UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

HEITOR COLTRO DE ANDRADE

Implementação de um Algoritmo de Redução de Ruído para Aparelhos Auditivos Binaurais em FPGAs

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Engenharia da Computação

Orientador: Prof. Dr. Gabriel Luca Nazar
Co-orientador: Prof. Dr. Fábio Pires Itturriet

Porto Alegre
2023

AGRADECIMENTOS

Gostaria de agradecer primeiramente aos meus pais por me proporcionarem as condições para traçar o meu próprio caminho. Além disso, gostaria de agradecer aos meus amigos de longa data, por ajudarem a deixar os momentos tensos da faculdade mais agradáveis. Por fim gostaria de agradecer aos meus orientadores: Gabriel Nazar e Fábio Itturriet, por terem sido de imensa ajuda e estarem sempre disponíveis ao longo da realização desse trabalho.

RESUMO

Nos últimos anos, o uso de aparelhos auditivos vem crescendo constantemente. Ainda assim, há muitos desafios pendentes no desenvolvimento dos mesmos. Dentre estes destaca-se o problema de redução de ruído, visto que, ao utilizar-se técnicas comuns de ampliação do sinal, aumenta-se não só o áudio de interesse como também barulhos e ruídos ao redor do usuário, causando grande desconforto. Já existem soluções teóricas na literatura para esse problema. No entanto, tais soluções acabam sendo muito custosas devido à alta dificuldade de atender às restrições de área e energia que um aparelho auditivos proporciona, custo esse que acaba sendo passado para o usuário final. Todavia, os recentes avanços nas tecnologias de rede permitem uma abordagem de computação de borda, onde o processamento desses algoritmos poderia ser feito através de uma arquitetura dedicada, de forma a atender os requisitos de latência do problema. O presente trabalho apresenta a implementação de um sistema de redução de ruído para aparelhos auditivos binaurais. Para essa redução, foi utilizado um filtro de Wiener adaptativo multicanal implementado a nível RTL utilizando aritmética de ponto fixo.

Palavras-chave: FPGA. Filtro de Wiener. Redução de Ruído. Audição.

Implementatinon of a Noise Reduction Algorithm for Binaural Hearing Aids Devices in FPGAs

ABSTRACT

The use of hearing aids has been steadily increasing in recent years. However, there are still several pending challenges regarding their development. Among such challenges, the problem of noise reduction stand out because the use of common signal amplification techniques results not only the desired audio, but also the noise surrounding the user, causing discomfort. Theoretical solutions for this problem already exist in academic studies. However, such solutions end up often being too expensive due to the challenging nature of the area and energy constraints of hearing devices, which often translates into an increased cost for the end user. The recennt advances in network technology, however, would allow edge computing approaches to the problem, where such algorithms could be run on dedicated architectures, making it easier to comply with the problems latency requirements. This work shows the implementation of a noise reduction system for binaural hearing aids devices. The noise reduction was accomplished with a RTL implementation of an adaptive multichannel Wiener filter utilizing fixed point arithmetic.

Keywords: FPGA, Wiener Filter, Noise Reduction, Hearing.

LISTA DE ABREVIATURAS E SIGLAS

MWF	Multichannel Wiener Filter
FPGA	Field Programmable Gate Array
ASIC	Application Specific Integrated Circuit
HLS	High Level Synthesis
CLB	Configurable Logic Block
RTL	Register Transfer Level
LUT	Look Up Table
DSP	Digital Signal Processor
ILD	Interaural Level Delay
ITD	Interaural Time Delay
CPU	Central Processing Unit
VAD	Voice Activity Detector
FFT	Fast Fourier Transform
FF	Flip Flop
IFFT	Inverse Fast Fourier Transform

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A audição é essencial para a comunicação humana e, consequentemente, para a nossa convivência em sociedade. No entanto, esse sentido pode ter seu funcionamento prejudicado por fatores como exposição constante a ruídos de alta intensidade, doenças, acidentes, envelhecimento, etc. A depender do grau de perda auditiva, a depreciação da audição pode causar desde dificuldade para entender conversas em grupo e dificultar a convivência em sociedade a até mesmo riscos à própria saúde. Um estudo de 2012 (LIN; FERRUCCI, 2012) mostrou que adultos de meia idade com deficiências auditivas são mais propensos a sofrerem quedas, com a chance de sofrer o acidente aumentando em aproximadamente 1,4 vezes para cada 10 dB de audição perdida. Além disso, a perda auditiva prejudica a percepção dos ruídos ao nosso redor, o que pode nos colocar em situações de risco, como, por exemplo, ter dificuldade em escutar se um carro está próximo quando desejamos atravessar a rua.

Em muitos casos, a pessoa é afetada por problemas de audição em ambos os ouvidos, o que resulta em desafios adicionais para os aparelhos auditivos. Uma abordagem possível para aparelhos que funcionam de forma independente, chamados bilaterais, seria simplesmente amplificar todos os sons ao redor do usuário e os redirecionar para as orelhas do mesmo. Essa abordagem simplista, no entanto, contém alguns problemas. O primeiro seria o não discernimento entre ruídos e sons de interesse do usuário. A amplificação uniforme de todos os sons do ambiente consequentemente resulta em ruídos também muito mais altos, o que pode gerar desconforto e até mesmo confusão nos usuários. Outro problema é o fato de que, muitas vezes, os usuários não possuem uma perda auditiva uniforme em todo o espectro da frequência. Aumentar uniformemente todos os sinais sonoros faria com que faixas de frequência na qual o usuário escuta bem ficassem prejudicadas, sendo então necessário um filtro de áudio customizado para cada usuário. A principal desvantagem dessa abordagem, todavia, é a perda da localização das fontes de ruído.

De forma simplificada, o cérebro humano utiliza principalmente duas características do som para determinar a origem da fonte sonora: a diferença de tempo interaural (ITD em inglês), que é a diferença de tempo entre a chegada do som em cada orelha em relação a chegada na outra e a diferença de nível interaural (ILD em inglês), que consiste na diferença de intensidade do sinal entre as orelhas. Ao amplificar independentemente os sons de cada orelha, no entanto, os aparelhos auditivos bilaterais acabam alterando erro-

neamente a percepção dessas pistas, o que causa grande desconforto para os usuários. Por exemplo, ao conversar com uma pessoa, o usuário com esse problema sentirá que a voz da pessoa está vindo do seu lado, enquanto claramente vê a boca da pessoa a sua frente se movimentando. Outro caso bem conhecido é o chamado *cocktail party* (CHERRY, 1953), que consiste no efeito de ter vários ruídos a sua volta, como as conversas paralelas de uma festa. Para pessoas com esse problema, tais situações parecem como se os ruídos todos estivessem à sua frente, ao invés de à sua volta.

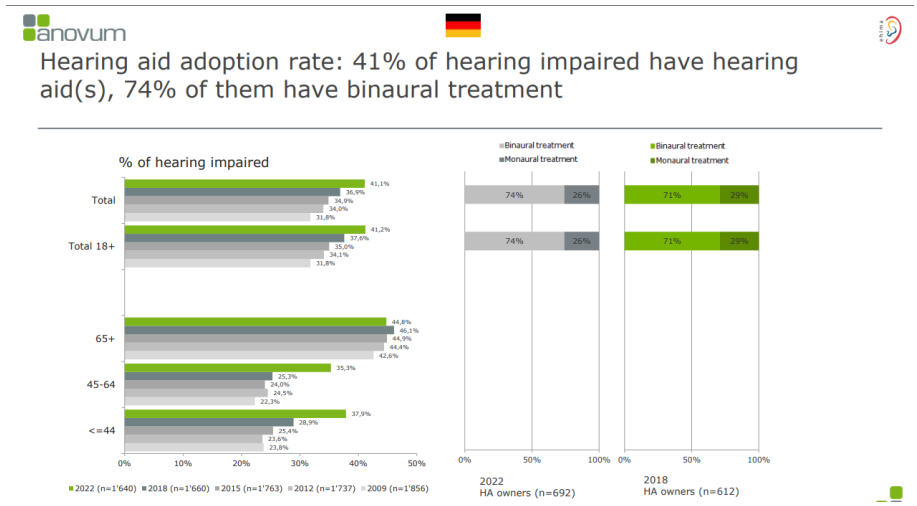
Para resolver esses desafios, são utilizados os chamados aparelhos binaurais, que se diferenciam dos anteriores por permitirem a comunicação entre os aparelhos de cada orelha, de forma a permitir técnicas mais avançadas de redução de ruído, amplificação eficaz dos sons e preservação das pistas de localização sonoras.

A *European Hearing Instrument Manufactures Association* (EHIMA), organização composta por seis dos maiores fabricantes de aparelhos auditivos do mundo, realiza a cada 3 ou 4 anos um estudo comparativo entre países europeus, denominado *Eurotrak*, a respeito do mercado de aparelhos auditivos. Em sua última versão, realizada em 2022, o estudo mostrou que há cada vez mais uma procura das pessoas com deficiências auditivas por aparelhos auditivos, com um aumento médio de 4,33% na adoção de aparelhos nos países analisados (Inglaterra, Itália, Dinamarca, França, Alemanha e Países Baixos), mantendo a tendência de crescimento contínuo da taxa desde 2009.

Entre outras estatísticas, o estudo também realiza uma pesquisa sobre a satisfação dos usuários em relação aos seus aparelhos. Na grande maioria dos países, a principal fonte de insatisfação dos usuários com os aparelhos bem como o principal motivo do não uso do mesmo estava relacionado a problemas de mal funcionamento na presença de ruído. Tais informações podem ser vistas nas figuras 1.1 e 1.2, retiradas da pesquisa da Alemanha de 2022 (EHIMA, 2022). É possível notar, ainda, a baixa taxa de melhora nas situações de insatisfação dos usuários. Essa demora no surgimento de melhorias nos aparelhos exemplificam os desafios dessa área.

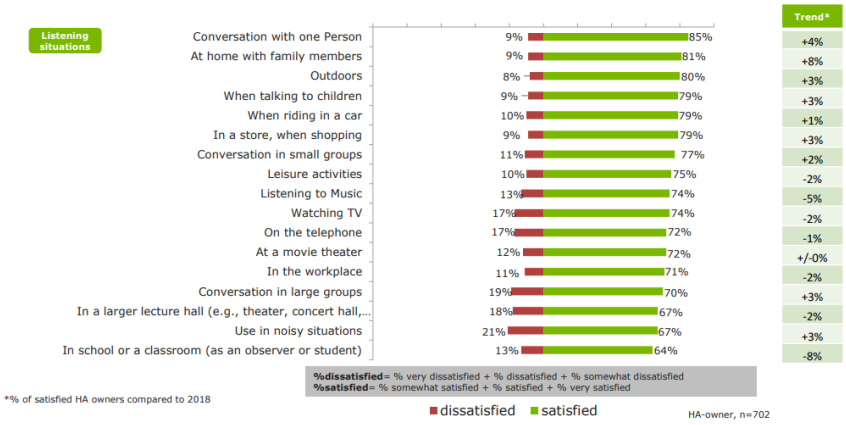
Não se sabe ao certo as tecnologias utilizadas nos aparelhos auditivos dos grandes fabricantes do ramo. No entanto, é seguro supor o uso de *hardware* dedicado nos mesmos devido aos restritos requisitos inerentes ao problema, os quais CPUs tradicionais, ou até mesmo DSPs (Digital Signal Processor), teriam dificuldades em atender. Projetos de aparelhos auditivos são de grande desafio justamente por terem duras restrições a todas as três variáveis de performance comumente usadas para analisar a qualidade de projetos de *hardware*: Energia, Desempenho e Área, popularmente conhecidas pela sigla em inglês

Figura 1.1: Porcentagem de uso de aparelhos auditivos por pessoas com deficiências auditivas



(EHIMA, 2022)

Figura 1.2: Taxa de satisfação de usuários de aparelhos em diferentes situações



(EHIMA, 2022)

"PPA"(Power, Performance, Area). Essa exigência em todas as três variáveis restringe significativamente a possibilidade de *tradeoffs* de uma variável para a outra, dificultando significativamente o projeto de *hardware* para essa aplicação.

As restrições relacionadas a energia decorrem da necessidade do uso de baterias nos aparelhos. É esperado que a bateria dos mesmos dure um dia inteiro, no mínimo, para que a recarga não seja um empecilho no dia a dia dos usuários. Dessa forma, é essencial que o *hardware* utilizado consuma pouca energia. Outra grande dificuldade no quesito energia está relacionada à restrição de área: utilizar baterias de grande capacidade, o que a princípio seria uma solução simples para o problema de energia, tornaria o equipamento grande demais ou pesado demais para ser usado confortavelmente, violando os requisitos de área e volume.

O principal desafio de desempenho relacionado ao projeto desses dispositivos é

a latência: os aparelhos auditivos devem ter uma resposta muito rápida de forma a não causar desconforto aos usuários. De acordo com o estudo (BURWINKEL; MCKINNEY; GALSTER, 2017), a latência máxima considerada aceitável em aparelhos auditivos é de $10m$s. A latência do aparelho começa a ficar desconfortável para o usuário a medida que fica aparente a diferença de tempo entre o som escutado e o acontecimento observado que gerou o gerou. Por exemplo, um usuário certamente se sentiria desconfortável se, ao derrubar algum objeto no chão, ele escutasse o barulho gerado somente depois de ver o objeto tocando o solo. Esse problema é ainda mais grave durante conversas, pois muitas pessoas, especialmente aquelas com problemas auditivos, utilizam da leitura labial para entender melhor o que a outra pessoa está falando. No caso de uma latência elevada, no entanto, esse hábito passa a ser uma desvantagem, pois a dessincronização entre lábios e palavras escutadas dificultam o entendimento das mesmas. Além das outras variáveis já discutidas (energia e área), outro fator que dificulta o cumprimento dos requisitos de latência são os algoritmos utilizados para o processamento de fala, que pode utilizar múltiplos microfones acoplados e consequentemente demandar alto poder de processamento.

Uma das técnicas mais exploradas nos últimos anos é a de filtragem Wiener multicanal (MWF, do inglês *Multichannel Wiener Filter*). Essa abordagem utiliza características estatísticas do sinal de áudio para separar a fala do ruído, o que se mostra uma abordagem mais robusta do que outras técnicas populares, como as baseadas em filtragem espacial (*beamforming*), que necessitam de uma estimativa da origem espacial da fala, e, consequentemente, possuem seu desempenho severamente reduzido no caso de uma estimação incorreta. A técnica de MWF, no entanto, pode introduzir erros na localidade sonora do áudio filtrado. Embora existam estudos sobre métodos para mitigar esse problema (ITTURRIET, 2019), sobretudo utilizando as características de ILD e de ITD citados anteriormente, tais métodos fogem do escopo do presente trabalho.

Outro problema relacionado ao uso do MWF é a sua complexidade de processamento. Aparelhos auditivos devem ser pequenos e leves de modo a prover um uso confortável para seus usuários. Tais requisitos, aliados à necessidade de existir uma bateria acoplada aos aparelhos, limitam consideravelmente a capacidade de processamento capaz de ser embarcada nos mesmos.

Uma forma de contornar esse problema seria usar um hardware dedicado para esse processamento como dispositivo para computação de borda (*edge computing*), de forma a reduzir a quantidade de processamento necessário nos aparelhos auditivos em si, o que tornaria o projeto dos mesmos mais flexíveis. Essa abordagem possui vantagens e

desvantagens. As vantagens, além da diminuição do nível de processamento do lado do usuário, consistem na possibilidade de lidar com o processamento de múltiplos usuários com uma quantidade menor de dispositivos. Afinal, sem a limitação extrema de área dos aparelhos auditivos, seria possível que um *hardware* lide com o processamento de múltiplos usuários simultaneamente. A principal desvantagem, por outro lado, consiste na latência adicional gerada pela necessidade de transmissão dos dados processados e das requisições, que passam a ter uma dependência de fatores externos ao *hardware*, como velocidade da rede e distância entre usuário e o local do *hardware* dedicado. Tal problema, no entanto, vem se tornando menos impactante devido aos avanços na área de redes. Tecnologias como a rede 5G, por exemplo, estão permitindo a crescente popularização da computação de borda nos mais diversos dispositivos eletrônicos.

Um método prático de implementar a abordagem de computação de borda seria o uso de FPGAs (*Field Programable Gate Array*) para o projeto do *hardware* dedicado de processamento. Tais dispositivos possuem a capacidade de serem reconfiguráveis, o que permite eventuais atualizações nos algoritmos de processamento sem a necessidade de troca dos aparelhos pelo usuário. Além disso, o projeto em um FPGA é muito mais barato do que os feitos em um ASIC (*Application Specific Integrated Circuit*) e são mais baratos que os mesmos para pequenas quantidades, o que provavelmente seria o caso para dispositivos colocados na borda, ao contrário de caso fossem embutidos nos aparelhos auditivos.

Dessa forma, o presente trabalho tem como objetivo a implementação do algoritmo MWF para redução de ruído em áudios. Será utilizada a linguagem de descrição de *hardware* Verilog bem como a ferramenta Vivado para fazer o projeto RTL do algoritmo tendo um FPGA como dispositivo alvo. Serão explorados os resultados obtidos quanto a latência e área em relação a viabilidade do projeto, bem como a sua eficiência no cumprimento dos requisitos da aplicação.

2 FUNDAMENTAÇÃO TEÓRICA

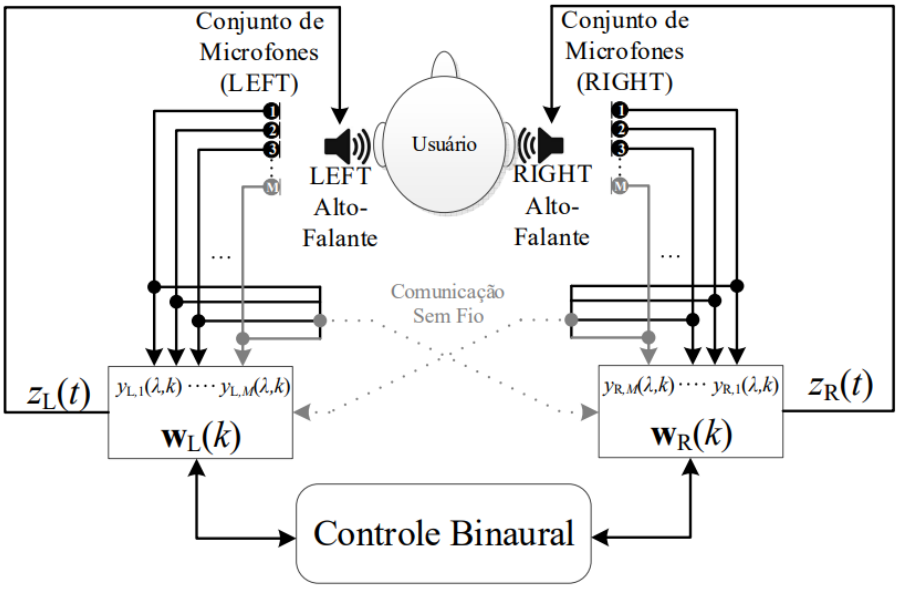
2.1 Filtro de Wiener Multicanal

O filtro de Wiener (WIENER, 1949) é uma técnica de filtragem classificada como uma técnica estatística. Técnicas estatísticas são muito utilizadas quando o sinal de interesse e o ruído podem ocupar as mesmas faixas de frequência, impossibilitando o uso de filtros seletores de frequência. Filtros estatísticos utilizam um sistema LTI (*Linear Time Invariant*) para produzir uma saída que seja a mais parecida possível com o sinal desejado. Para isso, é usado um sinal de erro de estimação para ajustar os coeficientes do filtro de forma a minimizar o valor desse sinal. Por definição, o filtro estatístico cujo sinal de erro de estimação é o erro quadrático médio é classificado como filtro de Wiener.

O filtro de Wiener multicanal já foi extensamente discutido na literatura em relação ao seu uso em aparelhos auditivos binaurais (ITTURRIET, 2019). O esquemático básico de um sistema binaural é apresentado na figura abaixo. O sistema consiste em *M* microfones para cada aparelho, sendo todos os sinais captados por ambos os lados compartilhados pelos dois aparelhos auditivos. O bloco *Controle Binaural* é responsável pela atualização dos coeficientes do filtros, denominados W_L e W_R, nos aparelhos através das entradas captadas pelos microfones. Os sinais filtrados Z_L e Z_R, as saídas do sistema, são enfim enviados aos seus respectivos alto-falantes, havendo um em cada aparelho.

Na figura 2.1 pode ser ver um esquemático do sistema. Cada microfone representa um canal de entrada do sistema, onde, para cada orelha, há um microfone chamado de referência. A utilidade dos mesmos se deve pois, durante o início do funcionamento do sistema, o projeto passa por um período de estimação, que consiste em um período inicial onde os coeficientes do filtro (representados na figura pelas letras $W_L(k)$ e $W_R(k)$) não são atualizados, somente as chamadas matrizes de coerência. Durante essa etapa, na ausência de coeficientes adequados para realizar a filtragem, o sistema apenas passa o sinal captado pelos microfones de referência para os canais de saída, que são os alto-falantes. O motivo dessa etapa de estimação é para que os coeficientes futuramente façam uma convergência mais rápida para os seus valores ideais. Vale notar também, na figura, as setas pontilhadas denominadas "Comunicação Sem Fio". Tais setas ilustram o fato de que o aparelho em cada ouvido utiliza as informações de ambas as orelhas (ou seja, tanto os sinais captados por si mesmo quanto o do aparelho vizinho) para atualizar os seus coeficientes, sendo por esse motivo o sistema em questão classificado como binaural.

Figura 2.1: Esquemático de sistema binaural



fonte: Fábio Itturriet, 2019

Os sinais captados pelos microfones são passados para o domínio da frequência utilizando a técnica de STFT (*Short Time Fourier Transform*). Essa técnica se difere da Transformada de Fourier tradicional pois, ao invés de aplicar a transformada no sinal inteiro, o sinal (ainda no domínio do tempo) é primeiro separado em diversas partes menores, denominadas frames, antes de ser passado para o domínio da frequência. Essa técnica permite que tenhamos conhecimento das mudanças do sinal no domínio da frequência ao longo do tempo, algo que não é obtido com a transformada de Fourier tradicional.

Após a passagem para o domínio da frequência, o sinal de entrada de cada microfone *i* da esquerda (*L*) e da direita (*R*) pode ser representado pela seguinte equação, onde λ representa o frame e *k* representa a frequência:

$$y_{L,i}(\lambda,k) = x_{L,i}(\lambda,k) + v_{L,i}(\lambda,k)$$

$$y_{R,i}(\lambda,k) = x_{R,i}(\lambda,k) + v_{R,i}(\lambda,k)$$

o sinal *x* representa o sinal de fala e *v* o sinal de ruído. Agrupando os sinais de todos os microfones, temos:

$$\mathbf{y}(\lambda,k) = \begin{bmatrix} y_L(\lambda,k) \\ y_R(\lambda,k) \end{bmatrix} \tag{2.1}$$

onde o sinal $\mathbf{y}(\lambda, k)$ pode ser descrito como $\mathbf{y}(\lambda, k) = \mathbf{x}(\lambda, k) + \mathbf{v}(\lambda, k)$ com $\mathbf{x}$ e $\mathbf{v}$ sendo definidos de maneira similar ao vetor $\mathbf{y}(\lambda, k)$.

Utilizando um VAD (*Voice Activity Detector*) externo ao sistema, é possível detectar se o sinal de entrada em um determinado instante é composto por fala ou somente por ruído (em outras palavras, se $\mathbf{x}(\lambda, k) = \mathbf{0}$ ou não). Com isso, representamos as matrizes de coerência do sinal de entrada $\boldsymbol{\Phi_{yy}(k)}$, do ruído $\boldsymbol{\Phi_{vv}(k)}$ e da fala $\boldsymbol{\Phi_{xx}(k)}$ como:

$$\boldsymbol{\Phi_{yy}} = E\{\boldsymbol{y}(\lambda, k)\boldsymbol{y}^H(\lambda, k)\}$$

$$\boldsymbol{\Phi_{vv}} = E\{\boldsymbol{v}(\lambda, k)\boldsymbol{v}^H(\lambda, k)\}$$

$$\boldsymbol{\Phi_{xx}} = \boldsymbol{\Phi_{yy}} - \boldsymbol{\Phi_{vv}}$$

onde o operador $E\{.\}$ representa o valor esperado da variável dentro das chaves e $(.)^H$ representa o hermitiano transposto. A matriz de coerência do sinal de fala pode ser representada dessa forma pois consideramos o sinal de fala e o de ruído como independentes um do outro.

Em seguida, escolhemos um microfone de cada aparelho para serem os microfones de referência, sendo esses geralmente os microfones frontais ($i = 1$). O sinal de referência de cada microfone é representado como:

$$y_{L,ref} = \boldsymbol{q}_L^T \boldsymbol{y}(\lambda, k)$$

$$y_{R,ref} = \boldsymbol{q}_R^T \boldsymbol{y}(\lambda, k)$$

onde $\boldsymbol{q}_L$ e $\boldsymbol{q}_R$ são vetores de de seleção. Como estamos escolhendo apenas um microfone como referência, apenas um valores desse vetores será igual a 1 e todos os outros serão iguais a 0. considerando $M = 3$, temos:

$$\boldsymbol{q}_L^T = [1, 0, 0, 0, 0, 0]$$

$$\boldsymbol{q}_R^T = [0, 0, 0, 1, 0, 0]$$

Os sinais de saída $Z_L(\lambda, k)$ e $Z_R(\lambda, k)$, por sua vez, são calculados através da multiplicação dos sinais de entrada com os coeficientes do filtro, cujas dimensões são *2M*:

$$Z_L(\lambda, k) = \boldsymbol{w}_L^H(k)\boldsymbol{y}(\lambda, k)$$

$$Z_R(\lambda, k) = \boldsymbol{w}_R^H(k)\boldsymbol{y}(\lambda, k)$$

os sinais de saída são em seguida passados de volta ao domínio do tempo antes de serem enviados aos microfones dos aparelhos.

Por fim, temos a função de custo do MWF. A função de custo é utilizada para fazer uma estimativa do mínimo erro quadrático médio do sinal de fala em cada microfone de referência. A equação de custo é apresentada a seguir:

$$\boldsymbol{J}_{MWF}(k, \boldsymbol{w}_L(k), \boldsymbol{w}_R(k)) = \boldsymbol{E}\left\{ \left\| \begin{matrix} y_L(\lambda, k) \\ y_R(\lambda, k) \end{matrix} \right\|^2 \right\} \tag{2.2}$$

onde $\|\cdot\|^2$ representa a norma euclideana.

O filtro descrito resulta no algoritmo simplificado que pode ser visto no Algoritmo 1. Note que os valores de *initCoef* $= \{1, 0, ..., 0, 1, 0\}$, *corrMatrixCoef* $= 0,999$, *mwfCoef* $= eye(12, 12)$ (sendo eye() uma função padrão do MATLAB que retorna a matriz identidade da dimensão especificada) e *betaCoef* $= 0,02$ são constantes já definidas no algoritmo referência.

2.2 Field Programable Gate Array

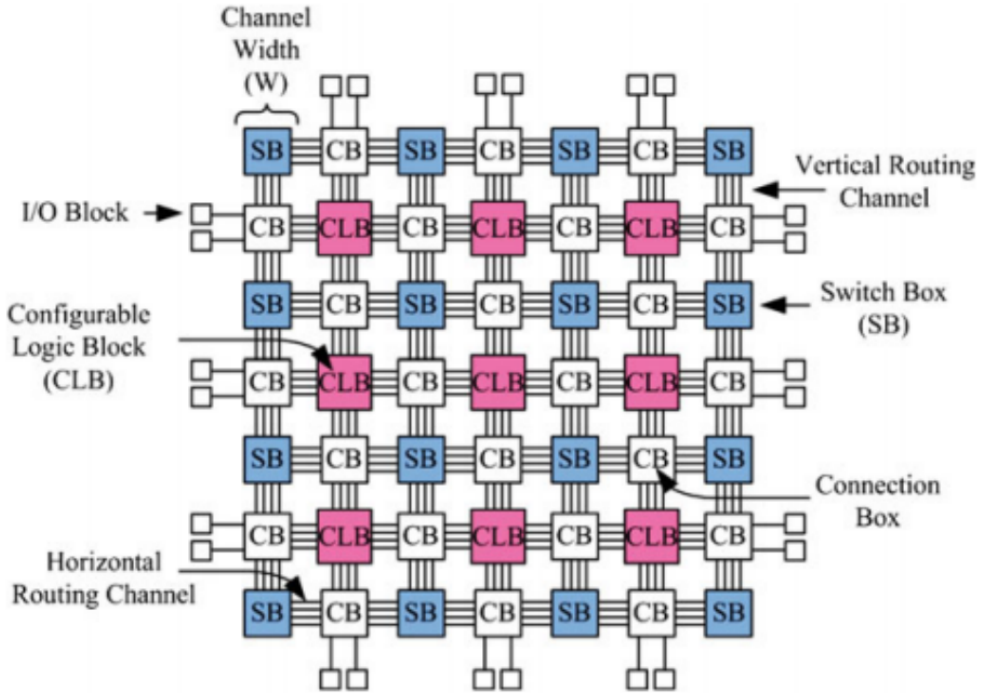
Um FPGA constitui um circuito integrado reconfigurável, ou seja, é capaz de ter a sua função modificada mesmo após a manufatura. FPGAs são compostos por blocos lógicos configuráveis (chamados de CLBs), interconexões entre os CLBs e portas de input e output, além de blocos de memória. A reconfiguração do circuito consiste em modificar a lógica implementada nos CLBs a serem utilizados bem como alterar as interconexões, de forma a conectar os CLBs entre sí e com as entradas e saídas. Um CLB consiste em uma LUT cuja saída está conectada a um multiplexador que, por sua vez, está conectado a um Flip-Flop (FF), de forma que pode-se escolher entre uma lógica combinacional ou sequencial. A estrutura básica de um FPGA pode ser vista na figura 2.2, enquanto a de um CLB pode ser visto na figura 2.3.

O fluxo genérico de desenvolvimento de uma aplicação em FPGA começa com a definição dos requisitos e arquitetura do sistema. Com isso concluído, a arquitetura projetada é descrita em uma linguagem de HDL (*Hardware Description Language*), em geral VHDL ou Verilog. Em seguida, após verificações funcionais do projeto em HDL, é

Algorithm 1 Algoritmo simplificado do filtro de Wiener Multicanal

$contFrame, contV, contY \leftarrow 0$
$frame \leftarrow \{0\}$
$filterCoef \leftarrow initCoef$
$frameVad \leftarrow 0$
for $amostra = 1 : NumAmostras$ **do**
  $frame \leftarrow frame[1:256]\&amostra$
  $frameVad \leftarrow frameVad|sampleVad$
  if $amostra\%64 == 0$ **then**
    $contFrame \leftarrow contFrame + 1$
    for $i = 1 : 256$ **do**
      $phase[i] \leftarrow (-1)^i * (0.9997 + 0.0245i)^{-contFrame*64*i}$
    end for
    $FreqFrame \leftarrow phase * fft(janela * frame)$
    if $contFrame <= EST\_MODE$ **then**
      if $frameVad == 0$ **then**
        $\Phi_{vv} \leftarrow \Phi_{vv} + FreqFrame * conj(FreqFrame)$
        $contV \leftarrow contV + 1$
      else
        $\Phi_{yy} \leftarrow \Phi_{yy} + FreqFrame * conj(FreqFrame)$
        $contY \leftarrow contY + 1$
      end if
      if $contFrame == EST\_MODE$ **then**
        $\Phi_{vv} \leftarrow \Phi_{vv}/contV$
        $\Phi_{yy} \leftarrow \Phi_{yy}/contY$
        $\Phi_{xx} \leftarrow \Phi_{yy} - \Phi_{vv}$
      end if
      $freqOutFrame \leftarrow FreqFrame[q_L] + FreqFrame[q_R]$
    else
      if $frameVad == 0$ **then**
        $\Phi_{vv} \leftarrow corrMatrixCoef * \Phi_{vv} + (1 - corrMatrixCoef) * FreqFrame * conj(FreqFrame)$
      else
        $\Phi_{yy} \leftarrow corrMatrixCoef * \Phi_{yy} + (1 - corrMatrixCoef) * FreqFrame * conj(FreqFrame)$
        $\Phi_{xx} \leftarrow \Phi_{yy} - \Phi_{vv}$
      end if
      $filterCoef \leftarrow mwfCoef * \Phi_{yy} * filterCoef + betaCoef * \Phi_{xx}$
      $freqOutFrame \leftarrow filterCoef * FreqFrame$
    end if
    $FinishedFrame \leftarrow janela * ifft(conj(phase) * freqOutFrame)$
  end if
end for

Figura 2.2: Esquema básico de um FPGA

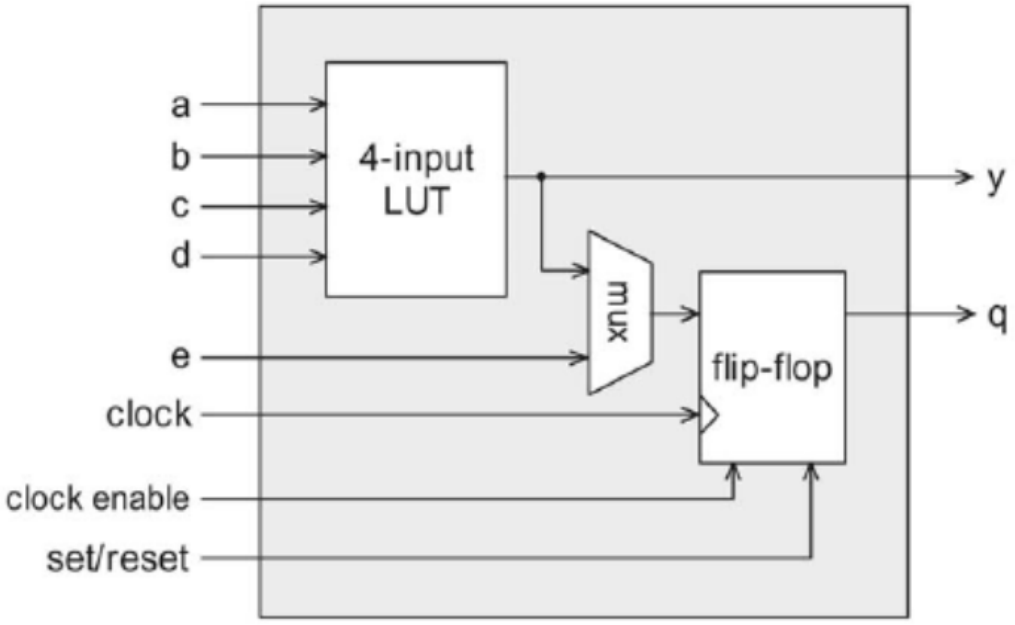


fonte: (FAROOQ; MARRAKCHI; MEHREZ, 2012)

utilizado um software para que o mesmo passe pelo processo de síntese, que consiste em traduzir a linguagem HDL para interconexões (chamados *nets*), LUTs (*Look Up Tables*, unidade básica de implementação de lógica), *flip-flops*, entre outros, que serão usados no FPGA. No final, esse processo gera um arquivo chamado *netlist* contendo o circuito lógico do projeto. Nesse ponto, há o processo de implementação, em que o *netlist* gerado anteriormente é mapeado para o FPGA. Nessa etapa também é onde ocorre a verificação dos requisitos de tempo estipulados (*timing constraints*), ou seja, se as frequências de clock escolhidas para projeto respeitam os caminhos críticos da arquitetura.

Por fim, o processo gera um arquivo final chamado *bitstream*, que é carregado no FPGA, fazendo com que o FPGA passe a se comportar como o especificado no projeto. Devido à sua alta capacidade de processamento paralelo, os FPGAs se mostram uma ótima opção de aceleração de algoritmos que exigem alto processamento de dados. No contexto do presente trabalho, essa capacidade permite o processamento simultâneo de cada canal do aparelho auditivo individualmente, o que permite uma redução valiosa na latência do sistema e consequentemente um maior conforto para o usuário.

Figura 2.3: Esquema básico de um CLB



fonte: (PONT et al., 2007)

2.3 Trabalhos Relacionados

O uso do MWF adaptativo já foi extensamente versado na literatura. Em (KUKLASINSKI, 2017), são exploradas as vantagens de se utilizar uma configuração binaural em relação a uma bilateral. Já em (CORNELIS, 2011), é feita uma análise teórica da abordagem binaural, bem como uma validação experimental do modelo utilizando métodos estatísticos para estimação de erros. De forma mais abrangente, (CORNELIS, 2010) explora variantes do algoritmo, sobretudo em relação à preservação das pistas acústicas binaurais, algo também extensamente explorado em (ITTURRIET, 2019).

Em termos de implementações, em (REIS et al., 2019) o filtro é implementado utilizando um DSP de modelo *ADSP-21489*, da empresa *Analog Devices*. O DSP em questão possui suporte para operações em ponto flutuante e aceleradores independentes para a transformada de Fourier. Outra implementação que merece destaque é a feita em (FRANCISCO, 2021), caso em que o algoritmo MWF foi implementado em um FPGA mediante ferramentas de síntese de alto nível e também se baseia em operações de ponto flutuante. Cumpre pontuar que o presente trabalho se diferencia dos demais citados principalmente devido ao uso de ponto fixo em suas operações bem como na implementação direta em RTL do projeto, o que permite um controle maior sobre os recursos utilizados pela placa.

3 PROPOSTA DE TRABALHO E METODOLOGIA

O presente trabalho propõe implementação alternativa às mostradas anteriormente, através de um projeto feito diretamente em RTL e com foco em computação de borda. O uso de computação de borda, embora nos imponha restrições de latência mais rígidas, nos permite aliviar as duras restrições de área encontradas no processamento diretamente feita no aparelho, nos permitindo um design que possa ser mais rápido e, consequentemente, nos permita compensar essa latência de transmissão adicional.

Neste capítulo serão discutidas as metodologias, estruturas e organização realizadas ao longo do projeto, cujo código fonte pode ser encontrado na referência (ANDRADE, 2023). O presente trabalho partiu de um código em MATLAB e outro em C++ (destinado para uso em ferramentas de HLS, embora nenhum módulo em HLS tenha sido utilizado no presente trabalho), já desenvolvidos no grupo de pesquisa. Tais códigos implementam o algoritmo do MWF e serviram como base para entender o algoritmo e suas operações. Para este projeto, foi utilizado o IP (*Intelectual Property*) de FFT (*Fast Fourier Transform*) próprio da Xilinx para a conversão entre os domínios de tempo e de frequência. A verificação do projeto quanto a corretude foi feita comparando os valores obtidos em simulação com os valores do código em MATLAB.

Ao contrário do algoritmo base, que utiliza ponto flutuante, a arquitetura criada utiliza valores em ponto fixo. Os principais motivos para essa decisão são o de melhorar a velocidade de execução e reduzir o uso de recursos físicos do FPGA. No entanto, é esperado que essas melhorias venham ao custo de uma perda na precisão dos valores calculados. Ao contrário da representação em ponto flutuante, que é capaz de representar uma grande extensão de valores com bastante precisão, a representação em ponto fixo apresenta uma troca entre boa precisão e bom alcance de representação, sendo essa troca definida pela posição do ponto decimal na palavra de bits: uma representação com mais bits reservados a parte inteira do que a fracionária resultará em uma arquitetura com melhor alcance de representação, porém com pouca precisão fracionária, e vice-versa.

A arquitetura inicial do projeto foi decidida em 32 bits devido ao fato desse ser o maior valor permitido pelos *cores* de FFT (32 bits para a parte real do valor complexo e 32 bits para a parte imaginária, ambos com representação em complemento de 2). Além disso, uma inspeção inicial sobre o os valores do arquivo de áudio de entrada mostrou que uma configuração de 12 bits para a parte inteira e, consequentemente, 20 bits para a parte fracionária seria adequada para atender o alcance e precisão requeridos pelo caso de

teste disponível, resultando em aproximações bem pequenas dos valores originais. Tanto o tamanho da palavra de bits quanto a distribuição dos mesmos entre parte inteira e fracionária foram feitas de forma parametrizável para permitir experimentos com configurações diferentes. O projeto foi criado através da ferramenta Xilinx versão 2021.1, utilizando o *device* "zc7a200tiffg1156-1L", sendo essa a melhor opção disponível na versão utilizada.

Mesmo com a configuração inicial sendo o bastante para representar adequadamente os sinais de entrada, o grande número de operações matemáticas, sobretudo de multiplicações, do projeto torna a opção de verificar o mesmo para os resultados dessas operações algo inviável. Com isso em mente, foram implementados nos multiplicadores complexos uma lógica de detecção de overflow e implementação de saturação. Essa capacidade permite que, em uma eventual ocorrência de estouro de representação, a corretude do sistema seja menos afetada. A operação de saturação consiste em substituir o valor da palavra que sofreu overflow pelo menor valor representável possível pela palavra caso o resultado correto seja negativo, e pelo maior valor possível no caso contrário. Como exemplo, no caso de de uma palavra de 32 bits com uma distribuição de 12 e 20 bits para as partes inteira e fracionária, respectivamente, um overflow de um valor positivo seria substituído pelo valor $2047,9999990463257$ e outro de valor negativo seria substituído por $-2048,0$. Ainda em relação aos multiplicadores, a quantidade dos mesmos instanciada foi decidida experimentalmente ainda no início do design do projeto. Inicialmente cogitou-se utilizar 64 multiplicadores. No entanto, os valores de recursos utilizados por essa quantidade foram considerados demasiados grandes. O teste seguinte, com 32 instâncias, foi considerado satisfatório para o uso no design. A quantidade de recursos foi obtida realizando somente a síntese dos multiplicadores.

Tabela 3.1: Recursos usados em diferentes quantidades de multiplicadores

nº de instâncias	*LUTs*	*FFs*	*DSPs*
32	14088	12802	512
64	124466	34024	740

Fonte: Acervo pessoal

A distribuição de bits para a parte inteira em relação à parte fracionária foi feita através de testes utilizando a biblioteca *fxpmath* (ALCARAZ, 2022). Os testes foram realizados comparando todos os valores do arquivo de input original do código MATLAB (em ponto flutuante) com seus valores em ponto fixo. Com a configuração utilizada, encontrou-se um erro máximo, em módulo, de $1,6181819879365607$. Para criar o testbench do projeto, os valores usados no Matlab foram passados para arquivos texto utili-

zando essa biblioteca, já passados para uma representação em hexadecimal de ponto fixo. Esses arquivos texto foram então usados para inicializar os valores iniciais de memórias e coeficientes, além do input do projeto.

As entradas do design criado correspondem a dois arquivos de texto: o primeiro representa os samples de áudio de entrada, com cada linha contendo 6 samples (um para cada microfone do sistema) de valor complexo, totalizando 283.552 samples. O segundo arquivo de texto contém o valor do sistema de VAD, que para o presente trabalho foi considerado externo ao sistema, atrelado a cada sample por linha, consequentemente tendo o mesmo número de linhas do arquivo de input.

Em concordância com o algoritmo original, o sistema projetado "executa" suas atividades a cada 64 amostras recebidas, sendo o envio das mesmas pelo testbench pausado até a conclusão das operações pendentes. O valor 64 se deve pelos valores de tamanho de frame, janela e overlap utilizados nos testes. Sendo o tamanhos dos frames 256 bits, o tamanho da janela 128 bits e a quantidade de overlap 50%, a quantidade de bits do frame de entrada que deve ser atualizada a cada iteração passa a ser 64. Os valores desses atributos foram mantidos os mesmos em todos os testes realizados.

Uma vez que o frame de entrada tenha sido atualizado, o mesmo é multiplicado sample a sample por uma janela antes de ser enviado ao *core* de FFT. A janela utilizada é uma variação específica da janela de Hamming . Umas vez que o frame esteja no domínio da frequência, este passa por uma mudança de fase (através de uma multiplicação) e logo em seguida, caso não tenha todas as amostras iguais a zero, o mesmo é usado para atualizar a matriz de coerência $\mathbf{\Phi}_{vv}$ no caso de ser um frame que só contenha ruído ou $\mathbf{\Phi}_{yy}$ no caso de haver pelo menos um sample que contenha voz (o nosso sinal de interesse). Durante um número inicial de frames pré estabelecido, denominado *estimation mode*, o frame em seguida passa por uma filtragem trivial, e é logo mais passado de volta para o domínio do tempo, sendo a filtragem trivial a multiplicação dos samples correspondentes aos microfones de referência por um, enquanto o restante é multiplicado por zero. Uma vez no domínio do tempo, a parte real do frame resultante é somada ao sinal de output resultante, utilizando um método conhecido como *overlap and add*. É importante notar que o arquivo texto do sinal de saída possui dois valores por linha, sendo um para cada alto-falante do sistema. Além disso, durante a fase de *estimation mode*, o sinal passado para a saída corresponde ao sinal captado pelos microfones de referência de cada lado.

Após o período de estimação, o funcionamento do sistema é levemente alterado. Uma das principais diferenças está na atualização das matrizes de coerência. Estas agora

são atualizadas também em função de um coeficiente adicional do sistema responsável por regular a relevância de valores anteriores das matrizes, sendo uma espécie de “fator de esquecimento” da mesma. Além disso, agora também é feita a atualização da matriz do sinal de fala $\mathbf{\Phi}_{xx}$ através da subtração da matriz de sinal total pela de ruído.

Após a atualização das matrizes, temos agora a atualização dos coeficientes dos filtros com base nas matrizes de coerência e logo em seguida a filtragem do sinal no domínio da frequência, utilizando os coeficientes recém atualizados. Finalmente, o sinal resultante é repassado para o domínio do tempo onde o mesmo método de *overlap and add* é utilizado. O algoritmo é concluído após todas as amostras tenham sido processadas e, no caso de um número de amostras não divisível por 64, são colocados zeros como uma espécie de preenchimento para que o último frame possa ser processado. O presetne projeto foi separado em 4 módulos principais, que são explicados em mais detalhes a seguir.

3.1 top_level

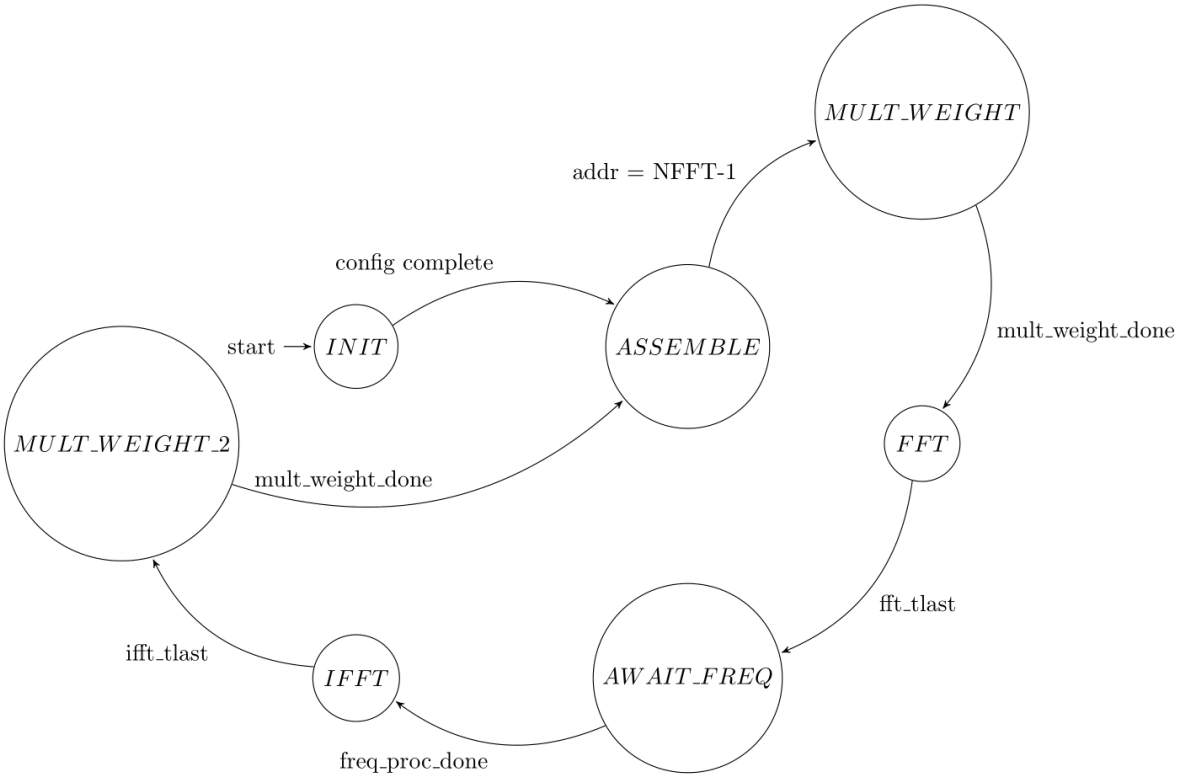
Sendo o módulo topo do projeto, este é responsável pelas operações realizadas no domínio do tempo, pelo controle dos *cores* de FFT e por receber os samples individuais do sinal da entrada e do arquivo de VAD e montar o frame, levando em consideração o tamanho de janela e a porcentagem de overlap entre os frames. Os estados criados para esse módulo levam em consideração as etapas realizadas no mesmo, que são: montagem do frame, multiplicação pela janela de análise, realização da FFT, espera pelo processamento em frequência, realização da IFFT (Inverse Fast Fourier Transform) e multiplicação pela janela de síntese. A etapa de espera pelo processamento em frequência é o estado em que é esperado a conclusão do processamento do módulo seguinte. Com exceção do módulo de divisão, que foi criado por Will Green de "Project F"(GREEN, 2020), todos os outros módulos são de autoria do presente autor.

A imagem 3.2 mostra o diagrama de estados que controla esse módulo. O estado inicial, chamado *INIT*, realiza a configuração inicial dos *cores* de FFT. Após essa configuração inicial, passamos para a montagem do frame no estado *ASSEMBLE*. A montagem de um frame consiste em descartar as 64 amostras mais antigas do frame antigo e concatenar as 64 novas amostras ao mesmo, criando assim o novo frame a ser processado. Uma vez que o frame esteja pronto, o mesmo é passado para o estado $MULT\_WEIGHT$, onde o frame é multiplicado pela janela (no projeto chamada de *weight*). Após o fim da

multiplicação, o frame resultante passa pelo *core* de FFT no estado seguinte, passando o frame para o domínio da frequência, onde finalmente o frame será processado no domínio da freqência pelo módulo frequency_proc, no estado *AWAIT_FRAME*.

Uma vez que os processos do módulo frequency_proc tenham terminado, o frame resultante é passado de volta para o domínio do tempo no estado $IFFT$. Uma vez no domínio do tempo, o frame é multiplicado novamente pela janela no estado $MULT\_WEIGHT\_2$, concluindo, assim, o processamento do frame.

Figura 3.2: diagrama de estados do top_level



fonte: Acervo pessoal

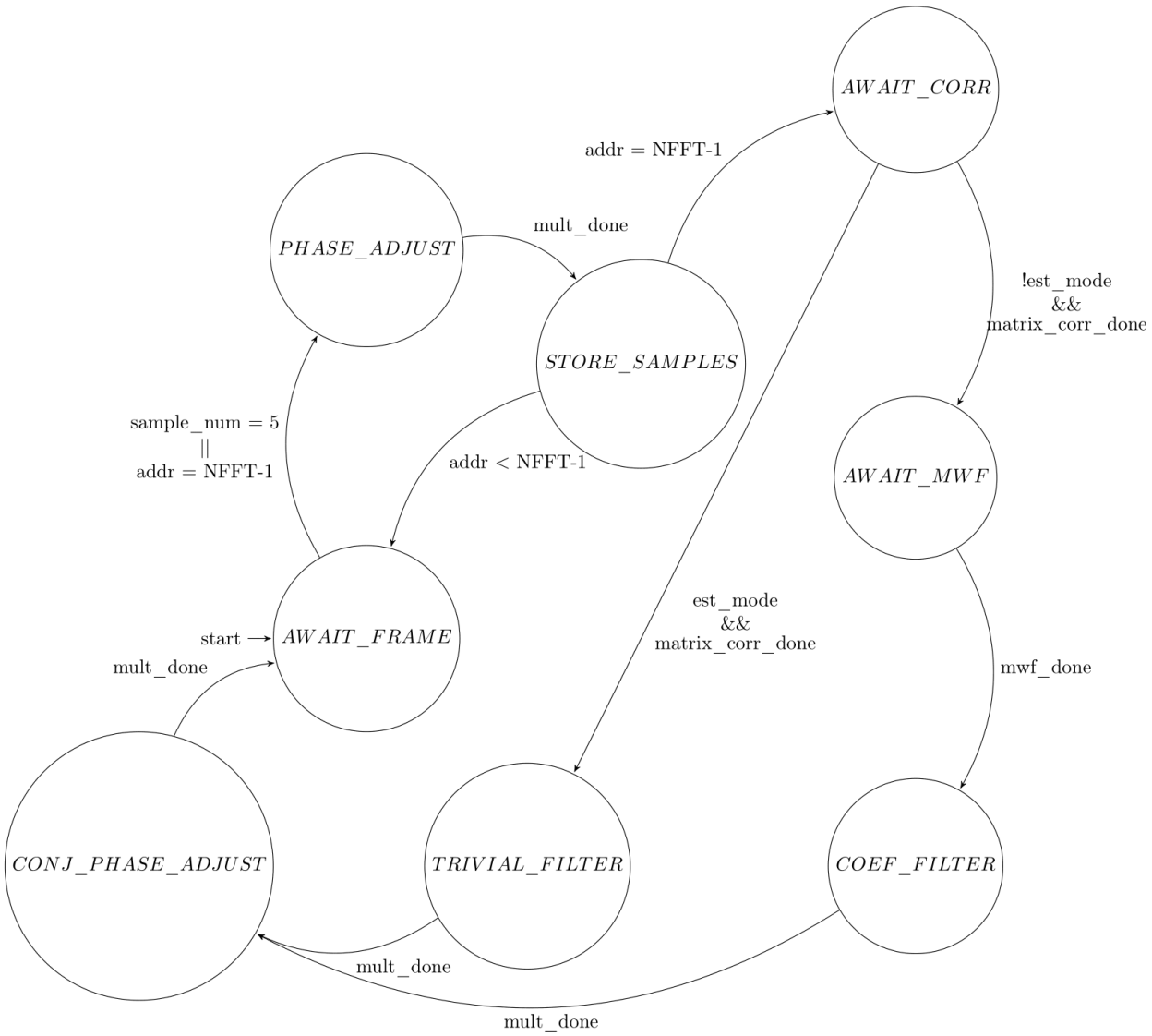
3.2 frequency_proc

O presente módulo engloba todo o processamento no domínio da frequência, incluindo os dois módulos subsequentes (matrix_corr e mwf). As operações do módulo frequency_proc começão após o frame ter sido passado para o domínio da frequência. Inicialmente, o frame de input é multiplicado por valores de fase constantes instanciados em um buffer, cujos valores armazenados são somente usados para leitura. Essa correção

de fase é feita em 5 amostras por vez, visto que cada amostra corresponde a concatenação de 6 valores, resultando em 30 multiplicações simultâneas por vez. Após o término dessas multiplicações, os valores resultantes são armazenados em um buffer que armazena o frame "corrigido". Essas etapas de coleta das amostras, multiplicação das mesmas pela fase e armazenamento do resultado em um buffer se repete continuamente nos estados $AWAIT\_FRAME$, $PHASE\_ADJUST$ e $STORE\_SAMPLES$, respectivamente, até que todo o frame tenha passado por esse processo.

A próxima etapa é a atualização das matrizes de coerência, que será explicada melhor na seção seguinte. Uma vez concluída essa etapa,o passo seguinte varia de acordo com o modo de operação do sistema: caso o mesmo esteja em modo de estimação, o sistema pula o processamento feito no estado $AWAIT\_MWF$ e realiza a chamada filtragem trivial no estado $TRIVIAL\_FILTER$. A filtragem trivial consiste em apenas repassar adiante os valores de amostras dos microfones de referência, sem nenhuma modificação. Caso o sistema esteja no modo de pós estimação, porém, o sistema realiza primeiro a atualização dos coeficientes do filtro no estado $AWAIT\_MWF$ (processo explicado melhor na seção do módulo mwf) e em seguida passa para o estado de $COEF_FILTER$, onde o frame é processado pelo filtro de fato. Independente do modo de operação realizado, o passo seguinte é feito no estado *CONJ_PHASE_ADJUST*, onde o frame resultante é multiplicado pelo valor conjugado da mesma constante de fase usada anteriormente. Finalmente, após a conclusão dessas multiplicações, o frame resultante está pronto para voltar ao domínio do tempo.

Figura 3.3: diagrama de estados do frequency_proc



fonte: Acervo pessoal

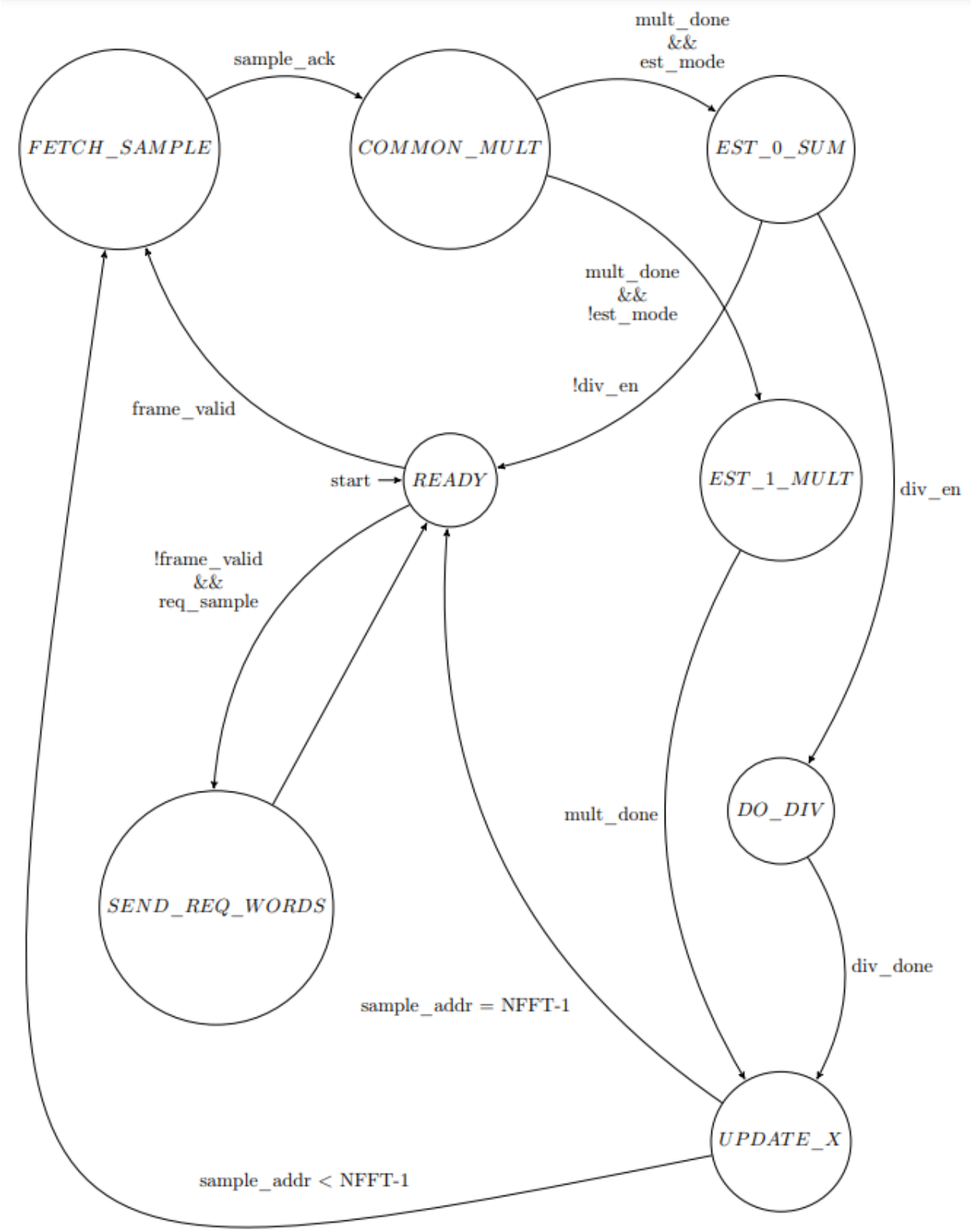
3.3 matrix_corr

O módulo matrix_corr é responsável pela atualização das matrizes de coerência. O processo de atualização começa depois que o módulo recebe o sinal *frame_valid* em 1, indicando que o frame foi atualizado. Primeiramente, o módulo faz requisições de algumas amostras do frame. Uma vez que isso tenha acontecido, as amostras passam por uma multiplicação no estado $COMMON\_MULT$. De forma semelhante ao módulo anterior, o presente módulo tem seu funcionamento diferente baseado no modo de ope-

ração atual do sistema. No caso do modo de estimação, o resultado da multiplicação de $COMMON\_MULT$, comum a ambos os modos de operação, passa por um processo de soma no estado EST_0_SUM, concluindo o processamento das samples e repetindo o processo até que o frame inteiro tenha sido processado. Na última ocorrência do modo estimação, ocorre a normalização dos valores das matrizes, no estado DOz_DIV, seguido da primeira atualização da matriz de coerência para a fala, no estado $UPDATE\_X$. No modo de pós estimação, após o primeiro processo de multiplicação, ocorre um segundo processo de multiplicação para que então o processo de atualização da matriz de fala seja realizada. Essa sequência de ações se repte até que todo o frame seja processado.

Após a atualização das matrizes, o presente módulo realiza a função de envio de informações das matrizes para o módulo de atualização dos coeficientes do filtro (explicado a seguir), através do estado $SEND\_REQ\_WORDS$. O diagrama de estados desse módulo pode ser visto na figura 3.4. Vale ressaltar, também, que as divisões realizadas nesse módulo são feitas através dos módulos de divisão em ponto fixo instanciados no matrix_corr, sendo os módulos de divisão feitos por Will Green (GREEN, 2020). O projeto RTL dos módulos de divisão é o único presente no projeto que não foi feito inteiramente pelo presente autor.

Figura 3.4: diagrama de estados do matrix_corr



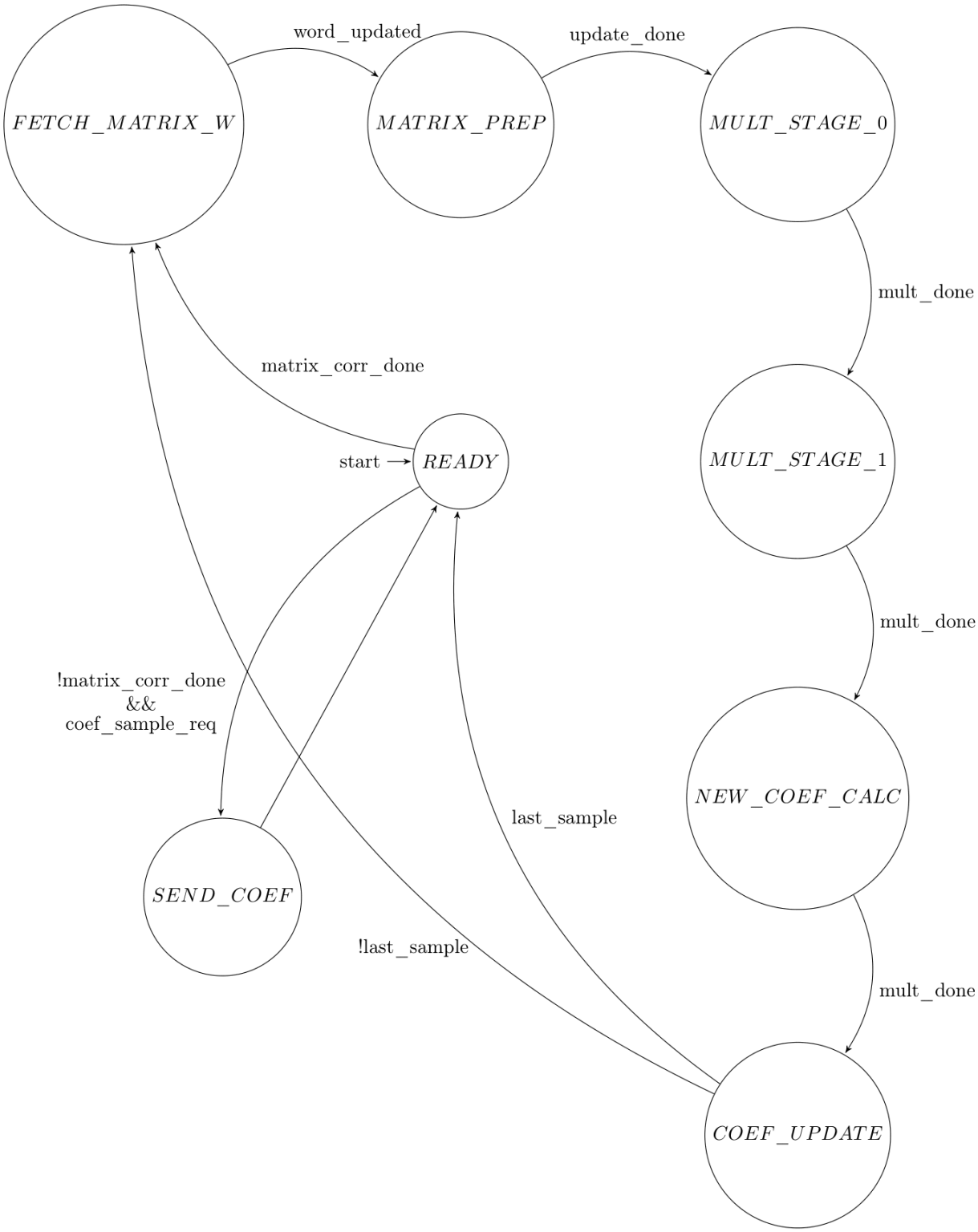
fonte: Acervo pessoal

3.4 mwf

O presente módulo tem como principal função a atualização dos coeficientes do filtro. Para isso, o módulo utiliza apenas informações das matrizes de coerência já atualizadas e os coeficientes anteriores, não sendo necessário dados do frame em sí. As atividades desse módulo se iniciam após o recebimento do sinal *matrix_corr_done*, indicando o término da atualização das matrizes. Após esse recebimento, o presente módulo solicita palavras das matrizes no módulo anterior no estado $FETCH\_MATRIX\_W$. Após o recebimento das palavras, o módulo passa para o estado $MATRIX\_PREP$, onde se monta uma matriz própria de dados, que não possui relação com as matrizes de coerência, necessária para as operações que virão a seguir. As operações realizadas em seguida são uma série de multiplicações e somas representadas pelos estágios subsequentes $MULT\_STAGE\_0$, $MULT\_STAGE\_1$ e NEW_COEF_CALC, onde os novos coeficientes ficam finalmente prontos. Em seguida, os coeficientes do filtro são atualizados pelos novos valores calculados no estado $COEF\_UPDATE$. Esse processo se repete continuamente até que todas as palavras das matrizes tenham sido utilizadas.

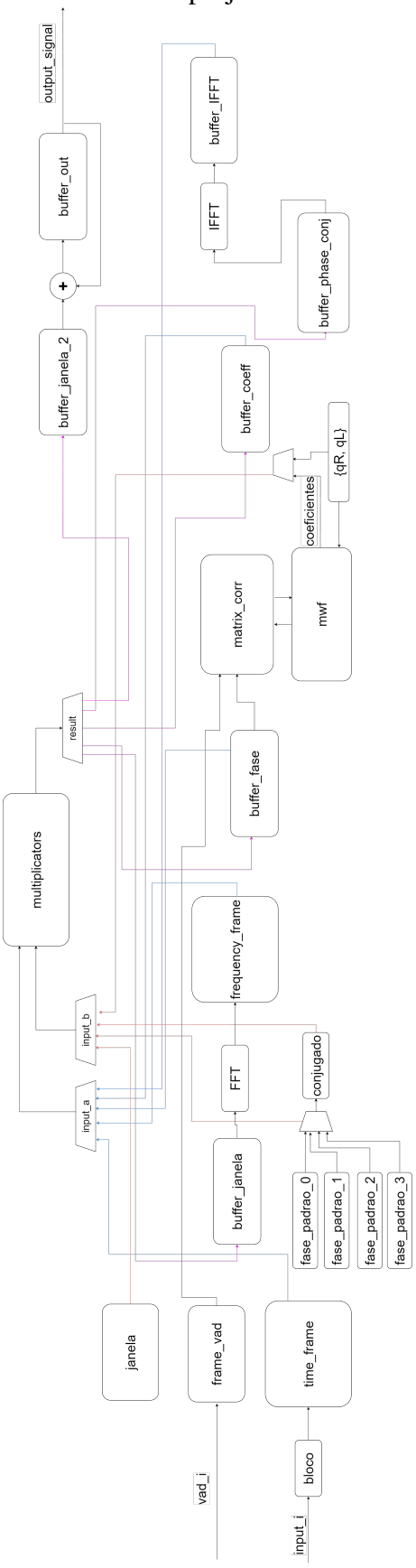
Uma vez que os coeficientes estejam completamente atualizados, o presente módulo passa a ter a função de prover continuamente os valores dos novos coeficientes para a filtragem realizada em seguida no módulo "pai", através do estado $SEND\_COEF$. Essa sequência de operações pode ser visualizada através do diagrama de estados da figura 3.5.

Figura 3.5: diagrama de estados do mwf



fonte: Acervo pessoal

Figura 3.1: esquemático do sistema projetado com os módulos apresentados



fonte: acervo pessoal

4 RESULTADOS EXPERIMENTAIS

A partir do projeto realizado, foram coletados dados quanto ao desempenho e recursos utilizados pelo mesmo. Os valores de desempenho foram coletados através de simulações lógicas enquanto os valores de utilização de recursos foram retirados dos relatórios de síntese na ferramenta *Vivado*.

Em relação ao desempenho, o processamento de um frame possui uma latência de 17.692 ciclos de clock quando em período de estimação, ou seja, quando ainda não são realizadas as operações de atualização dos coeficientes do filtro. Após esse período inicial, a latência de processamento passa a ser $26,696$ ciclos. Considerando uma frequência de clock de 50MHz, teremos uma latência de $353,84\mu$s e $533,92\mu$s, respectivamente.

Além disso, é preciso considerar o tempo de montagem do frame. Cada nova iteração do algoritmo deve receber 64 amostras antes de começar a sua execução. Ao todo, considerando o recebimento das novas amostras e o deslocamento das demais amostras necessárias, temos uma latência de 578 ciclos, ou $11,56\mu$s, para a montagem do frame. Os tempos de latência de cada módulo individualmente, para cada modo de operação, estão listados nas tabelas 4.1 e 4.2.

Tabela 4.1: latências por módulo, modo de estimação

módulo	*ciclos*	*tempo (μs)*
top_level	$3,558$	$71,16$
frequency_proc	$3,892$	$77,84$
matrix_corr	$10,242$	$204,84$
mwf	-	-

Fonte: Acervo pessoal

Tabela 4.2: latências por módulo, modo de pós estimação

módulo	*ciclos*	*tempo (μs)*
top_level	$3,558$	$71,16$
frequency_proc	$3,892$	$77,84$
matrix_corr	$14,848$	$296,96$
mwf	$4,398$	$87,96$

Fonte: Acervo pessoal

Analisando cada módulo individualmente, é possível perceber que o módulo de atualização das matrizes de coerência é o maior gargalo do sistema, com no mínimo $10,242$ ciclos. Essa limitação se deve pela grande quantidade de multiplicações complexas necessárias para a sua completude, sendo o processo feito para múltiplas matrizes.

Devido à natureza de tempo real da aplicação, buscou-se priorizar latência em relação a vazão para sistema. Mesmo no caso contrário, no entanto, algumas propriedades do algoritmo limitam o uso de técnicas que ampliariam a vazão, como o pipeline. O principal exemplo é a necessidade do conhecimento de todo o frame para atualizar corretamente as matrizes de coerência, o que impede o início do processamento desse bloco à medida que o frame está sendo montado, mesmo com alguns dados já disponíveis. Ainda assim, foram utilizados estágios de pipeline em algumas partes do código, especialmente em operações em matrizes.

Quanto aos recursos utilizados, a ferramenta de síntese apontou o uso de 579.981 LUTs, 896.232 Flip-Flops (FFs) e 480 DSPs, além de 2 BRAMs. Dos recursos utilizados, destaca-se o alto uso de FFs devido ao alto nível de paralelismo utilizado nas operações de matrizes, o que impede a ferramenta de síntese de implementar tais estruturas como memórias. Futuramente, planeja-se reduzir a quantidade de paralelismo utilizado de forma a obter um equilíbrio melhor entre os dois extremos. Em contrapartida, o valor relativamente baixo de DSPs utilizados é resultado da utilização de uma quantidade fixa de módulos de multiplicação instanciados, utilizando-se ao todo 32 multiplicadores complexos instanciados. Futuramente, pretende-se explorar um maior número de multiplicadores, e, consequentemente, de DSPs, de forma a agilizar mais as diversas operações aritméticas do algoritmo.

Tabela 4.3: recursos utilizados na síntese final do projeto

LUTs	*FFs*	*DSPs*	*BRAMs*
579981	896232	480	2

Fonte: Acervo pessoal

5 CONCLUSÃO

O presente trabalho apresentou a implementação do algoritmo do Filtro de Wiener Adaptativo em RTL com destino a um FPGA. No projeto realizado, buscou-se explorar, dentro do possível, o uso de paralelismo para acelerar a enorme quantidade de operações matemáticas do algoritmo. Para amenizar os grandes custos de área decorrentes dessa abordagem, foi utilizado um valor parametrizável de multiplicadores complexos, sendo o valor final escolhido experimentalmente. Também foram utilizados multiplexadores bem como máquinas de estados finitos para realizar o controle e sincronização das entradas dos mesmos, bem como para decidir para qual módulo receberia o resultado da operação.

Mesmo com os esforços de reduzir a área com a limitação de multiplicadores, ainda houve um grande custo em recursos utilizados. A principal razão para esse gasto está nos *buffers* utilizados entre os estágios do algoritmo, e, mais especificamente, no modo de acesso aos seus dados. Devido ao acesso em mais de uma palavra por vez bem como à ausência de sinais explícitos de *write enable* e de *read enable*, a ferramenta de síntese não foi capaz de sintetizar alguns desses módulos em memória, optando, em vez disso, em transformá-los em Flip Flops. Futuramente, seram exploradas abordagens mais econômicas em relação aos recursos, principalmente através do uso de mais BRAMs

Apesar disso, a proposta do uso de implementações como essa na borda, ao contrário do uso direto pelo usuário, faz com que os requisitos de uso de área, bem como os requisitos de energia, não sejam tão restritivos quanto o outro caso. Ademais, existem placas no mercado capazes de atender a essa demanda de recursos, como a placa *Ultrascale+ Alveo U50*, da Xilinx.

Em relação à latência, como visto anteriormente, o tempo de latência médio aceitável, ou seja, sem causar desconforto ao usuário, é de cerca de $10ms$. Sendo assim, o presente projeto foi capaz de atender aos requisitos com uma ótima margem. Essa margem é um incentivo ainda maior para a exploração de arquiteturas mais econômicas no futuro.

6 TRABALHOS FUTUROS

O presente trabalho abre um bom leque de possibilidades quanto a futuros trabalhos baseando-se nas discussões apresentadas. Dentre essas opções, destacam-se:

- Implementação e análise de um algoritmo de VAD a ser utilizado em conjunto com o presente algoritmo.
- Implementação de um projeto RTL com versão em ponto flutuante ao invés de ponto fixo.
- Implementação de um *core* próprio para realização de FFTs e IFFTs, explorando algoritmos alternativos ao tradicional Cooley-Tuckey.
- Implementação de versão utilizando ferramentas de alto nível (HLS), como feito em (FRANCISCO, 2021), porém mantendo arquitetura de ponto fixo.
- Implementação de outros algoritmos de redução de ruídos, não baseados no Filtro de Wiener.
- Prototipação do projeto em uma placa física para testes mais robustos.

REFERÊNCIAS

ALCARAZ, F. **fxpmath library**. 2022. Https://github.com/francof2a/fxpmath. Acessado em: em 20 de Agosto de 2023.

ANDRADE, H. C. de. **código fonte do projeto**. 2023. <https://github.com/HeitorCAndrade/TG>. Acessado em: em 04 de Setembro de 2023.

BURWINKEL, J.; MCKINNEY, M. F.; GALSTER, J. Acceptable hearing aid throughput delay for listeners with hearing loss under noisy conditions. **American Auditory Society**, 2017.

CHERRY. Some experiments on the recognition of speech, with one and with two ears. **The Journal of The Acoustical Society of America**, 1953.

CORNELIS, B. Theoretical analysis of binaural multimicrophone noise reduction techniques. **IEEE Transactions on Audio Speech and Language Processing**, 2010.

CORNELIS, B. Performance analysis of multichannel wiener filter-based noise reduction in hearing aids under second order statistics estimation errors. **IEEE Transactions on Audio Speech and Language Processing**, 2011.

EHIMA. **EHIMA Surveys**. 2022. Https://www.ehima.com/surveys/. Acessado em: em 20 de Agosto de 2023.

FAROOQ, U.; MARRAKCHI, Z.; MEHREZ, H. **Tree-based heterogeneous FPGA architectures: application specific exploration and optimization**. [S.l.]: Springer, 2012.

FRANCISCO, G. **Aceleração em hardware de algoritmo de redução de ruído com preservação de cenário acústico para aparelhos auditivos binaurais**. 2021.

GREEN, W. **project F. Division in Verilog**. 2020. Https://projectf.io/posts/division-in-verilog/. Acessado em: em 31 de Agosto de 2023.

ITTURRIET, F. **PRESERVAÇÃO PERCEPTUALMENTE RELEVANTE DA DIFERENÇA DE TEMPO INTERAURAL EM APARELHOS AUDITIVOS BINAURAIS**. Thesis (PhD) — Universidade Federal de Santa Catarina, 2019.

KUKLASINSKI, A. Multichannel wiener filters in binaural and bilateral hearing aids — speech intelligibility improvement and robustness to doa errors. **Journal of the Audio Engineering Society**, 2017.

LIN, F.; FERRUCCI, L. Hearing loss and falls among older adults in the united states. **Archives of internal medicine (Chicago, Ill. : 1908)**, American Medical Association, v. 172, n. 4, p. 369–371, feb. 2012. ISSN 2168-6106.

PONT, M. T. S. et al. Table-based recursive method for function evaluation. **AIP Conference Proceedings**, 2007.

REIS, A. et al. Impelmentação em tempo real de um sistema de redução de ruído binaural com preservação da função de transferência interaural. **Simpósio Brasileiro de Telecomunicações e Processamento de Sinais**, 2019.

WIENER. Extrapolation, interpolation, and smoothing of stationary time series: With engineering applications. **MIT Press**, 1949.