Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Unidade Acadêmica de Engenharia Elétrica

RELATÓRIO DE ESTÁGIO

Título: **Desenvolvimento de Aplicações para Plataforma de Tratamento Digital de Sinais DSP**

Aluno: **Miguel Augusto de Souza Falcão**
Matrícula: **20411225**
Orientador: **Raimundo Carlos Silvério Freire**

Campina Grande
Fevereiro de 2010

RELATÓRIO DE ESTÁGIO

Desenvolvimento de Aplicações para Plataforma de Tratamento Digital de Sinais DSP

Empresa: *APDISAR - Association Pour le Développement de l'ESISAR*

Endereço: 54, Rue Barthélemy de Laffemas
26902 - Valence CEDEX 9 - France

Telefone: 04 75 75 94 00 **E-mail:** Chantal.Robach@grenoble-inp.fr
Fax: 04 75 43 56 42 **Web site:** http://esisar.grenoble-inp.fr

Aluno: Miguel Augusto de Souza Falcão
Matrícula: 20411225

Data do estágio	**09/02/2008 – 03/07/2008**
Orientador na UFCG	**Raimundo Freire**
Orientador na ESISAR	**Saad Boukhili**
Orientador na APDISAR	**Yvan Duroc**

Agradecimentos

Este estágio de fim de curso é resultado do programa bilateral de mobilidade acadêmica BRAFITEC financiado pelo governo brasileiro através do MEC e CAPES e pelo governo francês através do MAE, MENESR e DREIC, onde a coordenação francesa é feita pelo CDEFI.

Agradeço a todos os organismos responsáveis pelo programa BRAFITEC, as minhas instituições de ensino UFCG e ESISAR, assim como aos professores que a mim deram um voto de confiança e me auxiliaram no desenvolvimento do projeto de estágio: Yvan Duroc, Saad Boukhili, Raimundo Freire e Glauco Fontgalland.

No âmbito pessoal, agradeço a minha família que me apoiou todos os dias e também aos meus companheiros de estágio, pessoas estas que partilharam comigo todas as dificuldades e vitórias: Diego Buriti e Gilles Fritz.

"If I have seen a little further it is by standing on the shoulders of Giants." *– Isaac Newton*

Índice

Agradecimentos 3
Lista de Figuras 5
Lista de Listagens 6
Lista de Tabelas 7
Acrônimos 8
Apresentação do Laboratório 9
Especificações do Projeto 9
1 - Introdução 11
2 - Ferramentas de Desenvolvimento 12
2.1 - DSP TMS 320VC5416 12
2.2 - Kit de Inicialização 5416 DSP 12
2.3 - Code Composer Studio 13
3 - Desenvolvimento de Aplicação de Base 15
3.1 - Estudo do Código Fonte 15
3.1.1 - Função "main()" 15
3.1.2 - Definição das Variáveis Globais Ligadas ao Tratamento 16
3.1.3 - Controle de Tarefas do Programa 16
3.1.4 - Tratamento do Sinal 17
3.2 - Execução e Testes do Programa "audio.c" 19
3.3 - Conversão Estéreo/Mono 21
3.3.1 - Transmissão Estéreo ao Nível do Programa "audio.c" 22
3.3.2 - Transformação Estéreo/Mono 23
3.3.3 - Codificação da Conversão Estéreo/ Mono 24
3.4 – Efeito de Eco 24
3.4.1 - Compreensão Teórica do Efeito de Eco 25
3.4.2 - Implementação do Efeito de Eco 25
3.4.3 - Execução e Testes 28
4 - Filtragem Digital 30
4.1 - Convolução no Tempo Discreto e Resposta Frequencial 30
4.2 - Filtro à Resposta Impulsional Finita 31
4.3 - Filtro à Resposta Impulsional Infinita 31
4.3.1 - Forma Discreta Tipo I 32
4.3.2 - Forma Discreta Tipo II 33
4.3.3 - Forma Discreta Tipo II em Cascata 34
4.4 - Obtenção dos Coeficientes de Filtros Digitais 35
4.5 - Implementação de um Filtro FIR 35
4.5.1 - Execução e Testes do Filtro FIR 37
5 - Transmissão Digital do Sinal 42
5.1 - Modulação 42
5.2 - Modulação Analógica e Digital 44
5.2.1 - Modulação Analógica 44
5.2.2 - Modulação Digital 45
5.2.2.1 - ASK – Modulação por Chaveamento de Amplitude 45
5.2.2.2 - FSK – Modulação por Chaveamento de Frequência 47
5.2.2.3 - PSK – Modulação por Chaveamento de Fase 48
6 - Considerações Finais 51
7 - Referências Bibliográficas 52
8 - Estimação Financeira 53
9 - Resumo e Abstract 54

Lista de Figuras

Figura 2.1: Kit de Inicialização 5416 DSP 10
Figura 2.2: IDE Code Composer 11
Figura 3.1: Fluxograma representativo do programa ***"audio.c"*** 15
Figura 3.2: Fluxo de dados do programa ***"audio.c"*** 15
Figura 3.3: Configuração para os testes do programa ***"audio.c"*** 16
Figura 3.4: Parâmetros dos gráficos de entrada e saída do DSK 17
Figura 3.5: Entrada e saída do DSP utilizando o arquivo de áudio ***"sinus1760Hz"*** 17
Figura 3.6: Parâmetros do gráfico de saída do DSK no domínio frequencial 18
Figura 3.7: Representação no domínio frequencial da saída do DSP utilizando o arquivo de áudio ***"sinus1760Hz"*** 18
Figura 3.8: Exemplificação da estereofonia 18
Figura 3.9: Fluxo de dados das vias esquerda e direita de um sistema de áudio estéreo 19
Figura 3.10: Saída do fluxo de dados pelas vias esquerda e direita de um sistema de áudio estéreo 20
Figura 3.11: Tratamento estéreo/mono 20
Figura 3.12: Seleção da saída de áudio: mono ou estéreo 21
Figura 3.13: Abstração do buffer circular, onde sua última posição é conectada à primeira 22
Figura 3.14: Parâmetros para os gráficos de entrada e saída do efeito de eco 26
Figura 3.15: Entrada e saída do efeito de eco cujo sinal de entrada é um seno de 1760 Hz para um ***"D=4800"*** e ***"g=0.8"*** 26
Figura 3.16: Entrada e saída do efeito de eco cujo sinal de entrada é uma onda quadrada de 440 Hz para um ***"D=4800"*** e ***"g=0.8"*** 26
Figura 4.1: Forma direta para implementação do filtro FIR 28
Figura 4.2: Forma direta tipo I para implementação do filtro IIR 29
Figura 4.3: Forma direta tipo II para a implementação do filtro IIR 30
Figura 4.4: Forma cascata para implementação do filtro IIR 31
Figura 4.5: Filtro IIR de quarta ordem fatorizado em seções de segunda ordem na forma direta tipo II 31
Figura 4.6: Parâmetros para o gráfico ***"Entrada/Saída no domínio temporal"*** 35
Figura 4.7: Parâmetros para o gráfico ***"Entrada/Saída no domínio frquencial"*** 36
Figura 4.8: Parâmetros para o gráfico ***"Saída no domínio frequencial"*** 36
Figura 4.9: Entrada/Saída no domínio temporal para um sinal composto por três senos (1 kHz, 8 kHz e 10 kHz) 37
Figura 4.10: Entrada/Saída no domínio frequencial para um sinal composto por três senos (1 kHz, 8 kHz e 10 kHz) 37
Figura 4.11: Resposta impusional obtida a partir do SPTool para o filtro passa-baixas concebido 38
Figura 4.12: Resposta impusional para o filtro passa-baixas implementado no DSP 38
Figura 5.1: Exemplo de modulação onde ***"m(t) = sin(w_m.t)"***, ***"p(t) = sin(w_c.t)"*** e ***"w_c = 25.w_m"*** 39
Figura 5.2: Análise frequencial do exemplo de modulação 40
Figura 5.3: Exemplo de modulação angular para um sinal útil de dois níveis 42
Figura 5.4: Modulação ASK 43
Figura 5.5: Modulação ASK com quatro níveis de saída diferentes (4-ASK) 44
Figura 5.6: Modulação FSK 45
Figura 5.7: Modulação PSK 45

Lista de Listagens

Listagem 3.1: Chamada às bibliotecas e definições da função ***"main()"*** do programa ***"audio.c"*** 12
Listagem 3.2: Declaração das variáveis globais que são utilizadas para o tratamento do sinal 13
Listagem 3.3: Bloco de tratamento do sinal 14
Listagem 3.4: Código para a conversão estéreo/mono 21
Listagem 3.5: Seleção do tratamento estéreo/mono escolhendo ***"mode"*** 1 21
Listagem 3.6: Variáveis necessárias à implementação do efeito de eco 22
Listagem 3.7: Implementação do efeito de eco 24
Listagem 3.8: Seleção do efeito de eco: o valor da variável ***"mode"*** é colocado em 2 25
Listagem 4.1: Arquivo ***"coef_fir_pb_5kHz.h"*** 33
Listagem 4.2: Mudanças de ***"fir.c"*** em relação à ***"audio.c"*** 33
Listagem 4.3: Algoritmo para a filtragem FIR 34
Listagem 4.4: Variáveis utilizadas para selecionar o modo de entrada do programa e a amplitude do impulso e do degrau 34
Listagem 4.5: Seleção da entrada para o filtro FIR 35

Lista de Tabelas

Tabela 3.1: Diferença entre equações teóricas e práticas do efeito de eco 24
Tabela 3.2: Processo de obtenção teórica da saída ***"y(n)"*** 25
Tabela 3.3: Processo para obter a saída ***"outputData[i]"*** ao nível do programa 25
Tabela 4.1: Características do filtro FIR estudado 32
Tabela 5.1: Comparação entre modulação analógica e digital 41
Tabela 5.2: Relação entre amplitude/fase e bits para uma modulação ASK a quatro níveis de saída diferentes (4-ASK) 43
Tabela I: Descrição dos custos do estágio de fim de curso 47

Acrônimos

ABU	*Automatic Buffering Unit*
AM	*Amplitude Modulation*
A/N	*Analogique / Numérique*
APDISAR	*Association Pour le Développement de l'ESISAR*
ARM	*Acorn RISC Machine*
ASK	*Amplitude Shift Keying*
BRAFITEC	*Programme d'échange BRésil France TEChnologie*
BSP	*Buffered Serial Port*
CAPES	*Coordenação de Aperfeiçoamento de PEssoal de nivel Supérior*
CCS	*Code Composer Studio*
CDEFI	*Conférence des Directeurs des Ecoles françaises d'Ingénieurs*
DMA	*Direct memory access*
DREIC	*Direction des Relations Européennes et internationales et de la Coopération*
DSK	*DSP Starter Kit*
DSP	*Digital Signal Processor*
EMIF	*External Memory Interface*
ESISAR	*École Supérieure d'Ingénieurs en Systèmes Industriels Avancés Rhône-Alpes*
FIR	*Finite Impulse Response*
FM	*Frequency Modulation*
FPGA	*Field-Programmable Gate Array*
FSK	*Frequency Shift Keying*
IDE	*Integrated Development Environment*
IIR	*Infinite Impulse Response*
JTAG	*Joint Test Action Group*
LCIS	*Laboratoire de Conception et d'Intégration des Systèmes*
McBSP	*Multi-Channel BSP*
MAE	*Ministère de les Affères Etrangers*
MEC	*Ministério da Educação*
MENESR	*Ministère de l'Education et l'Enseignement Supérieur et de la Recherche*
MODEM	*MOdulation / DEModulation*
N/A	*Numérique / Analogique*
PM	*Phase Modulation*
PSK	*Phase Shift Keying*
RTDX	*Real-Time Data Exchange*
SACCO	*Systèmes Autonomes Complexes et Communicants*
SPTool	*Signal Processing Tool*
SRAM	*Static Random Access Memory*

Apresentação do Laboratório

1. Plataforma SACCO para Sistemas Autônomos Complexos e Comunicantes

No âmbito geral dos "Sistemas embarcados comunicantes" da ESISAR (*École Supérieure d'Ingénieurs en Systèmes Industriels Avancés Rhône-Alpes*), a plataforma SACCO (*Systèmes Autonomes Complexes et Communicants*) é atualmente organizada em torno de quatro pólos de competência:

- Pólo processador ARM para sistemas embarcados;
- Pólo DSP para tratamento de sinais e comando de sistemas;
- Pólo circuitos lógicos programáveis (FPGA);
- Pólo automação industrial.

Esta plataforma possibilita oferecer uma resposta global aos problemas relativos aos sistemas compostos por unidades autônomas de controle/comando que realizam a aquisição de grandezas físicas (sensores), tratam estas informações (unidade de tratamento) e agem sobre o ambiente (atuadores), sejam eles no escopo da formação educacional (inicial e contínua), nas ações de transferência de tecnologia ou das atividades de pesquisas realizadas no laboratório LCIS.

Posicionamento e objetivos:

- Aumentar a prestação de formação continuada e de acompanhamento das empresas dentro do desenvolvimento de seus produtos;
- Captalizar os conhecimentos e compartilhar as competências adquiridas dentro dos projetos industriais do ESISAR;
- Formar os estudantes através de uma "abordagem sistema" completa;
- Melhorar a ligação entre pesquisa e formação. [1]

2. APDISAR

A APDISAR (*Association Pour le Développement de l'ESISAR*) é uma associação de professores e funcionários da ESISAR que buscam promover a escola dentro de empresas. Com esse objetivo, um escritório de estudo faz a prestação de serviços às empresas da região, sendo os estudos e as implementações nele realizados confiadas aos membros permanentes da escola. [2]

Especificações do Projeto

Este estágio se insere dentro do pólo DSP para o tratamento de sinal da plataforma SACCO. O objetivo do projeto é implementar aplicações com processadores de sinais.

A primeira parte do projeto é a adaptação à plataforma de desenvolvimento (kit de inicialização TMS320VC5416 e *software* Code Composer Studio) e implementação de aplicações simples (como modulação senoidal, efeito de eco, efeito de reverberação). Na segunda parte do projeto, filtros genéricos FIR e IIR (*Finite*/*Infinite Impulse Response*) são abordados. A última parte foca o desenvolvimento de um modem PSK (*Phase Shift Keying*).

Para cada um das aplicações desenvolvidas, um relatório de acompanhamento é redigido (em inglês ou francês). Ele deverá incluir a análise teórica, a descrição dos recursos utilizados, programas implementados (C ou assembler), simulações realizadas e exemplos.

Suporte	Fotocópia dos Trabalhos Práticos, CD-ROM e livros sobre C5416
Softwares à disposição	MATLAB, Code Composer Studio
Hardware à disposição	Stater kit da Texas Instrument, o DSK5416
Domínios do Estágio	Tratamento do sinal, Processadores de sinal, Comunicação

1. Introdução

Os processadores digitais de sinais são usados em uma larga gama de áreas, tais como: telecomunicações, reconhecimento vocal, aplicações médicas, automação, instrumentação, etc. Devido a essa grande quantidade de aplicações, estes dispositivos angariaram um posto significativo nos cursos universitários, sendo eles uma forma de baixo custo de introduzir o processamento digital de sinais em tempo real para os estudantes de engenharia e informática.

Este tema de estágio, *"Desenvolvimento de Aplicações para Plataforma de Tratamento Digital de Sinais DSP",* tem como objetivo criar uma plataforma para o estudo teórico e prático utilizado para a formação inicial dos alunos, educação continuada e como ferramenta de apresentação da plataforma de tecnologia SACCO da ESISAR. Os desenvolvimentos são, desta forma, de aplicações de processamento de sinal digital utilizando a família de DSPs TMS320 fabricado pela Texas Instruments. Exemplos práticos fundamentais à construção de uma base sólida nos estudos da DSP são feitas em conjunto com uma discussão teórica sobre o tema abordado.

Os *hardwares* e *software*s utilizados para o desenvolvimento deste tema são discutidos no segundo capítulo, apresentando o DSP TMS320VC5416, o kit de inicialização 5416 DSK e o Code Composer Studio (CCS).

Um exemplo de aplicação é desenvolvido no capítulo três. Este exemplo utiliza o codec de áudio do DSK 5416 para permitir a captura de amostras de áudio na entrada da placa, em seguida seu tratamento e a reconstrução em forma analógica na saída de áudio. Este exemplo é genérico, ele nos possibilita escrever algoritmos de processamento desejados e em seguida o inserir no código exemplos de forma modulada. Algumas aplicações clássicas no aprendizado do DSP são então apresentadas: conversão estéreo/mono, efeito de eco e efeito de reverberação. O capítulo quatro também usa esse exemplo para desenvolver um dos temas mais importantes do processamento de sinal, a filtragem digital, onde filtros de resposta finita (FIR) e infinita (IIR) ao impulso são abordados.

O capítulo cinco trata da modulação e demodulação de sinais (modulação em amplitude, freqüência e fase) e faz uma introdução aos princípios gerais de comunicações. Estes temas servem de base para o estudo e a concepção de um modem PSK (*Phase Shift Keying*).

Uma versão mais detalhada, em lingua francesa, deste documento está disponível [3]. Ele tem a forma de tutorial e pode ser usado como texto auxiliar para a aprendizagem de processamento digital de sinais com DSPs.

2. Ferramentas de Desenvolvimento

Este capítulo apresenta as ferramentas utilizadas ao longo deste documento para o estudo e concepção de sistemas para processamento digital de sinais. Tais ferramentas incluem o DSP TMS320VC5416, o DSP Starter Kit 5416 (DSK 5416) e Code Composer Studio (CCS) com o seu ambiente de desenvolvimento integrado (IDE).

2.1 DSP TMS320VC5416

O TMS320VC5416 (ou simplesmente DSP 5416) é um processador digital de sinais a ponto fixo de 160 MHz, produzido pela empresa Texas Instruments. A arquitetura em que este DSP se baseia é a "Harvard Avançada e Modificada" com três barramentos para a memória de dados e um barramento para a memória de programa. Os principais componentes deste DSP são uma unidade lógica aritmética (ALU) de 40 *bits*, dois acumuladores de 40 *bits*, uma unidade de multiplicação e adição, um registrador de deslocamento, oito registradores auxiliares e uma pilha implementada via *software*. Outra característica importante é o seu conjunto de comandos que contém instruções monocíclicas especializadas concebidas especificamente para o processamento de sinal. [4]

A família deste DSP, a TMS320C54x, é hoje uma das mais utilizadas entre todas as famílias de DSP. O seu principal domínio de aplicação é a comunicação através de telefones celulares. O grande sucesso da TMS320C54x é atribuído ao bom desempenho em consumo, velocidade e preço, conseguidos em grande parte graças ao conjunto de instruções especializadas supracitadas.

As regiões de memória separadas permitem o acesso simultâneo às instruções de dados e de programa, possibilitando um alto grau de paralelismo. Em um único ciclo de operação, duas operações de leitura e escrita podem ser realizadas. As instruções que utilizam armazenamento paralelo e as instruções para aplicações específicas podem utilizar plenamente esta arquitetura. Além disso, os dados podem ser transferidos entre a memória de dados e de programa. Este paralelismo permite o suporte para um poderoso conjunto de operações aritméticas, lógicas e manipulação de *bits*, onde todas essas operações podem ser executadas em um ciclo de máquina.

O DSP 5416 dispõe igualmente de mecanismos de controle para lidar com interrupções, para a repetição de operações e para as chamadas de funções. Outras características importantes são as 128 k-palavras de memória interna rápida, três portas seriais multi-canais com *buffer* (McBSP), um registo temporizador na própria placa e um controlador de acesso direto à memória (DMA) com seis canais.

2.2 Kit de Inicialização 5416 DSP

O kit de inicialização DSP 5416 (ou 5416 DSK) é uma plataforma de baixo custo que permite aos usuários avaliarem e desenvolverem aplicações para os DSPs da família C54X. Ele é um conjunto poderoso que fornece todo o *hardware* e *software* necessários para o desenvolvimento de várias aplicações de processamento de sinal. [5]

Os principais componentes do kit são um DSP TMS320VC5416, um codec de áudio estéreo PCM3002, mémoria Flash e SRAM no chip, além de quatro interruptores DIP e quatro LEDs, ambos acessíveis aos usuários como forma de entrada e saída.

A ferramenta de desenvolvimento Code Composer Studio (CCS) é incluída no DSK 5416, o que fornece ao usuário um ambiente integrado de desenvolvimento para a programação em C ou assembly. O software CCS se comunica com o DSP usando um emulador JTAG (*Joint Test Action Group*) presente no chip, utilizando uma interface USB. A placa com seus componentes é ilustrada na Figura 2.1.

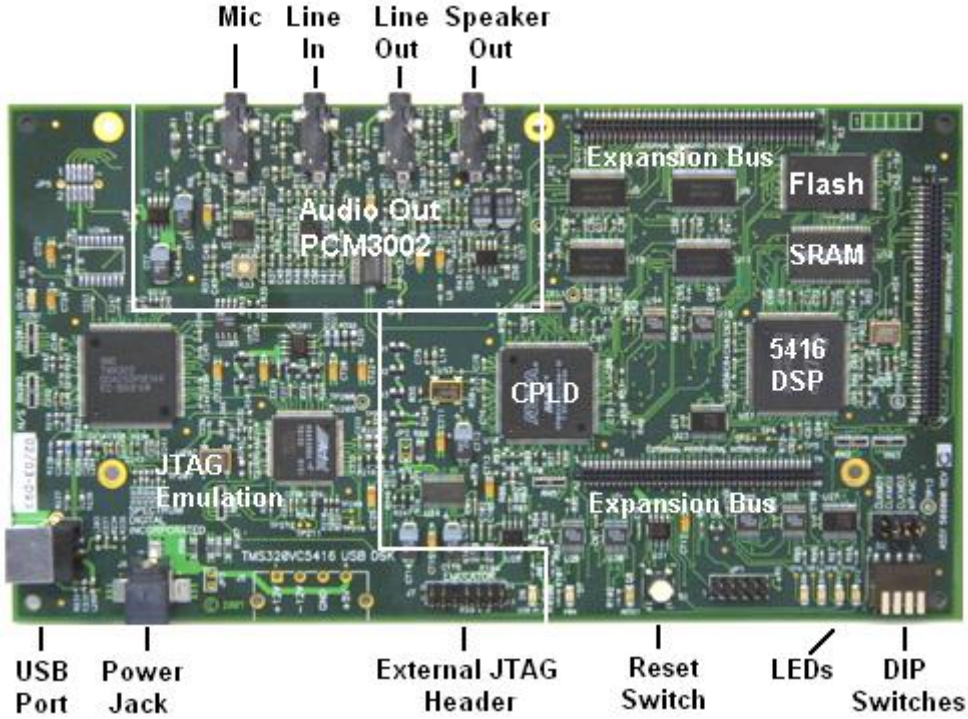

Figura 2.1: Kit de Inicialização 5416 DSP. [5]

O DSP TMS320VC5416 é o principal componente do sistema, ele tem uma quantidade significativa de memória interna, o que permite que aplicações típicas de tratamento de sinais possam ter todo o código fonte e os dados armazenados no chip. Para o caso onde a aplicação demanda mais memória, há a possibilidade de utilizar interfaces de memória externa (EMIF). O DSK inclui uma memória flash não-volátil para armazenar o código de partida e uma SRAM externa, que serve de exemplo de inclusão de memória externa no sistema. O DSK implementa a lógica necessária para gerenciar todos os componentes da placa, utilizando para tanto oito registradores lógicos programáveis que podem ser usado para definir vários parâmetros da placa.

Sendo os DSPs muitas vezes utilizados na implementação de aplicativos para processamento de áudio, o DSK possui um chip codec de áudio, o PCM3002. Este codec converte em digital o sinal de entrada analógico. Em seguida, essas amostras são processadas pelo algoritmo utilizado no DSP, reconvertidas em forma analógica e enviado para a saída analógica do DSK. O DSK usa portas seriais bufferizadas para receber os dados utilizados pelo codec e para enviar à saída após o tratamento.

No mais, o DSK tem 4 diodos eletroluminescents (LEDs) e 4 micro interruptores para permitir aos usuários interagir com os programas, utilizando os LEDs como saídas e as chaves como entradas.

Como fonte de energia, o DSK usa uma tensão externa de +5 V. Internamente, essa entrada de +5 V é convertida em 1,6 V e 3,3 V usando um regulador de voltagem. A tensão de 1,6 V é utilizada na alimentação do DSP, enquanto que uma das tensões de 3,3 V é usada pelas portas seriais de entrada/saída e outros componentes do DSK. Uma fonte de tensão de 3,3 V e 1 A também é disponibilizada para placas de expansão, sendo possível também alimentar tais placas com tensão de 12 V e - 12 V, quando o conector de alimentação externa é utilizado.

2.3 Code Composer Studio

O Code Composer Studio (CCS) é uma ferramenta de desenvolvimento da Texas Instruments. Ele fornece um ambiente de desenvolvimento integrado (IDE) e inclui ferramentas para a criação de códigos: compilador de linguagem C, assembly e alguns componentes de apoio como um *linker*. Devido as suas funcionalidades gráficas e a possibilidade de observar a execução do programa em tempo real, o processo de depuração se torna uma tarefa bem menos complicada. O DSK vem com uma versão do CCS já adaptada a sua placa de desenvolvimento. [6]

A Figura 2.2 mostra a interface do usuário do IDE do CCS. Ele é um editor para a criação de código-fonte e um gerenciador de projeto. Ele possibilita também o exame do comportamento do programa em execução. O IDE oferece igualmente a opção de utilização automática de componentes, como o compilador, desta forma o programador não tem necessidade de lidar com todos os problemas operacionais para cada ferramenta individual manualmente.

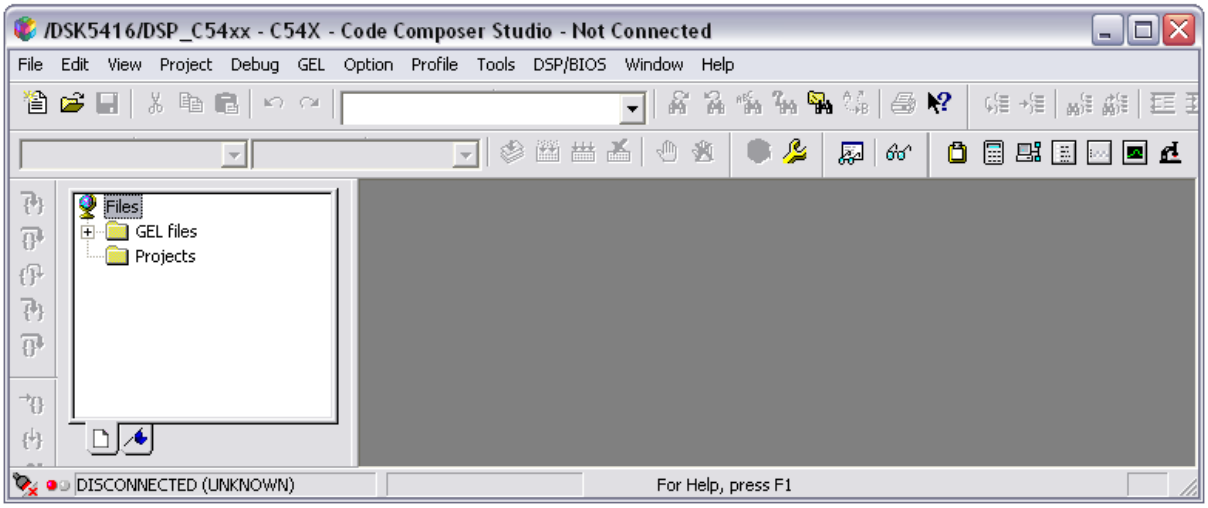

Figura 2.2: IDE Code Composer.

O compilador compila o código fonte em C (***".C"***) para produzir um arquivo em linguagem assembly (***". asm"***). O assembly monta o código ***".asm"*** para produzir um objeto no nível de código de máquina (***".obj"***). Em seguida o *linker* conecta os arquivos objeto e bibliotecas de objetos para gerar um arquivo executável (***".out "***) para ser carregado e executado diretamente no DSP ou no simulador.

No escopo da depuração de projetos, o CCS possui um bom número de funcionalidades tais como os pontos de interrupção (***"Breakpoints"***), a observação de variáveis (***"Watch Window"***), a observação do conteúdo da memória e de registos (***"Memory View"***), vários tipos de gráficos são fornecidos, assim como a opção de exibir em paralelo o código em linguagem C com o código assembly equivalente e a execução do programa em modo passo-a-passo.

O DSK inclui um dispositivo especial, o emulador JTAG, que pode acessar diretamente os registradores e o estado da memória do chip 5416 por intermédio de uma porta JTAG padrão. Quando o usuário deseja acompanhar o andamento de seu programa, o CCS envia os comandos para o emulador através de um *host* USB para verificar os dados de interesse. Esta análise pode ser efetuada usando o RTDX (*Real-Time Data Exchange)*, pois o mesmo permite a troca de dados entre o *host* e o alvo, assim como sua análise em tempo real, sem interromper o programa. Este método de depuração é extremamente poderoso, porque os programas podem ser verificados sem a utilização de disposições especiais de análise, tais como sondas externas ou *softwares* de monitoração.

3. Desenvolvimento de Aplicação de Base

Uma aplicação de base é apresentada neste capítulo: um programa genérico que permitirá utilizar o codec de áudio do DSK 5416 para realizar a captura de amostras de áudio na entrada da placa, processar tais entradas, para, em seguida, enviar as amostras tratadas à saída de áudio do DSK.

Todos os algoritmos deste capítulo e do seguinte foram inseridos na parte de tratamento deste código de base que utiliza a variável ***"mode"*** para determinar, utilizando uma instrução ***"switch ()"***, que tipo de tratamento é realizado. A partir deste exemplo, aplicações clássicas para a aprendizagem do tratamento de sinais são apresentadas: conversão estéreo/mono, efeito de eco e efeito de reverberação.

A criação de projetos, adição de arquivos, utilização das ***"Watch Window"*** ou dos ***"Breakpoints"*** não são aqui descritas. Para obter suporte à utilização de tais ferramentas de desenvolvimento, consulte a versão mais completa deste documento, Capítulo 3 em [7].

3.1 Estudo do Código Fonte

O código fonte deste exemplo está disponível em [8]. O primeiro passo no estudo do programa é abrir o código fonte contendo a função principal, ***"audio.c"***. Para facilitar a compreensão deste código, ele é dividido em partes menores que são estudadas separadamente.

3.1.1 Função "main()"

A primeira parte do programa trata das chamadas as bibliotecas e definições da função ***"main()"***. Esta primeira parte é ilustrada na Listagem 3.1.

```
1     // Progama de base para o tratamento do sinal
2     #include "pcm3002.h"
3
4     Void main(Void)
5     {
6           // Frequência de amostragem défaut
7           Uint32 freq = 48000;
8
9           // Inicialização e configuração do codec de áudio
10          PCM3002_init();
11          PCM3002_setup(NULL);
12          PCM3002_setFreq(freq);
13
14          // Ativa a interrupção via software ligada ao codec de áudio
15          PCM3002_start();
16
17          // Fim do main: devolve a execução ao DSP/BIOS (idle loop)
18          // DSP/BIOS chamará automaticamente a função blockProcessing
19          // a cada bloco de amostras disponível
20    }
```

Listagem 3.1: Chamada as bibliotecas e definições da função ***"main()"*** do programa "***audio.c".***

Na linha 2, pode-se observar a chamada à biblioteca ***"pcm3002.h"***. Esta biblioteca, em conjunto com a ***"plio.h"***, tem o papel de simplificar o uso do codec de áudio PCM3002. Elas criam uma abstração que permite usar o codec sem a necessidade de conhecê-lo no baixo nível. Para usar o codec, só é preciso lhe inicializar utilizando a função ***"PCM3002_init()"*** (linha 10), depois o configurar com o comando ***"PCM3002_setup(NULL)"*** (linha 11) e definir a taxa de amostragem "***PCM3002_setFreq(freq)"*** (linha 12). No tocante às configurações do codec, as opções escolhidas foram as padrões (parâmetro ***"NULL"*** passado durante a chamada da função) e a freqüência de amostragem foi definida como o maior valor suportado pelo codec, que é de 48 kHz.

Após as definições e configurações do PCM3002, faz-se necessário iniciar a sua execução (linha 15) para que ele comece a buscar na entrada analógica do DSK os dados a serem tratados. O codec converte em digital as entradas analógicas amostradas a partir das portas séries bufferizadas.

3.1.2 Definição das Variáveis Globais Ligadas ao Tratamento

A segunda parte do código é mostrada na Listagem 3.2. O programa ***"audio.c"*** utiliza dois *buffers* de dados durante sua execução. O primeiro, ***"inputData[]"*** (linha 7), armazena as entradas amostradas, enquanto que o segundo, ***"outputData[]"*** (linha 8) guarda os dados já tratados e que são enviados até a saída analógica do DSK.

Cada buffer possui 512 posições de 16 bits. As posições de ***"inputData[]"*** só são preenchidas quando houver as 512 amostras necessárias para lhe completar. Esta funcionalidade de esperar por uma determinada quantidade de dados antes de gravar as informações em uma variável é fornecida pela porta serial *bufferizada* (BSP). Esta porta apresenta uma unidade de auto-bufferização (ABU - *Automatic Buffering Unit*) que, possuindo um *buffer* próprio, permite estocar automaticamente uma determinada quantidade de dados recebida na entrada da placa. Utilizando-se desta funcionalidade, não se faz necessário tratar individualmente cada amostra, mas somente o conjunto de 512 a cada vez. Esta estrutura de tratamento do projeto ***"audio.pjt"*** é denominada de tratamento por blocos. Tal forma possibilita um desempenho mais elevado em comparação ao tratamento com amostras individuais, já que as trocas de dados com o codec são geradas no nível *hardware* pelo DSP. Utilizando desta abordagem os dados são transmitidos à função de tratamento por blocos de 512 valores. [9]

Para definir qual é o tratamento utilizado durante a execução do programa, a variável ***"mode"*** (linha 11) é avaliada através de uma estrutura de seleção ***"switch()"***.

```
1     // Variáveis globais ligadas ao tratamento
2
3     // LEN: número de amostras direita+esquerda por bloco
4     #define LEN 512
5
6     // Buffers de entrada e de saída
7     Int16 inputData[LEN];
8     Int16 outputData[LEN];
9
10    // Modo de funcionamento
11    Int16 mode = 0;
```

Listagem 3.2: Declaração das variáveis globais que são utilizadas para o tratamento so sinal

3.1.3 Controle de Tarefas do Programa

Para esta parte, apenas o recebimento de dados é descrito, já que a emissão de dados segue um caminho simétrico. A porta serial gerencia o envio da amostra analógica (convertida em digital pelo codec de áudio) para o registrador de entrada DRR (ou DXR para a saída). O DMA permite a transferência de dados entre este registrador e a memória principal do DSP sem interromper o tratamento. A utilização deste recurso utiliza o conceito de PIP, uma abstração em nível de *software* de transferência de dados específica para a plataforma DSP/BIOS [10] da Texas Instrument (plataforma esta que não tem seu conteúdo abordado neste documento).

O DSP funciona em sua essência como um *loop* de espera (*idle loop*), e gera automaticamente as entradas/saídas utilizando o codec de áudio (com seu ABU e DMA) em tarefa de plano de fundo. No momento em que o *buffer* de entrada é preenchido pelo DMA, a interrupção de *software* ***"swiBlockProcessing()"*** é gerado pelo DSP/BIOS. É devido a utilização destas interrupções que o programa não precisa de um *loop* infinito dentro da função ***"main()"***.

A ***"swiBlockProcessing()"*** é configurada no arquivo ***"audio.cdb"*** para chamar a função ***"blockProcessing()"***, onde o sinal é processado e os tratamentos implementados são postos. O sinal é recebido e transmitido em blocos de ***"LEN"*** amostras. Como o codec é estéreo, essas ***"LEN"***

amostras são metade do canal esquerdo e metade do direito. Para cada bloco de entrada recebido, um bloco de saída deve ser transmitido.

Sendo as entradas/saídas geredas sob a forma de interrupções, é essencial que o tempo de processamento da função ***"blockProcessing()"*** seja menor que o tempo entre duas ocorrências da interrupção ***"swiBlockProcessing()"***.

3.1.4 Tratamento do Sinal

A última etapa do programa lida com o tratamento do sinal propriamente dito, apresentado na Listagem 3.3.

```
1    Void blockProcessing(Void)
2    {
3        Int16 i;
4        Int16 *src, *dst;
5
6        // 1) Define os apontadores para entrada e saída
7        PCM3002_allocateBuffers(&src,&dst);
8
9        // 2) Copia as amostras de entrada em um buffer auxiliar
10       for (i = 0; i < LEN ; i++)
11           // Preenche o array inputData com as amostras de entrada
12           inputData[i] = (short)(*src++);
13
14       // 3) Tratamento
15       switch (mode)
16       {
17           case 0:
18               // Tratamento simples: copia as entradas na saída
19               for (i = 0; i < LEN ; i++)
20                   outputData[i] = inputData[i];
21           break;
22
23           case 1:
24               /* ... */
25           break;
26       }
27
28       // 4) Copia os dados tratados no apontador de saída
29       for (i = 0; i < LEN ; i++)
30           *dst ++ = (short)outputData[i];
31
32       // 5) Libera os apontadores src e dst
33       PCM3002_releaseBuffers();
34   }
```

Listagem 3.3: Bloco de tratamento do sinal.

No final do ***"main()"*** o controle da execução do programa é dada ao DSP/BIOS (*idle loop*). No módulo do DSP/BIOS há a definição da interrupção ***"swiBlockProcessing()"***. A função ***"blockProcessing()"*** (linha 1) é chamada automaticamente por essa interrupção cada vez que um bloco de LEN amostras está disponível no codec.

Quando ***"blockProcessing()"*** é chamada, o DSP/BIOS têm disponíveis dois *buffers*: um de recepção, ***"inputData[]"***, que contém as LEN amostras recebidas, e um de destino, ***"outputData[]"*** de mesmo tamanho, que é prenchido com os dados tratados. A estrutura desta função é composta por cinco etapas:

1. ***Recuperação dos endereços do buffer fonte "*src" e do buffer de destino "*dst":*** a cada chamada à ***"blockProcessing()"***, dois apontadores, ***"*src"*** e ***"*dst"*** são criados (linha 4) e associados respectivamente aos *buffers* de entrada e saída do codec de áudio usando a

função ***"PCM3002_allocateBuffers()"*** (linha 7). Esses apontadores são utilizados para acessar os dados de entrada ou copiar os valores na saída do codec.

2. ***Cópia das amostras de entrada em um buffer auxiliar:*** a segunda etapa (linhas 9 a 12) é de copiar as amostras de entrada no *buffer* ***"inputData[]"***;

3. ***Tratamento:*** o tratamento em si compõe a terceira parte do código (linhas 14 a 26). A variável global ***"mode"*** é usada para selecionar o método de tratamento do programa. No exemplo há apenas um modo (linhas 17 a 21): copiar a entrada na saída do DSK. O resultado de todos os tratamentos é armazenado em ***"outputBuffer[]"***;

4. ***Cópia dos dados de saída no buffer "*dst":*** o quarto passo é copiar os dados processados para a saída do DSK, acessível a partir do ponteiro ***"*dst"*** (linhas 28 a 30);

5. ***Fim do tratamento:*** a última etapa é a de desalocar os apontadores criados com o comando ***"PCM3002_releaseBuffers()"*** (linha 33). Como a função ***"blockProcessing()"*** é chamada várias vezes durante a execução do programa (todas as vezes que há 512 amostras disponíveis), caso os ponteiros não fossem desalocadas, poderia haver um problema de fuga de memória. Após este comando, o fim do bloco de processamento é relatado ao DSP/BIOS e a execução do programa saí de ***"blockProcessing()"***.

Um fluxograma do programa ***"audio.c"*** pode ser visto na Figura 3.1, onde um conversor A/D e um conversor D/A são utilizados após a entrada analógica e antes da saída analógica, respectivamente. Na Figura 3.2 é ilustrado o fluxo de dados a partir das amostras "brutas" encontradas na entrada do DSK até a saída com os dados já processados.

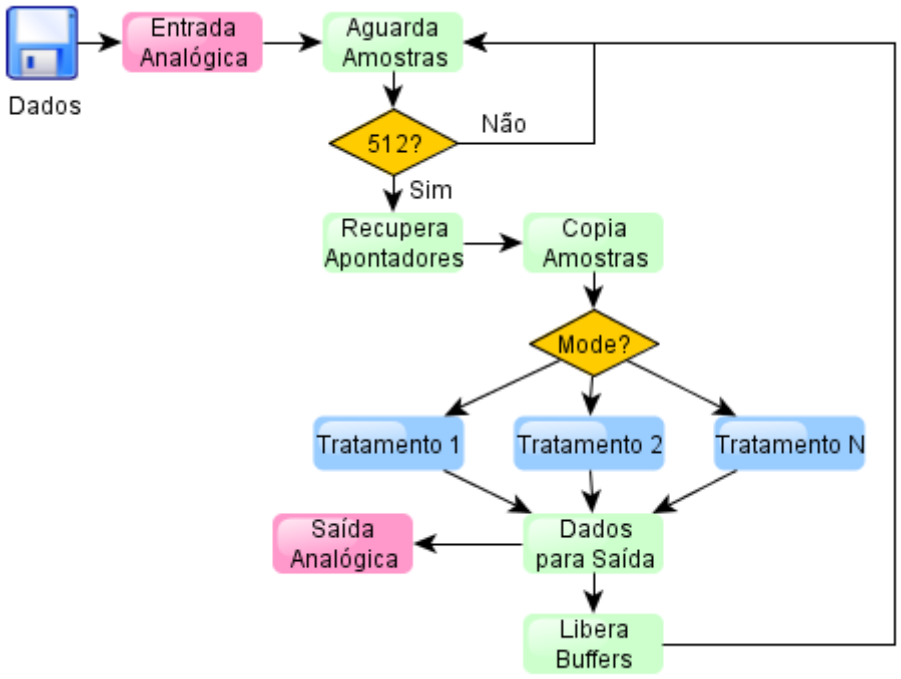

Figura 3.1: Fluxograma representativo do programa ***"audio.c"***.

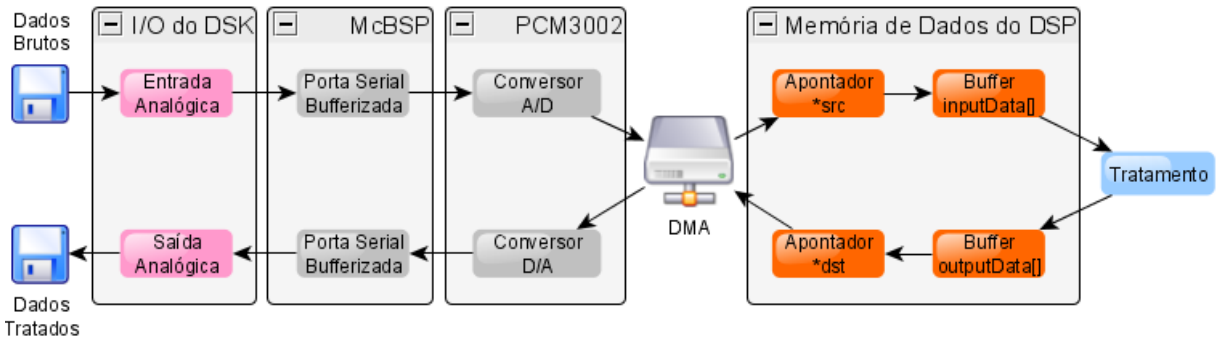

Figura 3.2: Fluxo de dados do programa ***"audio.c"***.

Durante a execução do programa os dados são capturados na entrada para o DSK, para em seguida passarem através da porta serial bufferizada que os levará até o conversor analógico/digital do codec PCM3002. A passagem de dados para a memória de programa do DSP é coordenado pelo DMA, o que permite a transferência de dados para o ponteiro ***"*src"*** na memória principal sem interromper o tratamento. Para o tratamento, o conteúdo do apontador ***"*src"*** é copiado no *buffer* ***"inputData[]"***. Após o tratamento, os dados seguem o caminho inverso: buffer ***"outputData[]"***, apontador ***"*dst"***, conversor digital/analógico do PCM3002, porta serial bufferizada e saída analógica.

3.2 Execução e Testes do Programa "audio.c"

Para executar e testar o programa, as entradas e saídas da placa devem ser conectadas como mostradas na Figura 3.3. Um cabo *jack/jack* transmite a saída de áudio do computador para a entrada da placa do DSK e um fone de ouvido é conectados à saída do DSK. O papel de fonte de sinal é desempenhado pelo computador.

Como ilustrado na Figura 3.2, o DSK utiliza seu codec para realizar as conversões A/D e D/A. Desta forma, o sinal analógico que entra em ***"Line In"*** (ou ***"Mic In"***) é convertido em digital e enviado para o chip do DSP. O sinal de saída do DSP é convertido para analógico e enviado para a saída ***"Line Out"*** (ou ***"Spkr Out"***).

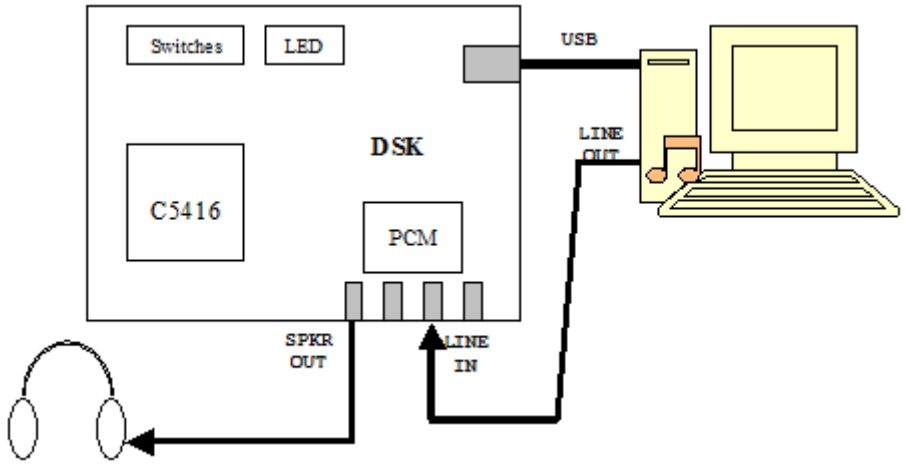

Figura 3.3: Configuração para os testes do programa ***"audio.c"***. [9]

Para iniciar os testes, um sinal de áudio foi tocado pelo computador em plano de fundo. A opção de repetição do *player* de áudio foi marcada para que no final do arquivo ele recomessasse automaticamente a sua reprodução.

Durante a execução do programa, é possível verificar que o sinal foi transmitido de forma idêntica para a saída da placa DSK escutando a música com os fones de ouvido.

É importante notar que o sinal chega diretamente no DSK na forma analógica através do cabo de áudio *jack/jack* e que não há qualquer comunicação direta entre o CCS e o *player*.

O sinal tipo seno (***"sinus1760Hz"***, disponível em [11]) também foi utilizado para verificar que o programa executado realizava corretamente a tarefa de copiar na saída a entrada do DSK.

Os gráficos para a entrada e saída do DSK foram exibidos no CCS e ilustrados na Figura 3.5. Os parâmetros utilizados para a obtenção dos gráficos são apresentados na Figura 3.4 (os parâmetros omitidos permanecem com seus valores padrão).

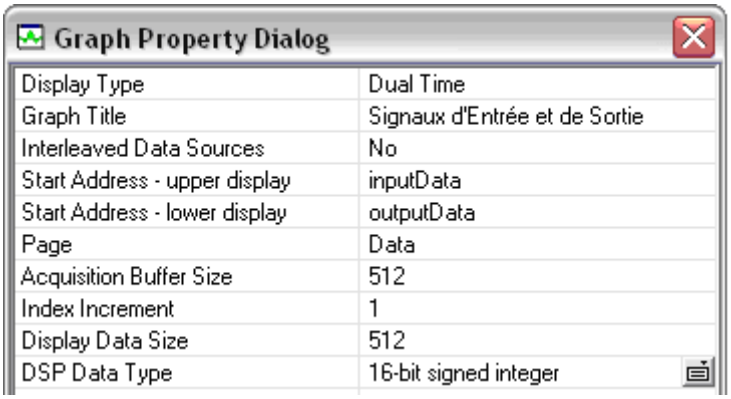

Figura 3.4: Parâmetros para o gráfico de entrada e saída do DSK.

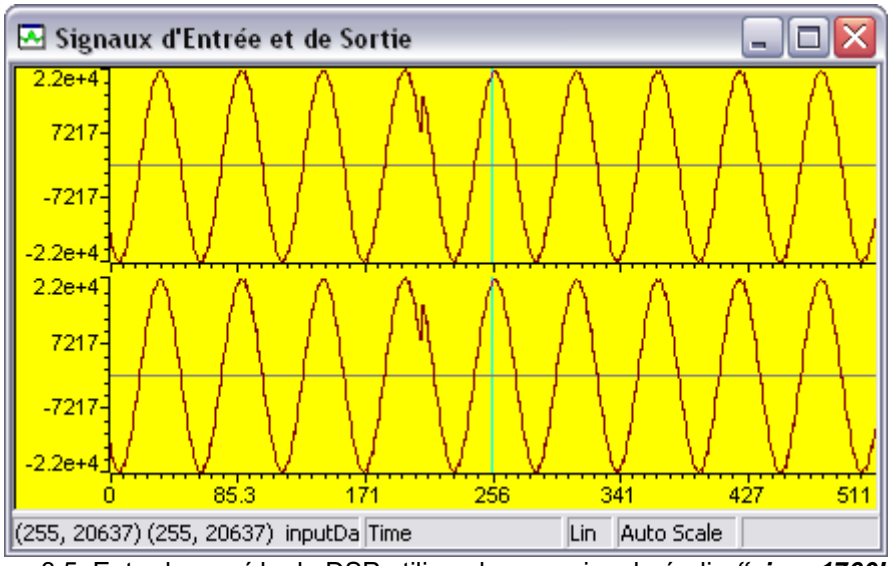

Figura 3.5: Entrada e saída do DSP utilizando o arquivo de áudio ***"sinus1760Hz"***.

Cosntata-se que a entrada foi satisfatoriamente copiada na saída e a curva obtida para ambos é um seno, como esperado. A curva tem uma ligeira degeneração. Esse problema surge devido ao arquivo de áudio usado possuir apenas três segundos e ser tocado em sucessivas repetições.

Outro gráfico relevante é a representação em freqüência dos sinais obtidos, utilizando para tal a transformada rápida de Fourier. Para obter esse gráfico, os parâmetros para o gráfico de saída da Figura 3.6 foram utilizados (os parâmetros omitodos permanecem com seus valores padrão). Um parâmetro particulamente importante definido para este gráfico é ***"Sampling Rate"***, parâmetro este que define a escala do eixo de frequência do gráfico. No domínio temporal é também possível utilizar esse parâmetro, sendo o seu efeito exibir o eixo horizontal na forma de tempo e não em número de amostras.

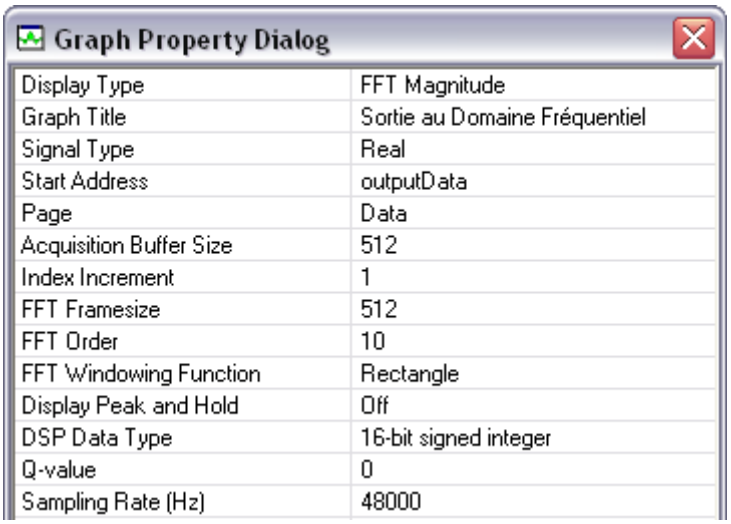

Figura 3.6: Parâmetros para o gráfico de saída do DSK no domínio frequencial.

Conforme observado na Figura 3.7, a única componente de freqüência relevante é em cerca de 880 Hz (1760 Hz dividido por 2). A explicação de ela ocorrer em 880 e não em 1760 Hz é que o arquivo de áudio ***"sinus1760Hz"*** é um arquivo mono, mas a entrada do codec é estéreo. Como há duas vezes mais informação para a representação de um sinal estéreo que para o mono (dados da via direita mais os dados da via esquerda), o aspecto final ao nível do DSK é um sinal cuja freqüência é metade da esperada.

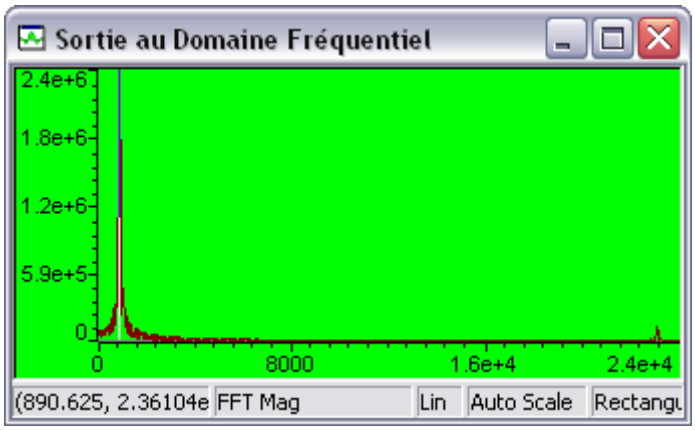

Figura 3.7: Representação no domínio frequencial da saída do DSP utilizando o arquivo de áudio ***"sinus1760Hz"***.

3.3 Conversão Estéreo/Mono

A primeira aplicação prática abordada foi de tratar um sinal de áudio estéreo para que ele se tornasse mono. Suponha uma conversa entre três pessoas, como mostrado na Figura 3.8. Mesmo que o homem do meio estivesse com os olhos fechados, ele saberia, intuitivamente, onde se localizam os outros dois quando eles falassem. Esta capacidade de distinguir a localização das fontes sonoras é chamada de estereofonia.

Figura 3.8: Exemplificação da estereofonia.

O áudio estereofônico, ou simplemente estéreo, é então um tipo de reprodução sonora baseada no fato de que temos duas orelhas, um atributo que nos permite localizar a fonte, distância e direção do som. Sistemas de reprodução estéreos simulam este efeito de percepção do som. Isto é conseguido

através de dois canais independentes possuidos pelo sistema. Desta forma, por exemplo, durante a gravação de uma canção em um estúdio, o cantor pode ter sua voz captada por um microfone independente, enquanto os instrumentos musicais são captados por outro. Quando se ouve esta gravação, tem-se a impressão de que a voz do cantor vem de um lado do ambiente enquanto que os sons dos instrumentos vêm de outro.

O áudio monofônico, ou mono, só possui um canal de modo que todos os sons são capturados por um único microfone. Assim, a reprodução de áudio mono não fornece a capacidade de reproduzir os efeitos de profundidade e de diferentes posições, mesmo que se utilizassem várias caixas de som para sua reprodução.

3.3.1 Transmissão Estéreo no Nível do Programa "audio.c"

No escopo do exemplo apresentado, a saída do computador, usada para fornecer o áudio para a entrada DSK, é um sistema estéreo, como também o é o codec PCM3002. Desta forma, como se tem necessidade de fornecer informações diferentes para cada canal, a cada "segundo de música" que chega aos fones de ouvido, precisa-se enviar "um segundo de música" para o canal de direita e "um segundo de música" para o de esquerda. Portanto, se um sistema mono requer um único conjunto de dados para ambos os canais (já que ambos tocarão o mesmo som), para o sistema estéreo é preciso um conjunto de dados individuais para cada canal, requerindo o dobro de informações.

A representação do fluxo de dados é ilustrada na Figura 3.9. Após deixar o computador pelo cabo *jack/jack* (de dois canais), os dados são enviados por dois fios diferentes para a entrada analógica do DSK. O conjunto BSP e PCM3002 recebe os dados ainda distintos, mas os armazenarão na memória do programa já juntos, intercalando um dado da via esquerda com um dado da via direita.

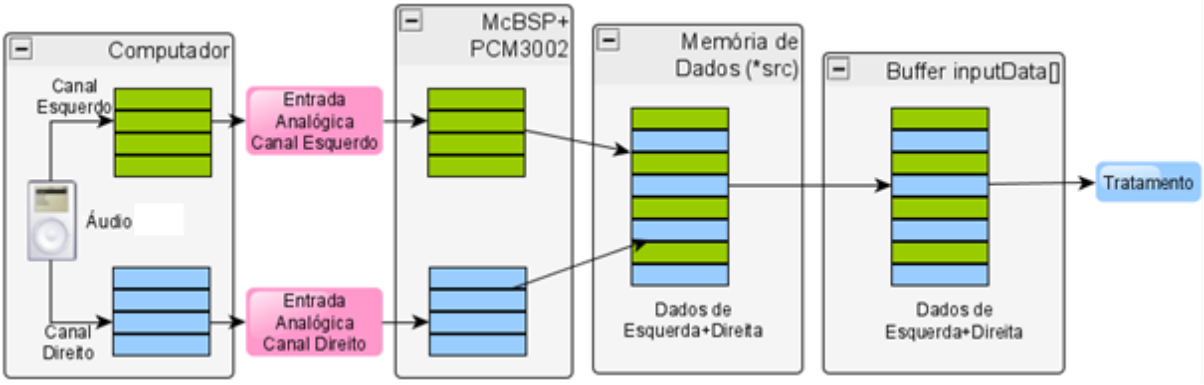

Figura 3.9: Fluxo de dados dos canais esquerdo e direito de um sistema de áudio estéreo.

O endereço onde os dados são armazenados é guardado pelo apontador ***"*src"***. Esse apontador é usado para copiar os dados para o *buffer* de entrada ***"inputData[]"***. Como o buffer de entrada tem 512 posições, 256 são usadas para armazenar os dados do canal da esquerda e 256 para os de direita. O processo inverso é observado após o tratamento de ***"mode = 0"***, como mostrado na Figura 3.10.

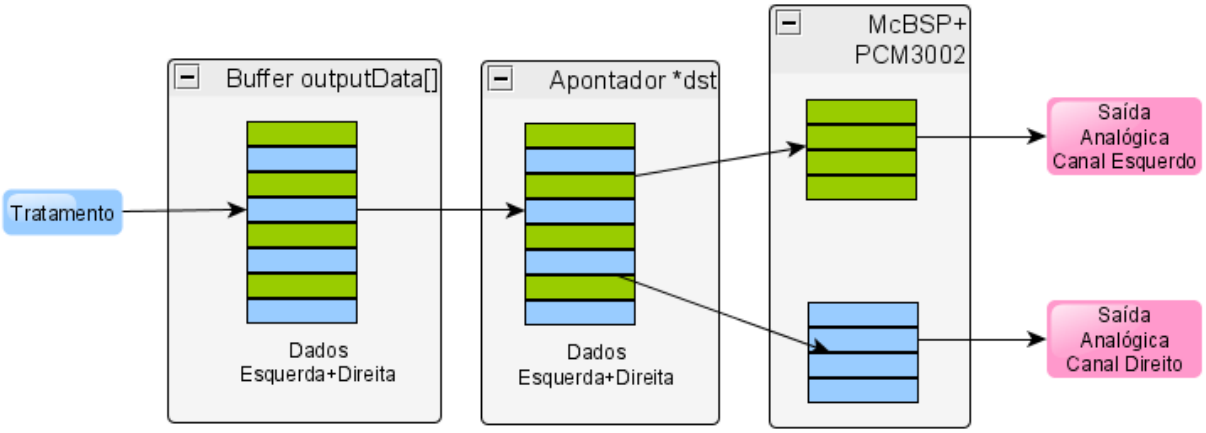

Figura 3.10: Saída do fluxo de dados dos canais esquerdo e direito de um sistema de áudio estéreo.

Como este tratamento só copiou os dados da entrada para a saída, a expectativa é de que os dados sejam iguais no buffer ***"outputData[]"*** e em ***"inputData[]"***.

Quando o codec de áudio recebe dados cujo endereço é indicado por um ponteiro (no nosso caso o ponteiro ***"*dst"***), ele enviará, de forma intercalada e sequência, os dados para o canal esquerdo e para o canal direito até a interface de saída do DSK.

3.3.2 Transformação Estéreo/Mono

Para converter o sinal de áudio estéreo em um sinal de áudio mono, o seguinte procedimento pode ser adotado: sabe-se que no sistema mono, embora possam existir vários canais, todos devem tocar o mesmo som. No exemplo apresentado dois canais estão disponíveis, o esquerdo e o direito. Desta maneira, para realizar o processo de conversão de estéreo para mono, é preciso enviar os mesmos dados para ambos os canais. A principal questão é: "se o que chega ao codec são dados estéreos, o que se deveria colocar como dados nos dois canais para se ter uma reprodução mono?".

Para cada "segundo de música" duas informações diferentes estão disponíveis, sendo uma correspondente ao canal esquerdo e outra ao direito. Como cada uma dessas informações fornece detalhes diferentes, faz-se necessário misturar esses dois sinais e obter a sua média para em seguida enviar esse novo sinal para ambos os canais.

Como o *buffer* de entrada tem 512 posições, a média fornecerá apenas 256 valores. Se esses 256 valores são enviados para o codec de áudio, ele não saberá que o que ele está recebendo se trata de uma informação mono que deve ser enviada igualmente para os dois canais. O codec, independente do que ele receba, enviará sequencialmente um dado para a via esquerda e outro para a via direita. Em consequência, é necessário copiar a mesma informação em duas posições consecutivas do *buffer* de saída para se obter um som mono, realizando para tal uma operação de "espansão" de um valor em duas posições consecutivas da memória. Todo esse processo é ilustrado na Figura 3.11: os dados de direita e de esquerda são recebidos separadamente, em seguida eles são armazenados juntos na memória do programa. Depois a média entre os valores deve ser realizada e o resultado armazenado em duas posições consecutivas da memória.

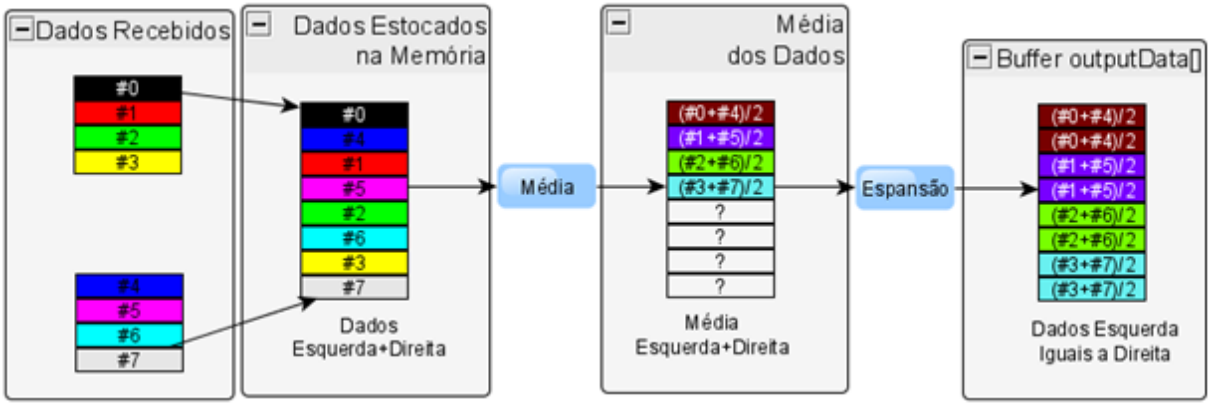

Figura 3.11: Tratamento estéreo/mono.

3.3.3 Codificação da Conversão Estéreo/Mono

Para alterar o código ***"audio.c"*** para que ele realize a conversão, o ***"case 1:"***, que estava vazio, foi editado. Como a partir do *buffer* ***"inputData[]"*** os dados intercalados esquerdos e direitos estão disponíveis, é necessário obter a média entre duas posições consecutivas e armazená-lo no *buffer* de saída ***"outputData[]"***. É necessário também que esses dados sejam armazenados em duas posições consecutivas da memória. Estas duas etapas podem ser realizadas separadamente, mas um custo computacional menor é conseguido se forem feitas em conjunto, como mostrado na Listagem 3.4. Neste código, sua execução iterará através do laço ***"for"*** ***"LEN/2"*** vezes, já que a variável ***"i"*** é incrementada de dois em dois (linha 3). A cada iteração, a média das duas posições consecutivas de ***"inputData[]"*** é armazenada em duas posições ***"outputData[]"*** (linhas 5 e 6).

```
1     case 1:
2           // Conversão Estéreo/Mono
3           for (i = 0; i < LEN; i+=2)
4           {
5                 outputData[i] = inputData[i]/2 + inputData[i+1]/2;
6                 outputData[i+1] = outputData[i];
7           }
8     break;
```

Listagem 3.4: Código para a conversão estéreo/mono.

Para utilizar este novo tratamento, o modo de funcionamento deve ser alterado para 1, Listagem 3.5.

```
// Modo de funcionamento
short mode = 1;
```

Listagem 3.5: Seleção do tratamento estéreo/mono selecionando ***"mode"*** 1.

Um teste estéreo encontrado no site [12] foi usado para validar a conversão. A fim de verificar a diferença entre os dois sistemas de áudio, a variável ***"mode"*** foi exibida na aba ***"Watch1"*** de ***"Watch Window"***. Seu valor foi alterado entre 0 e 1 durante a execução do programa para chavear a saída entre estéreo e mono, Figura 3.12.

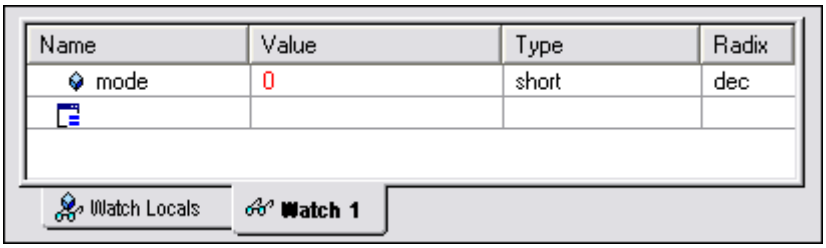

Figura 3.12: Seleção da saída de áudio: mono ou estéreo.

3.4 Efeito de Eco

Quando falamos geramos ondas mecânicas que se propagam através do meio. No momento em que essas ondas atingem uma superfície, uma parte é refletida e volta para os nossos ouvidos. O tempo entre o momento em que falamos até ouvirmos a onda refletida é determinada pela distância entre nós e a superfície que reflete o som.

Como o ouvido humano não pode distinguir dois sons similares quando não há uma separação mínima de 0,1 segundo entre eles, a percepção da onda refletida não é sempre garantida. Considerando a velocidade média do som no ar, cerca de 340 m/s, este 0,1 segundo representa 34 metros que o som tem que percorrer para ter esse atraso. Assim, se a distância para o obstáculo é inferior a 17 metros (17 metros a ida mais 17 metros a volta totalizando 34 metros), não se detecta a diferença entre o som emitido e o som recebido. Para distâncias superiores, quando tal percepção é obtida, chama-se o áudio refletido de eco.

3.4.1 Compreensão Teórica

Ao nível matemático, o efeito de eco é um filtro cuja equação de diferença é dada por:

$$y(n) = x(n) + g.x(n-D)$$

onde ***"D"*** é o valor do atraso em número de amostras, ***"g"*** um ganho real, ***"y"*** a saída do sistema e ***"x"*** sua entrada.

Como a taxa de amostragem (definida pela variável ***"freq"***) vale 48000 Hz no projeto, a relação entre o atraso ***"d"*** em segundos e do atraso ***"D"*** em número de amostras é dado por:

$$D = d*freq$$
$$d = D/freq$$
$$d = D/48000$$

Sendo o valor mínimo para ouvir os ecos igual a 0,1 segundo, é necessário um valor de atraso ***"D"*** superior a 4800 amostras para conseguir o efeito de eco.

Utilizando-se este valor mínimo, a equação de diferença se torna ***"y(n) = x(n) + g.x (n-4800)"***. A saída do sistema é composta então pela entrada atual adicionada à entrada anterior, que foi amostrada a 4800 ciclos atrás. O programa ***"audio.c"*** atual não tem capacidade de fornecer essa entrada precedente, uma vez que não há uma tabela para armazenar os valores passados que chegaram à entrada do sistema.

Fez-se necessário então, adicionar uma tabela suplementar para armazenar tais valores. Um *buffer* circular de 4800 posições foi criado para guardar os valores de ***"x"***. *Buffer* circular é um *array* cujo conteúdo é escrito e lido de uma forma cíclica, ou seja, sua última posição é sucedida pela primeira, conforme ilustrado na Figura 3.13. Dessa forma, se um algoritmo de leitura lê a última posição do *buffer* e deve continuar a ler, a leitura irá retornar ao início do *buffer* e de lá continuará. O mesmo se aplica aos algoritmos de escrita, sendo a única diferença que a escrita em uma posição não vazia faz com que haja perda do conteúdo original.

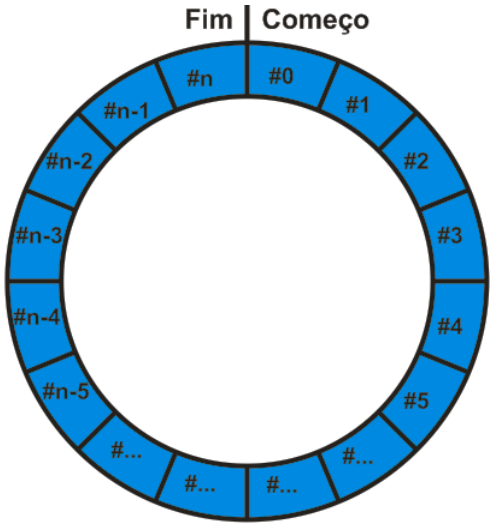

Figura 3.13: Abstração do *buffer* circular, onde sua última posição é conectada a primeira.

Quando a leitura ou escrita de dados chega ao limite da capacidade do *buffer*, o seu índice, que possui a função de indicar a última posição onde dados foram estocados, é indexada para começo do buffer.

3.4.2 Implementação

Para a concepção de tal filtro, o código ***"audio.c"*** foi adaptado para possuir um *buffer* circular assim como um ***"case"*** que implementasse a filtragem de eco. A declaração das variáveis necessárias a este processo é ilustrada na Listagem 3.6.

```
1     // Definições para utilização da filtragem de eco
2     #define BUFSIZE 19200
3     Int16 inputCircBuf[BUFSIZE]; // buffer circular de entrada
4     Int16 inputBufIdx = 0;       // índice do buffer circulair de entrada
5
6     Uint16 D = 4800;
7     Int16 G = 13107; // 0.8 em formato Q14
8     Int16 k = 14;    // Defasagem do Qk
```

Listagem 3.6: Variáveis necessárias para a implementação do efeito de eco.

Para armazenar o sinal que é necessário no cálculo da saída do filtro (as entradas precedentes), o *buffer* circular de captura "***inputCircBuf[]"*** foi criado (linha 3). O tamanho do *buffer* é definido pela constante ***"BUFSIZE"*** que vale 19200 (linha 2). Com este tamanho e a frequência de amostragem de 48 kHz, pode-se alcançar uma defasagem máxima de 0,4 segundo. O valor padrão desta variável de atraso é 4800, o que representa o valor teórico mínimo para se observar o efeito do eco: 0,1 segundo (linha 6).

A variável ***"inputBufIdx"*** (linha 4) é utilizada para guardar a posição atual do *buffer* circular. É a partir desta variável que é possível verificar se o *buffer* foi completamente preenchido e se já a partir da próxima gravação os dados deverão ser armazenados no início da estrutura de dados.

As últimas variáveis criadas são dois interios de 16 bits ***"G"*** e ***"k"***. A variável ***"G"*** (linha 7) controla o ganho do eco. Uma observação pertinente em relação a ela deve ser feita: observando novamente a equação do filtro: ***"y(n) = x(n) + g.x(n-D)"***, pode-se constatar que o parâmetro original ***"g"***, supostamente um real, foi substituído por ***"G"***, um inteiro de 16 bits.

Na realidade, existem duas abordagens diferentes para representar os números reais com precisão finita: a representação com ponto fixo e a representação com ponto flutuante. Cada tipo de DSP é concebido para poder trabalhar de uma forma otimizada com uma destas representações. De toda forma, um DSP, a exemplo do TMS320VC5416, concebido para trabalhar com ponto fixo, poderá também efetuar cálculos com ponto flutuante, porém seu desempenho não é tão bom. Desta maneira,

apesar de ser possível utilizar uma variável ***"float g"*** para representar o ganho do efeito de eco, optou-se por utilizar a representação em ponto fixo para se alcançar uma maior compatibidade com o DSP alvo [13].

Apesar de no primeiro momento poder parecer diferente, utilizar ***"float g = 0.8"*** terá o mesmo efeito de utilizar ***"Int16 G = 13107"*** graças à propriedade de representação dos números fracionários utilizando variáveis do tipo inteiras.

Esta forma de representação é chamada de ***"Representação binária de números fracionários em formato de ponto fixo"***. No escopo de tal método, os *bits* da variável são utilizados para diferentes tipos de informações: um *bit* é utilizado para o sinal, outra parte para armazenar a parte inteira do número e o resto para a parte fracionária. A expressão ***"formato Q_k"*** é geralmente utilizada para indicar uma representação com ***"k"*** *bits* para a parte fracionária.

Assim, o ganho ***"g"***, um número real, terá uma representação binária com ponto fixo ***"G"***, com precisão finita de 16 *bits* em formato Q_{14} (14 *bits* para representar a parte fracionária) da seguinte forma:

$$G \rightarrow b_{0(Sinal)} b_{1(Inteiro)}, b_{2(Fração)} \ldots b_{15(Fração)}$$

Para este caso, um *bit* é utilizado para o sinal, um *bit* para a parte inteira e quatorze *bits* para a parte fracionária. A relação entre ***"G"*** e ***"g"*** é dada por:

$$G = g * 2^k$$
$$G = g * 2^{14}$$

Como ***"G"*** é um inteiro sinalizado de dezesseis bits, seu intervalo de valores é de $[-2^{15}, 2^{15}-1]$, ou seja, [-32768, 32767]. Conhecendo tais valores, o intervalor alcançado por ***"g"*** em Q_{14} é:

$$g_{\min} = \frac{G_{\min}}{2^k} = \frac{-2^{15}}{2^{14}} = -2$$
$$g_{\max} = \frac{G_{\max}}{2^k} = \frac{2^{15}-1}{2^{14}} = 1.99993896484375$$

Devemos lembrar que nenhuma variável de programa é utilizada em representação Q_k. Assim, depois de cada multiplicação de um número inteiro por um número em formato Q_{14}, faz-se necessário recolocar o valor obtido em formato inteiro, que é obtida de acordo com a fórmula que segue:

$$Valor(Inteiro) = \frac{Valor(Q_k)}{2^k}$$

Para usar corretamente o *buffer* circular, foi necessário criar instruções cujos papéis eram de verificar se o fim do *buffer* havia sido atingido quando quiséssemos escrever ou se o índice era inferior a zero quando na leitura de dados (que se dá de modo regressivo). Essas funções são respectivamente ***"circularBufferWrite()"*** e ***"circularBufferRead()"***.

Após escrever essas duas funções, o acesso ao *buffer* circular foi alcançado. O próximo passo foi criar um novo ***"case"*** no ***"switch"*** e implementar a função de transferência do filtro, como mostrado na Listagem 3.7.

```
1    case 2:
2        // Efeito de eco
3        for (i = 0; i < LEN ; i++)
4        {
5            outputData[i] = inputData[i] + ((Int32)G *
                          (Int32)circularBufferRead(inputCircBuf,
```

```
                                &inputBufIdx, BUFSIZE, D)>>k);
6            circularBufferWrite(inputCircBuf, &inputBufIdx,
                                 BUFSIZE, inputData[i]);
7        }
8    break;
```

Listagem 3.7: Implementação do efeito de eco.

Assim como para os demais ***"cases",*** há ***"LEN"*** iterações, uma para cada amostra recebida, e a cada iteração a saída é calculada como a entrada atual mais a multiplicação do ganho por um valor deslocado do *buffer* circular (linha 5). Após o cálculo da saída, o *buffer* circular é atualizado com o valor atual de ***"inputData[]"*** (linha 6).

A Tabela 3.1 ilustra a comparação entre a rotina da linha 5 com a fórmula do filtro, ***"y(n) = x(n) + g.x(n-D)"***, mostrando a diferença entre cada parte destas duas equações.

Teórico	Prático
y(n)	outputData[i]
x(n)	inputData[i]
g.x(n-D)	((Int32)G*(Int32)circularBufferRead(inputCircBuf,&inputBufIdx,BUFSIZE,D)>>k)

Tabela 3.1: Diferenças entre a equação teórica e prática para o efeito de eco.

Embora a relação entre as primeiras linhas seja clara (***"y(n) -> outputData[i]"*** e ***"x(n) () -> inputData[i]"***), a mesma clareza não é observada para a última. Nela, ***"g"*** é associado com ***"G"*** e ***"x(n-D)"*** é associado com ***"circularBufferRead(inputCircBuf,&inputBufIdx,BUFSIZE,D)"***. Há, entratanto, dois problemas que tornam a linha de comando um pouco mais complexa: a promoção para o *tipo* ***(Int32)***, ou *cast*, e do deslocamento ***">> k"***.

A variável ***"G"*** é representada em Q_{14} e ***"inputData[]"*** em inteiro puro, ou seja Q_0, uma vez que ele não utiliza qualquer um de seus *bits* para representar a parte fracionária. Após ter efetuado a multiplicação entre o ganho ***"G"*** e o valor defasado de ***"inputData[]"***, um outro valor em formato Q_{14} é obtido. Como após essa operação o resultado da multiplicação é somado à ***"inputData[i]"*** que é Q_0, é necessário converter este resultao Q_{14} em Q_0. Pare efetuar esta operação é necessário dividir o número em Q_k por 2^k. O operador de deslocamento ***">>"*** realiza esta divisão binária. Como a variável ***"k"*** vale 14, o produto é dividido por 2^{14} e se tornará um número Q_0 que poderá ser somado à ***"inputData[]"***.

Em relação ao *cast* ***(Int32),*** ele é necessário porque na linguagem C a multiplicação entre dois números do mesmo tipo tem como resultado outro de mesmo tipo, ou seja: ***"Int16 * Int16 = Int16"***. Como a multiplicação entre dois números de 16 bits pode ter uma resposta de até 31 bits, é preciso converter os dois operandos em ***"Int32"*** para não perder informação nesta operação.

Para uma melhor compreensão do comando, um exemplo é considerado. A título de simplificação ***"circularBufferRead(inputCircBuf,&inputBufIdx,BUFSIZE,D)"*** é chamdo de ***"inputData[i-D]"***. Seja o valor de ***"g"*** igual a 0,8, ***"inputData[i]"*** a 10000 e ***"inputData[i-D]"*** a 1000. A operação desejada é: ***"outputData[i] = inputData[i] + g.inputData[i-D]"***. Os passos para obter o resultado de forma teórica e prática são ilustrados nas Tabelas 3.2 e 3.3, respectivamente.

Equação teórica	y(n) = x(n) + g.x(n-D)
Substituição dos valores	y(n) = 10000 + 0.8*1000
Multiplicação	y(n) = 10000 + 800
Adição	y(n) = 10800

Tabela 3.2: Processo para obtenção teórica da saída ***"y(n)"***.

Equação prática	outputData[i] = inputData[i] + g.inputData[i-D]
Substituição dos valores	outputData[i] = 10000 + 0.8*1000
Conversão de "g" en Q_{14}	G = g * 2^{14} = 0.8 * 2^{14} = 13107
Substituição de "g" por "G"	outputData[i] = 10000 + 13107*1000
Conversão dos operandos em "Int32"	outputData[i] = 10000 + 13107[32 bits]*1000[32 bits]
Multiplicação	outputData[i] = 10000 + 13107000[32 bits]

Conversão do resultado em Q_0	13107000 / 2^{14} = 800
Substituição do resultado Q_0	outputData[i] = 10000 + 800
Adição	outputData[i] = 10800

Tabela 3.3: Processo para obtenção, em nível de programa, a saída ***"y(n)"***.

3.4.3 Execução e Testes

Para verificar o comportamento do programa, uma música foi executada em *background*, como anteriormente, e o efeito de eco observado. A variável de controle ***"mode"***, foi definida como 2 para selecionar este novo tratamento, Listagem 3.8.

```
// Modo de funcionamento
Int16 mode = 2;
```

Listagem 3.8: Seleção do efeito de eco: o valor da variável "***mode"*** é colocado em 2.

O eco não é facilmente percebido quando o valor de atraso, ***"D=4800"***, é usado. Foi preciso então abrir a ***"Watch Window"*** e exibir a variável ***"D"*** para alterar seu valor e observar como a saída de áudio se comporta para diferentes valores de atraso. Com valores pequenos, como 1200 amostras (0,025 segundo) o eco não foi escutado. Para valores maiores que 9600 ele era percebido facilmente. No valor limite, ***"D = 19200 (0,4 segundos)"***, os dois sons eram praticamente disjuntos um do outro.

Foi possível também reproduzir sinais conhecidos, como seno ou onda quadrada, e visualizar a partir de gráficos os *buffers* ***"inputData[]"*** e ***"outputData[]"***. Para o caso de uma onda senoidal, o efeito do eco equivale a somar dois senos defasados, obtendo-se uma nova senoide cuja amplitude é modificada de acordo com o atraso ***"D"*** e o ganho ***"G"***. A entrada e saída do efeito de eco obtida com os parâmetros da Figura 3.14, estão ilustrados na Figura 3.15. Os parâmetros que não são mostrados na figura guardam os valores padrão.

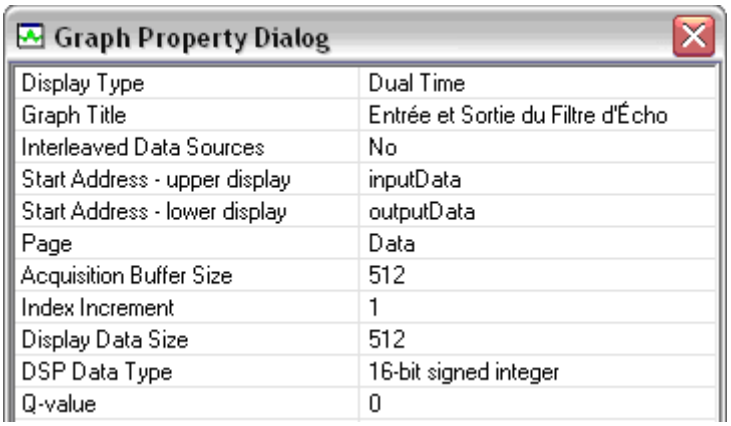

Figura 3.14 : Parâmetros para os gráficos de entrada e saída do efeito de eco.

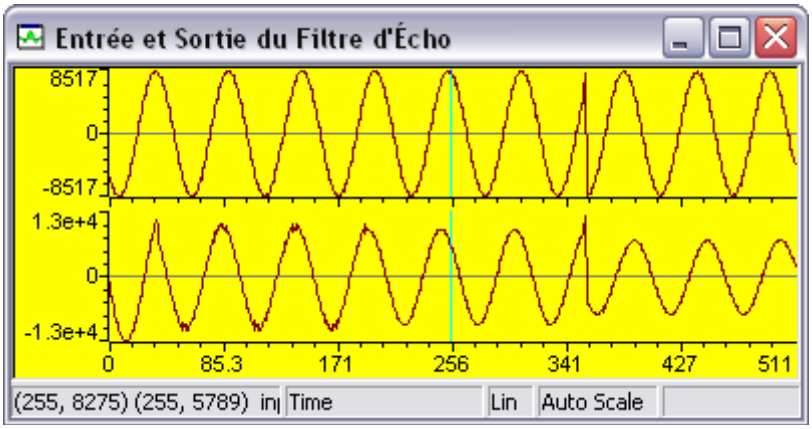

Figura 3.15: Entrada e saída do efeito de eco tendo como sinal de entrada um seno de 1760 Hz, ***"D=4800"*** e ***"g=0.8"***.

Se a entrada é alterada para um sinal quadrado, um sinal com três estágios de amplitude é obtido, o que corresponde à soma de um sinal quadrado com este mesmo sinal, porém defasado, Figura 3.16.

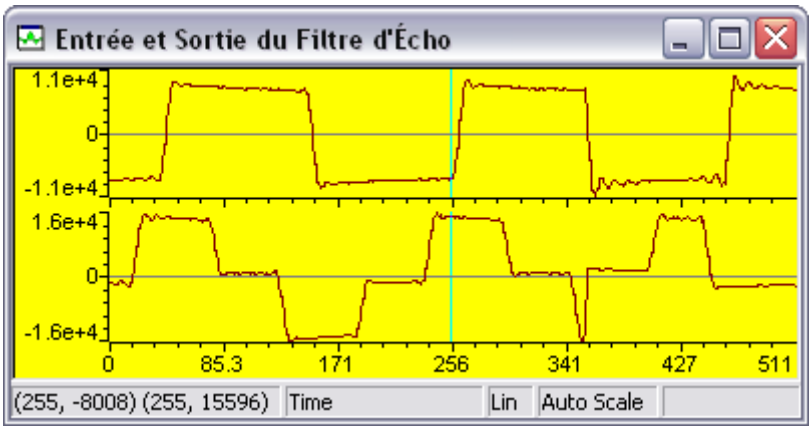

Figura 3.16: Entrada e saída do efeito de eco tendo como sinal de entrada uma onda quadrada de 440 Hz, ***"D=4800"*** e ***"g=0.8"***.

4. Filtragem Digital

Para os sistemas de transmissão da informação, um filtro é um componente cuja função é de retirar partes não desejadas do sinal, ou de extrair os componentes de frequência específica. Há dois tipos principais de filtros, os digitais e os analógicos. Mesmo sendo o objetivo final de ambos igual, eles são muito diferentes no que diz respeito a seus projetos, implementação e modo de funcionamento.

Enquanto que para a criação de filtros analógicos componentes como resistores, indutores e capacitores são utilizados para produzir o efeito de filtragem desejado, para os filtros digitais um processador é utilizado para efetuar os cálculos que desempenharão o papel de filtrar o sinal. Este processador pode ser o de um computador pessoal, *notebook* ou de um DSP.

Outra diferença entre estes dois tipos de filtros é o sinal que cada um trata. Se para os filtros analógicos os sinais analógicos são utilizados diretamente no tratamento, para a filtragem digital estes sinais devem ser convertidos em digitais através do processo de amostragem e codificados em formato binário. É este novo sinal binário que será tratado pelo filtro. Após o tratamento, o sinal digital obtido é reconstituído em formato analógico.

Deve-se notar também que para o filtro digital, o sinal é representado por uma sucessão de números no lugar de uma tensão ou corrente elétrica, como ocorre para os filtros analógicos.

Atualmente, os filtros digitais são muito utilizados em diversos domínios da tecnologia, como por exemplo, as telecomunicações, podendo-se, inclusive, atribuir o grande desenvolvimento deste domínio à popularização da filtragem digital e a sua substituição dos filtros analógicos. Os motivos desta substituição decorrem de diversas vantagens da versão digital em relação à analógica, como por exemplo:

- Os filtros digitais são programáveis e seu funcionamento é determinado por uma rotina armazenada em sua memória. Desta maneira, a rotina poderá ser modificada de uma forma rápida e simples alterando linhas de código, ao contrário do filtro analógico que necessitaria de mudanças em seu circuito eletrônico;

- Os filtros digitais são concebidos facilmente uma vez que eles podem ser testados e implementados em *desktops* ou DSP;

- Como os filtros analógicos são compostos diretamente de elementos elétricos como resistores, indutores e capacitores, sua característica funcional é influenciada por problemas decorrentes da variação de temperatura ou alteração de valor devido ao tempo de uso, por exemplo. Os fitros digitais não sofrem destes problemas, já que são extremamente estáveis, obtendo assim resultados mais precisos.

4.1 Convolução no Tempo Discreto e Resposta Frequencial

Para compreender bem o funcionamento de um filtro digital, é necessário ter o conhecimento da teoria que rege a convolução.

Seja ***"x[n]"*** a entrada de um sistema discreto, linear e invariante no tempo (LTI), a saída ***"y[n]"*** deste sistema pode ser calculada pela convolução entre ***"x[n]"*** e a sua resposta impulsional ***"h[n]"***. A formulação matemática para esta convolução no tempo discreto é mostrada na Equação 4.1 [14].

$$y[n] = \sum_{k=-\infty}^{\infty} x[k].h[n-k] \qquad (4.1)$$

Sabendo que a transformada Z da convolução no tempo discreto entre dois sinais é o produto da transformação de cada um, tem-se:

$$Y(z) = \sum_{n=-\infty}^{\infty} y[n].z^{-n} = X(z).H(z) \qquad (4.2)$$

Onde ***"X(z)"*** e ***"H(z)"*** são dados respectivamente pela Equação 4.3 e 4.4.

$$X(z) = \sum_{n=-\infty}^{\infty} x[n].z^{-n} \qquad (4.3)$$

$$H(z) = \sum_{n=-\infty}^{\infty} h[n].z^{-n} \qquad (4.4)$$

4.2 Filtro à Resposta Impulsional Finita

Se a resposta impulsional de um filtro é igual à zero fora de um intervalo de inteiros {0, 1, 2, ..., N-1}, a convolução ilustrada pela Equação 4.1 é dada pela Equação 4.5.

$$y[n] = \sum_{k=0}^{N-1} x[k].h[n-k] = \sum_{k=0}^{N-1} h[k].x[n-k] \qquad (4.5)$$

Um filtro deste tipo é dito de resposta impulsional finita, ou não recursiva, de ordem N. Um esquema de implementação de filtros FIR é ilustrado na Figura 4.1. Trata-se de uma linha de atrasos representada por blocos ***"z^{-1}"*** e um conjunto de coeficientes multiplicativos cujos valores são iguais à resposta impulsional do filtro ***{h[0], h[1], ..., h[N-1]}***.

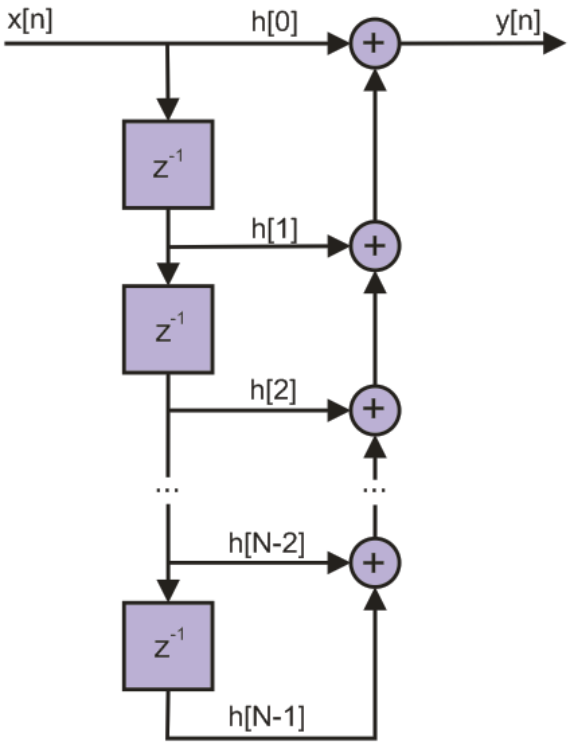

Figura 4.1: Forma direta para implementação do filtro FIR.

4.3 Filtro à Resposta Impulsional Infinita

Um filtro com uma resposta impulsional ***"h[n]"*** infinita é dito IIR. Se ***"h[n]"*** é a soma de exponenciais amortecidas, a função de transferência do filtro, ***"H(z)"***, obtida utilizando a transformada Z, é fornecida pela divisão de dois polinômios finitos em função de z, como ilustrado pela Equação 4.6.

$$H(z) = \frac{Y(z)}{X(z)} = \frac{a_0 + a_1.z^{-1} + a_2.z^{-2} + ... + a_N.z^{-N}}{1 + b_1.z^{-1} + b_2.z^{-2} + ... + b_M.z^{-M}} = \frac{A(z)}{B(z)} \quad (4.6)$$

4.3.1 Forma Direta Tipo I

A solução da Equação 4.6 pode ser obtida de diversas maneiras. A mais simples a nível matemático é a dita *"forma direta tipo I"*, encontrada isolando-se ***"Y(z)"*** e ***"X(z)"*** na Equação 4.6, como visto nas Equações 4.7, 4.8 e 4.9.

$$Y(z).B(z) = X(z).A(z) \quad (4.7)$$

$$Y(z).\left(1 + \sum_{k=1}^{M} b_k.z^{-k}\right) = X(z).\left(\sum_{k=0}^{N} a_k.z^{-k}\right) \quad (4.8)$$

$$Y(z) = \sum_{k=0}^{N} a_k.X(z).z^{-k} - \sum_{k=1}^{M} b_k.Y(z).z^{-k} \quad (4.9)$$

Utilizando a transformada Z inversa, a representação temporal da Equação 4.9 é dada por Equação 4.10.

$$y[n] = \sum_{k=0}^{N} a_k.x[n-k] - \sum_{k=1}^{M} b_k.y[n-k] \quad (4.10)$$

Com esta equação, observa-se que para obter o valor atual da saída ***"y[n]"*** é necessário calcular a soma da entrada atual (multiplicada por ***"a_0"***), mais ***"N"*** entradas precedentes (multiplicadas por seus ***"a_k"*** correspondentes) e ***"M"*** saídas precedentes (multiplicadas por seus ***"b_k"*** correspondentes). Decorrente da utilização de saídas anteriores para o cálculo da atual, este filtro é dito recursivo. Denomina-se a implementação que utiliza a Equação 4.10 de *direta* uma vez que os coeficientes da função de transferência (***"a_k"*** e ***"b_k"***) são utilizados diretamente na equação diferencial do filtro, como ilustrado na Figura 4.2.

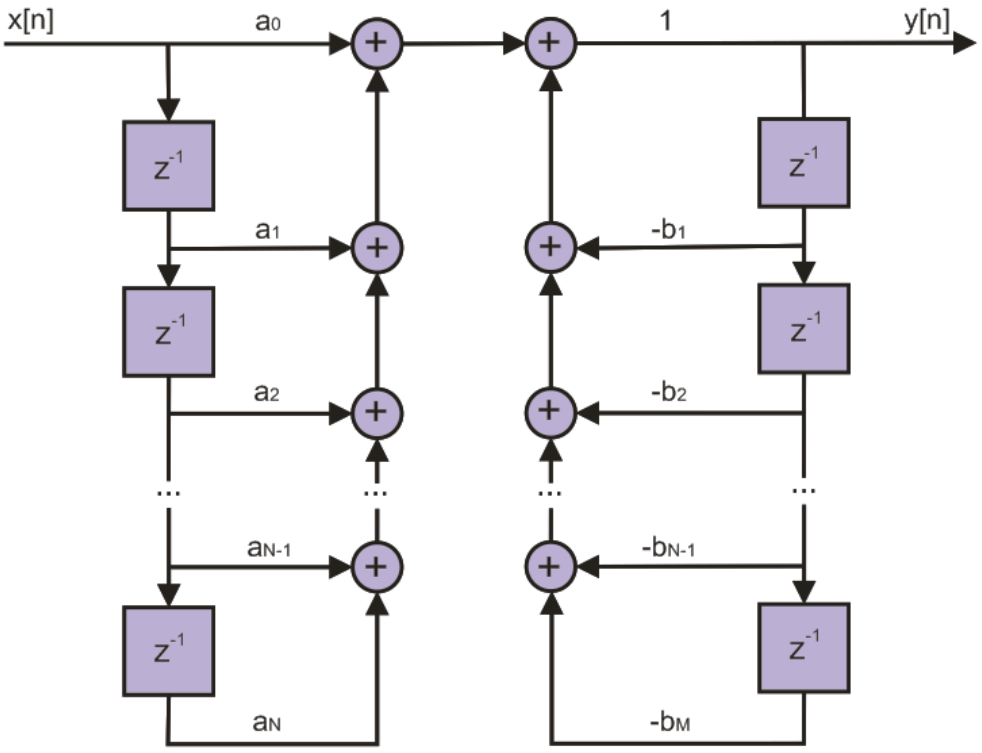

Figura 4.2: Forma direta tipo I para implementação do filtro IIR.

Para um filtro de ordem ***"N"*** (supondo ***"N = M"***), esta estrutura possuirá ***"2*N"*** elementos de atraso, representados por ***"z^{-1}"*** na Figura 4.2. Assim, se tivéssemos um filtro de ordem ***"N = 10"***, vinte elementos de atraso seriam necessários.

4.3.2 Forma Direta Tipo II

Apesar de ao nível matemático a forma do tipo I ser a mais simples, ela utiliza muitos elementos de atraso (memória do processador) e para as aplicações práticas ela é substituída por outras opções que apresentam uma implementação menos custosa, como a *"forma direta tipo II"*, ilustrada na Figura 4.3, nela apenas metade dos elementos de atraso do tipo I são utilizados. Se para o tipo I era necessário memória para as entradas e as saídas anteriores, para o tipo II é requerida apenas memória para uma variável intermediária auxiliar denominada ***"u[n]"***.

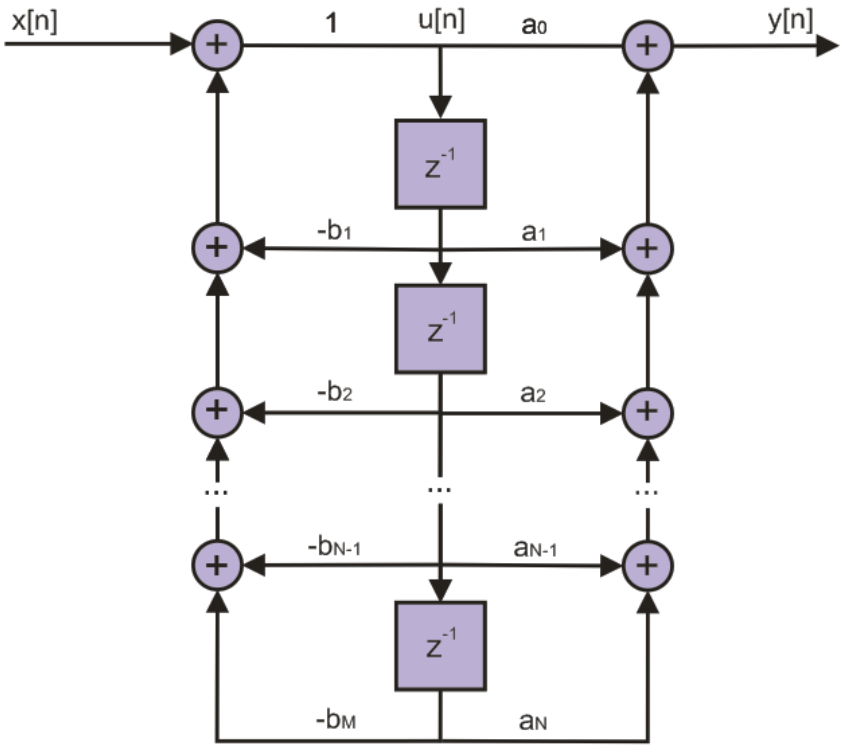

Figura 4.3: Forma direta tipo II para a implementação do filtro IIR.

Para alterar a Equação 4.10, possibilitando que ela possa ser escrita em função de ***"u[n]"***, utiliza-se o processo seguinte: seja a variável auxiliar ***"U(z)"*** dada pela Equação 4.11:

$$U(z) = \frac{X(z)}{B(z)} \qquad (4.11)$$

Das Equações 4.4 e 4.11, tem-se:

$$Y(z) = \frac{A(z).X(z)}{B(z)} = A(z).U(z)$$

$$Y(z) = U(z).(a_0 + a_1.z^{-1} + a_2.z^{-2} + ... + a_N.z^{-N}) \qquad (4.12)$$

Isolando-se ***"X(z)"*** em (4.11):

$$X(z) = U(z).B(z) = U(z).(1 + b_1.z^{-1} + b_2.z^{-2} + ... + b_M.z^{-M}) \qquad (4.13)$$

Utilizando a transformada Z inversa para a Equação 4.13, obtem-se:

$$x[n] = u[n] + b_1.u[n-1] + b_2.u[n-2] + ... + b_M.u[n-M] \quad (4.14)$$

Isolando ***"u[n]"*** em função de ***"x"*** a partir de (4.14):

$$u[n] = x[n] - b_1.u[n-1] - b_2.u[n-2] - ... - b_M.u[n-M] \quad (4.15)$$

Para se obter ***"y[n]"*** em função de ***"u"***, aplica-se a tranformada Z inversa em (4.12):

$$y[n] = a_0 u[n] + a_1.u[n-1] + a_2.u[n-2] + ... + a_N.u[n-N] \quad (4.16)$$

Pode-se utilizar (4.15) e (4.16) para representar o filtro na forma direta tipo II: a variável auxiliar ***"u[n]"*** é satisfeita por (4.15) enquanto que a saída ***"y[n]"*** é satisfeita por (4.16).

4.3.3 Forma Direta Tipo II em Cascata

Considerando que os valores de ***"M"*** e ***"N"*** são iguais, pode-se escrever a Equação 4.17 como uma multiplicação de fatores de primeira ordem:

$$H(z) = \frac{a_0 + a_1.z^{-1} + a_2.z^{-2} + ... + a_N.z^{-N}}{1 + b_1.z^{-1} + b_2.z^{-2} + ... + b_M.z^{-N}} = C\prod_{k=1}^{N}\frac{z - z_k}{z - p_k} \quad (4.17)$$

onde ***"C"*** é um ganho e "z_k" e "p_k" são respectivamente os zeros e pólos da função de transferência.

Pode-se também fatorisar (4.17) em termos de primeira e segunda ordem, Equação 4.18. Esta forma de representação é dita em cascata e sua estrutura é mostrada na Figura 4.4.

$$H(z) = C.H_1(z).H_2(z)...H_r(z) \quad (4.18)$$

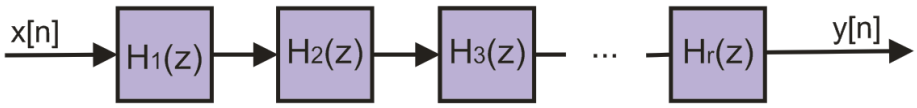

Figura 4.4: Forma em cascata para implementação do filtro IIR.

Para a implementação do filtro, cada um de seus fatores ***"H_r"*** (chamados também de seções), podem ser representados pela forma direta tipo II. Assim, se, por exemplo, deseja-se cirar um filtro de quarta ordem, pode-se fatorizar a função de transferência em dois termos de segunda ordem (***"H1"*** e ***"H2"***) e lhes colocar sob a forma direta tipo II, como ilustrado na Figura 4.5.

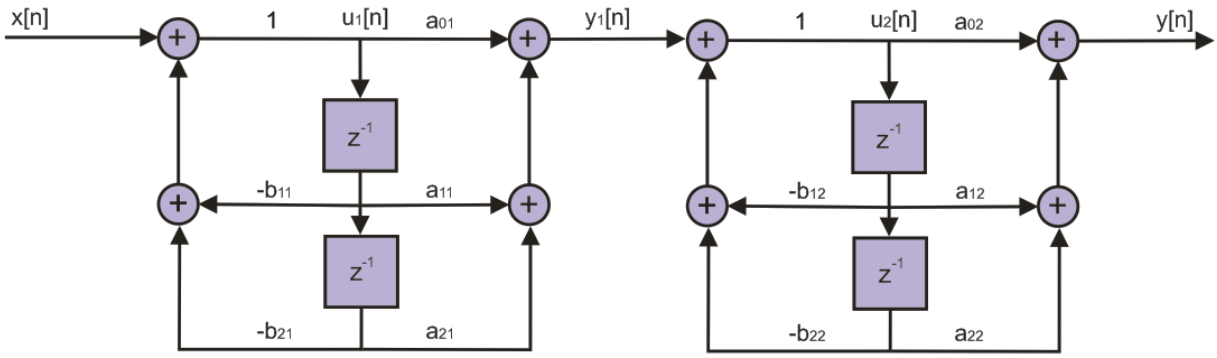

Figura 4.5: Filtro IIR de quarta ordem fatorizado em seções de segunda ordem na forma direta tipo II.

A função de transferência ***"H(z)"*** para uma estrutura em cascata é dada pela Equação 4.19.

$$H_r(z) = \prod_{i=1}^{N/2} \frac{a_{0i} + a_{1i}.z^{-1} + a_{2i}.z^{-2}}{1 + b_{1i}.z^{-1} + b_{2i}.z^{-2}} \qquad (4.19)$$

Para o exemplo da Figura 4.5, a função de transferência fatorizada em termos como (4.19) se tornará:

$$H(z) = \frac{(a_{01} + a_{11}.z^{-1} + a_{21}.z^{-2})}{(1 + b_{11}.z^{-1} + b_{21}.z^{-2})} \cdot \frac{(a_{02} + a_{12}.z^{-1} + a_{22}.z^{-2})}{(1 + b_{12}.z^{-1} + b_{22}.z^{-2})} \qquad (4.20)$$

4.4 Obtenção dos Coeficientes dos Filtros Digitais

A escolha da utilização de um filtro FIR ou IIR depende da aplicação. Os filtros IIR são indicados para os casos com especificação em amplitude. Em contrapartida, sua fase é essencialmente não linear. Já os filtros FIR podem possuir fase perfeitamente linear, o que é fundamental para certas aplicações, a exemplo da transmissão de dados.

É possível sintetizar filtros FIR cujas especificações em amplitude são rígidas, embora os IIR sejam mais bem adaptados à síntese de filtros clássicos tais como os passa-baixas, passa-altas, rejeita-faixa, etc.

Para um mesmo desempenho (para especificações em amplitude, por exemplo) a ordem de um FIR é geralmente superior a ordem de um IIR. Portanto, a realização FIR é mais complexa em número de coeficientes a serem calculados e a quantidade de memória.

Tendo decidido qual o filtro mais adequado para a aplicação, o próximo passo é de lhe caracterizar para a implementação. Nesta etapa, há outra diferença importante entre filtros FIR e IIR: o procedimento para a caracterização de um filtro digital, que consiste em obter os coeficientes ***"a"*** e ***"b"***, envolve métodos distintos para os dois tipos.

Uma maneira simples e prática para obter os coeficientes de filtros digitais é a utilização do *toolbox* SPTool (*Signal Processing Tool - Graphical User Interface*) disponíveis no MATLAB. Além disso, é possível exportar um arquivo gerado a partir da ferramenta contendo os coeficientes do filtro e então usá-los no código-fonte em linguagem C. Esta abordagem foi adotada, uma vez que possibilita uma rápida caracterização do filtro, assim como facilidade na integração do mesmo no projeto base ***"audio.pjt"***. Para obter informações adicionais a respeito do processo de obtenção dos coeficientes com SPTool ver [15].

4.5 Implementação de um Filtro FIR

Como exemplo da aplicação de um filtro digital, o filtro FIR é discutido nesta seção. O projeto básico ***"audio.pjt"*** foi complementado, adicionando a funcionalidade desta filtragem.

O primeiro passo foi calcular os coeficientes do filtro utilizando o SPTool. Como exemplo, foi implementado um filtro passa-baixas, cujas características são apresentadas na Tabela 4.1.

Tipo de Resposta	Passa-baixas
Tipo do Filtro	FIR (WINDOW)
Método de Janelamento	Hamming
Ordem do Filtro	50
Frequência de Amostragem	48 kHz
Frequência de Corte	5 kHz

Tabela 4.1: Características do filtro FIR estudado.

O arquivo mostrado na Listagem 4.1 foi obtido através do SPTool. Trata-se de um vetor contendo os coeficientes do filtro em formato Q_{15}, representando a função de transferência ***"H(z)"***. Como o filtro é de ordem de 50, tem-se um vetor de 51 posições.

```
/************************************************
 Coeficientes do Filtro FIR:
```

```
      Tipo de Resposta           => Passa-baixas
      Tipo do Filtro             => FIR(WINDOW)
      Método de Janelamento      => Hamming
      Ordem do Filtro            => 50
      Frequência de Amostragem   => 48 kHz
      Frequência de Corte        => 5 kHz
 ***********************************************/

// Quantité de coefficients
#define NUM_COEF 51

Int16 h[NUM_COEF] = {  -20,     0,    26,   51,   63,   44,
                       -15,  -101,  -178, -194, -106,   87,
                       325,   494,   470,  184, -326, -887,
                     -1229, -1075,  -248, 1228, 3104, 4958,
                      6319,  6818,  6319, 4958, 3104, 1228,
                      -248, -1075, -1229, -887, -326,  184,
                       470,   494,   325,   87, -106, -194,
                      -178,  -101,   -15,   44,   63,   51,
                        26,     0,   -20};
```

Listagem 4.1: Arquivo ***"coef_fir_pb_5kHz.h"***.

O código fonte utilizado nos capítulos anteriores foi substituído por um novo. O arquivo que o contém, denominado ***"fir.c"*** é semelhante ao ***"audio.c"***, a diferença observada trata principalmente da remoção de partes do código que não são mais necessárias, como a estrutura ***"switch"*** usado para selecionar o tratamento desejado. Adições feitas podem ser observadas na Listagem 4.2.

```
1     #include "coef_fir_pb_5kHz.h"
2     […]
3
4     #define BUFSIZE NUM_COEF
5     […]
6
7     Int16 i, j;
```

Listagem 4.2: Mudanças de ***"fir.c"*** em relação à ***"áudio.c"***.

Na linha 1 é incluída a chamada ao arquivo que contém os coeficientes do filtro passa-baixas, ***"coef_fir_pb_5kHz.h"***. A variável ***"NUM_COEF"*** (que corresponde ao número de coeficientes do filtro) é definida na Listagem 4.1. Esta constante corresponde também à ordem máxima de atrasos. Desta maneira deve-se atribuir o tamanho do *buffer* circular a esse valor e minimizar assim o uso da memória (linha 4). A última mudança é a criação de uma variável de iteração adicional ***"j"***. Esta variável é necessária para implementar a convolução.

O próximo passo foi elaborar a rotina de filtragem FIR. A Equação 4.5, recordada a seguir, foi utilizada.

$$y[n] = \sum_{k=0}^{N-1} h[k].x[n-k] \qquad (4.5)$$

Esta equação mostra que a saída atual ***"y[n]"*** é a convolução entre ***"h[n]"*** e ***"x[n]"***. A título de ilustração, para um caso simples, onde a ordem do filtro seja ***"N = 3"***, a Equação 4.5 é escrita da seguinte forma:

$$y[n] = \sum_{k=0}^{2} h[k].x[n-k] = h[0].x[n] + h[1].x[n-1] + h[2].x[n-2] \qquad (4.21)$$

Com o vetor ***"h[n]"*** definido em ***"coef_fir_pb_5kHz.h"***, utiliza-se o *buffer* circular para armazenar as entradas anteriores. Desta forma, (4,21) torna-se:

y[n] = h[0].circularBufferRead([...], atraso = 0) +
h[1].circularBufferRead([...], atraso = 1) +
h[2].circularBufferRead([...], atraso = 2)

Observando esta expressão, constata-se que para sua implementação é necessário, a cada iteração: armazenar a entrada atual no *buffer* circular e utilizar um laço ***"for"*** para realizar a soma de todos os "***h[n].x[n-k]"***.

Estes passos podem ser realizados, para qualquer ordem do filtro, utilizando-se a Listagem de 4.3. Na primeira linha, a declaração ***"for"*** é colocada para repetir o processo para cada uma das 512 amostras. Em seguida tem-se o procedimento de armazenamento no *buffer* circular, linha 3. As linhas 6 e 7 realizam a convolução propriamente dita.

Optou-se por trabalhar com os coeficientes do filtro em formato Q_{15}, desta forma, necessita-se utilizar o *cast* ***"(Int32)"*** e em seguida o operador de deslocamento binário ***">> 15"*** para obter o resultado em formato inteiro.

```
1     for (i = 0; i < LEN ; i++)
2     {
3           circularBufferWrite(inputCircBuf, &inputBufIdx, BUFSIZE,
                                inputData[i]);
4
5           outputData[i] = 0;
6           for (j = 0; j < NUM_COEF; j++)
7                outputData[i] += ((Int32)h[j] * (Int32)circularBufferRead(
                               inputCircBuf, &inputBufIdx, BUFSIZE, j))>>15;
8     }
```

Listagem 4.3: Algoritmo para a filtragem FIR.

4.5.1 Execução e Testes do Filtro FIR

Para esta parte, uma estrutura ***"swtich"*** é utilizada para mudar os sinais de entrada do filtro:

1. Entrada analógica amostrada;
2. Entrada impulsional;
3. Entrada em degrau.

Esta opção permite que se possa observar melhor as características dos filtros obtidos. Para tanto, uma nova variável global de seleção é criada, assim como uma variável para definir a amplitude do impulso e do degrau, Listagem 4.4.

```
Int16 mode_entree = 1;
Int16 valeur_entree = 32767; // Maior (Int16)
```

Listamge 4.4: Variáveis utilizadas para selecionar o modo de entrada do programa e a amplitude do impulso e do degrau.

Em seguida, a seção "***2) Copia as amostras de entrada em um buffer auxiliar"*** foi substituída pela apresentada na Listagem 4.5.

```
    // 2) Seleção da entrada do programa
      switch (mode_entree)
      {
            case 0:
                  // Entrada analógica amostrada
                  src_p = src;
                  // Conversão Stereo-Mono (uma via)
                  for (i = 0; i < LEN ; i++)
                  {
                        inputData[i]       = *(src_p+=2);
```

```
                        inputData[i+LEN/2] = inputData[i];
                  }

            break;

            case 1:
                  // Entrada Impulsional
                  inputData[0] = valeur_entree;
                  for (i = 1; i < LEN ; i++)
                        inputData[i] = 0;
            break;

            case 2:
                  // Entrada em degrau
                  for (i = 0; i < LEN ; i++)
                        inputData[i] = valeur_entree;
            break;
      }
```

Listagem 4.5: Seleção da entrada para o filtro FIR.

Os gráficos necessários aos testes foram os seguintes:

1. Entrada/Saída no domínio temporal;
2. Entrada no domínio frequencial, e;
3. Saída no domínio frequencial.

Os parâmetros definidos para cada um desses três gráficos são vistos nas Figuras 4.6, 4.7 e 4.8, respectivamente. Os parâmetros omitidos guardam seus valores padrões.

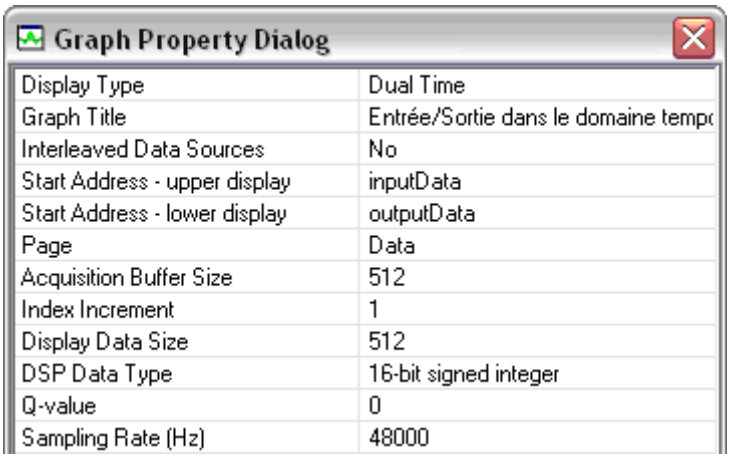

Figura 4.6: Parâmetros para o gráfico ***"Entrada/Saída no domínio temporal"***.

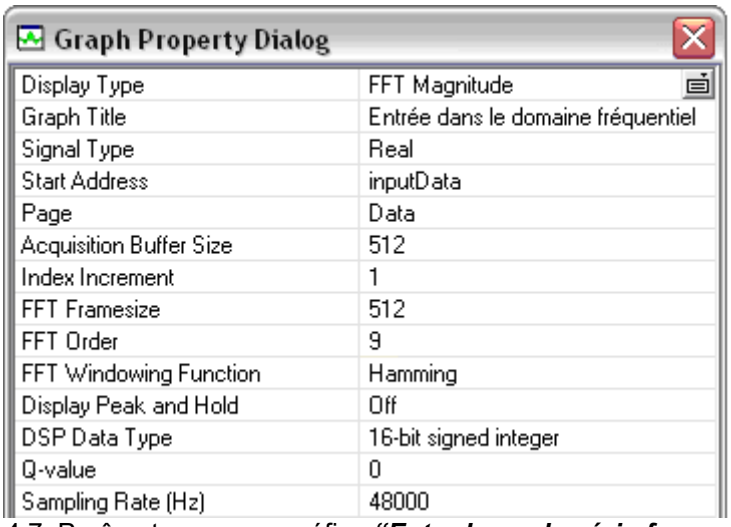

Figura 4.7: Parâmetros para o gráfico ***"Entrada no domínio frequencial"***.

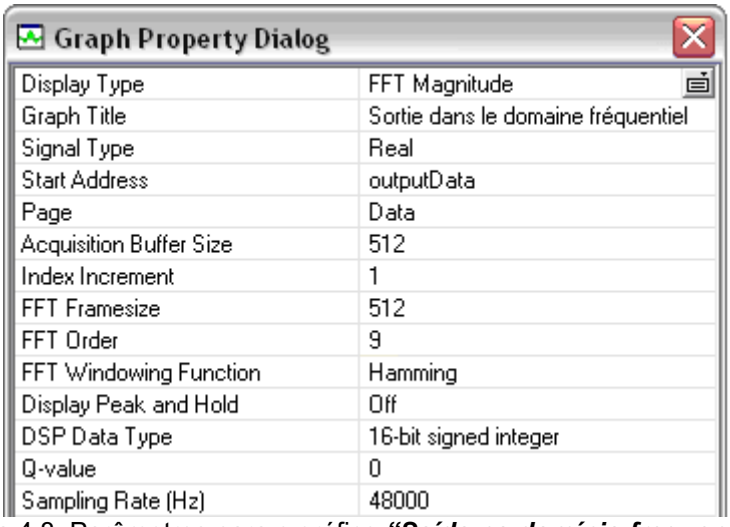

Figura 4.8: Parâmetros para o gráfico ***"Saída no domínio frequencial"***.

O MATLAB foi usado para criar uma entrada *"pseudo-analógica"* que pudesse ilustrar a influência do filtro em diferentes componentes de freqüência de um sinal. O sinal gerado pela rotina de teste foi, ao nível do CCS, a composição entre um seno de 1 kHz, outro de 8 kHz e um último de 10 kHz.

O ***"mode_entree = 0"*** 'foi selecionado para que usasse o sinal gerado no MATLAB. Os gráficos de entrada/saída no domínio do tempo são mostrados na Figura 4.9, enquanto que para o domínio da freqüência tem-se a Figura 4.10.

Ao observar a Figura 4.10, é possível verificar que o filtro reliza bem o seu papel: remover componentes de freqüência superior a 5 kHz. Se na entrada vemos as componentes de 1 kHz, 8 kHz e 10 kHz, na saída a única componente que resta é o de 1 kHz, conforme o esperado.

Estas observações podem ser feitas também no domínio do tempo, Figura 4.9: se para o sinal de entrada todos os três senos estão misturados, no final do tratamento a saída irá conter apenas o seno de 1 kHz.

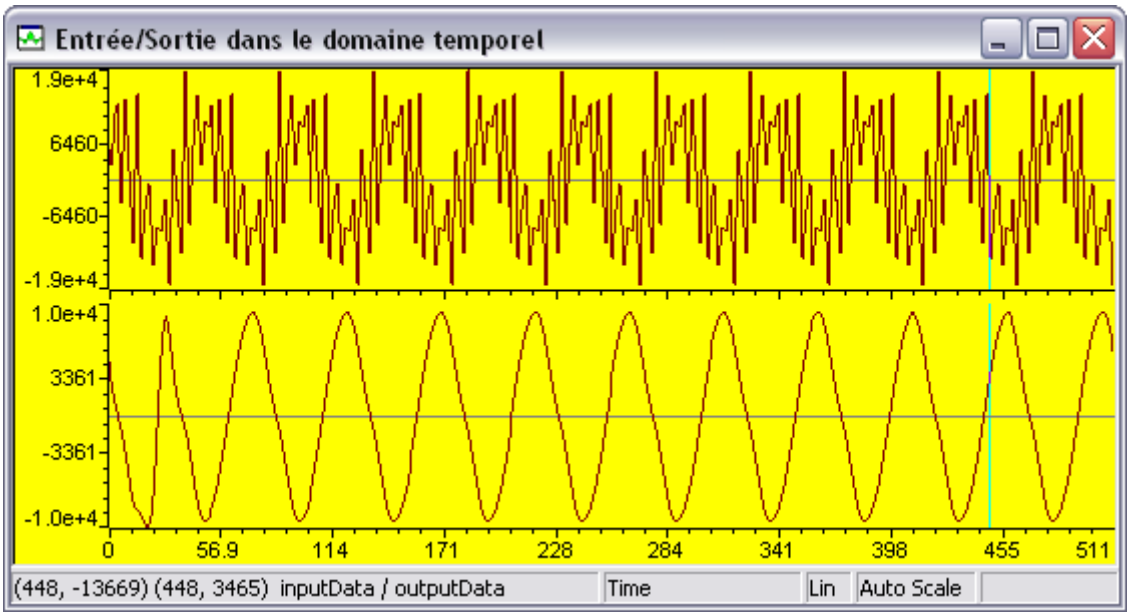

Figura 4.9: Entrada/Saída no domínio temporal para um sinal composto por três senos (1 kHz, 8 kHz e 10 kHz).

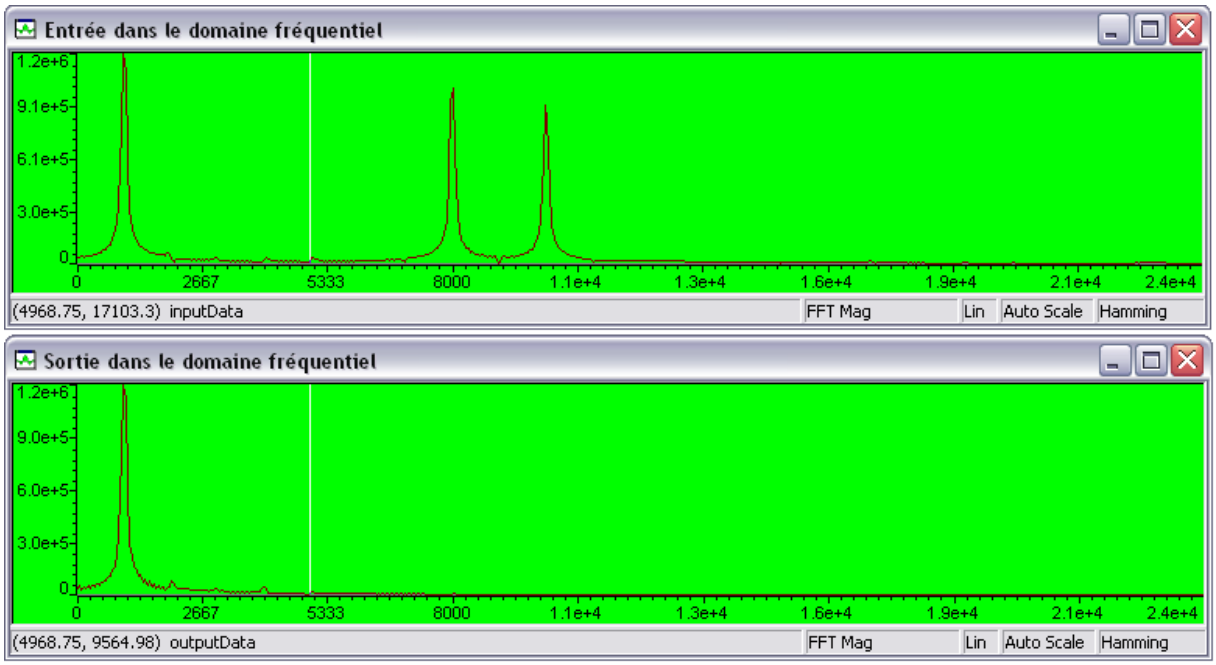

Figura 4.11: Entrada/Saída no domínio frequencial para um sinal composto por três senos (1 kHz, 8 kHz e 10 kHz).

Este resultado condiz com o que era esperado e ao que foi criado utilizando o SPTool. O último passo para confirmar essa validação é comparar a resposta impulsional do filtro concebido no DSP com a respota teórica do SPTool, Figura 4.12.

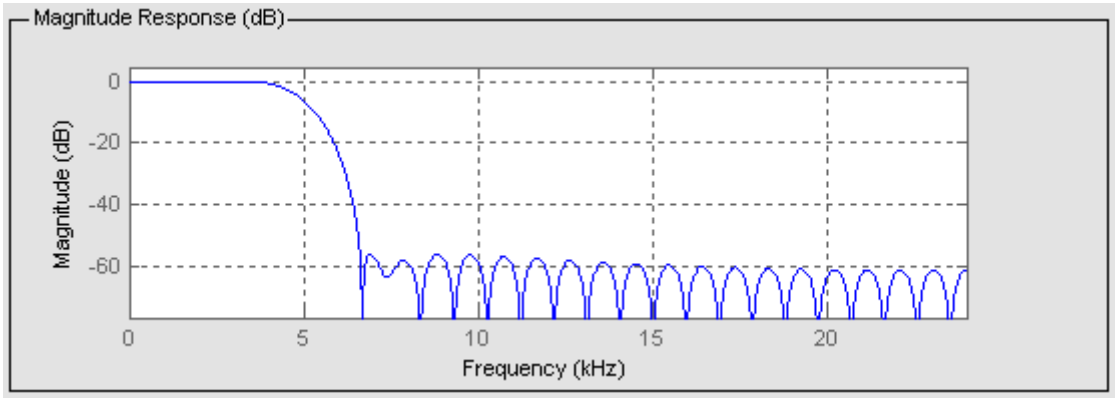

Figura 4.12: Resposta impulsional obtida para o filtro passa-baixas a partir do SPTool.

Para esta etapa, a variável ***"mode_entree"*** foi alterada para ***"1"*** em ***"Watch Window"*** para que a entrada fosse o impulso. Como o gráfico da saída do CCS é exibido em escala linear, foi necessário mudar o parâmetro ***"Magnitude Display Scale"*** para que o gráfico se tornasse logarítmico e que a comparação com a Figura 4.21 pudesse ser feita. A saída do filtro no domínio da freqüência é então dada pela Figura 4.13, coerente com a 4.12.

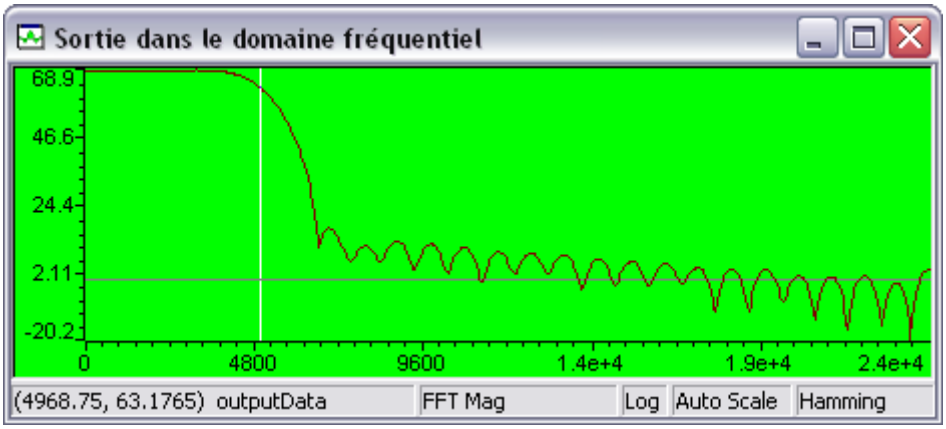

Figura 4.13: Resposta impulsional para o filtro passa-baixas implementado no DSP.

5. Transmissão Digital do Sinal

5.1 Modulação

O processo de modulação é definido como a transformação de um sinal em banda base que contém informações úteis para um novo sinal modulado de modo que ele se torne melhor adaptada ao meio de transmissão a ser utilizado. A modulação imprime o sinal de informação sobre um sinal adaptado ao meio de propagação.

O sinal que contém informações úteis, denominado modulante, é usado para modular um sinal de alta frequência, ou portadora, que normalmente é uma senóide cuja frequência é bem maior do que as componentes do sinal útil.

Um exemplo ilustrativo de modulação é mostrado na Figura 5.1, onde se tem, respectivamente, o sinal útil ***"m(t)"***, a portadora ***"p(t)"*** e o sinal modulado resultante ou sinal transmitido ***"s(t)"***.

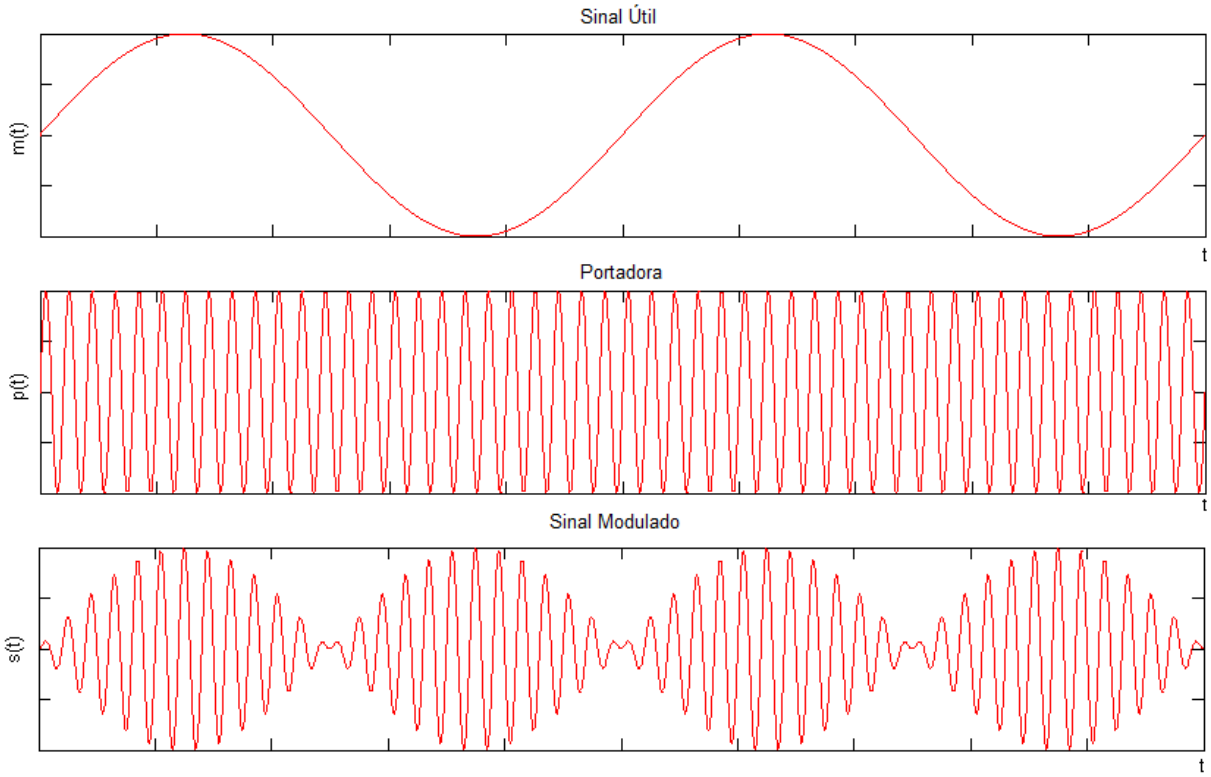

Figura 5.1: Exemplo de modulação onde ***"m(t) = sen(w_m.t)"***, ***"p(t) = sen(w_c.t)"*** e ***"w_c = 25.w_m"***.

Para este caso, utilizou-se a modulação em amplitude (AM) e a fórmula que define a relação entre estes três sinais é:

$$s(t) = m(t).p(t) \qquad (5.1)$$

Pode-se analisar esta equação no domínio frequencial utilizando-se a transformada de Fourier:

$$S(w) = M(w) * P(w) \qquad (5.2)$$

$$S(w) = M(w) * [\frac{\pi}{j}\delta(w - w_c) - \frac{\pi}{j}\delta(w + w_c)] \qquad (5.3)$$

$$|S(w)| = \pi[M(w - w_c) + M(w + w_c)] \qquad (5.4)$$

Por (5.4), pode-se notar que o efeito da modulação é de copiar as componentes do sinal modulante em torno da frequência da portadora, como visto na Figura 5.2, obtida a partir da transformada de Fourier da Figura 5.1

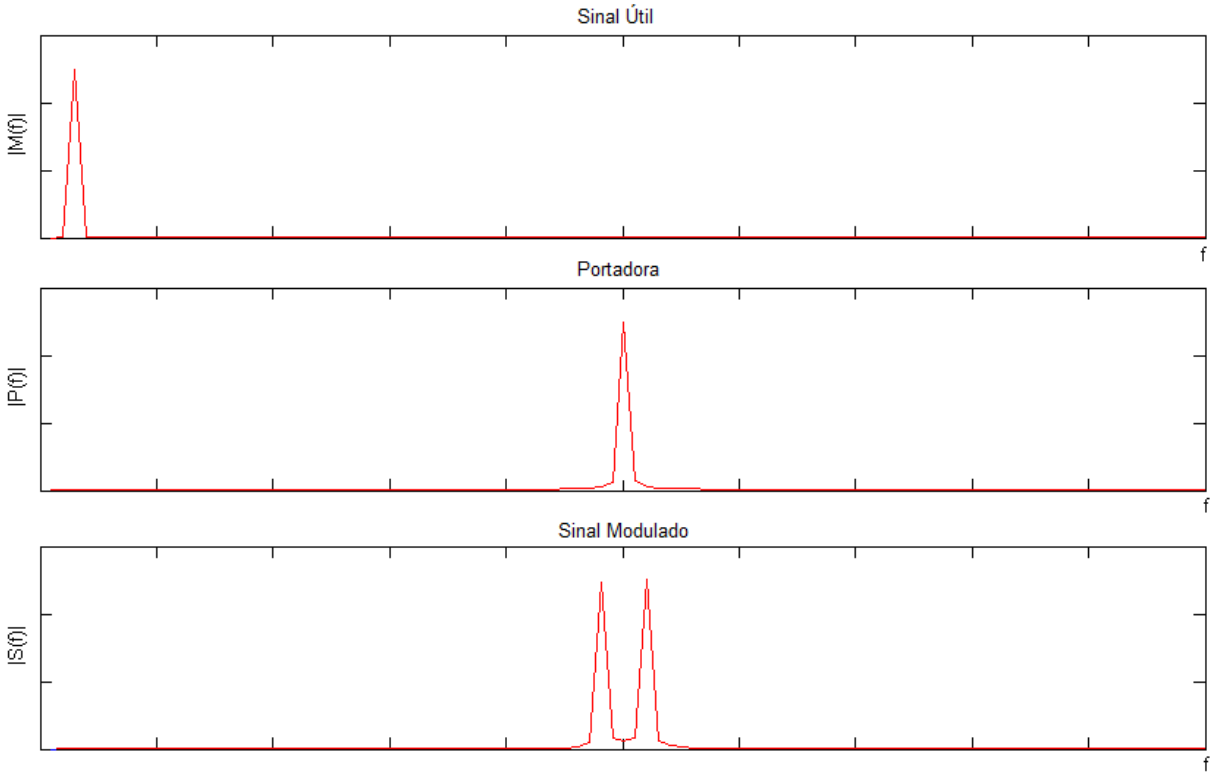

Figura 5.2: Análise frequencial do exemplo de modulação.

Devido a esta translação de freqüências, este método de transmissão de dados apresenta duas grandes vantagens sobre a transmissão direta em banda base:

- *Redução do tamanho das antenas de recepção e transmissão:* como o tamanho de uma antena é inversamente proporcional à frequência do sinal que ela deve ser capaz de receber, quanto maior é a frequência, menor é o seu tamanho. Como exemplificação, pode-se citar o caso de uma antena simples, o dipolo de meia onda. Para esta antena, o seu tamanho ***"L"*** é definido como:

$$L = \frac{\lambda}{2} = \frac{1}{2}\frac{c}{f}$$

onde ***"λ"*** é o comprimento de onda, ***"c"*** a velocidade da luz (≈300000 km/s) e ***"f"*** a frequência do signal.

O sinal modulante de áudio transmitido por uma estação de rádio FM varia de 100 Hz a 15 kHz. Para fazer a transmissão deste sinal diretamente em banda base (considerando como 7,5 kHz a freqüência média do sinal transmitido), a antena necessária para capturar este sinal deveria ter um tamanho de:

$$L_{banda\text{-}base} = \frac{1}{2}\frac{c}{f} = \frac{1}{2}\frac{3.10^8}{7{,}5.10^3} = 20km$$

Enquanto que para uma modulação em 100 MHz desse mesmo sinal, o tamanho da antena diminuiria consideravelmente:

$$L_{\mathrm{mod}ulada} = \frac{1}{2}\frac{c}{f} = \frac{1}{2}\frac{3.10^8}{100.10^6} = 1{,}5m$$

- *Partilha do espectro com outros sinais de mesma banda base:* estações de rádio FM são normalmente distribuídas na faixa de freqüência de 88 MHz a 108 MHz, cada um com uma faixa ocupada de 200 kHz. Sem modulação, o sinal de cada rádio em banda base seria misturado, ocasionando a impossibilidade da recuperação do sinal desejado [16].

Duas etapas são incluídas para a transmissão do sinal modulado: um transmissor deve realizar a modulação e um receptor a demodulação. Quando um mesmo equipamento é capaz de realizar as duas operações, é denominado de modem, contração para ***"MOdulador-DEModulador"***.

O objetivo do presente capítulo é de estudar as noções básicas da comunicação digital e implementar um modem QPSK (*Quadrature Phase Shift Keying*), utilizando o DSP. Para fornecer os conhecimentos básicos para a compreensão deste tema, a modulação analógica é introduzida e um exemplo de modem simples é desenvolvido, o modem AM.

5.2 Modulação Analógica e Digital

As formas de modulação mais conhecidas são as analógicas AM e FM já que as estações de rádio são projetadas para trabalhar com este tipo de modulação. Porém, estas formas analógicas "tradicionais" de transmissão da informação estão sendo, passo a passo, substituídas por sistemas digitais.

As vantagens dos métodos digitais envolvem a maior e mais confiável capacidade de transmissão. Fazendo com que a implementação de tais sistemas apresente um melhor custo-benefício.

Para todas as modulações, analógicos ou digitais, o processo de inserção das informações é obtido através da modificação de um parâmetro da portadora pelo sinal modulante. Este parâmetro pode ser a amplitude, a freqüência, a fase ou uma combinação delas. A Tabela 5.1 ilustra os principais tipos de modulação analógica e digital.

Parâmetro Modificado	Modulação	
	Analógica	Digital
Amplitude	AM (*Amplitude Modulation*)	ASK (*Amplitude Shift Keying*)
Frequência	FM (*Frequency Modulation*)	FSK (*Frequency Shift Keying*)
Fase	PM (*Phase Modulation*)	PSK (*Phase Shift Keying*)
Amplitude e Fase	QAM (*Quadrature Amplitude Modulation*)	QAM (*Quadrature Amplitude Modulation*)

Tabela 5.1: Comparação entre as principais modulações analógicas e digitais [17].

5.2.1 Modulação Analógica

Para a modulação analógica, os dois sinais envolvidos (modulante e portadora) estão em tempo contínuo e não haverá necessidade de passos intermédios de conversão de tempo contínuo em tempo-discreto como ocorrerá para a de modulação digital.

Os principais tipos de modulação analógica são a modulação em amplitude (AM), onde o sinal de informação é inserido na amplitude instantânea da portadora, e as modulações em ânuglo (FM e PM), onde o sinal de informação é posto na frequência ou na fase instantânea da portadora, respectivamente. A Figura 5.3 ilustra estas considerações. A Figura 5.1 ilustra a modulação AM, enquanto que a Figura 5.3 ilustra a modulação angular.

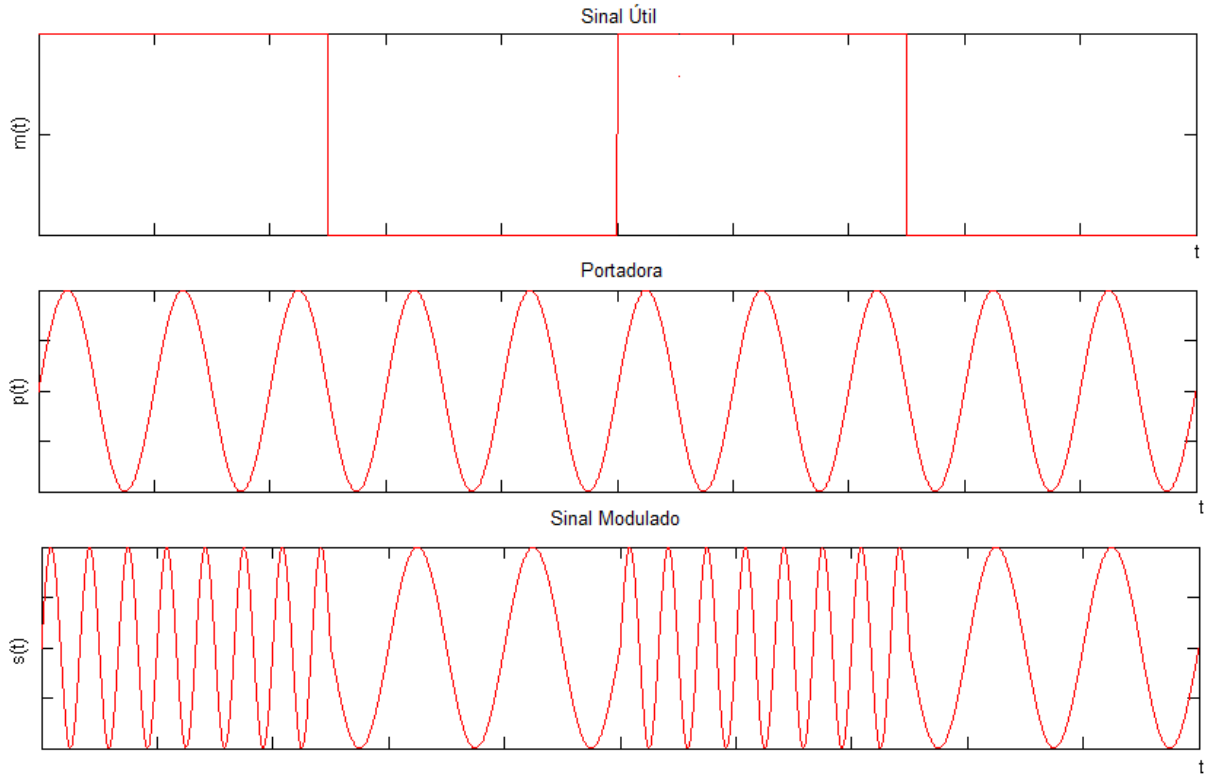

Figura 5.3: Exemplo de modulação angular para um sinal útil com dois níveis.

A modulação em amplitude é caracterizada por uma baixa dificuldade de execução e uma largura de banda necessária também baixa. No entanto, sua eficácia em termos de potência é pequena quando comparado aos métodos de modulação angular. Os métodos AM são ainda bastante utilizados para a transmissão de rádio/televisão, comunicação ponto a ponto e multiplexação telefônica. Os métodos angulares são mais difíceis de implementar, porém bem mais eficientes. Eles apresentam imunidade a ruído, o que facilita alcançar uma maior qualidade de recepção.

5.2.2 Modulação Digital

Assim como para a modulação analógica, os dados digitais também podem ser inseridos em uma portadora de várias maneiras. Os métodos de modulação digitais mais utilizados são o ASK, o FSK e o PSK.

5.2.2.1 ASK – Amplitude Shift Keying

O ASK é, entre todas as modulações digitais, a mais simples. Ela representa os símbolos ***"1"*** e ***"0"*** de um sistema digital pela presença ou ausência da portadora. A fase e a freqüência não são alteradas. A Figura 5.4 ilustra este método de modulação.

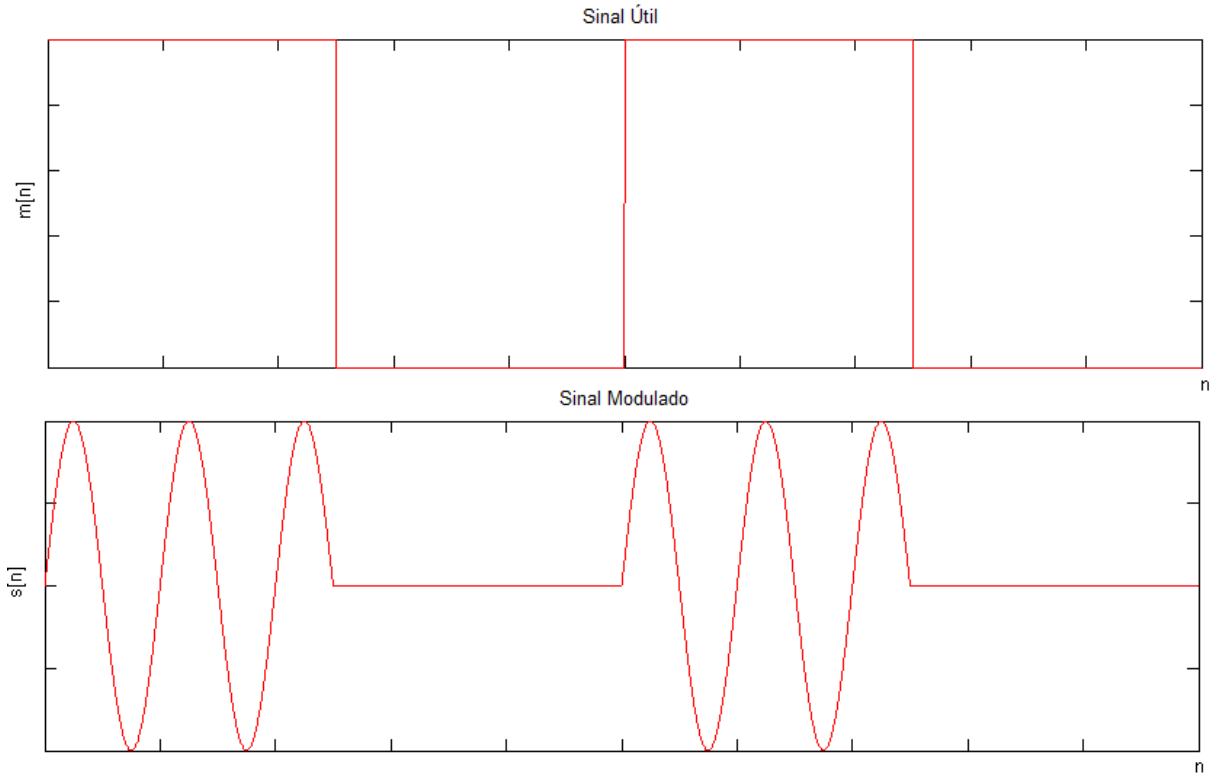

Figura 5.4: Modulação ASK.

É possível também aumentar a capacidade de transmissão de dados utilizando mais valores distintos para a amplitude. Se em vez de dois níveis diferentes para o sinal transmitido, utiliza-se quatro, pode-se associar cada um destes níveis a dois *bit*, o que dobraria a taxa de trnamissão em relação ao primeiro caso apresentado. Esta consideração é mostrada na Tabela 5.2, onde A_{max} é a amplitude máxima de saída. A Figura 5.5 ilustra essa modulação.

Amplitude da Portadora	Código Binário
A_{max}	11
$A_{max}/2$	10
$- A_{max}/2$	01
$-A_{max}$	00

Tabela 5.2: Relação entre amplitude/fase e *bits* para uma modulação ASK com quatro níveis (4-ASK).

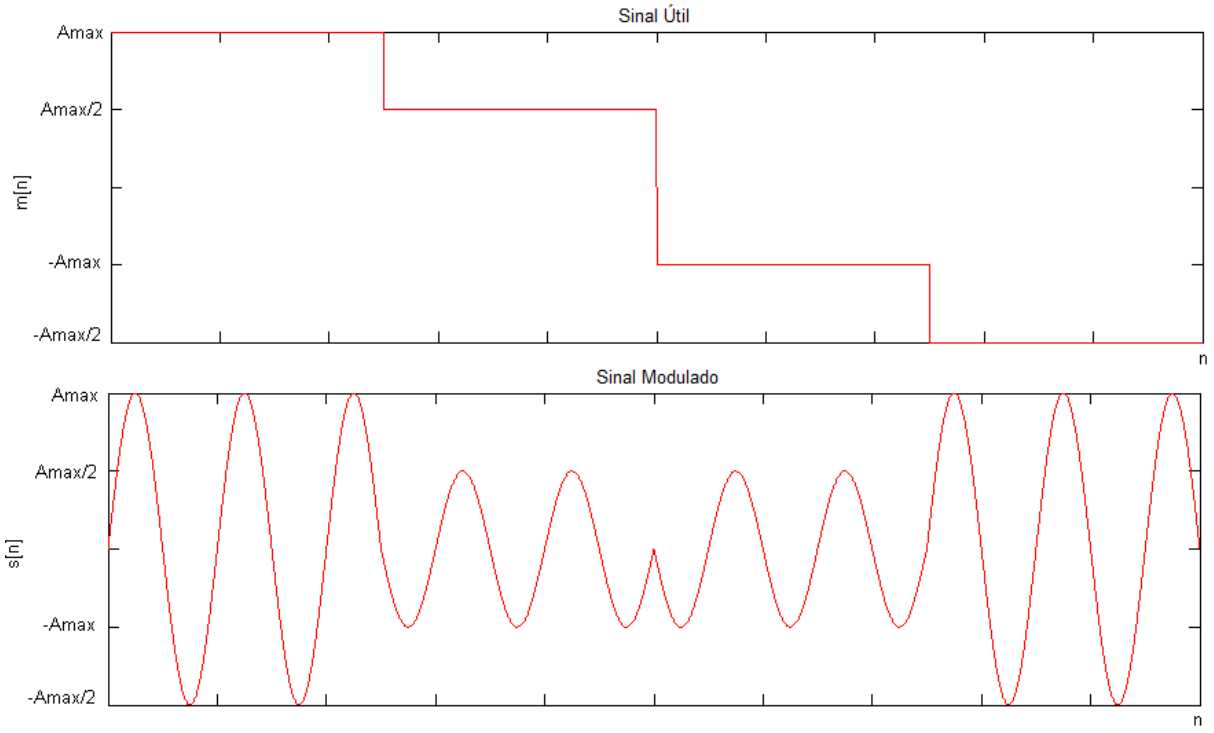

Figura 5.5: Modulação ASK com quatro níveis (4-ASK).

No tocante à demodulação do sinal, ao se utilizar o ASK mais simples (2-ASK), a envoltória do sinal transmitido pode ser usado diretamente para recuperar a informação útil. Uma filtragem simples e uma retificação permitem detectar esta envoltória. Para quantidades maiores de níveis de amplitudes (4-ASK, 8-ASK, etc.), o circuito necessário para demodular o sinal é mais complexo uma vez que o módulo de dois sinais distintos pode ser igual, como por exemplo, os bits ***"11 <-> 00"*** e ***"10 <-> 01"*** da Figura 5.5. Desta forma, deve-se conhecer não só a amplitude, mas também a fase do sinal a fim de recuperar os dados.

Deve-se também notar que quanto maior são os níveis de amplitude (e, por consequente, a taxa de transmissão de dados), maior é a dificuldade para o receptor demodular o sinal recebido e recuperar as informações enviadas. Esse problema é ainda pior quando na presença de ruído.

5.2.2.2 FSK – Frequency Shift Keying

Para a modulação FSK, os símbolos binários ***"1"*** e ***"0"*** são associados a duas frequências distintas de portadora. Para transmitir um ***"1"*** a frequência ***"F_1"*** é utilizada, enquanto que para transmitir um ***"0"*** a frequência ***"F_0"*** é associada, como mostrado na Figura 5.6.

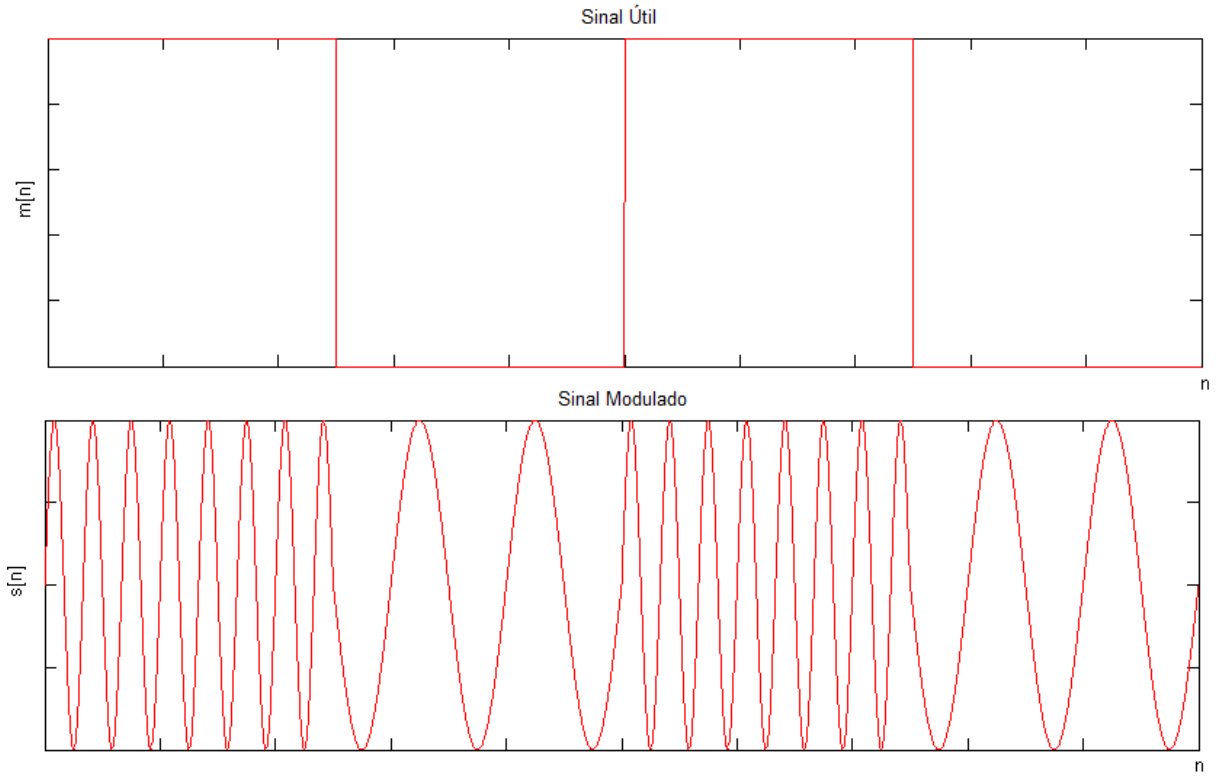

Figura 5.6: Modulação FSK.

5.2.2.3 PSK – Phase Shift Keying

Para esta modulação, a freqüência e a amplitude do sinal são mantidas constantes. A indicação do símbolo transmitido é observada através de uma mudança de fase no sinal modulado, como visto na Figura 5.7, onde uma mudança de 180° é observada para uma transição entre ***"1"*** e ***"0"***.

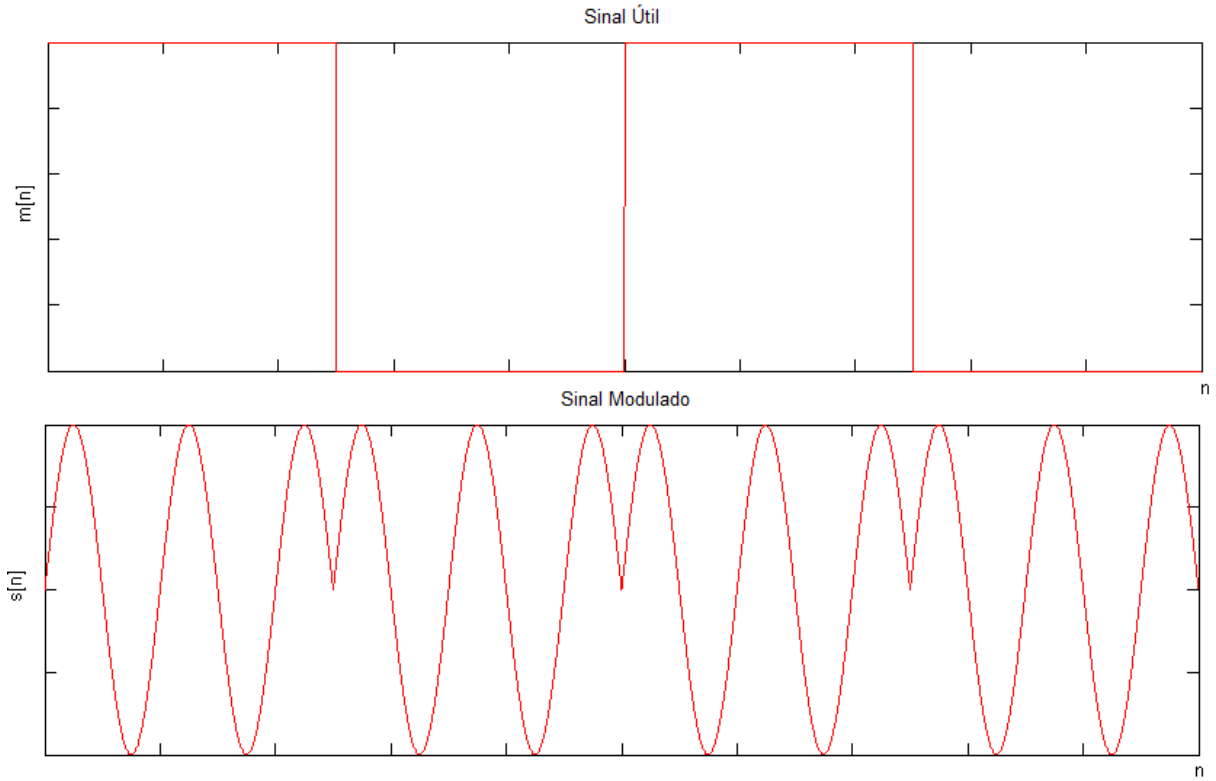

Figura 5.7: Modulação PSK.

Considerações Finais

A primeira parte do estágio consistiu na familiarização com a plataforma de desenvolvimento (*starter kit* TMS320VC5416 e *software* Code Composer Studio), além da implementação de aplicações simples, como efeito de eco e reverberação. Durante esta fase, observou-se um problema recorrente: a falta de documentação básica que nos permitisse construir aplicações realmente do zero. Muito embora a Texas Instruments ofereça uma vasta documentação para sua linha de DSPs, não há documentos relevantes que tratem tal problema. Desta maneira, a abordagem escolhida para desenvolver o projeto de estágio foi a busca de aplicações de baixo nível que nos permitissem utilizar o *hardware* do DSP de uma forma simples e assim criar uma camada de abstração de um nível superior para as aplicações propostas.

Para a etapa seguinte, dois filtros genéricos FIR e IIR foram elaborados e através dessas implementações pôde-se observar as principais características dos filtros digitais: filtros IIR mais adequado para o caso de especificações em amplitude e o FIR com sua fase perfeitamente linear, o que era um requisito essencial para aplicações como a transmissão da informação, assunto discutido na etapa seguinte.

A última parte consistia na implementação de um modem PSK. Foi realizado com sucesso o estudo teórico do tema e a primeira parte de sua codificação: a modulação. Os próximos passos envolveriam a execução do algoritmo de demodulação e o aumento da taxa de transmissão utilizando o sistema 4-PSK no lugar do 2-PSK utilizado naquele momento. Ao término do período do estágio os resultados encontrados para a demodulação não haviam sido satisfatórios, sendo a sua elaboração interrompida.

Como resultado final, uma documentação com quase 200 páginas foi elaborada, servindo de tutorial para os alunos ou profissionais que estejam adentrando nos domínios do DSP. Tal material possui um passo-a-passo da instalação e familiarização com o CCS, aplicações de base no tratamento de sinal, guia para a utilização do SPTool para obtenção dos coeficientes dos filtros digitais e introdução a transmissão da informação. Outro problema recorrente também é tratado neste documento, a configuração do CCS 3.3 para execução de projetos de versões anteriores. O presente relatório de estágio é uma versão resumida desta documentação.

Em uma visão mais geral, o projeto como um todo foi de valia para a formação profissional do estagiário, uma vez que o introduziu e o tornou apto a trabalhar com temas relevantes como o tratamento sinais e a transmissão digital da informação. Sem deixar de mencionar o ganho de maturidade no tocante a atividades de plano de projeto, escolhas de tecnologias e elaboração de projeto arquitetural. Havendo um aprimoramento na habilidade do estagiário em realizar estas tarefas que são de suma importância à sua formação acadêmica e ingresso no mercado de trabalho.

Apesar da última parte do estágio não haver sido totalmente concluída, o cliente se mostrou satisfeito com o trabalho desenvolvido, principalmente por se ter criado uma material de apoio sólido para o ensino de cursos práticos de tratamento de sinais e ter conseguido a adapatação de códigos para a nova plataforma de desenvolvimento de *software* da Texas Instruments, o CCS 3.3.

Através das ferramentas de colaboração e gerência, o cliente acompanhou e interagiu durante todo o desenvolvimento, ocorendo também reuniões semais com tanto o tutor da escola quanto o tutor da empresa.

Parte do conhecimento adquirido durante o curso de Engenharia Elétrica pôde ser posto em prática durante o desenvolvimento das atividades no laboratório. A integração com alunos, mestrandos e professores franceses com um vasto conhecimento em suas áreas propiciou um aprendizado extracurricular importante. A flexibilidade de horários de trabalho e o fácil acesso ao laboratório foram importantes, quando necessário para cumprir as tarefas determinadas.

No ambiente de trabalho, não foram identificados aspectos negativos significativos que merecessem ser aqui relatados. Talvez a falta destes aspectos fizesse com que este ambiente não fosse tão semelhante à maioria dos ambientes de trabalho comercial. Sendo assim, o estagiário não adquiriu uma noção mais realista de como poderia ser sua vida profissional no mercado de trabalho.

Referências Bibliográficas

[1] http://esisar.grenoble-inp.fr/75992650/0/fiche___pagelibre/&RH=SAR_PLAT – último acesso 13 de junho de 2009.

[2] http://esisar.inpgjc.com/references.php?id=5 – último acesso 13 de junho de 2009.

[3] http://intranetetu.esisar.inpg.fr/electronique/YvanDuroc/Divers/TraitementDSP.pdf – último acesso 13 de junho de 2009.

[4] G. Baudoin e F. Virolleau, “Les DSP Famille TMS320C54x Développement d'applications”, Dunod, Paris, 2000, Capítulo 4.

[5] ~\CCStudio_v3.3\docs\hlp\C5416DSK.HLP.

[6] http://dualist.stanford.edu/~ee265/labs/doc/CCS_usersguide.pdf – último acesso 13 de junho de 2009.

[7] http://intranetetu.esisar.inpg.fr/electronique/YvanDuroc/Divers/TraitementDSP.pdf, Capítulo 3 – último acesso 13 de junho de 2009.

[8] http://intranetetu.esisar.inpg.fr/electronique/YvanDuroc/Divers/projetAudio.rar – último acesso 13 de junho de 2009.

[9] http://uuu.enseirb.fr/~megret/Enseignement/DSP/TP/TP_DSP_Sujet.pdf – último acesso 13 de junho de 2009.

[10] http://focus.ti.com/docs/toolsw/folders/print/dspbios.html – último acesso 13 de junho de 2009.

[11] http://intranetetu.esisar.inpg.fr/electronique/YvanDuroc/Cours/EE530/Fichiers%20Support.rar - último acesso 13 de junho de 2009.

[12] http://www.youtube.com/watch?v=SVj-fMHxW0A – último acesso 13 de junho de 2009.

[13] G. Baudoin e F. Virolleau, "Les DSP Famille TMS320C54x Développement d'applications", Dunod, Paris, 2000, Capítulo 2.

[14] Steven A. Tretter, "Communication System Design Using DSP Algorithms with Laboratory Experiments for the TMS320C6713™ DSK", Springer, New York, 2008, Capítulo 3.

[15] R. Chassaing, "DSP Applications Using C and the TMS320C6x DSK", Wiley–Interscience Publication, New York, 2002, Anexo D.

[16] www.maua.br/arquivos/artigo/h/8c239ee2b10d164198633092eb65a2e9 - último acesso 13 de junho de 2009.

Estimação Financeira

Este estágio de fim de curso consistiu essencialmente de um estudo bibliográfico e concepção de programas para o tratamento digital de sinais utilizando DSP. Os custos do projeto podem ser divididos em três categorias:

- Custos de escritório (computador, eletricidade, cadeira, mesa, ar-condicionado, folha de papel, quadro, pincéis, livors, fotocópias, etc.);
- Custos de equipamentos técnicos (licença do *software* MATLAB e *Starter Kit* 5416 DSP);
- Bolsa do estagiário.

A estimação para os custos do presente projeto são mostradas na Tabela I.

Estimação Financeira	
Descrição das Despesas	Custo Estimado
Estrutura física da sala de pesquisa	500 €*
Material de escritório	100 €
Licença do *software* MATLAB	100 €*
Dois *Starters Kit* 5416 DSP	600 €
Bolsa do estagiário	1900 €
TOTAL	3200 €

Tabela I: Descrição dos custos do projeto de estágio.
*Preço equivalente a cinco meses de utilização.

RELATÓRIO DE ESTÁGIO
UFCG - INPG/ESISAR 2008/2009

Palavras Chave:

Tratamento do sinal, DSP, Comunicações, DSK5416, Filtragem Digital, PSK.

Resumo:

Os processadores digitais de sinais são utilizados dentro de uma larga gama de domínios como telecomunicações, automação, instrumentação, etc. Graças a essa grande quantidade de aplicações, estes dispositivos têm um lugar significativo dentro dos cursos universitários onde eles fornecem um meio de baixo custo para a introdução do tratamento digital de sinais em tempo real.

Este tema de projeto de fim de estudos apresenta o desenvolvimento de uma plataforma de estudos teóricos e práticos de DSP utilizado para a formação dos estudantes e também como uma vitrine da plataforma tecnológica SACCO. A família de DSP TMS320 foi utilizada para a implementação dos exemplos praáticos fundamentais a contrução de uma base sólida dentro do escopo da utilização de DSP. Tais aplicações incluem: conversão estéreo/mono, efeito de eco e reverberação, filtro FIR e modem PSK.

Keywords:

Signal Processing, DSP, Communications, DSK5416, Digital Filters, PSK.

Abstract:

Digital signal processors are used in a wide range of areas such as telecommunications, automation, instrumentation, etc. With this large amount of applications, these devices have a significant place in university's courses, providing a low cost way to introduce real-time digital signal processing.

This stage presents the development of a platform to a theoretical and practical DSP study that can be used for training students and also as a showcase for the SACCO platform. The TMS320 DSP family is used to implement practical examples that are necessaries to build a solid base to use the DSPs. This includes applications as: stereo to mono conversion, echo and reverberation filters, FIR filter and a PSK modem.