Implementação em FPGA de um Sistema Keyword Spotting Usando Redes Neurais Convolucionais

Natan Votre, Walter A. Gontijo e Eduardo L. O. Batista

***Resumo*— Este artigo apresenta a implementação em FPGA de um sistema de keyword spotting (KWS) utilizado para detectar palavras específicas em língua portuguesa. O sistema implementado é baseado no uso de MFCCs (*mel-frequency cepstral coefficients*) como entradas (features) para uma rede neural do tipo convolucional (CNN). Inicialmente, o sistema de KWS proposto é implementado e avaliado usando a linguagem Python e a biblioteca TensorFlow. Posteriormente, é feita a síntese em FPGA, bem como a avaliação de desempenho usando um kit de desenvolvimento apropriado. Os resultados experimentais obtidos mostram um excelente desempenho do sistema de KWS implementado, além da sua capacidade de processar dezenas de canais de áudio em tempo real.**

***Palavras-Chave*— KWS, FPGA, CNN, MFCC.**

***Abstract*— This paper presents the field-programmable-gate-array (FPGA) implementation of a keyword spotting (KWS) system focused on the detection of specific words in Portuguese language. The implemented system is based on the use of mel-frequency cepstral coefficients (MFCCs) as features for a convolutional neural network (CNN). Initially, the proposed KWS system is implemented and evaluated using the Python programming language along with the TensorFlow module. Afterwards, the FPGA synthesis of the proposed system is carried out as well as its evaluation using an appropriate development kit. The obtained results show that the implemented system is very effective and capable of processing dozens of audio channels in real time.**

***Keywords*— KWS, FPGA, CNN, MFCC.**

I. Introdução

O uso de redes neurais artificiais (*artificial neural networks* - ANNs) em aplicações práticas tem crescido exponencialmente nos últimos anos. Alguns exemplos dessas aplicações são tradução automática [1], colorização de imagens em tons de cinza [2], síntese de imagens [3], síntese de fala [4] e reconhecimento de fala [5]–[9]. Especificamente no contexto de reconhecimento de fala, uma tarefa que tem ganhado destaque é a detecção de palavras-chave (*keyword spotting* - KWS) [7]–[9]. Essa tarefa consiste na detecção de uma palavra específica dentro de um vocabulário extenso, frequentemente considerando um processamento de áudio em tempo real [6].

As implementações de redes neurais artificiais são tipicamente voltadas para computadores de propósito geral, contendo vários núcleos de processamento ou ainda *graphics processing units* (GPUs) [10]. Mais recentemente, plataformas alternativas para implementação de ANNs vêm sendo buscadas, visando obter melhorias de desempenho ou redução de custos computacionais e energéticos, os quais são essenciais em aplicações embarcadas [11]. Nesse contexto, os *field programmable gate arrays* (FPGAs) têm ganhado espaço significativo [9], [12].

Natan Votre, Walter A. Gontijo, Eduardo L. O. Batista, LINSE-Laboratório de Circuitos e Processamento de Sinais do Departamento de Engenharia Elétrica e Eletrônica da Universidade Federal de Santa Catarina, Florianopolis, SC, Brasil (e-mails: natan@linse.ufsc.br, walter@linse.ufsc.br e eduardo.batista@ufsc.br).

O presente trabalho de pesquisa é dedicado ao desenvolvimento de um sistema de KWS acústico em FPGA, capaz de reconhecer diferentes palavras-chave pré-treinadas em língua portuguesa. A arquitetura de rede neural base para o desenvolvimento desse sistema é a de uma rede neural convolucional (*convolutional neural network*, CNN). O desempenho do sistema desenvolvido será avaliado a partir de simulações e experimentos práticos.

II. Visão Geral

O desenvolvimento do presente trabalho está estruturado em duas etapas principais: a primeira realizada utilizando a linguagem Python e a segunda correspondendo à implementação em FPGA propriamente dita, conforme ilustrado na Fig. 1. A partir dessa figura, nota-se que a primeira etapa (em Python) constitui-se na construção de um banco de fala, aplicação de *data augmentation*, geração de cenários, treinamento e avaliação da CNN considerada. Por outro lado, na segunda etapa, tem-se a estrutura a ser implementada no FPGA, constituindo-se de uma etapa de *framing*, seguida pelo extrator de MFCCs, o processamento pela CNN e finalmente o decisor que define se uma das *keywords* (KW1, KW2, ...) foi ou não detectada. Nas próximas seções (III e IV), cada uma dessas etapas é descrita em detalhes.

III. Desenvolvimento em *Python*

A. Banco de Fala

O primeiro passo no desenvolvimento realizado usando a linguagem Python é a construção de um banco de fala em língua portuguesa composto por um conjunto de arquivos de áudio com 72 locutores ($N_l = 72$), correspondendo a 140 minutos de áudio. Ainda, são acrescentados, a tal banco de fala, diferentes tipos de ruídos, contidos em 961 arquivos ($N_n = 961$) que correspondem a 400 minutos de áudio. O banco de fala está disponível em [13].

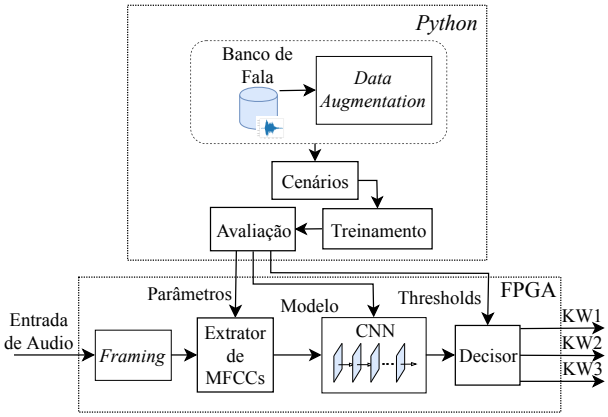

Fig. 1. Esquema geral do projeto proposto.

B. Data Augmentation

Como o banco de fala obtido é considerado relativamente pequeno, utilizou-se a técnica de *data augmentation* para expandir tal banco [14]. Os tipos de transformações de fala utilizados nesse contexto são *pitch shift*, *time shift* e *time stretch* [15].

C. Criação dos Cenários

Os cenários considerados no contexto do presente trabalho são baseados em MFCCs (*mel-frequency cepstral coefficients*) [16]. Em outras palavras, os MFCCs de trechos dos sinais de áudio contidos no banco de fala expandido serão usados como entradas para a CNN. Os parâmetros considerados para a extração desses MFCCs e seus respectivos valores são mostrados na Tabela I. Tais parâmetros estão de acordo com as recomendações encontradas em [16]–[22], enquanto os seus valores são obtidos empiricamente.

As *keywords* consideradas neste trabalho são "Sim Máquina", "Abre" e "Fecha". Assim, as classes utilizadas na saída da rede neural são: 0 = "OOV" (*Out Of Vocabulary*), 1 = "Sim Máquina", 2 = "Abre", 3 = "Fecha". Considerando tais classes, são elaborados os conjuntos de treinamento, teste e validação, devidamente balanceados. Os locutores utilizados no conjunto de treinamento não são utilizados nos conjuntos de teste e validação. A proporção entre os 3 conjuntos é de 80% para treinamento, 10% para teste e 10% para validação.

D. Treinamento

A ferramenta utilizada neste trabalho para elaborar, treinar e avaliar CNNs é o *TensorFlow* [23]. Em testes preliminares, foram consideradas 8 estruturas de CNN distintas, elaboradas empiricamente. Dentre essas estruturas, a que levou aos melhores resultados e será considerada no presente trabalho é a descrita na Tabela II. Em tal tabela, **Conv**, **Bnorm** e **FullC** representam, respectivamente, *convolutional layer*, *batch normalization* e *fully connected layer*. Além disso, a operação de *batch normalization* é representada por **Bnorm** e a função de ativação do tipo *rectifier linear unit* por **ReLU**. O resultado obtido para esta arquitetura foi de 97,72% de acurácia.

E. Avaliação

Após a obtenção do modelo da CNN, são avaliados os resultados na execução do sistema completo de KWS. Conforme apresentado na Fig. 1, na implementação em FPGA, a saída da CNN será enviada para um bloco Decisor. Tal bloco realiza a suavização da sequência de classificações e, através de limiares predefinidos, obtém a sequência de *keywords* identificadas. O valor de tais limiares foram obtidos utilizando o algoritmo *brute-force search* [24], no qual o *score* é a média geométrica entre as acurácias de cada classe. O melhor *score* obtido foi de 97,4%, com limiares dados por 0,5; 0,4; e 0,3. As taxas de detecções corretas foram de 98,31%, 99,15% e 97,75% para as *keywords* "Sim Máquina", "Abre" e "Fecha", respectivamente. Já as taxas de falso-positivo foram de 5,31% e 1,73% para voz e ruído, respectivamente.

TABELA I

PARÂMETROS UTILIZADOS NA EXTRAÇÃO DE MFCCS

Símbolo	Significado	Valor
f_s	Taxa de amostragem do sinal de entrada	8 kHz
N_{fr}	Número de amostras utilizadas em cada *frame*	512
N_{hop}	Número de amostras por passo no sinal de áudio	256
$window$	Janela utilizada	*Hanning*
N_{FFT}	Número de *bins* da FFT	512
f_i	Frequência inicial do banco de filtros Mel	100 Hz
f_o	Frequência final do banco de filtros Mel	4000 Hz
N_f	Número de filtros Mel	64
m	Número de elementos do vetor MFCC	35

TABELA II

MODELO DA CNN COM MELHOR DESEMPENHO

Camada	Bloco	Formato de Entrada	Formato de Saída
1	Conv1	15x35x1	11x31x4
	Bnorm1	11x31x4	11x31x4
	ReLU1	11x31x4	11x31x4
2	Conv2	11x31x4	8x27x6
	ReLU2	8x27x6	8x27x6
3	Conv3	8x27x6	7x23x8
	ReLU3	7x23x8	7x23x8
4	Conv4	7x23x8	6x19x10
	ReLU4	6x19x10	6x19x10
5	Conv5	6x19x10	6x16x12
	ReLU5	6x16x12	6x16x12
6	MaxPooling	6x16x12	3x8x12
7	Conv6	3x8x12	2x6x14
	ReLU6	2x6x14	2x6x14
8	FullC1	168	100
	ReLU7	100	100
9	FullC2	100	4

IV. IMPLEMENTAÇÃO EM FPGA

Nesta seção, a implementação dos blocos do sistema de KWS em FPGA é detalhada. A partir da Fig. 1, nota-se que essa implementação é dividida nos blocos: *Framing*, Extrator de MFCCs, CNN e Decisor. Tais blocos são sincronizados por um mesmo sinal de *clock*, o qual é também a referência de tempo para fins de avaliação.

A. Framing

Framing é o bloco de entrada da implementação em FPGA do sistema de KWS, sendo ele responsável por separar as amostras de entrada em *frames* ($N_{fr} = 512$ amostras, passo de $N_{hop} = 256$ amostras) e realizar o janelamento. Conforme apresentado na Tabela I, a taxa de amostragem f_s do sinal de entrada é 8 kHz e a janela é do tipo *Hanning*. O bloco *Framing* é implementado em ponto fixo com resolução de entrada de 16 bits e saída de 32 bits.

B. Extrator de MFCCs

O Extrator de MFCCs é responsável pelo cálculo dos MFCCs a partir dos *frames*. A Fig. 2 mostra o diagrama interno detalhado de tal extrator. Observa-se nessa figura que ele é composto pelos seguintes sub-blocos: *Power Spectrum Calculator*, *FFT Calculator*, *Mel-bank Filters Chain*, *Power to dB Chain*, *DCT Calculator* e *MFCC Matrix Shaping*. Note que os nomes desses blocos são representativos de suas funções, as quais são detalhadas em [16]. A implementação do Extrator de MFCCs é feita em ponto fixo com resolução de entrada de 32 bits e saída de 16 bits.

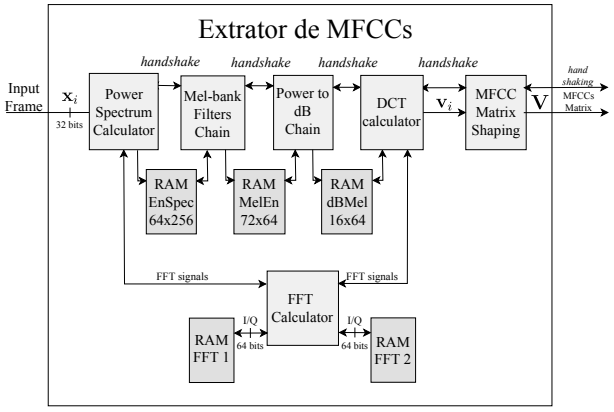

Fig. 2. Diagrama interno do Extrator de MFCCs.

C. CNN

A Fig. 3 mostra o diagrama detalhado da implementação da CNN em FPGA. Os blocos denominados Camada 1 até Camada 9 nesse diagrama correspondem às camadas descritas na Tabela II. A implementação é realizada em ponto fixo com resolução de 16 bits, ou seja, as entradas e saídas de todos os blocos possuem essa resolução. Além disso, a implementação das camadas é feita de forma independente, possibilitando um processamento paralelo no FPGA. A comunicação entre os blocos é feita através de sinais de *handshake* e o armazenamento de resultados é feito em memórias RAM externas aos blocos (veja Fig. 3).

Um aspecto importante a se ressaltar com relação à CNN implementada neste trabalho é a utilização de quantização dinâmica [25]. Essa técnica consiste em realizar a quantização das saídas e dos pesos de cada camada individualmente, visando obter uma melhor distribuição de valores de forma a evitar problemas como perda de precisão e *overflow* em implementações de ponto fixo.

Como mencionado anteriormente (veja Seção III-D e Tabela II), a implementação da CNN escolhida envolve camadas dos tipos convolucional (**Conv**), de *max-pooling* (**MaxPooling**) e *fully connected* (**FullC**), além da operação de *batch normalization* (**Bnorm**) e função de ativação do tipo *rectifier linear unit* (**ReLU**). A implementação desses elementos da CNN é descrita a seguir.

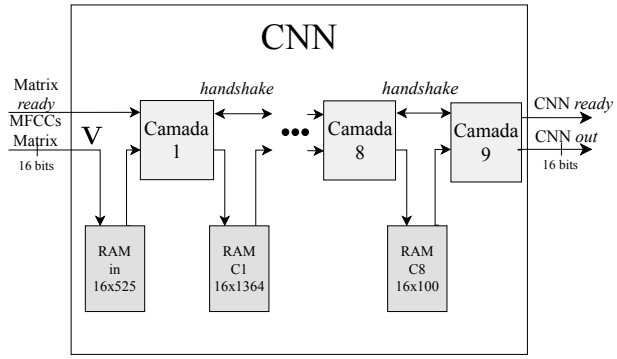

Fig. 3. Diagrama interno do bloco CNN.

*1) **Conv**:* A Figura 4 mostra o diagrama de interno do bloco **Conv**, o qual corresponde à implementação de uma camada do tipo convolucional. Conforme mostrado nessa figura, uma memória ROM é utilizada para armazenar os coeficientes dos

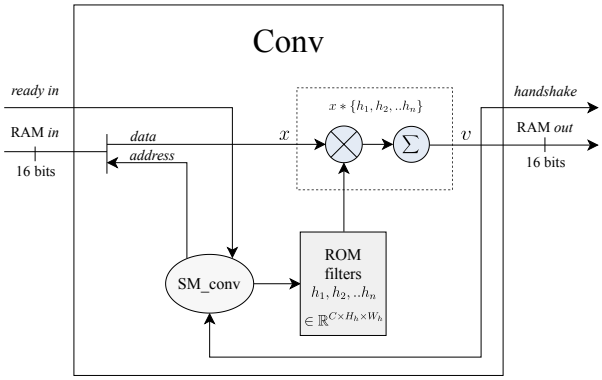

Fig. 4. Diagrama interno do bloco **Conv**.

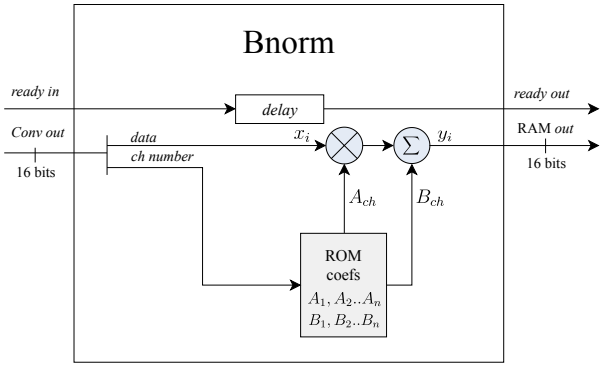

Fig. 5. Diagrama interno do bloco **Bnorm**.

filtros $h_1, h_2, ..h_n$. Além disso, a máquina de estados *SM_conv* controla a leitura dos dados das memórias RAM *in* e ROM *filters* utilizados para realizar os cálculos de convolução. As amostras de saída do bloco possuem resolução de 16 bits. Após finalizados os cálculos da convolução, o sinal *handshake* sinaliza ao próximo bloco que um tensor v está disponível na saída.

*2) **MaxPooling** e **FullC**:* Os blocos **MaxPooling** e **FullC** são implementados de forma similar ao bloco **Conv**, diferindo apenas nas operações matemáticas realizadas internamente. Para manter a concisão deste trabalho, a estrutura interna desses blocos não será detalhada aqui.

*3) **Bnorm**:* Este bloco realiza a operação de *batch normalization* [26]. A Figura 5 mostra o diagrama interno para implementação de tal bloco. Além das operações mais comuns, como somas e multiplicações, note que a **Bnorm** usa um bloco de atraso (*delay*) para sincronizar o sinal *ready in*, enviado por um bloco anterior (e.g., um bloco **Conv**), com a disponibilização do resultado em *RAM out*.

*4) **ReLU**:* Este bloco realiza o cálculo da função de ativação *rectifier linear unit*, da qual a saída $f(x)$ é o máximo entre 0 e a entrada x. Portanto, o bloco **ReLU** utiliza o bit de sinal do valor de entrada para selecionar entre colocar a saída para 0 ou repassar o valor de entrada para saída. Por simplicidade, a estrutura de tal bloco não será ilustrada aqui.

D. Decisor

O bloco Decisor, ilustrado na Figura 6, é responsável por obter a sequência de *keywords* identificadas a partir das classificações realizadas pelo bloco CNN. Em tal figura, o bloco *Max Function* recebe as predições realizadas pela CNN

e as classifica pelo maior valor, obtendo na saída os números correspondentes às classes consideradas (0="OOV", 1="Sim Máquina", 2="Abre", 3="Fecha"). Tais números são salvos na memória RAM *predict*, a qual armazena as últimas 14 classificações, sendo esse valor definido a partir de resultados experimentais preliminares. Na sequência, o bloco *Keywords Smoothing* realiza a suavização das classificações para cada *keyword*, obtendo probabilidades de ocorrência. Finalmente, no bloco *Distance Threshold Function*, são analisadas as probabilidades e, através dos *thresholds* pré-definidos (veja Seção III-E), a *keyword* identificada é obtida.

V. Resultados

Esta seção é dedicada à apresentação dos resultados obtidos com este trabalho de pesquisa. Assim, inicialmente o FPGA alvo é descrito em conjunto com os resultados de síntese obtidos. Na sequência, são apresentadas avaliações do KWS implementado, primeiramente, em termos de precisão em comparação com a implementação de referência e, posteriormente, em termos de desempenho na detecção das palavras-chave. Finalmente, o desempenho em tempo real do sistema implementado é discutido.

A. Resultados de Síntese

O dispositivo FPGA alvo usado neste trabalho é o EP4CE115F29C7 da família Cyclone IV E. A síntese do código do sistema de KWS para o FPGA foi realizada usando ferramenta Quartus Prime da Intel [27], enquanto as simulações foram realizadas usando o ModelSim [28]. Os resultados obtidos com a síntese são sumarizados na Tabela III. Note que o sistema proposto pôde ser implementado com folga no dispositivo alvo escolhido, utilizando 18% da capacidade total de memória, 15% dos multiplicadores disponíveis e apenas 4% do total de elementos lógicos. A frequência máxima f_{max} obtida para sinal de *clock* foi de 262 MHz.

Especificamente, com relação aos principais blocos do sistema, o Extrator de MFCCs utiliza cerca de 270 kbits de memória e realiza sua operação em 24904 ciclos de *clock*. O bloco CNN, por sua vez, utiliza 441 kbits de memória e realiza o processamento em 186148 ciclos de *clock*. Por fim, o bloco Decisor utiliza somente 60 bits de memória e realiza seu processamento em apenas 57 ciclos de *clock*.

B. Comparação de Blocos Implementados em Python e FPGA

Nesta seção, deseja-se avaliar os resultados da comparação entre a implementação em FPGA (em ponto fixo) com a de

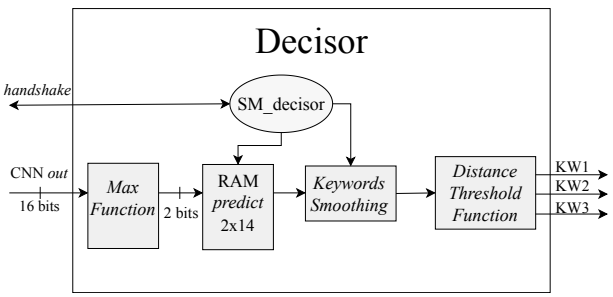

Fig. 6. Diagrama interno do Decisor.

TABELA III
Resultado de síntese no FPGA.

Família	Cyclone IV E
Dispositivo	EP4CE115F29C7
Total de elementos lógicos	5.023 / 114.480 (4%)
Total de registradores	3.423
Total de bits de memória	711.474 / 3.981.312 (18%)
Multiplicadores de 9 bits	78 / 532 (15%)
Frequência máxima (*speed grade* 3)	262 MHz

referência em Python, considerando a precisão e a quantização utilizada no FPGA. Essa avaliação é focada no Extrator de MFCCs e na CNN propriamente dita. Com respeito ao primeiro, a Fig. 7 mostra a comparação de um vetor de MFCCs obtido em *Python* com o correspondente obtido da implementação em FPGA. Observa-se, nessa figura, uma pequena diferença entre os vetores, a qual é justificada pelo erro de quantização envolvido na implementação em FPGA. De fato, o elemento que mais contribui para tal erro é o bloco *Power to dB Chain*, no qual o cálculo de um logaritmo na base 2 é realizado. Ainda, em testes por simulação, o erro quadrático médio (EQM) entre os vetores obtidos em *Python* e com o FPGA foi avaliado. O EQM calculado para os vetores de entrada do bloco *Power to dB Chain* foi de $2{,}11 \cdot 10^{-3}$, enquanto que, na saída desse bloco, o EQM obtido foi de $6{,}20 \cdot 10^{-1}$. Esse resultado mostra que tal bloco constitui-se realmente em uma importante fonte de perda de precisão. Testes de avaliação de EQM similares foram realizados também para o bloco CNN. Nesse caso, observou-se que o erro de quantização pode ser desconsiderado para tal bloco, especialmente devido ao uso da técnica de quantização dinâmica utilizada neste trabalho (veja Seção IV). Especificamente, o EQM calculado na saída do bloco CNN foi de $3{,}09 \cdot 10^{-2}$.

C. Avaliação do Sistema de KWS em FPGA

A avaliação do KWS em FPGA foi realizada usando o kit de desenvolvimento DE10-Standard [29], o qual permite processar arquivos de áudio utilizando o sistema operacional Ubuntu 16.04. Assim, o KWS implementado no FPGA recebe os arquivos de áudio do banco de fala desenvolvido (Seção III-A), produzindo resultados que são salvos em arquivos para permitir a avaliação das taxas de detecções corretas e de falsos positivos. Os *thresholds* utilizados no KWS são os apresentados na Seção III-E.

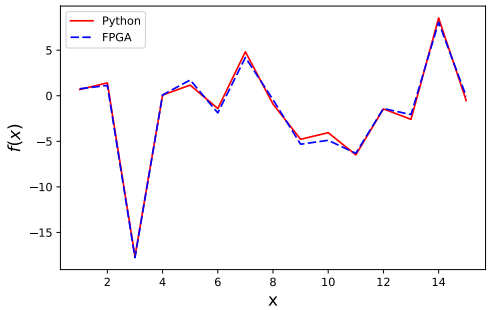

Fig. 7. Comparação de um vetor de MFCCs extraído pelo desenvolvimento em *Python* e pela implementação em FPGA.

A Tabela IV apresenta os resultados de desempenho do KWS em *Python* e em FPGA. Conforme mostrado nessa tabela, as taxas de detecções corretas são bastante próximas, com uma já esperada ligeira vantagem para implementação de referência em Python. Particularmente, na avaliação das taxas de falsos positivos, a implementação em FPGA foi capaz de obter um melhor desempenho para sinais de voz. Tais resultados mostram que foi possível obter uma implementação em FPGA bastante eficiente, sendo ela praticamente equivalente à desenvolvida em *Python*.

TABELA IV
COMPARAÇÃO ENTRE IMPLEMENTAÇÕES EM *Python* E FPGA

		Python	FPGA
Taxa de Detecção Correta	KW1	98,31%	96,90%
	KW2	99,15%	97,18%
	KW3	97,75%	96,62%
Taxa de Falso Positivo	Voz	5,31%	3,75%
	Ruído	2,12%	4,60%

D. Análise de Tempo Real

Considerando que os blocos *Framing*, Extrator de MFCCs, CNN e Decisor realizam o processamento em paralelo, a restrição de tempo de processamento do sistema de KWS desenvolvido é dada pelo bloco CNN, o qual requer um maior número de ciclos de *clock* para realizar sua operação. Assim, considerando que o número de ciclos requerido pela CNN é 186.148, o tempo de processamento do KWS é dado por aproximadamente $\frac{186.148}{f_{clk}}$, onde f_{clk} é a frequência de *clock*. Considerando, ainda, que a frequência de amostragem é 8 kHz e que o número de amostras para se ter uma nova janela é 256, tem-se que o período mínimo para processamento em tempo real do KWS é $\frac{256}{8.000} = 32$ ms. Portanto, a frequência mínima de *clock* para operação em tempo real é de $f_{min} = 186.148\frac{8.000}{256} = 5{,}9$ MHz. Assim, considerando que a frequência máxima de operação obtida com a síntese do sistema proposto é $f_{\max} = 262$ MHz, conclui-se que o sistema implementado em FPGA permite realizar o processamento do KWS a uma taxa 45 vezes maior do que a de tempo real, ou ainda que, por meio de multiplexação, é possível realizar o processamento de até 45 canais de áudio em tempo real.

VI. COMENTÁRIOS E CONCLUSÕES FINAIS

Neste artigo, a implementação em FPGA de um sistema para detecção de palavras-chave (*keyword spotting*) em língua portuguesa é apresentada. A implementação desenvolvida é baseada em extração de MFCCs, os quais são usados como *features* de entrada de uma CNN que faz a classificação/detecção de palavras-chave. Em contraste com implementações tradicionais de redes neurais, que usam CPUs e/ou GPUs, a implementação proposta considera apenas um único dispositivo FPGA. A síntese e implementação é realizada no kit DE10-Standard e seu funcionamento é avaliado através de simulações e experimentos. Os resultados obtidos mostram um excelente desempenho do sistema obtido, tanto em termos de taxas de acerto nas detecções, quanto de tempo de processamento e *throughtput*.

REFERÊNCIAS

[1] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint:1409.0473*, 2014.

[2] J. Hwang and Y. Zhou, "Image colorization with deep convolutional neural networks," Stanford University, Tech. Rep., 2016. Available: http://cs231n.stanford.edu/reports2016/ 219Report.pdf, Tech. Rep.

[3] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, "Generative adversarial text to image synthesis," *arXiv preprint:1605.05396*, 2016.

[4] R. Maia and R. Seara, "Um sistema TTS baseado em redes neurais profundas usando parametros síncronos de pitch," *Simposio Brasileiro de Telecomunicações e Processamento de Sinais*, 2017.

[5] K. Hwang, M. Lee, and W. Sung, "Online keyword spotting with a character-level recurrent neural network," *arXiv preprint:1512.08903*, 2015.

[6] C. T. Lengerich and A. Y. Hannun, "An end-to-end architecture for keyword spotting and voice activity detection," *arXiv preprint:1611.09405*, 2016.

[7] G. Chen, C. Parada, and G. Heigold, "Small-footprint keyword spotting using deep neural networks," in *2014 IEEE Int. Conf. Acoust., Speech and Sig. Process. (ICASSP)*, Mountain View, CA, USA, May 2014, pp. 4087–4091.

[8] M. Shah, J. Wang, D. Blaauw, D. Sylvester, H. S. Kim, and C. Chakrabarti, "*A fixed-point neural network for keyword detection on resource constrained hardware*," in *2015 IEEE Workshop on Signal Process. Syst. (SiPS)*, Hangzhou, China, Oct. 2015, pp. 1–6.

[9] S. Ö. Arik, M. Kliegl, and et al., "Convolutional recurrent neural networks for small-footprint keyword spotting," *arXiv preprint:1703.05390*, 2017.

[10] K.-S. Oh and K. Jung, "Gpu implementation of neural networks," *Pattern Recognition*, vol. 37, pp. 1311–1314, Jun 2004.

[11] J. Chase, B. Nelson, J. Bodily, Z. Wei, and L. Dah-Jye, "Real-time optical flow calculations on FPGA and GPU architectures: A comparison study," in *Proc. of the 16th IEEE Symp. on Field-Programmable Custom Computing Machines*, Palo Alto, CA, May 2008, pp. 173 – 182.

[12] S. Li, K. Choi, and Y. Lee, "Artificial neural network implementation in FPGA: A case study," in *2016 Int. SoC Design Conf. (ISOCC)*, Jeju, South Korea, 2016, pp. 297–298.

[13] N. Votre, "Repositório do TCC," https://gitlab.com/natan.votre/repositorio-tcc.

[14] T. Ko, V. Peddinti, D. Povey, and S. Khudanpur, "Audio augmentation for speech recognition," in *INTERSPEECH*, Dresden, Germany, 2015.

[15] J. Salamon and J. P. Bello, "Deep convolutional neural networks and data augmentation for environmental sound classification," *arXiv preprint:1608.04363*, 2016.

[16] J. R. Deller, J. H. L. Hansen, and J. G. Proakis, *Discrete-Time Processing of Speech Signals*. Hoboken, NJ: Wiley, 1999.

[17] W. Brent, *Physical and Perceptual Aspects of Percussive Timbre*. San Diego, CA: University of California, 2010.

[18] L. Rabiner, L. Rabiner, and B. Juang, *Fundamentals of Speech Recognition*. Upper Saddle River, NJ: Prentice Hall, 1993.

[19] B. Logan, "Mel frequency cepstral coefficients for music modeling," in *Proc. Int. Soc. Music Inf. Retrieval Conf. (ISMIR)*, Plymouth, MA, 2000.

[20] B. P. Lathi, *Linear Systems and Signals*, 2nd ed. Oxford, UK: Oxford University Press, 2009.

[21] V. Tiwari, "MFCC and its applications in speaker recognition," *International Journal on Emerging Technologies*, vol. 1, pp. 19–22, 2010.

[22] S. A. Khayam, "The discrete cosine transform: Theory and application," 2003.

[23] "Tensorflow repository," https://github.com/tensorflow/tensorflow.

[24] R. Stephens, *Essential Algorithms: A Practical Approach to Computer Algorithms*, 1st ed. Hoboken, NJ: Wiley, 2013.

[25] X. Wei, W. Liu, L. Chen, L. Ma, H. Chen, and Y. Zhuang, "FPGA-based hybrid-type implementation of quantized neural networks for remote sensing applications," in *Sensors 2019*, vol. 19, no. 924, 2019, pp. 297–298.

[26] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint:1502.03167*, 2015.

[27] Intel, "Quartus prime," https://www.intel.com/content/www/us/en/programmable/downloads/download-center.html.

[28] I. FPGA, "Modelsim," https://www.intel.com.br/content/www/br/pt/software/programmable/quartus-prime/model-sim.html.

[29] TerasIC, "DE10-standard FPGA development kit," https://www.terasic.com.tw/cgi-bin/page/archive.pl?No=1081.