Extração Eficiente de MFCCs em FPGA: Uma implementação Aberta e Flexível*

Julio N. Avelar[1], Vinicius P. M. Miguel[1], Tiago S. Zaparoli[1], Enzo P. Bertoloti[1], Gabriel Oliveira[1], Rodolfo Azevedo[1]

[1]Instituto de Computação – Universidade Estadual de Campinas (UNICAMP)
Caixa Postal – 13.083-852 – Campinas – SP – Brazil

{j241163, v260731, t237310, e248361, g247700}@dac.unicamp.br,
rodolfo@ic.unicamp.br

Abstract. *This paper presents a set of open-source and parameterizable IPs for MFCC extraction on FPGAs, optimized for edge and real-time applications. The implementation on a Kintex-7 FPGA processes an audio frame in just 56 µs, outperforming a high-performance CPU (AMD Ryzen 7 7700) by 1.62 times. The architecture is resource-efficient, using approximately 5900 LUTs and 31 DSPs, and maintains high precision with a mean absolute error (MAE) of 8.18, validating it as a high-performance, low-power solution for real-time audio analysis.*

Resumo. *Este artigo apresenta um conjunto de IPs de código aberto e parametrizáveis para extração de MFCCs em FPGAs, otimizado para aplicações de borda e de tempo real. A implementação em um FPGA Kintex-7 processa um quadro de áudio em apenas 56 µs, superando em 1.62 vezes um CPU de alto desempenho (AMD Ryzen 7 7700). A arquitetura é eficiente em recursos, utilizando cerca de 5900 LUTs e 31 DSPs, e mantém alta precisão (erro absoluto médio de 8.18%), validando-se como uma solução de alta performance e baixo consumo para análise de áudio em tempo real.*

1. Introdução

A detecção de padrões em áudios tem sido cada vez mais explorada em diferentes aplicações, desde assistentes de voz até ferramentas industriais para identificação de anomalias a partir de ruídos. Em especial, no contexto industrial e biomédico, a análise de sinais acústicos tem ganho destaque devido à facilidade de coleta e ao caráter pouco intrusivo dessa abordagem, que dispensa a instalação de sensores físicos diretamente em equipamentos e objetos. Segundo [Ye et al. 2025], métodos de manutenção preditiva baseados em sinais sonoros têm atraído crescente interesse da comunidade acadêmica e industrial, justamente por se tratarem de soluções de fácil implantação, capazes de capturar informações de alta dimensão.

Outra aplicação relevante é o uso da voz como forma de interface em ambientes industriais. Trabalhos como [Cohen et al. 2025] destacam a importância da comunicação vocal para a interação com robôs colaborativos, viabilizando comandos de voz que permitem aos operadores manterem as mãos livres para o acionamento de máquinas e a

*Este trabalho foi parcialmente financiado com recursos dos projetos CNPq 316067/2023-7 e FAPESP 2013/08293-7.

execução de outras atividades. Ademais, outras aplicações que compartilham desafios semelhantes são reconhecimento de fala [Dao et al. 2017], classificação de gêneros musicais [Wassi et al. 2015], análise de sons respiratórios [Bahoura and Ezzaidi 2013], etc. Neste cenário, a utilização de *Mel-frequency Cepstral Coefficients* (MFCC) [Abdul and Al-Talabani 2022] desponta como uma das abordagens mais comuns para extração de características de áudios.

Apesar de sua ampla utilização, o cálculo de MFCCs é um desafio computacional, pois sua computação é custosa e exige equilibrar velocidade e precisão. Diversas soluções têm explorado CPUs, GPUs e *Field-Programmable Gate Arrays* (FPGAs); enquanto CPUs e GPUs são amplamente usadas em soluções baseadas em software, tais soluções apresentam alto custo computacional e demandam hardware robusto. Em contrapartida, implementações em FPGAs permitem computação em tempo real em dispositivos de borda ou em cenários que exigem muitos coeficientes e/ou altas taxas de amostragem.

Com a recente evolução das CNNs (*Convolutional Neural Network*) e DNNs (*Deep Neural Network*) otimizadas para dispositivos de borda, a extração de características voltou a ser um gargalo computacional [Fariselli et al. 2021], o que reforça a relevância de arquiteturas eficientes de MFCC em hardware de baixo desempenho, como MCUs e FPGAs, onde técnicas de quantização e paralelismo tornam possíveis aplicações de áudio em tempo real.

Diante desse contexto, este artigo propõe um conjunto de IPs abertos (sob licença permissiva)[1] para a computação de MFCCs, desenvolvidos em SystemVerilog, totalmente parametrizáveis e compatíveis com diferentes famílias de FPGAs (Xilinx, Lattice e Gowin). Além disso, são apresentados comparativos de desempenho e custo em relação a implementações em CPUs, GPUs e outros trabalhos em FPGA disponíveis na literatura.

2. Trabalhos Relacionados

A literatura revela múltiplas estratégias para viabilizar a computação eficiente de MFCCs, envolvendo abordagens baseadas em software, utilizando CPUs e GPUs, bem como implementações em hardware específico, como FPGAs e *Application Specific Integrated Circuits* (ASICs).

No domínio de software, diversos trabalhos investigaram a implementação de MFCCs em GPUs, apresentando soluções capazes de oferecer elevado desempenho computacional [Kou et al. 2013, Michálek and Vaněk 2014, Yi and Talakoub 2008]. Apesar da eficiência, essas abordagens dependem de GPUs, que apresentam alto consumo de energia e custo significativo, limitando sua aplicabilidade em dispositivos de borda ou em cenários com restrições de potência. Em contraste, a presente proposta busca viabilizar o cálculo de MFCCs em aplicações de baixa potência, preservando a eficiência computacional e garantindo maior flexibilidade de integração em diferentes plataformas.

Por outro lado, implementações baseadas em ASICs como as investigadas em [Paul S et al. 2021, Wang et al. 2000, Nguyen et al. 2016], proporcionam alta performance e eficiência energética. Contudo, essa estratégia apresenta barreiras econômicas para produção em pequena escala, sendo viável principalmente em grandes volumes de fabricação. É importante notar que os IPs desenvolvidos para FPGA podem ser adap-

[1]Disponivel em: https://github.com/Unicamp-Odhin/MFCC_Core

tados para ASICs com modificações pontuais, mantendo a relevância da abordagem em contextos diversos.

Uma alternativa intermediária, tanto em termos técnicos quanto de custo, é a implementação em FPGAs, explorada em trabalhos como [Wassi et al. 2015, Bahoura and Ezzaidi 2013, Dao et al. 2017]. Nestes estudos, as soluções se restringem a FPGAs Xilinx, sendo que os dois primeiros utilizam o Xilinx System Generator (XSG) para modelagem em alto nível. Embora eficazes, essas abordagens apresentam limitações quanto à portabilidade e parametrização para diferentes famílias e fabricantes de FPGAs. Em contraste, a proposta deste trabalho apresenta IPs abertos e parametrizáveis, compatíveis com múltiplos fabricantes e famílias de FPGAs, promovendo maior flexibilidade e facilidade de integração.

De forma geral, os trabalhos analisados evidenciam a importância do uso de MFCCs e os esforços contínuos para otimizar sua implementação em hardware. No entanto, observa-se que muitas soluções permanecem dependentes de dispositivos proprietários ou de ferramentas específicas, restringindo sua generalização. Nesse contexto, a presente proposta se diferencia ao fornecer um conjunto de IPs abertos, totalmente parametrizáveis, compatíveis com diversas famílias de FPGAs.

3. Algoritmo de Implementação do MFCC

O processo de extração dos MFCCs envolve diversas etapas, realizadas predominantemente no domínio da frequência. O primeiro passo é a aplicação de uma pré-ênfase no sinal de áudio, com o objetivo de aumentar a amplitude das altas frequências e compensar a atenuação natural dessas componentes. A pré-ênfase é aplicada utilizando o filtro de diferença simples, dado por $y[n] = x[n] - \alpha\, x[n-1]$, com $0.90 \leq \alpha \leq 0.97$. Onde $x[n]$ é o sinal de áudio original, $y[n]$ é o sinal pré-enfatizado e α é o coeficiente de pré-ênfase.

Após a pré-ênfase, o sinal é segmentado em *frames* curtos, geralmente com duração entre 20 e 40 ms, deslocados em intervalos de 5 a 20 ms (*hop size*). Essa segmentação permite a análise local do áudio, assumindo que o sinal seja aproximadamente estacionário em pequenas janelas de tempo.

Cada *frame* recebe uma janela de *Hamming*, aplicada para reduzir os efeitos de descontinuidade nas bordas e dar ênfase à região central da janela. A função da janela de *Hamming* é definida por:

$$w[n] = 0.54 - 0.46 \cos\left(\tfrac{2\pi n}{N-1}\right), \quad 0 \leq n \leq N-1 \tag{1}$$

onde N é o número de amostras por *frame*. A aplicação da janela suaviza as transições entre os *frames* e minimiza efeitos de *spectral leakage* durante a análise no domínio da frequência.

Em seguida, é realizada a Transformada Rápida de Fourier (FFT) em cada *frame*, convertendo o sinal do domínio do tempo para o domínio da frequência:

$$X[k] = \sum_{n=0}^{N-1} x[n]\, e^{-j2\pi kn/N}, \quad 0 \leq k \leq N-1 \tag{2}$$

onde $X[k]$ representa o valor do espectro de frequência na posição k e N é o número de pontos da FFT. Normalmente, N é maior ou igual ao número de amostras do *frame*

e é escolhido como uma potência de 2, permitindo a utilização de algoritmos eficientes, como Radix-2 ou Radix-4. Caso N seja maior que o número de amostras, o vetor de entrada $x[n]$ é preenchido com zeros (*zero-padding*) até atingir o tamanho N, garantindo compatibilidade com a FFT e mantendo a resolução espectral desejada.

Após a FFT, apenas as primeiras $N/2 + 1$ saídas são utilizadas, devido à simetria do espectro para sinais reais. A potência em cada frequência é calculada e, em seguida, o espectro é mapeado para a escala Mel, que é perceptualmente relevante e imita a resposta do ouvido humano. As equações para o cálculo da potência e a conversão para a escala Mel são dadas respectivamente por:

$$P[k] = \frac{\Re(X[k])^2 + \Im(X[k])^2}{N} \qquad (3) \qquad\qquad m = 2595 \, \log_{10}\left(1 + \frac{f}{700}\right) \qquad (4)$$

Na Equação (3), $\Re(X[k])$ e $\Im(X[k])$ são as partes real e imaginária do espectro. A utilização da potência evita a operação de raiz quadrada, que é cara, e a normalização por N pode ser implementada como um simples *shift*. A Equação (4) define a conversão da frequência linear f (em Hz) para a perceptual m (em Mels).

Com base nesta escala, é construído um banco de filtros triangulares sobrepostos, como ilustrado na Figura 1. As frequências centrais desses filtros são linearmente espaçadas na escala Mel, o que resulta em filtros mais estreitos em baixas frequências e mais largos em altas frequências. A fórmula que define cada filtro triangular $H_m(k)$ é:

$$H_m(k) = \begin{cases} 0, & \text{se } k < f(m-1) \text{ ou } k > f(m+1) \\ \frac{k - f(m-1)}{f(m) - f(m-1)}, & \text{se } f(m-1) \leq k < f(m) \\ \frac{f(m+1) - k}{f(m+1) - f(m)}, & \text{se } f(m) \leq k \leq f(m+1) \end{cases}$$

onde k é o índice de frequência do espectro de potência $P[k]$, e $f(m-1)$, $f(m)$ e $f(m+1)$ são as frequências centrais (em *Hz*) do filtro anterior, atual e seguinte, respectivamente.

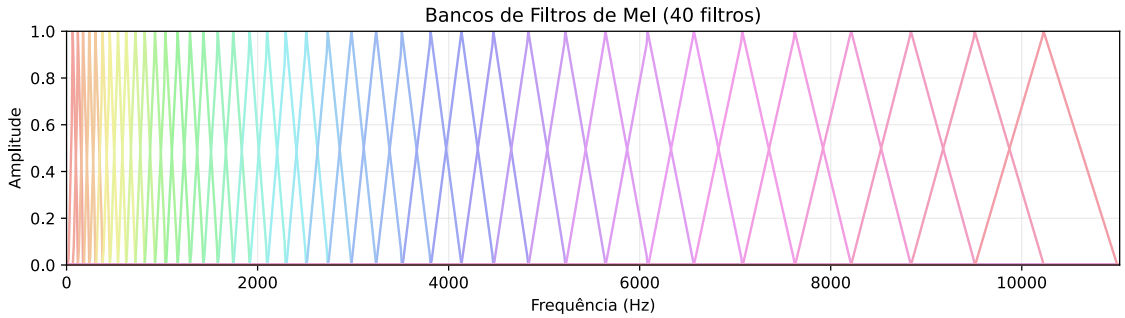


Figura 1. Banco de filtros triangulares sobrepostos na escala Mel. Note como os filtros são mais estreitos e próximos em baixas frequências e mais largos em altas frequências.

A energia em cada banda do filtro Mel, $S[m]$, é então calculada somando-se os valores do espectro de potência $P[k]$ ponderados pela resposta de cada filtro triangular. De posse da energia para cada uma das K bandas, calcula-se o logaritmo dessas saídas,

simulando a percepção humana de intensidade sonora. Por fim, é aplicada a *Transformada Discreta do Cosseno* (DCT) sobre o logaritmo dos coeficientes Mel para obter os MFCCs finais:

$$c_m = \sum_{k=1}^{K} \log_{10}(S[k]) \cos\left[m\frac{\pi}{K}\left(k - \frac{1}{2}\right)\right], \quad m = 1, 2, \ldots, M \tag{5}$$

onde $S[k]$ é a energia na banda k do filtro Mel, K é o número total de filtros Mel e M é o número de coeficientes MFCC desejados.

Essas etapas resultam em um vetor de características compacto e representativo do áudio, que preserva informações relevantes para tarefas como reconhecimento de fala e classificação de sons. O fluxograma completo deste processo é apresentado na Figura 2.

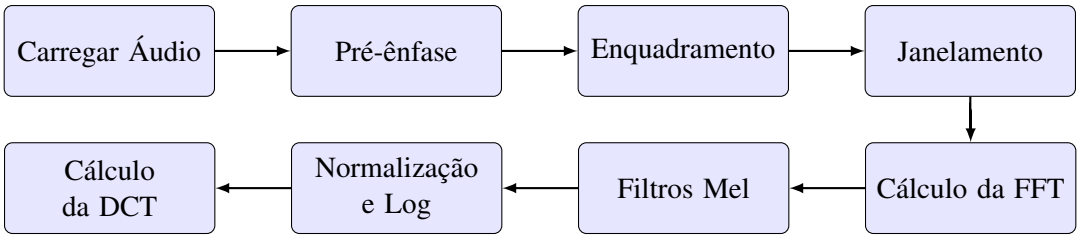


Figura 2. Fluxograma do processo de extração dos coeficientes MFCC

4. Implementação em FPGA do MFCC

A arquitetura proposta para a extração de características MFCC foi projetada para operar em tempo real, explorando o paralelismo inerente da utilização de FPGAs e a facilidade de construção de *Pipelines*. Para otimizar a latência e a vazão do sistema, diversas etapas do algoritmo foram reestruturadas. A etapa de pré-ênfase, por exemplo, foi integrada diretamente ao módulo de entrada de dados PCM (*Pulse-Code Modulation*), resultando em um tempo de aplicação desprezível. Similarmente, o processo de enquadramento (*framing*) é eficientemente realizado através de um *buffer* circular, enquanto os coeficientes da janela de *Hamming* são aplicados em paralelo aos dados de entrada por meio de uma estrutura em pipeline. Esta abordagem reduz consideravelmente o tempo de processamento, medido em ciclos de *clock*.

Para contornar a complexidade da aritmética de ponto flutuante em FPGAs, a implementação adota representação em ponto fixo com inteiros de 16 e 32 bits. Coeficientes e constantes, como os da janela de *Hamming* e dos filtros Mel, foram pré-calculados em software de precisão elevada e armazenados em tabelas de consulta (LUTs). A análise quantitativa do erro associado a essa escolha é apresentada na Seção 5.

Para apoiar a implementação em HDL, foi desenvolvido um modelo de referência em linguagem C. Este modelo possui a flexibilidade de operar tanto em ponto fixo quanto em ponto flutuante, cumprindo duas funções essenciais no projeto. Primeiramente, é utilizado para gerar dinamicamente as tabelas de consulta (LUTs) que são carregadas na FPGA. Em segundo lugar, serve como um *golden model* para a verificação funcional do hardware. O processo de verificação é realizado em um ambiente de co-simulação, onde as saídas dos módulos HDL são comparadas em tempo real com os resultados do modelo em C, permitindo a validação precisa e contínua do comportamento esperado.

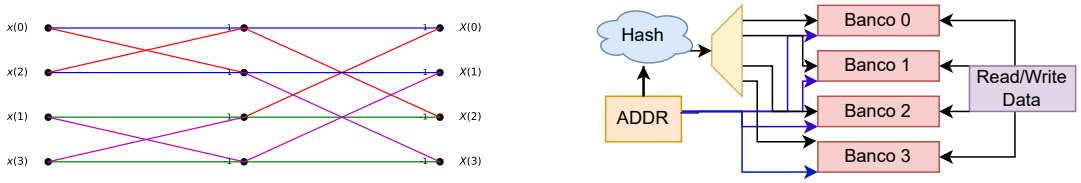
4.1. Geração de *Frames* e Janelamento de *Hamming*

O processamento inicia-se com a aquisição do sinal de áudio no formato PCM, cujas amostras passam por um módulo de pré-ênfase e são armazenadas em uma fila FIFO para desacoplar a taxa de entrada da velocidade de processamento. A partir dessa fila, uma unidade de controle alimenta um *buffer* circular que gera os quadros sobrepostos. Em paralelo, um módulo dedicado pondera cada amostra lida do *buffer* com o coeficiente correspondente da janela de *Hamming*, escrevendo o resultado sequencialmente no *buffer* de entrada da etapa de FFT.

Para garantir a integridade do fluxo, tal unidade de controle monitora a FIFO e emite um sinal de pausa (*stall*) em caso de esvaziamento (*underflow*), aguardando novos dados para continuar. Ao atingir o número de amostras de um quadro (N), a unidade realiza o *zero-padding* (se necessário para completar o *buffer* de entrada da FFT) e ativa o módulo da FFT. Em condições ideais de fluxo, este processo de enquadramento e janelamento é executado em $M + 2$ ciclos de clock, onde M é o tamanho da entrada da FFT e a constante de dois ciclos representa a latência inicial do pipeline.

4.2. FFT Radix-2

Para a implementação da FFT, optou-se pelo algoritmo Radix-2 devido à sua simplicidade estrutural e eficiência em hardware. Este algoritmo decompõe a transformada em operações elementares conhecidas como "borboletas" (*butterflies*), conforme ilustrado para um caso de 4 pontos na Figura 3a. A escolha do Radix-2, em detrimento de algoritmos de base superior como Radix-4 ou Radix-8, foi motivada pelo padrão de acesso à memória: cada borboleta requer apenas duas leituras e duas escritas, um requisito que se alinha favoravelmente com a arquitetura dos blocos de memória (BRAMs e LUTRAMs) disponíveis em FPGAs.



(a) Computação de uma borboleta para uma FFT de 4 pontos.

(b) Arquitetura de bancos de memória para paralelização de acesso.

Figura 3. Implementação da FFT Radix-2: (a) unidade básica de cálculo (*butterfly*) e (b) organização da memória para paralelização de acesso.

O cálculo de cada borboleta é definido pela Equação 6a. Os fatores de rotação complexos (*twiddle factors*, Eq. 6b) necessários para estas operações são constantes para uma dada FFT de tamanho N. Seguindo a estratégia adotada para os coeficientes da janela de *Hamming*, estes fatores foram pré-calculados e armazenados em tabelas de consulta (LUTs), minimizando a carga computacional em tempo de execução para apenas multiplicações e somas complexas.

$$x'[m] = x[m] + W_N^k \cdot x[n] \qquad x'[n] = x[m] - W_N^k \cdot x[n] \tag{6a}$$

$$W_N^k = e^{-j\frac{2\pi k}{N}} \tag{6b}$$

Uma característica da FFT Radix-2 iterativa é a necessidade de reordenar os dados de entrada para garantir o acesso correto durante os estágios de cálculo. Essa permutação

é implementada em hardware de forma eficiente por meio da inversão dos bits do endereço no momento da escrita no *buffer*, dispensando o percurso explícito do vetor exigido em software. A computação das borboletas, que envolve multiplicações e somas, é acelerada pelo uso dos blocos DSP (*Digital Signal Processing*) presentes em FPGAs modernos, capazes de executar uma operação completa a cada ciclo de *clock*. Com uma arquitetura em *pipeline* de quatro estágios: (1) cálculo dos endereços, (2) leitura da memória, (3) execução no DSP e (4) escrita do resultado — o tempo total da FFT é reduzido para $\frac{N \log_2(N)}{2}$ ciclos.

Um desafio dessa arquitetura é lidar com acessos simultâneos à memória, já que os BRAMs padrão não permitem múltiplas escritas no mesmo ciclo por porta. Sem cuidados, isso levaria a uma implementação ineficiente, com inferência de grande quantidade de Flip-Flops (FFs) e consequente aumento de área. Para contornar esse problema, adotou-se a técnica de *memory banking* [Zhou et al. 2017], dividindo a memória em quatro bancos independentes. Uma função de *hash* é aplicada ao endereço para selecionar o banco, garantindo que os dois operandos de cada borboleta nunca colidam. Esse arranjo permite acessos paralelos sem recorrer a memórias multi portas, embora à custa de maior consumo de blocos de memória (LUTRAMs e/ou BRAMs). A Figura 3b ilustra a organização dos bancos de memória.

Após a conclusão da FFT, é realizado o cálculo da potência do espectro ($|X[k]|^2$). Este cálculo também é realizado em um *pipeline* de quatro estágios: (1) leitura, (2) cálculo das potências, (3) soma e (4) escrita, executando em $\frac{N}{2} + 1$ ciclos. Ao seu término, um sinal de ativação é enviado ao módulo de filtros Mel. A latência total para este bloco, somando a FFT, o cálculo de potência e o preenchimento inicial dos *pipelines* (aproximadamente 3 ciclos para cada), é de $\frac{N \log_2(N)}{2} + \frac{N}{2} + 7$ ciclos de *clock*.

4.3. Aplicação dos Filtros Mel e Cálculo do Logaritmo

Após o cálculo da potência do espectro, aplica-se o banco de filtros Mel, implementado como um produto matriz-vetor entre o espectro de potência e a matriz de coeficientes pré-calculada. Como os filtros triangulares geram uma matriz altamente esparsa, adotou-se um formato comprimido de armazenamento: apenas os coeficientes não nulos são guardados, acompanhados do índice inicial e final das posições não nulas. Essa estratégia elimina multiplicações por zero e reduz significativamente a área de memória.

O cálculo do logaritmo da energia de cada banda, etapa subsequente do algoritmo, foi realizado por meio de uma aproximação eficiente baseada no logaritmo em base 2. A partir desse valor, obtém-se a energia de cada banda por meio da multiplicação por uma constante de escala. Esse procedimento reduz a complexidade da operação em hardware, preservando a precisão necessária para o cálculo dos MFCCs.

4.4. Transformada Discreta de Cossenos (DCT)

A etapa final para a extração das características é a aplicação da Transformada Discreta de Cossenos (DCT) sobre o vetor de energias Mel logarítmicas. De forma análoga às etapas anteriores, esta operação é otimizada tratando-a como um produto matriz-vetor com coeficientes pré-calculados. Os valores da base de cossenos da DCT são armazenados em uma tabela de consulta.

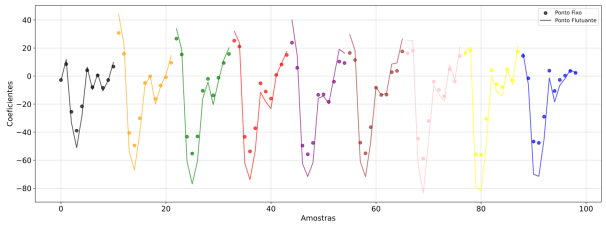
Cada coeficiente cepstral de saída (o elemento final do vetor MFCC) é calculado através do produto escalar entre o vetor de energias Mel e a linha correspondente da matriz DCT. Em hardware, isto é implementado de forma eficiente por uma unidade de multiplicação-acumulação (MAC). Ao final deste processo, o vetor de coeficientes cepstrais do quadro de áudio atual está completo e disponível para as etapas subsequentes de reconhecimento ou classificação.

5. Análise de Precisão da Aritmética de Ponto Fixo

A implementação da aritmética de ponto fixo no cálculo dos MFCCs foi realizada utilizando valores em `int32` e `int16`, nos formatos `q1.31` e `q1.15`, respectivamente. O uso de ponto fixo simplifica consideravelmente a implementação em FPGA, pois as operações são realizadas com inteiros e exploram de forma eficiente os blocos DSP disponíveis. No entanto, quando comparada à aritmética de ponto flutuante, essa abordagem ainda apresenta perda de precisão significativa.

Para mitigar erros de quantização, coeficientes como os fatores de rotação (*twiddle factors*) e o banco de filtros de Mel foram pré-calculados em ponto flutuante e armazenados em tabelas de consulta. Uma das principais fontes de erro decorre das operações de multiplicação. Para reduzir o acúmulo de erro, utilizou-se uma técnica de arredondamento que consiste em somar o menor valor representável no formato `q1.15` ou `q1.31` antes de truncar o resultado para o tamanho original do número.

A validação foi conduzida por meio da comparação entre a implementação em ponto fixo e um modelo de referência em ponto flutuante de dupla precisão. A Figura 4 apresenta a sobreposição das saídas para um mesmo *frame* de entrada (avaliação qualitativa), bem como a análise quantitativa baseada em métricas de erro.



(a) Sobreposição das saídas em ponto fixo (Y_{fixo}) e em ponto flutuante (Y_{ref}).

Métrica	Valor médio
MSE	123.96
MAE	8.18
Máx. Erro	22.82

(b) Resultados quantitativos médios da análise de erro.

Figura 4. Validação da implementação em ponto fixo: (a) comparação qualitativa e (b) análise quantitativa.

Os resultados indicam um erro quadrático médio (MSE) de aproximadamente 123.96, um erro absoluto médio (MAE) de 8.18 e um erro máximo em torno de 22.82. Esses valores demonstram que, embora existam discrepâncias pontuais, o desvio médio permanece baixo em relação à amplitude típica do sinal, indicando que o impacto do erro na extração dos MFCCs é limitado. Assim, a implementação em ponto fixo mantém boa fidelidade em comparação com a referência em ponto flutuante, mantendo o desempenho.

6. Resultados e Discussão

A Tabela 1 apresenta os resultados da síntese e implementação do núcleo MFCC em diferentes dispositivos FPGA e sob múltiplas configurações de parâmetros (F: tamanho

da FFT; M: número de filtros Mel; C: número de coeficientes; T: taxa de amostragem em kHz). São detalhados o consumo de recursos lógicos, memória, blocos de DSP, a frequência máxima de operação, o consumo dinâmico da implementação e o número de ciclos necessários para processar um quadro de áudio.

Tabela 1. Performance e recursos do núcleo MFCC por FPGA e configuração.

Dispositivo	Parâmetros (F/M/C/T)	LUTs	FFs	LUTRAMs	DSPs	Fmax (MHz)	DP (W)	N. ciclos
Artix-7 100T-1	0512 / 40 / 13 / 16.0	5,968	5,664	3,138	31	55.9	0.175	4211
	1024 / 60 / 20 / 44.1	6,852	6,024	6,220	31	54.4	0.185	9513
Kintex-7 325T-2	0256 / 20 / 07 / 08.0	4,029	2,349	1,602	31	76.36	0.105	2253
	0512 / 40 /13 / 16.0	5,867	5,668	3,138	31	74.45	0.175	4211
Virtex-7 690T-2	1024 / 60 / 20 / 44.1	6,655	6,027	6,220	31	75.24	0.185	9513
ECP5 45F	0256 / 20 / 07 / 08.0	39,990	2,725	3,128	34	13.44	-	2253

Os resultados evidenciam a alta flexibilidade da arquitetura proposta. A descrição em SystemVerilog permite um grau elevado de parametrização, viabilizando configurações otimizadas para diferentes cenários de aplicação, variando o tamanho da FFT, o número de coeficientes e a taxa de amostragem. Outro ponto de destaque é a portabilidade: o mesmo núcleo pôde ser sintetizado em FPGAs de baixo custo e em plataformas de alto desempenho, algo que diferencia esta solução de outras abordagens mais rígidas, limitadas a um dispositivo específico.

Vale ressaltar que a tabela reúne apenas resultados representativos. Dispositivos de famílias semelhantes (e.g., Kintex-7 e Virtex-7; Artix e Zynq-7000) apresentaram frequências de operação e área utilizada muito próximas para o mesmo design, sendo a principal diferença a disponibilidade total de recursos. Por motivos de concisão, entradas redundantes foram omitidas.

A Tabela 2 apresenta uma comparação direta com outras implementações em FPGA descritas na literatura, evidenciando que a solução proposta se mostra, em média, mais eficiente no uso de recursos lógicos e opera com um número relativamente reduzido de ciclos de *clock*. Como os trabalhos comparados não adotam uma padronização para o reporte do uso de LUTRAMs, estabeleceu-se como métrica de normalização que um BRAM18 equivale a 288 LUTs-6 configurados como LUTRAM, enquanto um BRAM32 corresponde a 576 LUTs-6 em modo LUTRAM. Ressalta-se, entretanto, que não foi realizada uma análise comparativa de consumo de energia, uma vez que os trabalhos de referência não apresentam essa métrica em seus resultados.

No contexto de desempenho, a Tabela 3 apresenta uma análise detalhada das implementações em CPU, GPU e ASIC. A solução proposta em FPGA (Kintex-7) se destaca ao processar um quadro em apenas 56 μs, superando CPUs de alto desempenho como o AMD Ryzen 7 7700 (91 μs) e implementações otimizadas de outras pesquisas, como a do Intel Core i7 2600 (123 μs). Além disso, o desempenho em FPGA é competitivo em relação a GPUs, como a AMD Radeon RX 7800XT (11 μs) e a NVIDIA GTX 580 (13 μs). A diferença de desempenho é atenuada pelo menor custo do sistema e pelo consumo energético significativamente reduzido, o que torna a solução mais atraente para

Tabela 2. Comparativo de recursos e performance com outras implementações, detalhando os parâmetros de FFT (F), filtros Mel (M) e coeficientes (C).

Dispositivo	Parâms. (F/M/C)	LUTs	FFs	LUTRAMs	DSPs	Fmax (MHz)	N. ciclos
Kintex-7 325T-2	512/40/13	5,867	5,668	3,138	31	74.45	4,211
Virtex-6 lx240t [Wassi et al. 2015]	-/24/24	8,193	11,577	12,672	82	155	-
Virtex-6 lx240t [Bahoura and Ezzaidi 2013]	1024/24/24	14,837	13,726	1,152	57	65.5	-
Virtex-6 lx240t [Dao et al. 2017]	128/-/12	7,353	11,567	2,304	10	139.0	-
Spartan 3A DSP 1800 [Dao et al. 2017]	128/-/12	11,581	11,335	4,320	11	52.0	-
Virtex 4 LX15 [Paul S et al. 2021]	-/24/39	4,092	1,984	-	14	-	2,740
Virtex 2 XC2V6000 [Ehkan et al. 2015]	256/-/17	39,452	35,104	4,032	33	-	1,178
Artix 7 XC7A100T [Zhao et al. 2024]	512/26/13	4,391	3,253	1,831	117	-	-
Zynq XCZU7EV [Tsai and Wang 2024]	512/20/13	14,496	28,387	19,102	5	150	13,755
Zedboard XC7Z020 [K et al. 2019]	-/40/12	22,090	22,106	19,917	129	49.34	-
Xilinx Zynq kit [Anshu et al. 2022]	256/-/10	14,150	15,210	11,792	55	80	-
Artix 7 XC7A100T [Boujelben and Bahoura 2018]	1024/24/15	16,563	17,074	1,152	87	101.74	-

aplicações em que o custo e a eficiência energética são fatores críticos.

Comparando com plataformas embarcadas de baixo custo, como VisionFive 2, Raspberry Pi 4 e Milk-V Jupiter, a solução proposta apresenta ganhos substanciais, com tempos de execução dezenas de vezes menores. Para garantir uma comparação justa, todos os testes em software foram realizados utilizando FFTs de 512 pontos, mesma configuração adotada na implementação em FPGA e nas referências da literatura. Essas implementações variam apenas em pequenos detalhes, como o número de coeficientes Ceps e a quantidade de filtros Mel, com destaque para as implementações de [Wang et al. 2000] e [Nguyen et al. 2016], que utilizam uma FFT de 256 pontos. Vale ressaltar que a versão para CPU utilizada neste trabalho é genérica e totalmente sequencial, sem otimizações arquiteturais específicas, embora tenha sido compilada com o nível de otimização `-O3` e a flag *--march=native*. Todos os resultados reportados referem-se ao tempo de processamento de um único quadro (*frame*), sendo que na GPU considera-se o tempo médio após processar um lote de frames.

Os resultados reforçam que a solução proposta é competitiva em desempenho, mesmo frente a arquiteturas especializadas (ASICs), e é muito superior a implementações puramente em software. A baixa latência, aliada à natureza autônoma do núcleo, abre

Tabela 3. Comparação de performance entre diferentes plataformas.

Trabalho	Plataforma	Precisão Numérica	Tempo/Quadro (μs)
Este Trabalho	**FPGA (Kintex-7)**	**Ponto Fixo**	**56**
	CPU (AMD Ryzen 7 7700)		91
	CPU (AMD Ryzen 7 5825u)		231
	CPU (Starfive VisionFive 2)		1,998
	CPU (Raspberry PI 4)		1,278
	CPU (Milk-V Jupiter)		1,320
	GPU (AMD RX 7800XT)		11
[Wang et al. 2000]	ASIC (0.6 μm - 50 MHz)	-	73
[Nguyen et al. 2016]	ASIC (0.13 μm - 500 MHz)	Ponto Flutuante	2,348
[Paul S et al. 2021]	ASIC (0.13 μm - 100 MHz)	Ponto Flutuante	27
[Kou et al. 2013]	GPU (NVIDIA GTX 580)	Ponto Flutuante	13
[Kou et al. 2013]	CPU (Intel Core i7 2600)	Ponto Flutuante	123

espaço para a integração em arquiteturas heterogêneas de SoC, como nas famílias Zynq (Xilinx) e Arria V (Altera). Nesses cenários, o núcleo MFCC pode atuar como um acelerador dedicado na lógica programável (PL), liberando o processador de aplicações (PS) para executar algoritmos de aprendizado de máquina ou outras tarefas de nível mais alto.

Por outro lado, a estratégia de *memory banking* adotada na FFT, embora eficaz para reduzir a latência, implica em maior consumo de LUTRAMs e pode dificultar a síntese em ferramentas específicas, como o Yosys, que tende a inferir grande quantidade de FFs para representar *buffers*. Uma evolução futura seria a adoção de esquemas de endereçamento mais sofisticados, otimizando o uso de memória em FPGAs de menor porte. Além disso, a redução de caminhos críticos pode aumentar a frequência máxima de operação, habilitando taxas de amostragem ainda mais elevadas. Planeja-se também ampliar a validação para FPGAs de outros fabricantes (e.g., Altera e Microchip), reforçando o caráter aberto e independente de fornecedor que norteia este projeto.

7. Conclusão

Este artigo apresentou um conjunto de IPs de código aberto e parametrizáveis para a extração eficiente de MFCCs em FPGAs, projetado para atender às demandas de aplicações de borda em tempo real. A arquitetura proposta demonstrou notável portabilidade entre diferentes fabricantes e um desempenho superior, processando um quadro de áudio em apenas 56 μs, o que representa um ganho de 1.62 vezes sobre uma CPU de alto desempenho. Este resultado, alcançado com uso eficiente de recursos, valida a abordagem como uma solução de baixa latência e consumo energético para análise de áudio. Os próximos passos se concentrarão na otimização dos caminhos críticos para elevar a frequência de operação, na ampliação do suporte a outros fabricantes de FPGAs e na integração com plataformas SoC para viabilizar sistemas de análise de áudio *end-to-end*.

Referências

Abdul, Z. K. and Al-Talabani, A. K. (2022). Mel Frequency Cepstral Coefficient and its Applications: A Review.

Anshu, Raghuvanshi, A., and Muchahary, D. (2022). Fpga design for efficient speech processing system.

Bahoura, M. and Ezzaidi, H. (2013). Hardware implementation of MFCC feature extraction for respiratory sounds analysis.

Boujelben, O. and Bahoura, M. (2018). Efficient fpga-based architecture of an automatic wheeze detector using a combination of mfcc and svm algorithms.

Cohen, Y., Faccio, M., and Rozenes, S. (2025). Vocal Communication Between Cobots and Humans to Enhance Productivity and Safety: Review and Discussion.

Dao, V.-L., Nguyen, V.-D., Nguyen, H.-D., and Hoang, V.-P. (2017). Hardware Implementation of MFCC Feature Extraction for Speech Recognition on FPGA. Cham.

Ehkan, P., Zakaria, F. F., Warip, M. N. M., Sauli, Z., and Elshaikh, M. (2015). Hardware Implementation of MFCC-Based Feature Extraction for Speaker Recognition.

Fariselli, M., Rusci, M., Cambonie, J., and Flamand, E. (2021). Integer-Only Approximated MFCC for Ultra-Low Power Audio NN Processing on Multi-Core MCUs.

K, N., Gadamsetty, M., and J Kailath, B. (2019). Fpga implementation of speech recognizer for isolated words.

Kou, H., Shang, W., Lane, I., and Chong, J. (2013). Efficient MFCC feature extraction on Graphics Processing Units.

Michálek, J. and Vaněk, J. (2014). An open-source GPU-accelerated feature extraction tool.

Nguyen, T., Pham, L., Nguyen, H., Bui, B., Ngo, D., and Hoang, T. (2016). A High Performance Dynamic ASIC-Based Audio Signal Feature Extraction (MFCC).

Paul S, B. S., Glittas, A. X., and Gopalakrishnan, L. (2021). A low latency modular-level deeply integrated MFCC feature extraction architecture for speech recognition. *Integration*, 76:69–75.

Tsai, T. and Wang, C. (2024). GMM-Based Speaker Verification System with Hardware MFCC in SoC Design. 83(19).

Wang, J.-C., Wang, J.-F., and Weng, Y.-S. (2000). Chip design of mel frequency cepstral coefficients for speech recognition. volume 6.

Wassi, G., Iloga, S., Romain, O., and Granado, B. (2015). FPGA-based real-time MFCC extraction for automatic audio indexing on FM broadcast data.

Ye, T., Peng, T., and Yang, L. (2025). Review on Sound-Based Industrial Predictive Maintenance: From Feature Engineering to Deep Learning. *Mathematics*, 13(11).

Yi, A. and Talakoub, O. (2008). Implementing a Speech Recognition System on a Graphics Processor Unit (GPU) using CUDA.

Zhao, C., Yamamura, N., Tsutsui, H., and Ohgane, T. (2024). Evaluation of Computational Cost and Result Accuracy in Design and Efficient Implementation of Log-Mel Spectrogram and MFCC Feature Extraction Using Fixed-Point Arithmetic on FPGA.

Zhou, Y., Al-Hawaj, K. M., and Zhang, Z. (2017). A New Approach to Automatic Memory Banking using Trace-Based Address Mining. FPGA '17.