Universidade Federal da Bahia
Escola Politécnica
Departamento de Engenharia Elétrica e de Computação
Docente: Tiago Trindade Ribeiro
Discentes: Heverton Reis, Ivon Luiz, Jonh Brian Lemos, Matheus Souto e Vitor Cavalcante
Semestre 2024.2

Projeto Semestral

Relatório Final do Projeto: Reconstruindo os Efeitos da Mesa Digital Vedo/Teyun A8 Utilizando o Kit TMS320C5502 eZdsp

Universidade Federal da Bahia
Escola Politécnica
Departamento de Engenharia Elétrica e de Computação
Docente: Tiago Trindade Ribeiro
Discentes: Heverton Reis, Ivon Luiz, Jonh Brian Lemos, Matheus Souto e Vitor Cavalcante
Semestre 2024.2

1. Introdução

O trabalho teve como objetivo implementar, em um sistema embarcado com DSP, os efeitos sonoros de uma mesa de som Vedo/Teyun A8. Para isso, foi utilizado o kit TMS320C5502 da Texas Instruments, com o qual foram aplicados conceitos de Processamento Digital de Sinais (PDS) previamente estudados.
A aplicação foi desenvolvida em linguagem C, proporcionando experiência prática na implementação de técnicas de PDS em hardware. O desenvolvimento em um sistema embarcado aprofundou a compreensão sobre manipulação digital de sinais em tempo real.
Nós ficamos responsáveis pela reconstrução dos seguintes efeitos: REV-HALL1, REV-ROOM2, REV-STAGE B, REV-STAGE D, REV-STAGE F, RET-STAGE Gb, FLANGER e TREMOLO, presentes na mesa Vedo/Teyun A8.

2. Revisão Bibliográfica

Nesta seção, são apresentados brevemente os principais métodos utilizados na implementação dos efeitos de áudio.

2.1 Reverb

O algoritmo de reverberação implementado foi baseado no modelo de Schroeder, que utiliza quatro filtros comb e um filtro passa-tudo para simular reflexões sonoras. O filtro comb cria uma sequência de repetições do sinal original com atenuação progressiva, enquanto o filtro passa-tudo suaviza a resposta do sistema, garantindo uma distribuição mais uniforme das reflexões. A implementação utiliza uma cadeia de filtros para alcançar um som mais natural.

2.2 Flanger

O efeito de flanger foi implementado a partir da modulação do atraso do sinal original. O sinal é duplicado, e uma das cópias é atrasada por uma pequena quantidade de tempo variável controlada por uma função seno (LFO). Esse deslocamento e a combinação dos sinais gera interferências construtivas e destrutivas que produzem o efeito característico de oscilação.

2.3 Pitch Shifter

Para pitch shifting, foram adotadas técnicas baseadas em buffers de atraso variáveis. O sinal de entrada é armazenado em um buffer circular e lido com um deslocamento variável. A variação desse deslocamento, controlada por uma função periódica, altera a frequência percebida do sinal sem modificar sua duração. A interpolação linear é utilizada para suavizar as transições entre os valores das amostras.

2.4 Tremolo

O tremolo foi desenvolvido utilizando diferentes formas de onda para modulação da amplitude do sinal, fazendo que a amplitude do sinal altere entre -1 e 1 a cada meio ciclo. Por conseguinte, a intensidade do som e suavização das transições varia periodicamente de acordo com a forma de onda escolhida: parabólica, triangular ou quadrada.
Para obtermos essas formas de onda, partimos da onda quadrada, a qual foi integrada para gerar a onda triangular. Posteriormente, a integração da onda triangular resultou na onda parabólica. Com esse método, é possível criar efeitos sutis ou intensos, dependendo da taxa de modulação aplicada.

3. Metodologia

O desenvolvimento do experimento foi estruturado em seis etapas principais:

3.1 Estudo do hardware

Durante a matéria, diversas atividades práticas nos permitiram adquirir um conhecimento sólido sobre a utilização e desenvolvimento na placa, o qual foi reaproveitado no trabalho final. Dentre os conhecimentos adquiridos, podemos citar a leitura e escrita de arquivos WAV, o processamento de sinais com os buffers da placa, o uso de DMA para saída de áudio, entre outros. No entanto, para o desenvolvimento do trabalho final, além de utilizarmos a base de conhecimento adquirida, buscamos aprofundar nosso conhecimento, revisando manuais do fabricante e a literatura disponível, a fim de verificar se havia algum atributo que pudesse facilitar o desenvolvimento.

3.2 Implementação inicial em Python

Para entender como os efeitos da mesa foram criados, utilizamos a biblioteca SOX para manipular arquivos WAV, buscando aproximá-los ao máximo das referências fornecidas pelo professor. A SOX nos permitiu combinar diversos efeitos com facilidade,

auxiliando-nos na compreensão da combinação utilizada pela mesa. Constatamos, por exemplo, que alguns efeitos, como alguns tipos de reverb, não utilizavam apenas um método, mas combinavam diferentes elementos, como o pitch shifter. A partir da experimentação com a biblioteca SOX em Python, pudemos compreender melhor os processos a serem replicados em linguagem C para atingir os efeitos desejados.

3.3 Estudo teórico dos efeitos

Com o objetivo de realizar uma aplicação prática de PDS, estudamos a teoria por trás dos efeitos presentes na mesa, como a geração de ondas quadradas, triangulares e parabólicas, a modulação de sinais, a aplicação de atrasos e filtros, e a técnica de clipping. Por se tratar de uma aplicação que realiza a conversão de sinal analógico para digital, os efeitos da amostragem e quantização se mostraram especialmente relevantes, principalmente em altas frequências. O estudo de todos esses elementos foi crucial para a implementação dos efeitos em linguagem C.

3.4 Implementação em C++ e tradução para C

Para facilitar a manipulação dos efeitos em um nível mais alto, mas ainda visando um código portável para C, optamos por uma implementação intermediária em C++. Essa etapa nos permitiu um maior controle sobre os algoritmos antes da conversão final para C.

Posteriormente, portamos os código para C, aproximando mais a implementação do que seria aplicado na placa.

3.5 Aplicação no placa DSP

Ao contrário da execução em C em um computador, a placa apresenta algumas limitações em relação ao processamento, memória e buffers. Além disso, o compilador também impõe restrições, seja pela ausência de algumas bibliotecas e funções, seja pela dificuldade em lidar com determinados processos. Por essas razões, a aplicação do código em C demandou algumas adaptações para garantir seu funcionamento correto na placa.

3.6 Implementação em tempo real

A aplicação em tempo real consistiu na integração dos nossos códigos de efeitos, desenvolvidos em C, com o kit DSP. Essa integração visava proporcionar uma interação autônoma e intuitiva com a placa, permitindo o processamento do áudio diretamente no dispositivo.

3.7 Aproximação dos parâmetros

A complexidade em portar os efeitos para a placa, com todas as suas limitações, impediu a realização de um ajuste fino dos parâmetros para replicar com exatidão os resultados da mesa Vedo/Teyun.

No entanto, nosso objetivo era gerar um conjunto de dados com diversos áudios, gerados pelo nosso código em C com parâmetros aleatórios. Com esse conjunto de dados, realizaríamos um pré-processamento, extraindo características relevantes de cada áudio, como amplitude, energia, média, variância, espectrograma, gammatones, frequências Mel, entre outras. Essas características formariam um dataframe para o treinamento de uma rede neural para cada efeito. Dessa forma, a rede seria capaz de identificar, com um grau de precisão aceitável, os parâmetros empregados na geração de um determinado efeito, mas já no contexto da nossa implementação.

3.8 Síntese do Processo de Desenvolvimento

Os efeitos foram programados inicialmente em Python para facilitar o entendimento dos algoritmos e a análise de sua eficácia. Em seguida, foram implementados em C++ como uma transição para um código mais próximo do C, permitindo uma melhor adaptação ao DSP. Finalmente, a implementação em C foi realizada para compatibilidade com o Kit TMS320C5502.

4. Interface Humano-Máquina (IHM)

A interface homem-máquina (IHM) foi inicialmente projetada para execução em tempo real, utilizando os dois botões da placa para navegação entre áudios com efeitos aplicados dinamicamente. A ideia era usar um botão para avançar para o áudio seguinte e outro para retornar ao anterior, com a reprodução iniciando automaticamente após a alteração do efeito. A troca de áudio durante a execução seria gerenciada por interrupções acionadas pelos botões.

No entanto, como nossa implementação não ocorreu em tempo real, optamos por uma interação diferente. O botão direito passou a ser responsável por navegar entre os efeitos disponíveis, enquanto o botão esquerdo, por aplicar o efeito selecionado ao áudio.
O fluxo de interação ocorre da seguinte forma:

1. O usuário navega pelos efeitos disponíveis utilizando o botão direito.
2. Ao pressionar o botão esquerdo, o efeito selecionado é aplicado ao áudio.
3. O áudio modificado é salvo e o sistema entra em um estado “pronto”.

4. O usuário pode então escolher entre executar o áudio com o efeito aplicado (reproduzido na saída da placa) ou retornar ao menu de seleção de efeitos.

5. Desafios

Durante o desenvolvimento do experimento, algumas dificuldades foram enfrentadas:

- **Conversão de código Python para C:** Houve desafios na tradução dos algoritmos para C, especialmente na manipulação de buffers e alocação de memória.
- **Configuração do DSP:** A interface de desenvolvimento e os drivers da placa DSP apresentaram dificuldades de configuração inicial.
- **Execução em tempo real:** Ajustes na estrutura dos algoritmos foram necessários para garantir a execução eficiente dos efeitos sem latência perceptível.
- **Testes e depuração:** A avaliação da qualidade dos efeitos exigiu testes extensivos, ajustando parâmetros como ganho, frequência de modulação e resposta de filtros.
- **Otimização para tempo de execução na placa:** Foi necessário ajustar a eficiência computacional dos algoritmos para que pudessem rodar de forma eficiente no hardware embarcado, exigindo reduções na complexidade computacional e otimização no uso de memória.
- **Processamento em tempo real:** Optou-se por não utilizar processamento em tempo real devido às limitações computacionais do hardware, o que exigiria otimizações mais complexas.

6. Resultados

Os efeitos de áudio foram implementados em python, C++ e C, mas decidimos comparar apenas o desempenho entre as linguagens python e C:

- **Tempo de processamento em python:** 0,218 segundos.
- **Tempo de processamento em C:** 0,116 segundos.

A implementação em C mostrou-se mais eficiente, reduzindo o tempo de processamento pela metade aproximadamente.

Na placa, não chegamos a medir formalmente o tempo de processamento de um áudio. No entanto, análises informais com o arquivo piano8kHz.wav, com 38 segundos de duração, indicaram um tempo de processamento aproximado de 30 segundos, em diferentes execuções. Observamos que a variação no tempo de processamento não estava

diretamente relacionada ao efeito aplicado, mas ao tamanho do áudio, devido à necessidade da placa segmentar a informação em buffers para processamento.

É importante destacar que o áudio processado pela placa apresentou, em alguns efeitos como o flanger, ruídos em intervalos regulares. Esses ruídos provavelmente estão relacionados ao intervalo entre os janelamentos do método utilizado.

7. Conclusão

Este trabalho explorou a implementação prática de efeitos sonoros, inspirados na mesa Vedo/Teyun A8, em um sistema embarcado com o DSP TMS320C5502. O processo compreendeu desde o estudo teórico de efeitos como reverb, flanger e tremolo, até a prototipagem em alto nível, utilizando python, e a implementação final em C, visando compatibilidade e desempenho otimizado no sistema embarcado. A experiência de implementar os algoritmos em um ambiente com recursos limitados aprofundou a compreensão sobre desenvolvimento embarcado e processamento digital de sinais em tempo real.

Apesar dos desafios enfrentados, como a conversão de código entre plataformas, a adaptação aos recursos da placa e as limitações do compilador, os resultados executados na placa foram promissores. Como esperado, a implementação em C se mostrou consideravelmente mais eficiente que a versão em python, reforçando a utilização de linguagens de mais baixo nível para processamento de sinais em tempo real.. Por fim, as análises aproximadas do tempo de processamento no kit mostram a diferença de eficiência do processamento de dispositivos embarcados, reforçando a necessidade de otimização constante de código.

8. Referências

- Schroeder, M. R. (1962). "Natural Sounding Artificial Reverberation".
- Kuo, S. M., Real-Time Digital Signal Processing: Implementations and
- Application, Wiley, 2003.
- Mesa Digital VEDO/A8
- Teoria dos efeitos de áudio
- Implementação em alto nível: SOX
- Manual do kit TMS320C5502 eZdsp
- Processador DSP TMS320C5502
- Code Composer Studio (CCS) - Texas Instruments

9. Repositório

https://github.com/IvonLuiz/digital_table_effects