# Integer-Only Approximated MFCC for Ultra-Low Power Audio NN Processing on Multi-Core MCUs

Marco Fariselli
*Greenwaves Technologies*
marco.fariselli@greenwaves-technologies.com

Manuele Rusci
*Università di Bologna*
*Greenwaves Technologies*
manuele.rusci@unibo.it

Joel Cambonie
*Greenwaves Technologies*
joel.cambonie@greenwaves-technologies.com

Eric Flamand
*Greenwaves Technologies*
eric.flamand@greenwaves-technologies.com

***Abstract*—Given the recent advances in the design of efficient Deep Neural Networks (DNN) for tiny edge devices, the feature extraction frontend has become a computation bottleneck for enabling audio processing on low-end MicroController Units (MCUs). To address this challenge, this work presents novel hardware-aware integer quantization schemes for the Mel-Frequency Cepstral Coefficients (MFCC) feature extractor. Our high-precision integer-only 32 bit approximated flow does not lead to accuracy degradation with respect to a full-precision implementation when feeding multiple DNN models for Audio Keyword Spotting applications. In contrast, a second low-precision 16-bit approximated MFCC algorithm presents a 0.6% lower accuracy but results 3× faster. Additionally, by leveraging on an 8-cores MCU, GAP8, our solution results 9.8× faster than the full precision MFCC deployed on an FPU-suited MCU. When integrated within an optimized end-to-end system for Keyword Spotting, a GAP8-based audio smart device presents an overall power consumption as low as 3.4mW, demonstrating up to 35 days of lifetime with a single AA battery.**

***Index Terms*—MelSpectrogram, MFCC, Keyword Spotting, MicroControllers, Multi-Core, TinyML**

## I. Introduction

TinyML is becoming pervasive and rapidly impacting the ecosystem of next-generation audio devices, such as hearables and voice assistants. These devices feature on-board processing capabilities to analyze the audio signal, e.g. human speech, as captured by microphone sensors. Because of their battery-powered real-time operation, the data processing engines are affected by severe latency and energy constraints, which demand optimized hardware-aware data analytics algorithms.

In the context of Keyword Spotting (KWS) systems, i.e. devices that recognize a given keyword(s) in a speech utterance, solutions based on Deep Neural Networks (DNNs) have shown the highest accuracy [1]. The KWS DNN models are commonly fed with a frequency-domain representation of the audio signal, e.g. Log-Mel filter bank energies (LFBE) or Mel-frequency cepstral coefficients (MFCC), which result to be the most effective transformations for this task.

In the view of smart audio devices, KWS algorithms have been recently demonstrated on low-power MicroControllers (MCUs) [1], which are typically preferred to hardwired integrated circuits because of the flexibility given by SW programmability. Unfortunately, current implementations of feature extraction modules, such as MFCC, leverage floating-point software libraries, which results in a computational bottleneck for the majority of low-end MCUs lacking of floating-point arithmetic units.

To address this issue, we present a hardware-aware approximated MFCC feature extractor and we provide a first optimized fixed-point software implementation for MCUs[1]. The proposed module, when combined with a KWS DNN model trained on MFCC full-precision features, preserves the accuracy of the classifier without the need of retraining. Moreover, we optimize our MFCC library to run on a multi-core RISC-V MCU, GAP8, by leveraging on the DSP-oriented instruction set and the parallel computation paradigm.

The contributions of this work are:

- We propose two optimized Mel Spectrogram integer-only approximation techniques: a high-precision (*HP32*) ultra-accurate and a faster low-precision (*LP16*) but less accurate version.
- We present an optimized software MFCC library for GAP8, which leverages on 8 core parallelism and SIMD low-precision instructions.

Thanks to the proposed techniques, we show an end-to-end KWS application running in real-time on a multi-core MCU class device (GAP8) and achieving an accuracy of $94\%$ on the Google Speech Commands dataset without relying on floating-point computation. In particular, the MFCC feature extractor takes as low as 6.8ms to process an audio frame (40 msec at 16kHz) and results $9.8\times$ faster, in terms of clock cycles, than a full-precision solution implemented on an FPU equipped MCU, such as an *STM32L4*. Also, by leveraging DVFS techniques, the proposed system achieves a power consumption of 2.4mW that leads to a battery lifetime of several weeks (1.2V & 2400mAh AA battery for 35 days), when considering a 1mW power cost of a microphone sensor.

## II. Related Work

Recent battery-operated smart audio devices integrate Audio Keyword Spotting as the processing pipeline frontend to reduce the energy consumption [2], [3]. In the latest years, DNN-based solutions have demonstrated to be highly accurate to solve this task. The work [1] introduced low-complexity (less than 80MOps) KWS solutions for tiny devices achieving up

[1]The Software Library is Open-Source at https://github.com/GreenWaves-Technologies/gap_sdk/tree/master/tools/autotiler_v3/Generators/MFCC

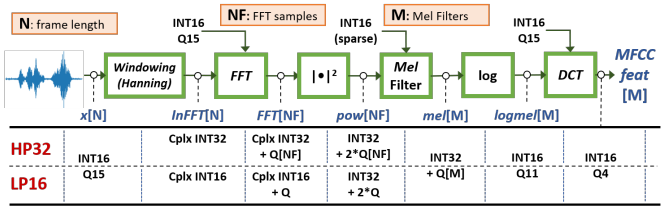


Fig. 1. Block diagram of the Integerized MFCC pipeline featuring high-precision (*HP32*) or low-precision (*LP16*) arithmetic. The datatype of intermediate results is reported in the bottom.

to 95% accuracy on the Google Speech Command dataset [4]. The proposed approaches included multiple kinds of DNNs, which were fed with the MFCC features extracted from the audio signal.

To efficiently bring Audio processing, including KWS, on deep edge devices, several works presented dedicated circuits for low-power applications [5]–[7]. To extract energy features over multiple frequency channels audio, [5] presented a mixed-signal implementation of the MFCC based on analog bandpass filters, which has been employed in [8] for keyword spotting applications. On the contrary, among the fully digital solutions, the work [6] presented an integrated KWS system featuring, as audio preprocessing, a circuit consisting of a serial FFT-based MFCC. To recover the accuracy of the KWS algorithm with respect to the Tensorflow baseline, the authors trained the DNN models on the approximated MFCC features but limiting the detection capabilities to a set of only two words. Vocell [7] also embeds integrated feature extraction, KWS and Speaker Verification on the same die. Despite the effectiveness and the low-energy consumption of the proposed algorithm approximations, these solutions lack flexibility which is instead featured by our work.

On the other side, DSP solutions for audio processing gain high flexibility thanks to software programmability. Recently, the work [1] firstly presented a DNN-based KWS solution for microcontrollers. The implementation relies on an optimized software backend CMISIS-NN [9] while the MFCC leverages an optimized floating-point implementation. Our work goes beyond this seminal paper by introducing an algorithm approximation of the MFCC extractor to leverage fixed-point arithmetic and by exploiting multi-core acceleration and DVFS techniques to improve energy efficiency.

## III. Feature Extraction for Audio DNN Processing

In this section, we focus on MFCC, which is a widely adopted approach for many DNN-based audio applications. In the following, we firstly describe the MFCC pipeline and then we present our approximated solutions for MCU targets featuring low-precision integer-only arithmetic.

### A. Background on MFCC

Figure 1 illustrates the MFCC components and the dataflow that applies on an audio frame $x$ of $N$ data points to produce $M$ frequency features. After a windowing filter, e.g. Hanning, the signal feeds an FFT block to compute the Discrete Time Fourier Transform (DFT). The filtered signal *InFFT* is zero-padded to reach a power-of-two size (*NF*). Then, the spectrogram of the FFT is extracted by computing the module of the complex values. Alternative MFCC implementations return the squared module of complex values (Periodogram), avoiding the highly expensive square root function required in the complex module computation. The dot product between the Periodogram $pow$ and a set of $M$ triangular filter banks, i.e. the *Mel* filters, provides the Mel energies values. This block aims at mimicking the non-linear human sound perception, by discriminating differently the low and high audio frequencies. For this reason, the components of *Mel* filters banks result highly sparse, i.e. the majority of the filters coefficients is zero. Lastly, the logarithmic values of the Mel Filter output (*LogMel*) feed a DCT block to produce the $M$ MFCC features.

### B. Integerized MFCC

Since Audio DNN models are typically trained on full-precision feature vectors, current available software implementations of the MFCC feature extractor for MCUs relies on floating-point arithmetic [1]. On the contrary, we target a hardware-aware approximation of the MFCC algorithm that (*i*) relies on integer-only arithmetic for devices lacking FPU engines, (*ii*) reduces the memory footprint of coefficient parameters and (*iii*) does not lead to accuracy degradation for Audio DNN inference on the target device.

Starting from the full-precision MFCC baseline described in Section III-A, we firstly target a high-precision integer-only flow, denoted as *HP32*, which closely approximate the baseline algorithm. In this solution, which is depicted in Fig. 1, the output of the windowing function is an array of *Cmplx int32* fixed-point values. Additionally, we use an *int8* vector $Q$ of N elements to store the exponent of the values, i.e. a per-element scaling factor $2^{-Q[i]}, i = 1, ..N$. Hence, we make use of 40 bits to represent every element but keeping the integer and exponent values into two separate arrays, i.e. not paying latency overhead for misaligned memory accesses. The core computation of the Radix-2 FFT, which consists of 2 additions and 2 multiplication, matches the scaling factors of the fixed-point operands by means of shift operations before the sum. The FFT products are computed by multiplying and shifting the fixed point values and the *int16* twiddles (fixed-point Q15). To compute the Mel energies, we multiply the *int32* data, i.e. the Periodogram with the non-zero elements of the sparse Mel filter banks (dot-product operation). We only store the first and the last indexes of the non-zero elements in the triangular shaped filters to reduce the memory cost. Also, to prevent overflow while computing the dot product, we evaluate at runtime the dynamic range of the operands by checking the position of the leading 1 in the bit representation of the input maximum value. For each Mel filter, we right shift the input elements if $pos\_lead\_one(max(input)) + 15 + 2 > 32$ (we empirically add +2 to take into account the sum over multiple products). Then, we store the applied element-wise shift factors in a buffer, which is used later during the log

computation. To compute the logarithm of a fixed-point data, we rely on the intrinsic properties of the logarithm operator:

$$log(x[i]) = log(X[i] \cdot 2^{-Q[i]}) = log(X[i]) - Q[i] \cdot log(2) \quad (1)$$

where $log(2)$ is stored as a fixed-point constant, while $log(X[i])$ is computed with a 3rd order Taylor series approximation with integer-only operations. The fixed-point *int16* elements produced by logarithm operator share the same scaling factor $Q$. Lastly, the DCT operation consists of a dot product between the log output and the *int16* DCT coefficients matrix (Q15 format). We empirically found that a Q11 format for the log and Q4 for the DCT outputs lead to a good trade-off between dynamic and precision of the real value numbers.

To improve the performance, i.e. reduce the latency, of the proposed approximated algorithm, we further reduce the bitwidth of the FFT and Periodogram input/output buffers to *int16*. On one hand, a lower bitwidth reduces the memory footprint of the data buffers and, on the other hand, enables the usage of 2x16 bit SIMD instructions. We refer to this low-precision flow as *LP16* MFCC.

## IV. Audio Processing on a Multi-Core MCU

In this section we describe an optimized implementation of the *HP32* and the *LP16* MFCC approximated algorithms on an energy-efficient multi-core MCU and the integration into an end-to-end Keyword Spotting (KWS) pipeline. The targeted processing platform is GAP8 [10], an ultra-low power System-on-Chip (SoC) that includes a cluster of 8 general-purpose RISC-V cores coupled with a 64kB scratchpad data memory tailored to accelerate compute-intensive tasks, e.g. DNN inference. Besides a large set of peripherals, the SoC features a single RISC-V core, i.e. the Fabric Controller (FC), and a 512kB on-chip L2 memory. The FC controls the software execution and can optionally offload tasks to the multi-core cluster. The FC and the Cluster subsystem feature separate Voltage-Frequency domains. Hence, DVFS can be individually applied and controlled by software. To efficiently support DSP operations (e.g. DNN inference), the Instruction Set of the 9 (8+1) embedded RISC-V cores features low-bitwidth SIMD vector instructions, including 4x8-bit and 2x16-bit MAC.

We rely on a data-parallelism paradigm to accelerate our Integerized MFCC pipeline of Fig. 1 on GAP8. The FFT features $s = log_2(N_F)$ computation stages, each one with $N/2$ butterflies, i.e. the low-level computation units. Within our multi-core implementation, the computation of the even and odd-numbered DFT outputs is distributed among all the cores. To optimize the reuse of the twiddle factors during the computation of the $N/2$ butterflies, we implement two nested loops: an outer loop $i = 0, .., N_F/2^{s+1}$ and an inner loop $j = 0, .., 2^s$. To improve the multicore speedup, the first $log_2(N_F) - 3$ stages are parallelized on the outermost loop while the last 3 stages on the innermost loop. For the Mel energy computation, each core computes the dot product between the input Periodogram and $\lfloor \frac{M}{N_{cores}} \rfloor$ Mel Filters. The same strategy is exploited in the DCT calculation. For the piecewise operations like square and log calculation, every core process a subset of the operands. Moreover, SIMD instructions are leveraged for the $LP16$ MFCC to compute the FFT butterflies and the Periodogram, making use of vectorized multiplications and sums between *Complex int16*.

The parallel optimized MFCC function is integrated into an end-to-end Keyword Spotting (KWS) pipeline. Given an audio clip of 1 sec (with a sampling rate of 16kHz), the computed MFCC features feed a DNN model for classifying the audio sample into one of the 10 speech command classes provided by the dataset [4]. We consider the audio classification DS-CNN models of [1], which achieve the best accuracy score among the presented ones. DNN models are quantized to 8 bit for the deployment on GAP8. A parallel C code implementation of the inference function is obtained thanks to the GAP*flow* toolset provided by the chip vendor.

To design an ultra-low-power and robust KWS audio application, we perform speech command classification every 0.5 sec, hence considering a 50% of overlapping between input audio clips. On the one side, the overlap leads to a higher detection accuracy but, on the other side, imposes a latency constraint more strict than a non-overlapped scenario because the inference time, including both MFCC and the DS-CNN, must be lower than $1\text{sec} - \text{overlap}$. Additionally, we apply a DVFS strategy, i.e. maximally reducing voltage and frequency, to minimize the power consumption under the aforementioned latency constraint.

## V. Experimental Results

In this section, we evaluate the proposed Integerized MFCC approximations in terms of latency and accuracy, as scored by the end-to-end KWS model, and we provide a system-level energy evaluation.

To replicate the setup of [1], we trained the DS-CNN classification models, namely *DS-CNN-SMALL*, *DS-CNN-MEDIUM* and *DS-CNN-LARGE*, on the MFCC features extracted from 1 sec audio clips of the Speech Commands dataset [4]. Concerning the MFCC, we rely on the full-precision Tensorflow implementation by setting the audio frame length to 40 ms and 20 ms of overlap, hence a total of 49 frames per audio clip to classify. Despite the MFCC block computes 40 features, we feed the CNNs with the first 10 coefficients of each frame, i.e. $10 \times 49$ MFCC features, without any accuracy penalty with respect to larger feature maps, demonstrating that the majority of the speech information resides in the first coefficients of the Mel spectrogram. Moreover, we apply 8bit quantization-aware training for the classifier. The trained and quantized CNN models are deployed on the GAP8 platform, in combination with our approximated MFCC functions *HP32* and *LP16*.

Figure 2A depicts the computational cost of each step of the MFCC for an individual 40ms frame, normalized with respect to the *HP32* single-core implementation. In the *HP32* version, 90% of the time is spent during the FFT task. The *LP16* shows a more balanced computation partitioning, but the FFT still remains the most expensive step, taking around 60% of the total time. The 8-cores implementation shows a $5.9\times$ speed up. The discrepancy from the theoretical maximum

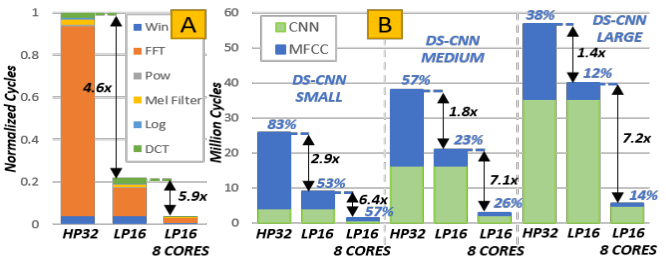


Fig. 2. (A) Inference time on single frame MFCC for each step, normalized to the total *HP32* computation cycles. (B) Number of cycles of both KWS pipeline blocks in case of *LP16* and *HP32* MFCC preprocessing and with the different sized CNNs. The MFCC percentage cost is indicated over each bar.

speed up of $8\times$ is due to two main factors: (*i*) most of the FFTs stages are parallelized on the outermost loop (Section IV), which results advantageous for the early steps but the number of iterations rapidly decreases along the stages causing an unbalanced execution on the 8 cores, (*ii*) the parallel Mel Filter sparse computation processes a subset of the filter banks on each core and, therefore, some of the cores receive large bandwidth filters while others very selective ones. This makes the computation unbalanced among the cores and leads to a speed-up of only $4.5\times$ on this block. Overall, the 8-core execution of our *LP16* MFCC algorithm resulted $9.8\times$ faster in terms of clock cycles than the floating-point MFCC running on an FPU-equipped MCU such as the *STM32L4*.

Figure 2B, on the other hand, shows the latency breakdown, in terms of clock cycles, of the end-to-end KWS pipeline when running on GAP8. For all the DS-CNN models, we report the performance for the single-core and 8-cores implementation, when leveraging the *HP32* or the *LP16* MFCC algorithm. In case of low-complexity DNN models, i.e. *DS-CNN-SMALL* or *DS-CNN-MEDIUM*, a less-approximate *HP32* MFCC algorithm leads to a processing time for feature extraction higher, respectively by $5.2\times$ and $1.3\times$, than the DNN inference time. Also for the largest network, the *HP32* MFCC takes 38% of the total processing time if running on a single core. With a $> 3\times$ speed up w.r.t. the high precision version, the *LP16* MFCC solution shows a more balanced computation taking the 53%, 23% and 14% for respectively the *DS-CNN-SMALL*, *DS-CNN-MEDIUM* and *DS-CNN-LARGE* KWS.

Table I reports the accuracy on the Speech Command dataset of the deployed KWS pipeline, which combines the quantized DS-CNN models with our Integer MFCC versions. These solutions are compared against the baseline model trained with full-precision MFCC features and quantized CNN (Tensorflow implementation). When featuring a *HP32* MFCC, the KWS DNN models results as accurate as the full-precision baseline. The LP16 approximation, instead, presents an accuracy degradation with respect to the baseline of less than 0.6% for all the models. Furthermore, we report in Table I the number of clock cycles to compute the MFCC feature extraction and the DS-CNN inference on the 8 cores of the GAP cluster.

Lastly, we analyzed the power consumption of a KWS always-on device when leveraging the most energy-efficient configuration, i.e. *LP16* MFCC coupled with a *DS-CNN-SMALL* model. Because the SoC cannot be powered off to guarantee continual audio data acquisition for KWS, we apply a DVFS technique to reduce the system average power consumption. To this aim, we set the SoC Voltage to 1V and a Fabric Controller & Cluster frequencies to respectively 10MHz and 7MHz. In this operating point, GAP8 shows an average power consumption of 2.4mW. If combined with a microphone sensor consuming 1mW, the system achieves a lifetime of 35 days when powered with a single 2000mAh (1,2V) AA battery.

TABLE I
ACCURACY AND PERFORMANCE OF MFCC+CNN KWS

| | TF-MFCC + TF Quant CNN | | HP32 MFCC + GAP CNN | | LP16 MFCC + GAP CNN | |
|---|---|---|---|---|---|---|
| CNN (#MAC) | Acc. | MCyc | Acc. | MCyc | Acc. | MCyc |
| SMALL (2.7M) | 93.45 | - | 93.50 | 3.1+0.6 | 92.86 | 0.8+0.6 |
| MEDIUM (9.8M) | 94.10 | - | 94.05 | 3.1+2.2 | 93.99 | 0.8+2.2 |
| LARGE (28.4M) | 94.62 | - | 94.72 | 3.1+6.2 | 94.09 | 0.8+4.8 |

## VI. Conclusion

In this work we presented integer-only arithmetic solutions for MFCC feature extractions. A high-precision approximate function showed a negligible accuracy loss with respect to a full-precision baseline, but resulting $3\times$ slower than a low-precision 16 bit approximation, showing a 0.6% lower accuracy on a KWS use case. Our solutions, when coupled to a DNN model for KWS, and deployed on the multi-core GAP8 presented a power cost of 2.4mW, which leads to a battery lifetime of 35 days.

## Acknowledgments

This work has been funded by the H2020-ECSEL CPS4EU European project (grant agreement 826276).

## References

[1] Y. Zhang, N. Suda, L. Lai, and V. Chandra, "Hello edge: Keyword spotting on microcontrollers," *arXiv preprint arXiv:1711.07128*, 2017.

[2] A. Gruenstein, R. Alvarez, C. Thornton, and M. Ghodrat, "A cascade architecture for keyword spotting on mobile devices," *arXiv preprint arXiv:1712.03603*, 2017.

[3] M. O. Khursheed, C. Jose, R. Kumar, G. Fu, B. Kulis, and S. K. Cheekatmalla, "Small footprint convolutional recurrent networks for streaming wakeword detection," *arXiv preprint arXiv:2011.12941*, 2020.

[4] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *arXiv preprint arXiv:1804.03209*, 2018.

[5] Y. Zhang, X. Qiu, Q. Li, F. Qiao, Q. Wei, L. Luo, and H. Yang, "Optimization and evaluation of energy-efficient mixed-signal mfcc feature extraction architecture," in *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2020, pp. 506–511.

[6] W. Shan, M. Yang, T. Wang, Y. Lu, H. Cai, L. Zhu, J. Xu, C. Wu, L. Shi, and J. Yang, "A 510-nw wake-up keyword-spotting chip using serial-fft-based mfcc and binarized depthwise separable cnn in 28-nm cmos," *IEEE JSSC*, vol. 56, no. 1, pp. 151–164, 2020.

[7] J. S. P. Giraldo, S. Lauwereins, K. Badami, and M. Verhelst, "Vocell: A 65-nm speech-triggered wake-up soc for 10-$\mu w$ w keyword spotting and speaker verification," *IEEE JSSC*, vol. 55, no. 4, pp. 868–878, 2020.

[8] Q. Li, S. Lin, C. Liu, Y. Liu, F. Qiao, Y. Wang, and H. Yang, "Ns-kws: joint optimization of near-sensor processing architecture and low-precision gru for always-on keyword spotting," in *Proc. of the ACM/IEEE ISLPED*, 2020, pp. 97–102.

[9] L. Lai, N. Suda, and V. Chandra, "Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus," *arXiv preprint arXiv:1801.06601*, 2018.

[10] E. Flamand, D. Rossi, F. Conti, I. Loi, A. Pullini, F. Rotenberg, and L. Benini, "Gap-8: A risc-v soc for ai at the edge of the iot," in *IEEE 29th Int. Conf. on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2018, pp. 1–4.