
941     if (HAL_I2CEx_ConfigAnalogFilter(&hi2c3, I2C_ANALOGFILTER_ENABLE) != HAL_OK)
942     {
943         Error_Handler();
944     }
945     /** Configure Digital filter
946     */
947     if (HAL_I2CEx_ConfigDigitalFilter(&hi2c3, 0) != HAL_OK)
948     {
949         Error_Handler();
950     }
951     /* USER CODE BEGIN I2C3_Init 2 */
952
953     /* USER CODE END I2C3_Init 2 */
954
955 }
956
957 /**
958  * @brief LTDC Initialization Function
959  * @param None
960  * @retval None
961  */
962 static void MX_LTDC_Init(void)
963 {
964
965     /* USER CODE BEGIN LTDC_Init 0 */
966
967     /* USER CODE END LTDC_Init 0 */
968
969     LTDC_LayerCfgTypeDef pLayerCfg = {0};
970
971     /* USER CODE BEGIN LTDC_Init 1 */
```

main.c

```

972
973 /* USER CODE END LTDC_Init 1 */
974 hltadc.Instance = LTDC;
975 hltadc.Init.HSPolarity = LTDC_HSPOLARITY_AL;
976 hltadc.Init.VSPolarity = LTDC_VSPOLARITY_AL;
977 hltadc.Init.DEPolarity = LTDC_DEPOLARITY_AL;
978 hltadc.Init.PCPolarity = LTDC_PCPOLARITY_IPC;
979 hltadc.Init.HorizontalSync = 40;
980 hltadc.Init.VerticalSync = 9;
981 hltadc.Init.AccumulatedHBP = 53;
982 hltadc.Init.AccumulatedVBP = 11;
983 hltadc.Init.AccumulatedActiveW = 533;
984 hltadc.Init.AccumulatedActiveH = 283;
985 hltadc.Init.TotalWidth = 565;
986 hltadc.Init.TotalHeigh = 285;
987 hltadc.Init.Backcolor.Blue = 0;
988 hltadc.Init.Backcolor.Green = 0;
989 hltadc.Init.Backcolor.Red = 0;
990 if (HAL_LTDC_Init(&hltadc) != HAL_OK)
991 {
992     Error_Handler();
993 }
994 pLayerCfg.WindowX0 = 0;
995 pLayerCfg.WindowX1 = 480;
996 pLayerCfg.WindowY0 = 0;
997 pLayerCfg.WindowY1 = 272;
998 pLayerCfg.PixelFormat = LTDC_PIXEL_FORMAT_RGB565;
999 pLayerCfg.Alpha = 255;
1000 pLayerCfg.Alpha0 = 0;
1001 pLayerCfg.BlendingFactor1 = LTDC_BLENDING_FACTOR1_PAxCA;
1002 pLayerCfg.BlendingFactor2 = LTDC_BLENDING_FACTOR2_PAxCA;
1003 pLayerCfg.FBStartAddress = 0xC0000000;
1004 pLayerCfg.ImageWidth = 480;
1005 pLayerCfg.ImageHeight = 272;
1006 pLayerCfg.Backcolor.Blue = 0;
1007 pLayerCfg.Backcolor.Green = 0;
1008 pLayerCfg.Backcolor.Red = 0;
1009 if (HAL_LTDC_ConfigLayer(&hltadc, &pLayerCfg, 0) != HAL_OK)
1010 {
1011     Error_Handler();
1012 }
1013 /* USER CODE BEGIN LTDC_Init 2 */
1014
1015 /* USER CODE END LTDC_Init 2 */
1016
1017 }
1018
1019 /**
1020  * @brief QUADSPI Initialization Function
1021  * @param None
1022  * @retval None
1023  */
1024 static void MX_QUADSPI_Init(void)
1025 {
1026
1027     /* USER CODE BEGIN QUADSPI_Init 0 */
1028
1029     /* USER CODE END QUADSPI_Init 0 */
1030
1031     /* USER CODE BEGIN QUADSPI_Init 1 */
1032
1033     /* USER CODE END QUADSPI_Init 1 */

```

main.c

```
1034 /* QUADSPI parameter configuration*/
1035 hqspi.Instance = QUADSPI;
1036 hqspi.Init.ClockPrescaler = 255;
1037 hqspi.Init.FifoThreshold = 1;
1038 hqspi.Init.SampleShifting = QSPI_SAMPLE_SHIFTING_NONE;
1039 hqspi.Init.FlashSize = 1;
1040 hqspi.Init.ChipSelectHighTime = QSPI_CS_HIGH_TIME_1_CYCLE;
1041 hqspi.Init.ClockMode = QSPI_CLOCK_MODE_0;
1042 hqspi.Init.FlashID = QSPI_FLASH_ID_1;
1043 hqspi.Init.DualFlash = QSPI_DUALFLASH_DISABLE;
1044 if (HAL_QSPI_Init(&hqspi) != HAL_OK)
1045 {
1046     Error_Handler();
1047 }
1048 /* USER CODE BEGIN QUADSPI_Init 2 */
1049
1050 /* USER CODE END QUADSPI_Init 2 */
1051
1052 }
1053
1054 /**
1055  * @brief RTC Initialization Function
1056  * @param None
1057  * @retval None
1058  */
1059 static void MX_RTC_Init(void)
1060 {
1061
1062     /* USER CODE BEGIN RTC_Init 0 */
1063
1064     /* USER CODE END RTC_Init 0 */
1065
1066     RTC_TimeTypeDef sTime = {0};
1067     RTC_DateTypeDef sDate = {0};
1068     RTC_AlarmTypeDef sAlarm = {0};
1069
1070     /* USER CODE BEGIN RTC_Init 1 */
1071
1072     /* USER CODE END RTC_Init 1 */
1073     /** Initialize RTC Only
1074     */
1075     hrtc.Instance = RTC;
1076     hrtc.Init.HourFormat = RTC_HOURFORMAT_24;
1077     hrtc.Init.AsynchPrediv = 127;
1078     hrtc.Init.SynchPrediv = 255;
1079     hrtc.Init.OutPut = RTC_OUTPUT_DISABLE;
1080     hrtc.Init.OutPutPolarity = RTC_OUTPUT_POLARITY_HIGH;
1081     hrtc.Init.OutPutType = RTC_OUTPUT_TYPE_OPENDRAIN;
1082     if (HAL_RTC_Init(&hrtc) != HAL_OK)
1083     {
1084         Error_Handler();
1085     }
1086
1087     /* USER CODE BEGIN Check_RTC_BKUP */
1088
1089     /* USER CODE END Check_RTC_BKUP */
1090
1091     /** Initialize RTC and set the Time and Date
1092     */
1093     sTime.Hours = 0x0;
1094     sTime.Minutes = 0x0;
1095     sTime.Seconds = 0x0;
```

main.c

```

1096 sTime.DayLightSaving = RTC_DAYLIGHTSAVING_NONE;
1097 sTime.StoreOperation = RTC_STOREOPERATION_RESET;
1098 if (HAL_RTC_SetTime(&hrtc, &sTime, RTC_FORMAT_BCD) != HAL_OK)
1099 {
1100     Error_Handler();
1101 }
1102 sDate.WeekDay = RTC_WEEKDAY_MONDAY;
1103 sDate.Month = RTC_MONTH_JANUARY;
1104 sDate.Date = 0x1;
1105 sDate.Year = 0x0;
1106 if (HAL_RTC_SetDate(&hrtc, &sDate, RTC_FORMAT_BCD) != HAL_OK)
1107 {
1108     Error_Handler();
1109 }
1110 /** Enable the Alarm A
1111 */
1112 sAlarm.AlarmTime.Hours = 0x0;
1113 sAlarm.AlarmTime.Minutes = 0x0;
1114 sAlarm.AlarmTime.Seconds = 0x0;
1115 sAlarm.AlarmTime.SubSeconds = 0x0;
1116 sAlarm.AlarmTime.DayLightSaving = RTC_DAYLIGHTSAVING_NONE;
1117 sAlarm.AlarmTime.StoreOperation = RTC_STOREOPERATION_RESET;
1118 sAlarm.AlarmMask = RTC_ALARMMASK_NONE;
1119 sAlarm.AlarmSubSecondMask = RTC_ALARMSUBSECONDMASK_ALL;
1120 sAlarm.AlarmDateWeekDaySel = RTC_ALARMDATEWEEKDAYSEL_DATE;
1121 sAlarm.AlarmDateWeekDay = 0x1;
1122 sAlarm.Alarm = RTC_ALARM_A;
1123 if (HAL_RTC_SetAlarm(&hrtc, &sAlarm, RTC_FORMAT_BCD) != HAL_OK)
1124 {
1125     Error_Handler();
1126 }
1127 /** Enable the Alarm B
1128 */
1129 sAlarm.Alarm = RTC_ALARM_B;
1130 if (HAL_RTC_SetAlarm(&hrtc, &sAlarm, RTC_FORMAT_BCD) != HAL_OK)
1131 {
1132     Error_Handler();
1133 }
1134 /** Enable the TimeStamp
1135 */
1136 if (HAL_RTCEx_SetTimeStamp(&hrtc, RTC_TIMESTAMPEDGE_RISING, RTC_TIMESTAMP_PIN_POS1) !=
HAL_OK)
1137 {
1138     Error_Handler();
1139 }
1140 /* USER CODE BEGIN RTC_Init 2 */
1141
1142 /* USER CODE END RTC_Init 2 */
1143
1144 }
1145
1146 /**
1147  * @brief SAI2 Initialization Function
1148  * @param None
1149  * @retval None
1150  */
1151 static void MX_SAI2_Init(void)
1152 {
1153
1154     /* USER CODE BEGIN SAI2_Init 0 */
1155
1156     /* USER CODE END SAI2_Init 0 */

```

main.c

```
1157
1158 /* USER CODE BEGIN SAI2_Init 1 */
1159
1160 /* USER CODE END SAI2_Init 1 */
1161 hsai_BlockA2.Instance = SAI2_Block_A;
1162 hsai_BlockA2.Init.Protocol = SAI_FREE_PROTOCOL;
1163 hsai_BlockA2.Init.AudioMode = SAI_MODEMASTER_TX;
1164 hsai_BlockA2.Init.DataSize = SAI_DATASIZE_16;
1165 hsai_BlockA2.Init.FirstBit = SAI_FIRSTBIT_MSB;
1166 hsai_BlockA2.Init.ClockStrobing = SAI_CLOCKSTROBING_FALLINGEDGE;
1167 hsai_BlockA2.Init.Synchro = SAI_ASYNCHRONOUS;
1168 hsai_BlockA2.Init.OutputDrive = SAI_OUTPUTDRIVE_DISABLE;
1169 hsai_BlockA2.Init.NoDivider = SAI_MASTERDIVIDER_ENABLE;
1170 hsai_BlockA2.Init.FIFOThreshold = SAI_FIFOTHRESHOLD_EMPTY;
1171 hsai_BlockA2.Init.AudioFrequency = SAI_AUDIO_FREQUENCY_48K;
1172 hsai_BlockA2.Init.SynchroExt = SAI_SYNCEXT_DISABLE;
1173 hsai_BlockA2.Init.MonoStereoMode = SAI_STEREOMODE;
1174 hsai_BlockA2.Init.CompandingMode = SAI_NOCOMPANDING;
1175 hsai_BlockA2.Init.TriState = SAI_OUTPUT_NOTRELEASED;
1176 hsai_BlockA2.FrameInit.FrameLength = 64;
1177 hsai_BlockA2.FrameInit.ActiveFrameLength = 1;
1178 hsai_BlockA2.FrameInit.FSDefinition = SAI_FS_STARTFRAME;
1179 hsai_BlockA2.FrameInit.FSPolarity = SAI_FS_ACTIVE_LOW;
1180 hsai_BlockA2.FrameInit.FSOffset = SAI_FS_FIRSTBIT;
1181 hsai_BlockA2.SlotInit.FirstBitOffset = 0;
1182 hsai_BlockA2.SlotInit.SlotSize = SAI_SLOTSIZE_DATASIZE;
1183 hsai_BlockA2.SlotInit.SlotNumber = 4;
1184 hsai_BlockA2.SlotInit.SlotActive = 0x0000FFFF;
1185 if (HAL_SAI_Init(&hsai_BlockA2) != HAL_OK)
1186 {
1187     Error_Handler();
1188 }
1189 hsai_BlockB2.Instance = SAI2_Block_B;
1190 hsai_BlockB2.Init.Protocol = SAI_FREE_PROTOCOL;
1191 hsai_BlockB2.Init.AudioMode = SAI_MODESLAVE_RX;
1192 hsai_BlockB2.Init.DataSize = SAI_DATASIZE_8;
1193 hsai_BlockB2.Init.FirstBit = SAI_FIRSTBIT_MSB;
1194 hsai_BlockB2.Init.ClockStrobing = SAI_CLOCKSTROBING_FALLINGEDGE;
1195 hsai_BlockB2.Init.Synchro = SAI_SYNCHRONOUS;
1196 hsai_BlockB2.Init.OutputDrive = SAI_OUTPUTDRIVE_DISABLE;
1197 hsai_BlockB2.Init.FIFOThreshold = SAI_FIFOTHRESHOLD_EMPTY;
1198 hsai_BlockB2.Init.SynchroExt = SAI_SYNCEXT_DISABLE;
1199 hsai_BlockB2.Init.MonoStereoMode = SAI_STEREOMODE;
1200 hsai_BlockB2.Init.CompandingMode = SAI_NOCOMPANDING;
1201 hsai_BlockB2.Init.TriState = SAI_OUTPUT_NOTRELEASED;
1202 hsai_BlockB2.FrameInit.FrameLength = 64;
1203 hsai_BlockB2.FrameInit.ActiveFrameLength = 1;
1204 hsai_BlockB2.FrameInit.FSDefinition = SAI_FS_STARTFRAME;
1205 hsai_BlockB2.FrameInit.FSPolarity = SAI_FS_ACTIVE_LOW;
1206 hsai_BlockB2.FrameInit.FSOffset = SAI_FS_FIRSTBIT;
1207 hsai_BlockB2.SlotInit.FirstBitOffset = 0;
1208 hsai_BlockB2.SlotInit.SlotSize = SAI_SLOTSIZE_DATASIZE;
1209 hsai_BlockB2.SlotInit.SlotNumber = 4;
1210 hsai_BlockB2.SlotInit.SlotActive = 0x0000FFFF;
1211 if (HAL_SAI_Init(&hsai_BlockB2) != HAL_OK)
1212 {
1213     Error_Handler();
1214 }
1215 /* USER CODE BEGIN SAI2_Init 2 */
1216
1217 /* USER CODE END SAI2_Init 2 */
1218
```

main.c

```
1219 }
1220
1221 /**
1222  * @brief SDMMC1 Initialization Function
1223  * @param None
1224  * @retval None
1225  */
1226 static void MX_SDMMC1_SD_Init(void)
1227 {
1228
1229     /* USER CODE BEGIN SDMMC1_Init 0 */
1230
1231     /* USER CODE END SDMMC1_Init 0 */
1232
1233     /* USER CODE BEGIN SDMMC1_Init 1 */
1234
1235     /* USER CODE END SDMMC1_Init 1 */
1236     hsd1.Instance = SDMMC1;
1237     hsd1.Init.ClockEdge = SDMMC_CLOCK_EDGE_RISING;
1238     hsd1.Init.ClockBypass = SDMMC_CLOCK_BYPASS_DISABLE;
1239     hsd1.Init.ClockPowerSave = SDMMC_CLOCK_POWER_SAVE_DISABLE;
1240     hsd1.Init.BusWide = SDMMC_BUS_WIDE_1B;
1241     hsd1.Init.HardwareFlowControl = SDMMC_HARDWARE_FLOW_CONTROL_DISABLE;
1242     hsd1.Init.ClockDiv = 0;
1243     /* USER CODE BEGIN SDMMC1_Init 2 */
1244
1245     /* USER CODE END SDMMC1_Init 2 */
1246
1247 }
1248
1249 /**
1250  * @brief SPDIFRX Initialization Function
1251  * @param None
1252  * @retval None
1253  */
1254 static void MX_SPDIFRX_Init(void)
1255 {
1256
1257     /* USER CODE BEGIN SPDIFRX_Init 0 */
1258
1259     /* USER CODE END SPDIFRX_Init 0 */
1260
1261     /* USER CODE BEGIN SPDIFRX_Init 1 */
1262
1263     /* USER CODE END SPDIFRX_Init 1 */
1264     hspdif.Instance = SPDIFRX;
1265     hspdif.Init.InputSelection = SPDIFRX_INPUT_IN0;
1266     hspdif.Init.Retries = SPDIFRX_MAXRETRIES_NONE;
1267     hspdif.Init.WaitForActivity = SPDIFRX_WAITFORACTIVITY_OFF;
1268     hspdif.Init.ChannelSelection = SPDIFRX_CHANNEL_A;
1269     hspdif.Init.DataFormat = SPDIFRX_DATAFORMAT_LSB;
1270     hspdif.Init.StereoMode = SPDIFRX_STEREO_MODE_DISABLE;
1271     hspdif.Init.PreambleTypeMask = SPDIFRX_PREAMBLETYPEMASK_OFF;
1272     hspdif.Init.ChannelStatusMask = SPDIFRX_CHANNELSTATUS_OFF;
1273     hspdif.Init.ValidityBitMask = SPDIFRX_VALIDITYMASK_OFF;
1274     hspdif.Init.ParityErrorMask = SPDIFRX_PARITYERRORMASK_OFF;
1275     if (HAL_SPDIFRX_Init(&hspdif) != HAL_OK)
1276     {
1277         Error_Handler();
1278     }
1279     /* USER CODE BEGIN SPDIFRX_Init 2 */
1280
```

main.c

```
1281  /* USER CODE END SPDIFRX_Init 2 */
1282
1283 }
1284
1285 /**
1286  * @brief SPI2 Initialization Function
1287  * @param None
1288  * @retval None
1289  */
1290 static void MX_SPI2_Init(void)
1291 {
1292
1293  /* USER CODE BEGIN SPI2_Init 0 */
1294
1295  /* USER CODE END SPI2_Init 0 */
1296
1297  /* USER CODE BEGIN SPI2_Init 1 */
1298
1299  /* USER CODE END SPI2_Init 1 */
1300  /* SPI2 parameter configuration*/
1301  hspi2.Instance = SPI2;
1302  hspi2.Init.Mode = SPI_MODE_MASTER;
1303  hspi2.Init.Direction = SPI_DIRECTION_2LINES;
1304  hspi2.Init.DataSize = SPI_DATASIZE_4BIT;
1305  hspi2.Init.CLKPolarity = SPI_POLARITY_LOW;
1306  hspi2.Init.CLKPhase = SPI_PHASE_1EDGE;
1307  hspi2.Init.NSS = SPI_NSS_SOFT;
1308  hspi2.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
1309  hspi2.Init.FirstBit = SPI_FIRSTBIT_MSB;
1310  hspi2.Init.TIMode = SPI_TIMODE_DISABLE;
1311  hspi2.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
1312  hspi2.Init.CRCPolynomial = 7;
1313  hspi2.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
1314  hspi2.Init.NSSPMode = SPI_NSS_PULSE_ENABLE;
1315  if (HAL_SPI_Init(&hspi2) != HAL_OK)
1316  {
1317    Error_Handler();
1318  }
1319  /* USER CODE BEGIN SPI2_Init 2 */
1320
1321  /* USER CODE END SPI2_Init 2 */
1322
1323 }
1324
1325 /**
1326  * @brief TIM1 Initialization Function
1327  * @param None
1328  * @retval None
1329  */
1330 static void MX_TIM1_Init(void)
1331 {
1332
1333  /* USER CODE BEGIN TIM1_Init 0 */
1334
1335  /* USER CODE END TIM1_Init 0 */
1336
1337  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
1338  TIM_MasterConfigTypeDef sMasterConfig = {0};
1339  TIM_OC_InitTypeDef sConfigOC = {0};
1340  TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0};
1341
1342  /* USER CODE BEGIN TIM1_Init 1 */
```

main.c

```
1343
1344 /* USER CODE END TIM1_Init 1 */
1345 htim1.Instance = TIM1;
1346 htim1.Init.Prescaler = 0;
1347 htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
1348 htim1.Init.Period = 65535;
1349 htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
1350 htim1.Init.RepetitionCounter = 0;
1351 htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
1352 if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
1353 {
1354     Error_Handler();
1355 }
1356 sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
1357 if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
1358 {
1359     Error_Handler();
1360 }
1361 if (HAL_TIM_PWM_Init(&htim1) != HAL_OK)
1362 {
1363     Error_Handler();
1364 }
1365 sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
1366 sMasterConfig.MasterOutputTrigger2 = TIM_TRGO2_RESET;
1367 sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
1368 if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
1369 {
1370     Error_Handler();
1371 }
1372 sConfigOC.OCMode = TIM_OCMODE_PWM1;
1373 sConfigOC.Pulse = 0;
1374 sConfigOC.OCpolarity = TIM_OCPOLARITY_HIGH;
1375 sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
1376 sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
1377 sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
1378 sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
1379 if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
1380 {
1381     Error_Handler();
1382 }
1383 sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
1384 sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
1385 sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;
1386 sBreakDeadTimeConfig.DeadTime = 0;
1387 sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
1388 sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
1389 sBreakDeadTimeConfig.BreakFilter = 0;
1390 sBreakDeadTimeConfig.Break2State = TIM_BREAK2_DISABLE;
1391 sBreakDeadTimeConfig.Break2Polarity = TIM_BREAK2POLARITY_HIGH;
1392 sBreakDeadTimeConfig.Break2Filter = 0;
1393 sBreakDeadTimeConfigAutomaticOutput = TIM_AUTOMATICOUTPUT_DISABLE;
1394 if (HAL_TIMEx_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig) != HAL_OK)
1395 {
1396     Error_Handler();
1397 }
1398 /* USER CODE BEGIN TIM1_Init 2 */
1399
1400 /* USER CODE END TIM1_Init 2 */
1401 HAL_TIM_MspPostInit(&htim1);
1402
1403 }
1404
```

main.c

```
1405
1406 static void MX_USART6_UART_Init(void)
1407 {
1408
1409     /* USER CODE BEGIN USART6_Init 0 */
1410
1411     /* USER CODE END USART6_Init 0 */
1412
1413     /* USER CODE BEGIN USART6_Init 1 */
1414
1415     /* USER CODE END USART6_Init 1 */
1416     huart6.Instance = USART6;
1417     huart6.Init.BaudRate = 115200;
1418     huart6.Init.WordLength = UART_WORDLENGTH_8B;
1419     huart6.Init.StopBits = UART_STOPBITS_1;
1420     huart6.Init.Parity = UART_PARITY_NONE;
1421     huart6.Init.Mode = UART_MODE_TX_RX;
1422     huart6.Init.HwFlowCtl = UART_HWCONTROL_NONE;
1423     huart6.Init.OverSampling = UART_OVERSAMPLING_16;
1424     huart6.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
1425     huart6.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
1426     if (HAL_UART_Init(&huart6) != HAL_OK)
1427     {
1428         Error_Handler();
1429     }
1430     /* USER CODE BEGIN USART6_Init 2 */
1431
1432     /* USER CODE END USART6_Init 2 */
1433
1434 }
1435
1436 /**
1437  * Enable DMA controller clock
1438  */
1439 static void MX_DMA_Init(void)
1440 {
1441
1442     /* DMA controller clock enable */
1443     __HAL_RCC_DMA2_CLK_ENABLE();
1444
1445     /* DMA interrupt init */
1446     /* DMA2_Stream4_IRQn interrupt configuration */
1447     HAL_NVIC_SetPriority(DMA2_Stream4_IRQn, 5, 0);
1448     HAL_NVIC_EnableIRQ(DMA2_Stream4_IRQn);
1449     /* DMA2_Stream7_IRQn interrupt configuration */
1450     HAL_NVIC_SetPriority(DMA2_Stream7_IRQn, 5, 0);
1451     HAL_NVIC_EnableIRQ(DMA2_Stream7_IRQn);
1452
1453 }
1454
1455 /* FMC initialization function */
1456 static void MX_FMC_Init(void)
1457 {
1458
1459     /* USER CODE BEGIN FMC_Init 0 */
1460
1461     /* USER CODE END FMC_Init 0 */
1462
1463     FMC_SDRAM_TimingTypeDef SdramTiming = {0};
1464
1465     /* USER CODE BEGIN FMC_Init 1 */
1466
```

main.c

```

1467  /* USER CODE END FMC_Init 1 */
1468
1469  /** Perform the SDRAM1 memory initialization sequence
1470  */
1471  hsdram1.Instance = FMC_SDRAM_DEVICE;
1472  /* hsdram1.Init */
1473  hsdram1.Init.SDBank = FMC_SDRAM_BANK1;
1474  hsdram1.Init.ColumnBitsNumber = FMC_SDRAM_COLUMN_BITS_NUM_8;
1475  hsdram1.Init.RowBitsNumber = FMC_SDRAM_ROW_BITS_NUM_12;
1476  hsdram1.Init.MemoryDataWidth = FMC_SDRAM_MEM_BUS_WIDTH_16;
1477  hsdram1.Init.InternalBankNumber = FMC_SDRAM_INTERN_BANKS_NUM_4;
1478  hsdram1.Init.CASLatency = FMC_SDRAM_CAS_LATENCY_3;
1479  hsdram1.Init.WriteProtection = FMC_SDRAM_WRITE_PROTECTION_DISABLE;
1480  hsdram1.Init.SDClockPeriod = FMC_SDRAM_CLOCK_PERIOD_2;
1481  hsdram1.Init.ReadBurst = FMC_SDRAM_RBURST_ENABLE;
1482  hsdram1.Init.ReadPipeDelay = FMC_SDRAM_RPIPE_DELAY_0;
1483  /* SdramTiming */
1484  SdramTiming.LoadToActiveDelay = 2;
1485  SdramTiming.ExitSelfRefreshDelay = 7;
1486  SdramTiming.SelfRefreshTime = 4;
1487  SdramTiming.RowCycleDelay = 7;
1488  SdramTiming.WriteRecoveryTime = 3;
1489  SdramTiming.RPDelay = 2;
1490  SdramTiming.RCDDelay = 2;
1491
1492  if (HAL_SDRAM_Init(&hsdram1, &SdramTiming) != HAL_OK)
1493  {
1494      Error_Handler( );
1495  }
1496
1497  /* USER CODE BEGIN FMC_Init 2 */
1498
1499  /* USER CODE END FMC_Init 2 */
1500 }
1501
1502 /**
1503  * @brief GPIO Initialization Function
1504  * @param None
1505  * @retval None
1506  */
1507 static void MX_GPIO_Init(void)
1508 {
1509     GPIO_InitTypeDef GPIO_InitStruct = {0};
1510
1511     /* GPIO Ports Clock Enable */
1512     __HAL_RCC_GPIOE_CLK_ENABLE();
1513     __HAL_RCC_GPIOG_CLK_ENABLE();
1514     __HAL_RCC_GPIOB_CLK_ENABLE();
1515     __HAL_RCC_GPIOD_CLK_ENABLE();
1516     __HAL_RCC_GPIOC_CLK_ENABLE();
1517     __HAL_RCC_GPIOA_CLK_ENABLE();
1518     __HAL_RCC_GPIOJ_CLK_ENABLE();
1519     __HAL_RCC_GPIOI_CLK_ENABLE();
1520     __HAL_RCC_GPIOK_CLK_ENABLE();
1521     __HAL_RCC_GPIOF_CLK_ENABLE();
1522     __HAL_RCC_GPIOH_CLK_ENABLE();
1523
1524     /*Configure GPIO pin Output Level */
1525     HAL_GPIO_WritePin(OTG_FS_PowerSwitchOn_GPIO_Port, OTG_FS_PowerSwitchOn_Pin,
1526                       GPIO_PIN_SET);
1527     /*Configure GPIO pin Output Level */

```

main.c

```
1528 HAL_GPIO_WritePin(GPIOI, ARDUINO_D7_Pin|ARDUINO_D8_Pin, GPIO_PIN_RESET);
1529
1530 /*Configure GPIO pin Output Level */
1531 HAL_GPIO_WritePin(LCD_BL_CTRL_GPIO_Port, LCD_BL_CTRL_Pin, GPIO_PIN_SET);
1532
1533 /*Configure GPIO pin Output Level */
1534 HAL_GPIO_WritePin(LCD_DISP_GPIO_Port, LCD_DISP_Pin, GPIO_PIN_SET);
1535
1536 /*Configure GPIO pin Output Level */
1537 HAL_GPIO_WritePin(DCMI_PWR_EN_GPIO_Port, DCMI_PWR_EN_Pin, GPIO_PIN_RESET);
1538
1539 /*Configure GPIO pin Output Level */
1540 HAL_GPIO_WritePin(GPIOG, ARDUINO_D4_Pin|ARDUINO_D2_Pin|EXT_RST_Pin, GPIO_PIN_RESET);
1541
1542 /*Configure GPIO pin : OTG_HS_OverCurrent_Pin */
1543 GPIO_InitStruct.Pin = OTG_HS_OverCurrent_Pin;
1544 GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
1545 GPIO_InitStruct.Pull = GPIO_NOPULL;
1546 HAL_GPIO_Init(OTG_HS_OverCurrent_GPIO_Port, &GPIO_InitStruct);
1547
1548 /*Configure GPIO pins : ULPI_D7_Pin ULPI_D6_Pin ULPI_D5_Pin ULPI_D3_Pin
1549                          ULPI_D2_Pin ULPI_D1_Pin ULPI_D4_Pin */
1550 GPIO_InitStruct.Pin = ULPI_D7_Pin|ULPI_D6_Pin|ULPI_D5_Pin|ULPI_D3_Pin
1551                      |ULPI_D2_Pin|ULPI_D1_Pin|ULPI_D4_Pin;
1552 GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
1553 GPIO_InitStruct.Pull = GPIO_NOPULL;
1554 GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
1555 GPIO_InitStruct.Alternate = GPIO_AF10_OTG_HS;
1556 HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
1557
1558 /*Configure GPIO pin : OTG_FS_VBUS_Pin */
1559 GPIO_InitStruct.Pin = OTG_FS_VBUS_Pin;
1560 GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
1561 GPIO_InitStruct.Pull = GPIO_NOPULL;
1562 HAL_GPIO_Init(OTG_FS_VBUS_GPIO_Port, &GPIO_InitStruct);
1563
1564 /*Configure GPIO pin : Audio_INT_Pin */
1565 GPIO_InitStruct.Pin = Audio_INT_Pin;
1566 GPIO_InitStruct.Mode = GPIO_MODE_EVT_RISING;
1567 GPIO_InitStruct.Pull = GPIO_NOPULL;
1568 HAL_GPIO_Init(Audio_INT_GPIO_Port, &GPIO_InitStruct);
1569
1570 /*Configure GPIO pin : OTG_FS_PowerSwitchOn_Pin */
1571 GPIO_InitStruct.Pin = OTG_FS_PowerSwitchOn_Pin;
1572 GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
1573 GPIO_InitStruct.Pull = GPIO_NOPULL;
1574 GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
1575 HAL_GPIO_Init(OTG_FS_PowerSwitchOn_GPIO_Port, &GPIO_InitStruct);
1576
1577 /*Configure GPIO pins : ARDUINO_D7_Pin ARDUINO_D8_Pin LCD_DISP_Pin */
1578 GPIO_InitStruct.Pin = ARDUINO_D7_Pin|ARDUINO_D8_Pin|LCD_DISP_Pin;
1579 GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
1580 GPIO_InitStruct.Pull = GPIO_NOPULL;
1581 GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
1582 HAL_GPIO_Init(GPIOI, &GPIO_InitStruct);
1583
1584 /*Configure GPIO pin : uSD_Detect_Pin */
1585 GPIO_InitStruct.Pin = uSD_Detect_Pin;
1586 GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
1587 GPIO_InitStruct.Pull = GPIO_NOPULL;
1588 HAL_GPIO_Init(uSD_Detect_GPIO_Port, &GPIO_InitStruct);
1589
```

main.c

```

1590  /*Configure GPIO pin : LCD_BL_CTRL_Pin */
1591  GPIO_InitStruct.Pin = LCD_BL_CTRL_Pin;
1592  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
1593  GPIO_InitStruct.Pull = GPIO_NOPULL;
1594  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
1595  HAL_GPIO_Init(LCD_BL_CTRL_GPIO_Port, &GPIO_InitStruct);
1596
1597  /*Configure GPIO pin : OTG_FS_OverCurrent_Pin */
1598  GPIO_InitStruct.Pin = OTG_FS_OverCurrent_Pin;
1599  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
1600  GPIO_InitStruct.Pull = GPIO_NOPULL;
1601  HAL_GPIO_Init(OTG_FS_OverCurrent_GPIO_Port, &GPIO_InitStruct);
1602
1603  /*Configure GPIO pins : TP3_Pin NC2_Pin */
1604  GPIO_InitStruct.Pin = TP3_Pin|NC2_Pin;
1605  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
1606  GPIO_InitStruct.Pull = GPIO_NOPULL;
1607  HAL_GPIO_Init(GPIOH, &GPIO_InitStruct);
1608
1609  /*Configure GPIO pin : DCMI_PWR_EN_Pin */
1610  GPIO_InitStruct.Pin = DCMI_PWR_EN_Pin;
1611  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
1612  GPIO_InitStruct.Pull = GPIO_NOPULL;
1613  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
1614  HAL_GPIO_Init(DCMI_PWR_EN_GPIO_Port, &GPIO_InitStruct);
1615
1616  /*Configure GPIO pin : LCD_INT_Pin */
1617  GPIO_InitStruct.Pin = LCD_INT_Pin;
1618  GPIO_InitStruct.Mode = GPIO_MODE_EVT_RISING;
1619  GPIO_InitStruct.Pull = GPIO_NOPULL;
1620  HAL_GPIO_Init(LCD_INT_GPIO_Port, &GPIO_InitStruct);
1621
1622  /*Configure GPIO pin : ULPI_NXT_Pin */
1623  GPIO_InitStruct.Pin = ULPI_NXT_Pin;
1624  GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
1625  GPIO_InitStruct.Pull = GPIO_NOPULL;
1626  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
1627  GPIO_InitStruct.Alternate = GPIO_AF10_OTG_HS;
1628  HAL_GPIO_Init(ULPI_NXT_GPIO_Port, &GPIO_InitStruct);
1629
1630  /*Configure GPIO pins : ARDUINO_D4_Pin ARDUINO_D2_Pin EXT_RST_Pin */
1631  GPIO_InitStruct.Pin = ARDUINO_D4_Pin|ARDUINO_D2_Pin|EXT_RST_Pin;
1632  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
1633  GPIO_InitStruct.Pull = GPIO_NOPULL;
1634  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
1635  HAL_GPIO_Init(GPIOG, &GPIO_InitStruct);
1636
1637  /*Configure GPIO pins : ULPI_STP_Pin ULPI_DIR_Pin */
1638  GPIO_InitStruct.Pin = ULPI_STP_Pin|ULPI_DIR_Pin;
1639  GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
1640  GPIO_InitStruct.Pull = GPIO_NOPULL;
1641  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
1642  GPIO_InitStruct.Alternate = GPIO_AF10_OTG_HS;
1643  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
1644
1645  /*Configure GPIO pin : RMII_RXER_Pin */
1646  GPIO_InitStruct.Pin = RMII_RXER_Pin;
1647  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
1648  GPIO_InitStruct.Pull = GPIO_NOPULL;
1649  HAL_GPIO_Init(RMII_RXER_GPIO_Port, &GPIO_InitStruct);
1650
1651  /*Configure GPIO pins : ULPI_CLK_Pin ULPI_D0_Pin */

```

main.c

```
1652 GPIO_InitStruct.Pin = ULPI_CLK_Pin|ULPI_D0_Pin;
1653 GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
1654 GPIO_InitStruct.Pull = GPIO_NOPULL;
1655 GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
1656 GPIO_InitStruct.Alternate = GPIO_AF10_OTG_HS;
1657 HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
1658
1659 }
1660
1661 /* USER CODE BEGIN 4 */
1662
1663 static void Display_DemoDescription(void)
1664 {
1665
1666     /* Set LCD Foreground Layer */
1667     BSP_LCD_Init();
1668     BSP_LCD_LayerDefaultInit(0, LCD_FB_START_ADDRESS);
1669
1670     BSP_LCD_DisplayOn();
1671     BSP_LCD_Clear(LCD_COLOR_BLUE);
1672     BSP_LCD_SetFont(&LCD_DEFAULT_FONT);
1673     BSP_LCD_SetTextColor(LCD_COLOR_DARKBLUE);
1674     BSP_LCD_SetFont(&Font16);
1675
1676     /* Display LCD messages */
1677
1678     BSP_LCD_DisplayStringAt(0, 10, (uint8_t *)"Instituto Federal de Santa Catarina",
1679                             CENTER_MODE);
1680     BSP_LCD_DisplayStringAt(0, 35, (uint8_t *)"Curso de Engenharia Eletronica",
1681                             CENTER_MODE);
1682     BSP_LCD_DisplayStringAt(0, 70, (uint8_t *)"Equalizador Parametrico Digital",
1683                             CENTER_MODE);
1684     BSP_LCD_DisplayStringAt(0, 100, (uint8_t *) "JOSE NICOLAU VARELA", CENTER_MODE);
1685 }
1686
1687
1688 /* USER CODE END Header_StartDefaultTask */
1689 void StartDefaultTask(void const * argument)
1690 {
1691     /* init code for USB_HOST */
1692     MX_USB_HOST_Init();
1693     /* USER CODE BEGIN 5 */
1694     /* Infinite loop */
1695     for(;;)
1696     {
1697         osDelay(1);
1698     }
1699     /* USER CODE END 5 */
1700 }
1701
1702 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
1703 {
1704     /* USER CODE BEGIN Callback 0 */
1705
1706     /* USER CODE END Callback 0 */
1707     if (htim->Instance == TIM6) {
1708         HAL_IncTick();
1709     }
1710     /* USER CODE BEGIN Callback 1 */
```

main.c

```
1711
1712  /* USER CODE END Callback 1 */
1713 }
1714
1715 void Error_Handler(void)
1716 {
1717
1718 }
1719
1720 #ifndef USE_FULL_ASSERT
1721
1722 void assert_failed(uint8_t *file, uint32_t line)
1723 {
1724
1725 }
1726 #endif /* USE_FULL_ASSERT */
1727
1728
1729
```

APÊNDICE 3

AudioProcess.c

```
1 /*
2  * AudioProcess.c
3  *
4  * Created on: Feb 6, 2021
5  * Author: José Nicolau VRELA
6  */
7
8 /* USER CODE BEGIN Header */
9
10 /* Includes -----*/
11 #include "main.h"
12 #include "cmsis_os.h"
13
14
15 /* Private includes -----*/
16 /* USER CODE BEGIN Includes */
17 #include "arm_math.h"
18 #include <stdlib.h>
19 #include <stdio.h>
20 #include "string.h"
21 #include <math.h>
22 #include "WM.h"
23
24 typedef enum
25 {
26     BUFFER_OFFSET_NONE = 0,
27     BUFFER_OFFSET_HALF = 1,
28     BUFFER_OFFSET_FULL = 2,
29 } BUFFER_StateTypeDef;
30
31
32 volatile uint32_t audio_rec_buffer_state;
33
34 #define VOL_IIR 74
35
36 #define AUDIO_BLOCK_SIZE ((uint32_t)2048)
37 #define SIGNAL_SAMPLES AUDIO_BLOCK_SIZE/2
38 #define AUDIO_BUFFER_IN AUDIO_REC_START_ADDR /* In SDRAM */
39 #define AUDIO_BUFFER_OUT (AUDIO_REC_START_ADDR + (AUDIO_BLOCK_SIZE * 2)) /* In SDRAM */
40 #define float_buffer_in (AUDIO_REC_START_ADDR + (AUDIO_BLOCK_SIZE * 4))
41 #define float_buffer_out (AUDIO_REC_START_ADDR + (AUDIO_BLOCK_SIZE * 6))
42 #define NUM_TAPS_FIR ((uint16_t)232)
43 #define numStages_IIR 1
44
45 #define BUFSIZE 9
46
47 unsigned char logmsgbuff[128];
48
49
50 /* USER CODE BEGIN PFP */
51 void eq_coef_calc(float* pCoeffs, uint16_t f0, int8_t G, float Q, uint16_t fs);
52 void eq_coef_LowShelf(float* pCoeffs, uint16_t f0, int8_t G, uint16_t fs);
53 void eq_coef_HighShelf(float* pCoeffs, uint16_t f0, int8_t G, uint16_t fs);
54 void audio_init_with_memory(void);
55 uint8_t CheckForUserInputTP(void);
56 const char * logmsg(text, value, base);
57 uint8_t CheckForUserInput(void);
58 void BSP_AUDIO_OUT_HalfTransfer_Callback(void);
59 void BSP_AUDIO_IN_TransferComplete_Callback(void);
60 void BSP_AUDIO_IN_HalfTransfer_Callback(void);
61 void FilterCoffCalc(void);
62 static void PID_X_Exec(void);
```

AudioProcess.c

```

63 static void GUIThread(void const * argument);
64
65 void audio_init_with_memory(void){
66
67     if (BSP_AUDIO_IN_OUT_Init(INPUT_DEVICE_INPUT_LINE_1, OUTPUT_DEVICE_HEADPHONE,
        SAI_AUDIO_FREQUENCY_48K, DEFAULT_AUDIO_IN_BIT_RESOLUTION, DEFAULT_AUDIO_IN_CHANNEL_NBR) ==
        AUDIO_OK)
68     {
69 //         BSP_LCD_SetBackColor(LCD_COLOR_WHITE);
70 //         BSP_LCD_SetTextColor(LCD_COLOR_DARKGREEN);
71 //         BSP_LCD_DisplayStringAt(0, BSP_LCD_GetYSize() - 95, (uint8_t *)"  AUDIO RECORD INIT
        OK ", CENTER_MODE);
72     }
73     else
74     {
75 //         BSP_LCD_SetBackColor(LCD_COLOR_WHITE);
76 //         BSP_LCD_SetTextColor(LCD_COLOR_RED);
77 //         BSP_LCD_DisplayStringAt(0, BSP_LCD_GetYSize() - 95, (uint8_t *)"  AUDIO RECORD INIT
        FAIL", CENTER_MODE);
78 //         BSP_LCD_DisplayStringAt(0, BSP_LCD_GetYSize() - 80, (uint8_t *)" Try to reset board
        ", CENTER_MODE);
79     }
80
81     /* Initialize SDRAM buffers */
82     memset((uint16_t*)AUDIO_BUFFER_IN, 0, AUDIO_BLOCK_SIZE*2);
83     memset((uint16_t*)AUDIO_BUFFER_OUT, 0, AUDIO_BLOCK_SIZE*2);
84     audio_rec_buffer_state = BUFFER_OFFSET_NONE;
85
86     /* Start Recording */
87     BSP_AUDIO_IN_Record((uint16_t*)AUDIO_BUFFER_IN, AUDIO_BLOCK_SIZE);
88
89     /* Start Playback */
90     // BSP_AUDIO_OUT_SetVolume(Vol);
91     BSP_AUDIO_OUT_SetAudioFrameSlot(CODEC_AUDIOFRAME_SLOT_02);
92     BSP_AUDIO_OUT_Play((uint16_t*)AUDIO_BUFFER_OUT, AUDIO_BLOCK_SIZE * 2);
93 }
94
95
96 ///////////////////////////////////CALCULAR OS COEFF IIR
    PEAKS////////////////////////////////////
97
98 void eq_coef_calc(float* pCoeffs,uint16_t f0, int8_t G, float Q, uint16_t fs){
99
100
101     float fc =(float) f0/((float)fs);
102     float K = tan(M_PI*((float) fc)/2);
103     float V;
104     float norm;
105
106
107
108     if (G >= 0 )
109     {
110         V = pow(10, ((float)G/20) );           //
111
112         norm = 1/( 1 + 1/Q * K + K * K );
113         pCoeffs[0] = ( 1 +(V/Q) * K + K * K ) * norm;
114         pCoeffs[1]= 2 * (K*K - 1 ) * norm;
115         pCoeffs[2] = ( 1 - (V/Q) * K + K * K ) * norm;
116         pCoeffs[3] = 2 * ( K * K - 1) * norm;
117         pCoeffs[4] = (1 - (1/Q)* K + K * K ) * norm;
118     }

```

AudioProcess.c

```

119         else
120         {
121             V = pow(10, ((float)(G*(-1))/20) );
122             norm = 1/( 1 + (V/Q) * K + K * K );
123             pCoeffs[0] = ( 1 +(1/Q) * K + K * K ) * norm;
124             pCoeffs[1] = 2 * (K*K - 1 ) * norm;
125             pCoeffs[2] = ( 1 - (1/Q) * K + K * K ) * norm;
126             pCoeffs[3] = 2 * ( K * K - 1) * norm;
127             pCoeffs[4] = (1 - (V/Q) * K + K * K ) * norm;
128         }
129
130 }
131 ////////////////////////////////////////////////////Coeff
132 LowShelf//////////////////////////////////////
133 void eq_coef_LowShelf(float* pCoeffs,uint16_t f0, int8_t G, uint16_t fs){
134
135     float fc =(float) f0/((float)fs);
136     float K = tan(M_PI*((float) fc)/2);
137     float V;
138     float norm;
139
140
141
142
143     if (G >= 0 )
144     {
145
146         V = pow(10, ((float)G/20) );
147         norm = 1/( 1 + sqrt(2) * K + K * K );
148         pCoeffs[0] = ( 1 +sqrt(2*V) * K + V * K * K ) * norm;
149         pCoeffs[1]= 2 * (V*K*K - 1 ) * norm;
150         pCoeffs[2] = ( 1 - sqrt(2*V) * K + V * K * K ) * norm;
151         pCoeffs[3] = 2 * ( K * K - 1) * norm;
152         pCoeffs[4] = (1 - sqrt(2)* K + K * K ) * norm;
153     }
154     else
155     {
156
157         V = pow(10, ((float)(G*(-1))/20) );
158
159         norm = 1/( 1 + sqrt(2*V) * K + V * K * K );
160         pCoeffs[0] = ( 1 + sqrt(2) * K + K * K ) * norm;
161         pCoeffs[1] = 2 * (K*K - 1 ) * norm;
162         pCoeffs[2] = ( 1 - sqrt(2) * K + K * K ) * norm;
163         pCoeffs[3] = 2 * ( V * K * K - 1) * norm;
164         pCoeffs[4] = (1 - sqrt(2*V) * K + V * K * K ) * norm;
165     }
166
167 }
168
169 void eq_coef_HighShelf(float* pCoeffs,uint16_t f0, int8_t G, uint16_t fs){
170
171
172     float fc =(float) f0/((float)fs);
173     float K = tan(M_PI*((float) fc)/2);
174     float V;
175     float norm, norm2, norm1;
176
177
178     if (G >= 0 )
179     {

```

AudioProcess.c

```

180
181     V = pow(10, ((float)G/20) );
182     norm = 1/( 1 + sqrt(2) * K + K * K );
183     pCoeffs[0] = ( V + sqrt(2*V) * K + K * K ) * norm;
184     pCoeffs[1] = ( 2 * (K*K - V) ) * norm;
185     pCoeffs[2] = ( V - sqrt(2*V) * K + K * K ) * norm;
186     pCoeffs[3] = ( 2 * ( K * K - 1) ) * norm;
187     pCoeffs[4] = ( 1 - sqrt(2)* K + K * K ) * norm;
188 }
189 else
190 {
191
192     V = pow(10, ((float)(G*(-1))/20) );
193
194     norm1 = 1/( V + sqrt(2*V) * K + K * K );
195     pCoeffs[0] = ( 1 + sqrt(2) * K + K * K ) * norm1;
196     pCoeffs[1] = ( 2 * (K*K - 1) ) * norm1;
197     pCoeffs[2] = ( 1 - sqrt(2) * K + K * K ) * norm1;
198
199     norm2 = 1/( 1 + sqrt(2/V)* K + (K * K)/V );
200     pCoeffs[3] = (2 * ((K * K /V) - 1) ) * norm2;
201     pCoeffs[4] = ( 1 - sqrt(2/V) * K + (K * K) /V ) * norm2;
202 }
203
204 }
205
206
207 static void ConvertSampleBufferToDMABuffer (int16_t * samplebuffer,uint8_t * dmabuffer,
208      uint32_t num_samples)
209 {
210     for (uint32_t i=0;i<num_samples;i++)
211     {
212         int16_t * p = (int16_t *)&dmabuffer[i*8];
213         *p = samplebuffer[i];
214         *(p+2) = samplebuffer[i];
215     }
216 }
217 }
218
219 static void ExtractSampleBufferToDMABuffer (int16_t * samplebuffer,uint8_t * dmabuffer,
220      uint32_t num_samples)
221 {
222     for (uint32_t i=0;i<num_samples;i++)
223     {
224         int16_t * samplePointer = (int16_t *)&dmabuffer[i*8];
225         *samplePointer = samplebuffer[i];
226         samplebuffer[i]= *samplePointer;
227     }
228 }
229 }
230
231 ///////////////////////////////////////////////////ATIVAR TS ////////////////////////////////////////
232 static void PID_X_Exec(void) {
233     TS_StateTypeDef TS_State;
234     static GUI_PID_STATE StatePID;
235     static int IsTouched;
236
237     BSP_TS_GetState(&TS_State);
238     StatePID.Layer = 0;
239     if (TS_State.touchDetected) {

```

AudioProcess.c

```
240     IsTouched = 1;
241     StatePID.Pressed = 1;
242     StatePID.x = (int)(TS_State.touchX[0]);
243     StatePID.y = (int)(TS_State.touchY[0]);
244     GUI_PID_StoreState(&StatePID);
245 } else {
246     if (IsTouched == 1) {
247         IsTouched = 0;
248         StatePID.Pressed = 0;
249         GUI_PID_StoreState(&StatePID);
250     }
251 }
252 }
253
254
255
256
```

APÊNDICE 4

WindowDLG.c

```
1
2
3 // USER START (Optionally insert additional includes)
4 #include "arm_const_structs.h"
5 // USER END
6
7 #include "DIALOG.h"
8 #include "main.h"
9
10 /*****
11 *
12 *      Defines
13 *
14 *****/
15 */
16 #define ID_WINDOW_0 (GUI_ID_USER + 0x00)
17 #define ID_SLIDER_0 (GUI_ID_USER + 0x01)
18 #define ID_SLIDER_1 (GUI_ID_USER + 0x02)
19 #define ID_SLIDER_2 (GUI_ID_USER + 0x03)
20 #define ID_SLIDER_3 (GUI_ID_USER + 0x04)
21 #define ID_SLIDER_4 (GUI_ID_USER + 0x05)
22 #define ID_SLIDER_5 (GUI_ID_USER + 0x06)
23 #define ID_SLIDER_6 (GUI_ID_USER + 0x07)
24 #define ID_SLIDER_7 (GUI_ID_USER + 0x08)
25 #define ID_SLIDER_8 (GUI_ID_USER + 0x09)
26 #define ID_SLIDER_9 (GUI_ID_USER + 0x0A)
27 #define ID_GRAPH_0 (GUI_ID_USER + 0x0B)
28 #define ID_SPINBOX_0 (GUI_ID_USER + 0x10)
29 #define ID_TEXT_0 (GUI_ID_USER + 0x12)
30 #define ID_TEXT_1 (GUI_ID_USER + 0x13)
31 #define ID_TEXT_2 (GUI_ID_USER + 0x14)
32 #define ID_TEXT_3 (GUI_ID_USER + 0x15)
33 #define ID_TEXT_4 (GUI_ID_USER + 0x16)
34 #define ID_SLIDER_10 (GUI_ID_USER + 0x17)
35 #define ID_SLIDER_11 (GUI_ID_USER + 0x18)
36 #define ID_TEXT_5 (GUI_ID_USER + 0x1B)
37 #define ID_BUTTON_0 (GUI_ID_USER + 0x1C)
38 #define ID_BUTTON_1 (GUI_ID_USER + 0x1D)
39 #define ID_TEXT_6 (GUI_ID_USER + 0x21)
40 #define ID_DROPDOWN_0 (GUI_ID_USER + 0x29)
41 #define ID_DROPDOWN_1 (GUI_ID_USER + 0x2A)
42 #define ID_DROPDOWN_2 (GUI_ID_USER + 0x2B)
43 #define ID_GRAPH_1 (GUI_ID_USER + 0x2C)
44 #define ID_TEXT_7 (GUI_ID_USER + 0x2F)
45 #define ID_TEXT_8 (GUI_ID_USER + 0x30)
46 #define ID_BUTTON_2 (GUI_ID_USER + 0x31)
47 #define ID_BUTTON_3 (GUI_ID_USER + 0x32)
48 #define ID_BUTTON_4 (GUI_ID_USER + 0x33)
49 #define ID_BUTTON_5 (GUI_ID_USER + 0x34)
50 #define ID_BUTTON_6 (GUI_ID_USER + 0x35)
51 #define ID_TEXT_9 (GUI_ID_USER + 0x36)
52 #define ID_TEXT_10 (GUI_ID_USER + 0x37)
53 #define ID_TEXT_11 (GUI_ID_USER + 0x38)
54 #define ID_TEXT_12 (GUI_ID_USER + 0x39)
55 #define ID_TEXT_13 (GUI_ID_USER + 0x3A)
56 #define ID_TEXT_14 (GUI_ID_USER + 0x3B)
57
58
59 // USER START (Optionally insert additional defines)
60 #define FFT_SIZE ((uint32_t)256)
61 static GUI_POINT Points[FFT_BUFFER];
62 //static GRAPH_DATA_Handle SineData;
```

WindowDLG.c

```

63 extern U16 dmaBuffer[FFT_BUFFER];
64 void FFT_Data2(void);
65 Led_TypeDef led1;
66 WM_HWIN hItem;
67 WM_HWIN hWin;
68 int Gmais,Gmenos;
69 // USER END
70
71 /*****
72 *
73 *      Static data
74 *
75 *****/
76 */
77
78 // USER START (Optionally insert additional static data)
79 static void _UserDraw(WM_HWIN hWin, int Stage);
80 static void _cbBk(WM_MESSAGE * pMsg);
81 // USER END
82
83 /*****
84 *
85 *      _aDialogCreate
86 */
87 static const GUI_WIDGET_CREATE_INFO _aDialogCreate[] = {
88 { WINDOW_CreateIndirect, "Window", ID_WINDOW_0, -1, -1, 480, 272, 0, 0x0, 0 },
89 { SLIDER_CreateIndirect, "G_PK1", ID_SLIDER_0, 185, 140, 19, 70, 8, 0x0, 32 },
90 { SLIDER_CreateIndirect, "G_PK2", ID_SLIDER_1, 265, 140, 20, 70, 8, 0x0, 0 },
91 { SLIDER_CreateIndirect, "G_PK3", ID_SLIDER_2, 335, 140, 19, 70, 8, 0x0, 0 },
92 { SLIDER_CreateIndirect, "lowShelf", ID_SLIDER_3, 125, 140, 21, 70, 8, 0x0, 0 },
93 { SLIDER_CreateIndirect, "HighShelf", ID_SLIDER_4, 415, 140, 20, 70, 8, 0x0, 0 },
94 { SLIDER_CreateIndirect, "FRQ_LS", ID_SLIDER_5, 100, 235, 50, 20, 0, 0x0, 0 },
95 { SLIDER_CreateIndirect, "FREQ_PK1", ID_SLIDER_6, 170, 235, 50, 20, 0, 0x0, 0 },
96 { SLIDER_CreateIndirect, "FREQ_PK2", ID_SLIDER_7, 250, 235, 47, 20, 0, 0x0, 0 },
97 { SLIDER_CreateIndirect, "FREQ_PK3", ID_SLIDER_8, 330, 235, 47, 20, 0, 0x0, 0 },
98 { SLIDER_CreateIndirect, "FREQ_HS", ID_SLIDER_9, 410, 235, 47, 20, 0, 0x0, 0 },
99 { GRAPH_CreateIndirect, "GraphEQ", ID_GRAPH_0, 78, 16, 400, 95, 0, 0x0, 0 },
100 { SPINBOX_CreateIndirect, "Spinbox", ID_SPINBOX_0, 484, 159, 80, 20, 0, 0x0, 0 },
101 { TEXT_CreateIndirect, "lsf", ID_TEXT_0, 101, 218, 53, 20, 0, 0x64, 0 },
102 { TEXT_CreateIndirect, "p1", ID_TEXT_1, 156, 218, 80, 20, 0, 0x64, 0 },
103 { TEXT_CreateIndirect, "p2", ID_TEXT_2, 238, 218, 80, 20, 0, 0x64, 0 },
104 { TEXT_CreateIndirect, "p3", ID_TEXT_3, 319, 218, 80, 20, 0, 0x64, 0 },
105 { TEXT_CreateIndirect, "hsf", ID_TEXT_4, 394, 218, 80, 20, 0, 0x64, 0 },
106 { SLIDER_CreateIndirect, "Fact_Q", ID_SLIDER_10, 20, 235, 50, 20, 0, 0x0, 0 },
107 { SLIDER_CreateIndirect, "OUT", ID_SLIDER_11, 35, 110, 28, 100, 8, 0x0, 0 },
108 { TEXT_CreateIndirect, "Text", ID_TEXT_5, 8, 211, 80, 20, 0, 0x64, 0 },
109 { BUTTON_CreateIndirect, "LIGA", ID_BUTTON_0, 16, 30, 53, 20, 0, 0x0, 0 },
110 { BUTTON_CreateIndirect, "DESL", ID_BUTTON_1, 16, 64, 53, 20, 0, 0x0, 0 },
111 { TEXT_CreateIndirect, "titulo", ID_TEXT_6, 200, 98, 172, 20, 0, 0x64, 0 },
112 { TEXT_CreateIndirect, "Fator Q", ID_TEXT_7, 50, 250, 42, 20, 0, 0x64, 0 },
113 { TEXT_CreateIndirect, "Text", ID_TEXT_8, 30, 86, 50, 20, 0, 0x64, 0 },
114 // { BUTTON_CreateIndirect, "ls", ID_BUTTON_2, 107, 118, 50, 13, 0, 0x0, 0 },
115 { BUTTON_CreateIndirect, "p1", ID_BUTTON_3, 170, 118, 50, 13, 0, 0x0, 0 },
116 { BUTTON_CreateIndirect, "p2", ID_BUTTON_4, 250, 118, 50, 13, 0, 0x0, 0 },
117 { BUTTON_CreateIndirect, "p3", ID_BUTTON_5, 322, 118, 50, 13, 0, 0x0, 0 },
118 // { BUTTON_CreateIndirect, "hs", ID_BUTTON_6, 405, 118, 50, 13, 0, 0x0, 0 },
119 { TEXT_CreateIndirect, "Text", ID_TEXT_9, 139, 192, 32, 20, 0, 0x64, 0 },
120 { TEXT_CreateIndirect, "Text", ID_TEXT_10, 204, 192, 39, 19, 0, 0x64, 0 },
121 { TEXT_CreateIndirect, "Text", ID_TEXT_11, 251, 192, 80, 20, 0, 0x64, 0 },
122 { TEXT_CreateIndirect, "Text", ID_TEXT_12, 325, 192, 80, 20, 0, 0x64, 0 },
123 { TEXT_CreateIndirect, "Text", ID_TEXT_13, 403, 192, 80, 20, 0, 0x64, 0 },
124 { TEXT_CreateIndirect, "ass", ID_TEXT_14, 5, 1, 71, 20, 0, 0x64, 0 },

```

WindowDLG.c

```

125 // USER START (Optionally insert additional widgets)
126 // USER END
127 };
128
129 /*****
130 *
131 *     Static code
132 *
133 *****/
134 */
135
136 // USER START (Optionally insert additional static code)
137
138 GUI_POINT _aPoint[] = {{0, 35}, {10, 35}, {50, 35}, {120, 35}, {150, 35}, {175, 35}, {478,
    35} }; // Some Points to be drawn in the graph
139 static GUI_POINT _Cursor[] = { {0, 0}, {5, -5}, {5, -10}, {-5, -10}, {-5, -5}, {0, 0} };
    // These points describing the cursor
140 static int CursorPos;
141
142 // _DrawCursor
143
144 static void _DrawCursor(int xOrg, int yOrg) {
145     int i;
146     int x0, x1;
147     int y0, y1;
148     GUI_SetColor(GUI_WHITE);
149     for (i = 0; i < 5; i++) {
150         x0 = xOrg + _Cursor[i].x;
151         y0 = yOrg + _Cursor[i].y;
152         x1 = xOrg + _Cursor[i + 1].x;
153         y1 = yOrg + _Cursor[i + 1].y;
154         GUI_DrawLine(x0, y0, x1, y1);
155     }
156 }
157 /*****
158 *
159 *     _cbGraph
160 */
161 static void _cbGraph(WM_MESSAGE * pMsg) {
162     int xPos, yPos;
163     GUI_RECT Rect;
164     switch (pMsg->MsgId) {
165     case WM_PAINT:
166         //
167         // Draw the GRAPH with the default settings
168         //
169         GRAPH_Callback(pMsg);
170         //
171         // Now draw a cursor over the GRAPH
172         //
173         WM_GetClientRect(&Rect); // Get the client rectangle to receive the bottom
    most y position of the GRAPH area
174         xPos = _aPoint[CursorPos].x; // Use CursorPos as index and get the x coordinate
    of the point
175         yPos = _aPoint[CursorPos].y; // Same again but use the lowest y position to calculate
    where the cursor should be drawn
176         _DrawCursor(xPos, yPos); // Draw cursor
177         break;
178     default:
179         //
180         // Handle all other messages by the default callback
181         //

```

WindowDLG.c

```

182     GRAPH_Callback(pMsg);
183     break;
184 }
185 }
186
187
188 // USER END
189
190 /*****
191 *
192 *     _cbDialog
193 */
194 static void _cbDialog(WM_MESSAGE * pMsg) {
195     WM_HWIN hItem;
196     int NCode;
197     int Id;
198     // USER START (Optionally insert additional variables)
199     static GRAPH_DATA_Handle hGraphData;
200     static GRAPH_DATA_Handle hGraphSC;
201     static int DataAttached;
202
203
204     int aux;
205     int aux2;
206     char buf[6];
207     int gainPk1;
208
209     // USER END
210
211     switch (pMsg->MsgId) {
212     case WM_INIT_DIALOG:
213         //
214         // Initialization of 'Window'
215         //
216         hItem = pMsg->hWin;
217         WINDOW_SetBkColor(hItem, GUI_MAKE_COLOR(0x000112211));
218         //
219         // Initialization of 'GraphEQ'
220         //
221         hItem = WM_GetDialogItem(pMsg->hWin, ID_GRAPH_0);
222         GRAPH_SetBorder(hItem, 3, 3, 3, 16);
223
224         // meu codigo
225
226         hGraphData = GRAPH_DATA_XY_Create(GUI_WHITE, 110, _aPoint, GUI_COUNTOF(_aPoint));
227         GRAPH_AttachData(hItem, hGraphData);
228
229         GRAPH_SetGridVis(hItem, 5);
230         hGraphSC = GRAPH_SCALE_Create( 85, GUI_TA_LEFT, GRAPH_SCALE_CF_HORIZONTAL, 160);
231         // GRAPH_AttachScale(hItem, hGraphSC);
232         // GRAPH_SCALE_SetFactor(hGraphSC, log(80));
233         GRAPH_SetGridDistX(hItem, 80);
234         GRAPH_SetGridDistY(hItem, 6);
235         GRAPH_SetColor(hItem, 0x111111, GRAPH_CI_BORDER);
236
237         // Initialization of 'Text'
238         //
239         hItem = WM_GetDialogItem(pMsg->hWin, ID_TEXT_0);
240         TEXT_SetText(hItem, "FREQ-LS");
241         TEXT_SetTextColor(hItem, GUI_MAKE_COLOR(0x0000FF00));
242         TEXT_SetTextAlign(hItem, GUI_TA_HCENTER | GUI_TA_VCENTER);
243         TEXT_SetFont(hItem, GUI_FONT_8_1);

```

WindowDLG.c

```

244 //
245 // Initialization of 'Text'
246 //
247 hItem = WM_GetDialogItem(pMsg->hWin, ID_TEXT_1);
248 TEXT_SetText(hItem, "FREQ-PK1");
249 TEXT_SetTextAlign(hItem, GUI_TA_HCENTER | GUI_TA_VCENTER);
250 TEXT_SetTextColor(hItem, GUI_MAKE_COLOR(0x0000FF00));
251 TEXT_SetFont(hItem, GUI_FONT_8_1);
252 //
253 // Initialization of 'Text'
254 //
255 hItem = WM_GetDialogItem(pMsg->hWin, ID_TEXT_2);
256 TEXT_SetText(hItem, "FREQ-PK2");
257 TEXT_SetTextAlign(hItem, GUI_TA_HCENTER | GUI_TA_VCENTER);
258 TEXT_SetTextColor(hItem, GUI_MAKE_COLOR(0x0000FF00));
259 TEXT_SetFont(hItem, GUI_FONT_8_1);
260 //
261 // Initialization of 'Text'
262 //
263 hItem = WM_GetDialogItem(pMsg->hWin, ID_TEXT_3);
264 TEXT_SetText(hItem, "FREQ-PK3");
265 TEXT_SetTextColor(hItem, GUI_MAKE_COLOR(0x0000FF00));
266 TEXT_SetTextAlign(hItem, GUI_TA_HCENTER | GUI_TA_VCENTER);
267 TEXT_SetFont(hItem, GUI_FONT_8_1);
268 //
269 // Initialization of 'Text'
270 //
271 hItem = WM_GetDialogItem(pMsg->hWin, ID_TEXT_4);
272 TEXT_SetText(hItem, "FREQ-HS");
273 TEXT_SetTextColor(hItem, GUI_MAKE_COLOR(0x0000FF00));
274 TEXT_SetTextAlign(hItem, GUI_TA_HCENTER | GUI_TA_VCENTER);
275 TEXT_SetFont(hItem, GUI_FONT_8_1);
276 //
277 // Initialization of 'Text'
278 //
279 hItem = WM_GetDialogItem(pMsg->hWin, ID_TEXT_5);
280 TEXT_SetText(hItem, "OUT");
281 TEXT_SetTextColor(hItem, GUI_MAKE_COLOR(0x0000ff00));
282 TEXT_SetFont(hItem, GUI_FONT_13HB_1);
283 TEXT_SetTextAlign(hItem, GUI_TA_HCENTER | GUI_TA_VCENTER);
284 //
285 // Initialization of 'LIGA'
286 //
287 hItem = WM_GetDialogItem(pMsg->hWin, ID_BUTTON_0);
288 BUTTON_SetText(hItem, "EQ-ON");
289 //
290 // Initialization of 'DESL'
291 //
292 hItem = WM_GetDialogItem(pMsg->hWin, ID_BUTTON_1);
293 BUTTON_SetText(hItem, "EQ-OFF");
294 //
295 // Initialization of 'titulo'
296 //
297 hItem = WM_GetDialogItem(pMsg->hWin, ID_TEXT_6);
298 TEXT_SetText(hItem, "PARAMETRIC EQUALIZER");
299 TEXT_SetFont(hItem, GUI_FONT_13HB_1);
300 TEXT_SetTextColor(hItem, GUI_MAKE_COLOR(0x0000ff));
301 //
302 //
303 // Initialization of 'Text'
304 //
305 hItem = WM_GetDialogItem(pMsg->hWin, ID_TEXT_7);

```

WindowDLG.c

```

306 TEXT_SetText(hItem, "Factor Q");
307 TEXT_SetTextAlign(hItem, GUI_TA_HCENTER | GUI_TA_VCENTER);
308 TEXT_SetTextColor(hItem, GUI_MAKE_COLOR(0x00FF0000));
309 TEXT_SetFont(hItem, GUI_FONT_13B_1);
310 //
311 // Initialization of 'Text'
312 //
313 hItem = WM_GetDialogItem(pMsg->hWin, ID_TEXT_8);
314 TEXT_SetText(hItem, "OUT");
315 TEXT_SetTextAlign(hItem, GUI_TA_HCENTER | GUI_TA_VCENTER);
316 TEXT_SetTextColor(hItem, GUI_MAKE_COLOR(0x0000FFCC));
317 TEXT_SetFont(hItem, GUI_FONT_13B_1);
318
319
320 // Initialization of 'p1'
321 //
322 hItem = WM_GetDialogItem(pMsg->hWin, ID_BUTTON_3);
323 BUTTON_SetText(hItem, "Q-PK-1");
324 BUTTON_SetBkColor(hItem, BUTTON_CI_DISABLED, GUI_RED);
325 BUTTON_SetFont(hItem, GUI_FONT_10S_1);
326 //
327 // Initialization of 'p2'
328 //
329 hItem = WM_GetDialogItem(pMsg->hWin, ID_BUTTON_4);
330 BUTTON_SetText(hItem, "Q-PK-2");
331 BUTTON_SetBkColor(hItem, BUTTON_CI_DISABLED, GUI_BLUE);
332 BUTTON_SetFont(hItem, GUI_FONT_10S_1);
333 //
334 // Initialization of 'p3'
335 //
336 hItem = WM_GetDialogItem(pMsg->hWin, ID_BUTTON_5);
337 BUTTON_SetText(hItem, "Q-PK-3");
338 BUTTON_SetBkColor(hItem, BUTTON_CI_DISABLED, GUI_YELLOW);
339 BUTTON_SetFont(hItem, GUI_FONT_10S_1);
340 //
341 // Initialization of 'Text'
342 //
343 hItem = WM_GetDialogItem(pMsg->hWin, ID_TEXT_9);
344 TEXT_SetText(hItem, "GAIN");
345 TEXT_SetTextAlign(hItem, GUI_TA_HCENTER | GUI_TA_VCENTER);
346 TEXT_SetTextColor(hItem, GUI_MAKE_COLOR(0x0000FF80));
347 TEXT_SetFont(hItem, GUI_FONT_8_ASCII);
348
349 // Initialization of 'Text'
350 hItem = WM_GetDialogItem(pMsg->hWin, ID_TEXT_10);
351 TEXT_SetText(hItem, "GAIN");
352 TEXT_SetTextAlign(hItem, GUI_TA_HCENTER | GUI_TA_VCENTER);
353 TEXT_SetTextColor(hItem, GUI_MAKE_COLOR(0x0000FF80));
354 TEXT_SetFont(hItem, GUI_FONT_8_1);
355
356 // Initialization of 'Text'
357
358 hItem = WM_GetDialogItem(pMsg->hWin, ID_TEXT_11);
359 TEXT_SetTextColor(hItem, GUI_MAKE_COLOR(0x0000FF80));
360 TEXT_SetText(hItem, "GAIN");
361 TEXT_SetTextAlign(hItem, GUI_TA_HCENTER | GUI_TA_VCENTER);
362 TEXT_SetFont(hItem, GUI_FONT_8_1);
363 //
364 // Initialization of 'Text'
365 //
366 hItem = WM_GetDialogItem(pMsg->hWin, ID_TEXT_12);
367 TEXT_SetTextColor(hItem, GUI_MAKE_COLOR(0x0000FF80));

```

WindowDLG.c

```

368 TEXT_SetTextAlign(hItem, GUI_TA_HCENTER | GUI_TA_VCENTER);
369 TEXT_SetText(hItem, "GAIN");
370 TEXT_SetFont(hItem, GUI_FONT_8_1);
371 //
372 // Initialization of 'Text'
373 //
374 hItem = WM_GetDialogItem(pMsg->hWin, ID_TEXT_13);
375 TEXT_SetTextColor(hItem, GUI_MAKE_COLOR(0x0000FF80));
376 TEXT_SetTextAlign(hItem, GUI_TA_HCENTER | GUI_TA_VCENTER);
377 TEXT_SetText(hItem, "GAIN");
378 TEXT_SetFont(hItem, GUI_FONT_8_1);
379 //
380 // Initialization of 'ass'
381 //
382 hItem = WM_GetDialogItem(pMsg->hWin, ID_TEXT_14);
383 TEXT_SetTextColor(hItem, GUI_MAKE_COLOR(0x0040ff80));
384 TEXT_SetTextAlign(hItem, GUI_TA_HCENTER | GUI_TA_VCENTER);
385 TEXT_SetText(hItem, "EQPR-JNV");
386 // USER START (Optionally insert additional code for further widget initialization)
387
388 hItem = WM_GetDialogItem(pMsg->hWin, ID_BUTTON_0); // init liga
389
390
391 // volume out
392 hItem = WM_GetDialogItem(pMsg->hWin, ID_SLIDER_11);
393 SLIDER_SetNumTicks(hItem, 10);
394 SLIDER_SetRange(hItem, -100, 0);
395 SLIDER_SetValue(hItem, -74);
396 aux = SLIDER_GetValue(hItem);
397 aux *=-1;
398
399 hItem = WM_GetDialogItem(pMsg->hWin, ID_SLIDER_10);
400 SLIDER_SetNumTicks(hItem, 20);
401 SLIDER_SetRange(hItem, 1, 20);
402 SLIDER_SetValue(hItem, 5);
403 Fact_Q = SLIDER_GetValue(hItem);
404
405 ///////////////////////////////////////////////////
406 // PEAK 1 GAIN G
407 hItem = WM_GetDialogItem(pMsg->hWin, ID_SLIDER_0);
408 SLIDER_SetNumTicks(hItem, 25);
409 SLIDER_SetRange(hItem, -12, 12);
410 SLIDER_SetValue(hItem, 0);
411 gainPK1 = SLIDER_GetValue(hItem);
412 gainPK1*=-1;
413
414 // PEAK 1 FREQ F0
415
416 hItem = WM_GetDialogItem(pMsg->hWin, ID_SLIDER_6);
417 SLIDER_SetNumTicks(hItem, 100);
418 SLIDER_SetRange(hItem, 250, 4000); // passo de 38 aproximadamente
419 SLIDER_SetValue(hItem, 1450);
420 frqPK1 = SLIDER_GetValue(hItem);
421 ///////////////////////////////////////////////////
422 // PEAK 2 GAIN G
423 hItem = WM_GetDialogItem(pMsg->hWin, ID_SLIDER_1);
424 SLIDER_SetNumTicks(hItem, 25);
425 SLIDER_SetRange(hItem, -12, 12);
426 SLIDER_SetValue(hItem, 0);
427 gainPK2 = SLIDER_GetValue(hItem);
428 gainPK2*=-1;
429

```

WindowDLG.c

```

430 // PEAK 2 FREQ F0
431
432 hItem = WM_GetDialogItem(pMsg->hWin, ID_SLIDER_7);
433 SLIDER_SetNumTicks(hItem, 100);
434 SLIDER_SetRange(hItem,500,6500); // passo de 60
435 SLIDER_SetValue(hItem, 3500);
436 frqPK2 = SLIDER_GetValue(hItem);
437 ///////////////////////////////////////////////////
438 // PEAK 3 GAIN G
439 hItem = WM_GetDialogItem(pMsg->hWin, ID_SLIDER_2);
440 SLIDER_SetNumTicks(hItem, 25);
441 SLIDER_SetRange(hItem,-12,12);
442 SLIDER_SetValue(hItem, 0);
443 gainPK3 = SLIDER_GetValue(hItem);
444 gainPK3*=-1;
445
446 // PEAK 3 FREQ F0
447
448 hItem = WM_GetDialogItem(pMsg->hWin, ID_SLIDER_8);
449 SLIDER_SetNumTicks(hItem, 100);
450 SLIDER_SetRange(hItem,1600,16000); // passo de 144
451 SLIDER_SetValue(hItem, 8800);
452 frqPK3 = SLIDER_GetValue(hItem);
453 ///////////////////////////////////////////////////
454 // LOW SHELF GAIN G
455 hItem = WM_GetDialogItem(pMsg->hWin, ID_SLIDER_3);
456 SLIDER_SetNumTicks(hItem, 25);
457 SLIDER_SetRange(hItem,-12,12);
458 SLIDER_SetValue(hItem, 0);
459 gainLS = SLIDER_GetValue(hItem);
460 gainLS*=-1;
461
462 // LOW SHELF FREQ F0
463
464 hItem = WM_GetDialogItem(pMsg->hWin, ID_SLIDER_5);
465 SLIDER_SetNumTicks(hItem, 100);
466 SLIDER_SetRange(hItem,50,1000);
467 SLIDER_SetValue(hItem, 225); // passo de 10 aproximadamente
468 frqLS = SLIDER_GetValue(hItem);
469 ///////////////////////////////////////////////////
470 // HIGH SHELF SHELF GAIN G
471 hItem = WM_GetDialogItem(pMsg->hWin, ID_SLIDER_4);
472 SLIDER_SetNumTicks(hItem, 25);
473 SLIDER_SetRange(hItem,-12,12);
474 SLIDER_SetValue(hItem, 0);
475 gainHS = SLIDER_GetValue(hItem);
476 gainHS*=-1;
477
478 // HIGH SHELF FREQ F0
479
480 hItem = WM_GetDialogItem(pMsg->hWin, ID_SLIDER_9);
481 SLIDER_SetNumTicks(hItem, 100);
482 SLIDER_SetRange(hItem,8000,18000); // passo de 100
483 SLIDER_SetValue(hItem, 13000);
484 frqHS = SLIDER_GetValue(hItem);
485 ///////////////////////////////////////////////////
486 /
487 sprintf(buf,"%d",aux); // exibir valores
488
489 ///////////////////////////////////////////////////

```

WindowDLG.c

```

490 // USER END
491 break;
492 case WM_NOTIFY_PARENT:
493     Id = WM_GetId(pMsg->hWinSrc);
494     NCode = pMsg->Data.v;
495     switch(Id) {
496     case ID_SLIDER_0: // Notifications sent by 'G_PK1'
497         switch(NCode) {
498         case WM_NOTIFICATION_CLICKED:
499             // USER START (Optionally insert code for reacting on notification message)
500             // USER END
501             break;
502         case WM_NOTIFICATION_RELEASED:
503             // USER START (Optionally insert code for reacting on notification message)
504             frq = frqPK1;
505             gain = gainPK1;
506             flag.bandPK1 = 1;
507             flag.band = 1;
508             // USER END
509             break;
510         case WM_NOTIFICATION_VALUE_CHANGED:
511             // USER START (Optionally insert code for reacting on notification message)
512             hItem = WM_GetDlgItem(pMsg->hWin, ID_SLIDER_0);
513                                     gainPK1 = SLIDER_GetValue(hItem);
514                                     gainPK1 *= -1;
515             hItem = WM_GetDlgItem(pMsg->hWin, ID_TEXT_10);
516                                     sprintf(buf,"%d",gainPK1);
517                                     TEXT_SetText(hItem, buf);
518             // USER END
519             break;
520             // USER START (Optionally insert additional code for further notification handling)
521             // USER END
522         }
523         break;
524     case ID_SLIDER_1: // Notifications sent by 'G_PK2'
525         switch(NCode) {
526         case WM_NOTIFICATION_CLICKED:
527             // USER START (Optionally insert code for reacting on notification message)
528             // USER END
529             break;
530         case WM_NOTIFICATION_RELEASED:
531             // USER START (Optionally insert code for reacting on notification message)
532             frq = frqPK2;
533             gain = gainPK2;
534             flag.bandPK2 = 1;
535             flag.band = 1;
536             // USER END
537             break;
538         case WM_NOTIFICATION_VALUE_CHANGED:
539             // USER START (Optionally insert code for reacting on notification message)
540             hItem = WM_GetDlgItem(pMsg->hWin, ID_SLIDER_1);
541             gainPK2 = SLIDER_GetValue(hItem);
542             gainPK2 *= -1;
543
544             hItem = WM_GetDlgItem(pMsg->hWin, ID_TEXT_11);
545                                     sprintf(buf,"%d",gainPK2);
546                                     TEXT_SetText(hItem, buf);
547             // USER END
548             break;
549             // USER START (Optionally insert additional code for further notification handling)
550             // USER END
551         }

```

WindowDLG.c

```

552     break;
553 case ID_SLIDER_2: // Notifications sent by 'G_PK3'
554     switch(NCode) {
555     case WM_NOTIFICATION_CLICKED:
556         // USER START (Optionally insert code for reacting on notification message)
557
558         // USER END
559         break;
560     case WM_NOTIFICATION_RELEASED:
561         // USER START (Optionally insert code for reacting on notification message)
562         frq = frqPK3;
563         gain = gainPK3;
564         flag.bandPK3 = 1;
565         flag.band = 1;
566
567         // USER END
568         break;
569     case WM_NOTIFICATION_VALUE_CHANGED:
570         // USER START (Optionally insert code for reacting on notification message)
571         hItem = WM_GetDlgItem(pMsg->hWin, ID_SLIDER_2);
572             gainPK3 = SLIDER_GetValue(hItem);
573             gainPK3 *= -1;
574         hItem = WM_GetDlgItem(pMsg->hWin, ID_TEXT_12);
575             sprintf(buf, "%d", gainPK3);
576             TEXT_SetText(hItem, buf);
577
578         // USER END
579         break;
580     // USER START (Optionally insert additional code for further notification handling)
581     // USER END
582     }
583     break;
584 case ID_SLIDER_3: // Notifications sent by 'lowShelf'
585     switch(NCode) {
586     case WM_NOTIFICATION_CLICKED:
587         // USER START (Optionally insert code for reacting on notification message)
588         // USER END
589         break;
590     case WM_NOTIFICATION_RELEASED:
591         // USER START (Optionally insert code for reacting on notification message)
592         frq = frqLS;
593         gain = gainLS;
594         flag.bandLS = 1;
595         flag.band = 1;
596         // USER END
597         break;
598     case WM_NOTIFICATION_VALUE_CHANGED:
599         // USER START (Optionally insert code for reacting on notification message)
600         hItem = WM_GetDlgItem(pMsg->hWin, ID_SLIDER_3);
601             gainLS = SLIDER_GetValue(hItem);
602             gainLS *= -1;
603
604         hItem = WM_GetDlgItem(pMsg->hWin, ID_TEXT_9);
605             sprintf(buf, "%d", gainLS);
606             TEXT_SetText(hItem, buf);
607
608             hItem = WM_GetDlgItem(pMsg->hWin, ID_GRAPH_0);
609
610             WM_Invalidate(hItem);
611
612             //    }
613     // USER END

```

WindowDLG.c

```

614         break;
615     // USER START (Optionally insert additional code for further notification handling)
616     // USER END
617 }
618 break;
619 case ID_SLIDER_4: // Notifications sent by 'HighShelf'
620     switch(NCode) {
621     case WM_NOTIFICATION_CLICKED:
622         // USER START (Optionally insert code for reacting on notification message)
623         // USER END
624         break;
625     case WM_NOTIFICATION_RELEASED:
626         // USER START (Optionally insert code for reacting on notification message)
627         frq = frqHS;
628         gain = gainHS;
629         flag.bandHS = 1;
630         flag.band = 1;
631         // USER END
632         break;
633     case WM_NOTIFICATION_VALUE_CHANGED:
634         // USER START (Optionally insert code for reacting on notification message)
635         hItem = WM_GetDlgItem(pMsg->hWin, ID_SLIDER_4);
636                                     gainHS = SLIDER_GetValue(hItem);
637                                     gainHS *= -1;
638         hItem = WM_GetDlgItem(pMsg->hWin, ID_TEXT_13);
639                                     sprintf(buf, "%d", gainHS);
640                                     TEXT_SetText(hItem, buf);
641         // USER END
642         break;
643     // USER START (Optionally insert additional code for further notification handling)
644     // USER END
645     }
646     break;
647 case ID_SLIDER_5: // Notifications sent by 'FRQ_LS'
648     switch(NCode) {
649     case WM_NOTIFICATION_CLICKED:
650         // USER START (Optionally insert code for reacting on notification message)
651         // USER END
652         break;
653     case WM_NOTIFICATION_RELEASED:
654         // USER START (Optionally insert code for reacting on notification message)
655         frq = frqLS;
656         frqPK = frq;
657         flag.bandLS = 1;
658         flag.band = 1;
659
660         //(Points[i].x, 270 - Points[i].y, 270);
661         // USER END
662         break;
663     case WM_NOTIFICATION_VALUE_CHANGED:
664         // USER START (Optionally insert code for reacting on notification message)
665         hItem = WM_GetDlgItem(pMsg->hWin, ID_SLIDER_5);
666                                     frqLS = SLIDER_GetValue(hItem);
667
668         hItem = WM_GetDlgItem(pMsg->hWin, ID_TEXT_0);
669                                     sprintf(buf, "FRQ = %d", frqLS);
670                                     TEXT_SetText(hItem, buf);
671
672         // USER END
673         break;
674     // USER START (Optionally insert additional code for further notification handling)
675     // USER END

```

WindowDLG.c

```

676     }
677     break;
678 case ID_SLIDER_6: // Notifications sent by 'FREQ_PK1'
679     switch(NCode) {
680     case WM_NOTIFICATION_CLICKED:
681         // USER START (Optionally insert code for reacting on notification message)
682         // USER END
683         break;
684     case WM_NOTIFICATION_RELEASED:
685         // USER START (Optionally insert code for reacting on notification message)
686         frq = frqPK1;
687         frqPK = frq;
688         flag.bandPK1 = 1;
689         flag.band = 1;
690
691         // USER END
692         break;
693     case WM_NOTIFICATION_VALUE_CHANGED:
694         // USER START (Optionally insert code for reacting on notification message)
695         hItem = WM_GetDlgItem(pMsg->hWin, ID_SLIDER_6);
696                 frqPK1 = SLIDER_GetValue(hItem);
697
698         hItem = WM_GetDlgItem(pMsg->hWin, ID_TEXT_1);
699                 sprintf(buf, "FRQ = %d", frqPK1);
700                 TEXT_SetText(hItem, buf);
701         // USER END
702         break;
703     // USER START (Optionally insert additional code for further notification handling)
704     // USER END
705     }
706     break;
707 case ID_SLIDER_7: // Notifications sent by 'FREQ_PK2'
708     switch(NCode) {
709     case WM_NOTIFICATION_CLICKED:
710         // USER START (Optionally insert code for reacting on notification message)
711         // USER END
712         break;
713     case WM_NOTIFICATION_RELEASED:
714         // USER START (Optionally insert code for reacting on notification message)
715         frq = frqPK2;
716         frqPK = frq;
717         flag.bandPK2 = 1;
718         flag.band = 1;
719         // USER END
720         break;
721     case WM_NOTIFICATION_VALUE_CHANGED:
722         // USER START (Optionally insert code for reacting on notification message)
723         hItem = WM_GetDlgItem(pMsg->hWin, ID_SLIDER_7);
724                 frqPK2 = SLIDER_GetValue(hItem);
725
726         hItem = WM_GetDlgItem(pMsg->hWin, ID_TEXT_2);
727                 sprintf(buf, "FRQ = %d", frqPK2);
728                 TEXT_SetText(hItem, buf);
729         // USER END
730         break;
731     // USER START (Optionally insert additional code for further notification handling)
732     // USER END
733     }
734     break;
735 case ID_SLIDER_8: // Notifications sent by 'FREQ_PK3'
736     switch(NCode) {
737     case WM_NOTIFICATION_CLICKED:

```

WindowDLG.c

```

738     // USER START (Optionally insert code for reacting on notification message)
739
740     // USER END
741     break;
742 case WM_NOTIFICATION_RELEASED:
743     // USER START (Optionally insert code for reacting on notification message)
744     frq = frqPK3;
745     frqPK = frq;
746     flag.bandPK3 = 1;
747     flag.band = 1;
748
749     // USER END
750     break;
751 case WM_NOTIFICATION_VALUE_CHANGED:
752     // USER START (Optionally insert code for reacting on notification message)
753     hItem = WM_GetDlgItem(pMsg->hWin, ID_SLIDER_8);
754     frqPK3 = SLIDER_GetValue(hItem);
755
756     hItem = WM_GetDlgItem(pMsg->hWin, ID_TEXT_3);
757     sprintf(buf, "FRQ = %d", frqPK3);
758     TEXT_SetText(hItem, buf);
759
760     // USER END
761     break;
762 // USER START (Optionally insert additional code for further notification handling)
763 // USER END
764 }
765 break;
766 case ID_SLIDER_9: // Notifications sent by 'FREQ_HS'
767     switch(NCode) {
768     case WM_NOTIFICATION_CLICKED:
769         // USER START (Optionally insert code for reacting on notification message)
770         // USER END
771         break;
772     case WM_NOTIFICATION_RELEASED:
773         // USER START (Optionally insert code for reacting on notification message)
774         frq = frqHS;
775         frqPK = frq;
776         flag.bandHS = 1;
777         flag.band = 1;
778         // USER END
779         break;
780     case WM_NOTIFICATION_VALUE_CHANGED:
781         // USER START (Optionally insert code for reacting on notification message)
782         hItem = WM_GetDlgItem(pMsg->hWin, ID_SLIDER_9);
783         frqHS = SLIDER_GetValue(hItem);
784
785         hItem = WM_GetDlgItem(pMsg->hWin, ID_TEXT_4);
786         sprintf(buf, "FRQ = %d", frqHS);
787         TEXT_SetText(hItem, buf);
788
789         // USER END
790         break;
791     // USER START (Optionally insert additional code for further notification handling)
792     // USER END
793     }
794 case ID_SLIDER_10: // Notifications sent by 'Fact_Q'
795     switch(NCode) {
796     case WM_NOTIFICATION_CLICKED:
797         // USER START (Optionally insert code for reacting on notification message)
798         // USER END
799         break;
800     case WM_NOTIFICATION_RELEASED:

```

WindowDLG.c

```

800 // USER START (Optionally insert code for reacting on notification message)
801 // flag.bandPK2 = 1;
802
803 // USER END
804 break;
805 case WM_NOTIFICATION_VALUE_CHANGED:
806 // USER START (Optionally insert code for reacting on notification message)
807 hItem = WM_GetDlgItem(pMsg->hWin, ID_SLIDER_10);
808 F_Qtext = SLIDER_GetValue(hItem);
809
810 hItem = WM_GetDlgItem(pMsg->hWin, ID_TEXT_7);
811 sprintf(buf, "%d", F_Qtext);
812 TEXT_SetText(hItem, buf);
813 Fact_Q = (float) F_Qtext;
814 flag.band = 1;
815
816 // USER END
817 break;
818 // USER START (Optionally insert additional code for further notification handling)
819 // USER END
820 }
821 break;
822 case ID_SLIDER_11: // Notifications sent by 'OUT'
823 switch(NCode) {
824 case WM_NOTIFICATION_CLICKED:
825 // USER START (Optionally insert code for reacting on notification message)
826 // USER END
827 break;
828 case WM_NOTIFICATION_RELEASED:
829 // USER START (Optionally insert code for reacting on notification message)
830
831 // USER END
832 break;
833 case WM_NOTIFICATION_VALUE_CHANGED:
834 // USER START (Optionally insert code for reacting on notification message)
835 hItem = WM_GetDlgItem(pMsg->hWin, ID_SLIDER_11);
836 aux = SLIDER_GetValue(hItem);
837 aux *= -1;
838 vol = aux;
839 BSP_AUDIO_OUT_SetVolume(aux);
840 hItem = WM_GetDlgItem(pMsg->hWin, ID_TEXT_8);
841 sprintf(buf, "%d", aux);
842 TEXT_SetText(hItem, buf);
843
844 // USER END
845 break;
846 // USER START (Optionally insert additional code for further notification handling)
847 // USER END
848 }
849 break;
850 case ID_BUTTON_0: // Notifications sent by 'LIGA'
851 switch(NCode) {
852 case WM_NOTIFICATION_CLICKED:
853 // USER START (Optionally insert code for reacting on notification message)
854 flag.flagFilt = 1 ;
855 // USER END
856 break;
857 case WM_NOTIFICATION_RELEASED:
858 // USER START (Optionally insert code for reacting on notification message)
859 // USER END
860 break;
861 // USER START (Optionally insert additional code for further notification handling)
862 // USER END

```

WindowDLG.c

```

862     }
863     break;
864 case ID_BUTTON_1: // Notifications sent by 'DESL'
865     switch(NCode) {
866     case WM_NOTIFICATION_CLICKED:
867         // USER START (Optionally insert code for reacting on notification message)
868         flag.flagFilt = 0;
869         // USER END
870         break;
871     case WM_NOTIFICATION_RELEASED:
872         // USER START (Optionally insert code for reacting on notification message)
873         // USER END
874         break;
875     // USER START (Optionally insert additional code for further notification handling)
876     // USER END
877     }
878     break;
879
880 case ID_BUTTON_3: // Notifications sent by 'p1'
881     switch(NCode) {
882     case WM_NOTIFICATION_CLICKED:
883         // USER START (Optionally insert code for reacting on notification message)
884
885         frqPK = frqPK1;
886
887         Flt = band1;
888
889         // USER END
890         break;
891     case WM_NOTIFICATION_RELEASED:
892         // USER START (Optionally insert code for reacting on notification message)
893         // USER END
894         break;
895     // USER START (Optionally insert additional code for further notification handling)
896     // USER END
897     }
898     break;
899 case ID_BUTTON_4: // Notifications sent by 'p2'
900     switch(NCode) {
901     case WM_NOTIFICATION_CLICKED:
902         // USER START (Optionally insert code for reacting on notification message)
903
904         frqPK = frqPK2;
905         Flt = band2;
906
907         // USER END
908         break;
909     case WM_NOTIFICATION_RELEASED:
910         // USER START (Optionally insert code for reacting on notification message)
911         // USER END
912         break;
913     // USER START (Optionally insert additional code for further notification handling)
914     // USER END
915     }
916     break;
917 case ID_BUTTON_5: // Notifications sent by 'p3'
918     switch(NCode) {
919     case WM_NOTIFICATION_CLICKED:
920         // USER START (Optionally insert code for reacting on notification message)
921
922         frqPK = frqPK3;
923         Flt = band3;

```

WindowDLG.c

```

924
925     // USER END
926     break;
927     case WM_NOTIFICATION_RELEASED:
928         // USER START (Optionally insert code for reacting on notification message)
929         // USER END
930         break;
931         // USER START (Optionally insert additional code for further notification handling)
932         // USER END
933     }
934     break;
935
936 }
937 break;
938 // USER START (Optionally insert additional message handling)
939 // USER END
940 default:
941     WM_DefaultProc(pMsg);
942     break;
943 }
944 }
945
946 /*****
947 *
948 *     Public code
949 *
950 *****/
951 */
952 /*****
953 *
954 *     CreateWindow
955 */
956 WM_HWIN CreateWindow(void);
957 WM_HWIN CreateWindow(void) {
958     WM_HWIN hWin;
959
960     hWin = GUI_CreateDialogBox(_aDialogCreate, GUI_COUNTOF(_aDialogCreate), _cbDialog,
961         WM_HBKWIN, 0, 0);
962     return hWin;
963 }
964
965 // USER START (Optionally insert additional public code)
966
967 static void _UserDraw(WM_HWIN hWin, int Stage) {
968
969     switch (Stage) {
970     case GRAPH_DRAW_FIRST:
971         break;
972     case GRAPH_DRAW_LAST:
973         GUI_SetColor(GUI_GREEN);
974
975         for (int i = 1; i < FFT_SIZE; i++) {
976             GUI_DrawRect(log(Points[i].x)*70, 79 - Points[i].y, log(Points[i].x)*70+1,
977                 79);
978             if((log(Points[i].x) > log(0.010*(frqPK - frqPK/(2*Fact_Q))))&& (log(Points
979 [i].x) < log(0.010*(frqPK + frqPK/(2*Fact_Q)))) // 4
980             {
981
982                 if(gain >= 0)

```

WindowDLG.c

```

983         {
984             Gmenos = 0;
985             Gmais = gain;
986         }
987         else{
988             Gmais =0;
989             Gmenos = gain;
990         }
991         switch(Flt){
992
993             case bandLS:
994
995
996                 GUI_DrawGradientV(log(Points[i].x)*70, 42 - 3*Gmais , log(Points
997 [i].x)*70, 42 - 3*Gmenos, GUI_BLUE,GUI_RED);
998                 break;
999             case band1:
1000                 GUI_DrawGradientV(log(Points[i].x)*70, 42 - 3*Gmais , log(Points
1001 [i].x)*70, 42 - 3*Gmenos, GUI_YELLOW,GUI_RED);
1002                 flag.bandPK1 =1;
1003                 break;
1004             case band2:
1005                 GUI_DrawGradientV(log(Points[i].x)*70, 42 - 3*Gmais , log(Points
1006 [i].x)*70, 42 - 3*Gmenos, GUI_RED,GUI_RED);
1007                 flag.bandPK2 =1;
1008                 break;
1009             case band3:
1010                 GUI_DrawGradientV(log(Points[i].x)*70, 42 - 3*Gmais , log(Points
1011 [i].x)*70, 42 - 3*Gmenos, GUI_WHITE,GUI_RED);
1012                 flag.bandPK3 =1;
1013                 break;
1014             case bandHS:
1015                 GUI_DrawGradientV(log(Points[i].x)*70, 42 - 3*Gmais , log(Points
1016 [i].x)*70, 42 - 3*Gmenos, GUI_BROWN,GUI_RED);
1017                 break;
1018         }
1019
1020
1021     }
1022     else
1023     {
1024         GUI_SetColor(GUI_GREEN);
1025     }
1026
1027 }
1028 break;
1029 }
1030 }
1031
1032 static void _cbBk(WM_MESSAGE * pMsg) {
1033     switch (pMsg->MsgId) {
1034         case WM_PAINT:
1035             GUI_SetBkColor(GUI_BLACK);
1036             GUI_Clear();
1037             break;
1038         default:
1039             WM_DefaultProc(pMsg);

```

WindowDLG.c

```

1040         break;
1041     }
1042 }
1043
1044
1045
1046 //////////////////////////////////////////////////FFTGRAPHIC//////////////////////////////////////
1047 void FFT_Data2(void) {
1048     // WM_HWIN hWin;
1049     static arm_rfft_instance_q15 fft_instance;
1050     static q15_t output[FFT_SIZE * 2]; //has to be twice FFT size
1051     unsigned char input[FFT_BUFFER/2];
1052     for (int i = 0; i < FFT_BUFFER/2; i++) {
1053         dmaBuffer[i] += 35536;
1054         dmaBuffer[i]/=256;
1055         input[i]=dmaBuffer[i];
1056     }
1057     if (arm_rfft_init_q15(&fft_instance, 256, 0, 1) != ARM_MATH_SUCCESS) {
1058         //error
1059     }
1060     arm_rfft_q15(&fft_instance, (q15_t*) input, output);
1061     arm_abs_q15(output, output, FFT_SIZE);
1062
1063     for (int i = 0; i < FFT_SIZE; i++) {
1064
1065         Points[i].x = i;
1066         Points[i].y =pow(2 ,(log(output[i])));
1067         //Points[i].y = log(output[i]*4)*10;
1068
1069     }
1070
1071
1072     hItem = WM_GetFirstChild(WM_HBKWIN);
1073     hItem = WM_GetDialogItem(hItem, ID_GRAPH_0);
1074     GRAPH_SetUserDraw(hItem, _UserDraw); // Enable custom draw function
1075     WM_InvalidateWindow(hWin);
1076
1077 }
1078 /*****
1079 *
1080
1081 // USER END
1082
1083 /***** End of file *****/
1084

```