Vocell: A 65-nm Speech-Triggered Wake-Up SoC for 10-μW Keyword Spotting and Speaker Verification

Juan Sebastian P. Giraldo, Steven Lauwereins, *Student Member, IEEE*, Komail Badami, *Student Member, IEEE*, and Marian Verhelst, *Senior Member, IEEE*

***Abstract*— The use of speech-triggered wake-up interfaces has grown significantly in the last few years for use in ubiquitous and mobile devices. Since these interfaces must always be active, power consumption is one of their primary design metrics. This article presents a complete mixed-signal system-on-chip, capable of directly interfacing to an analog microphone and performing keyword spotting (KWS) and speaker verification (SV), without any need for further external accesses. Through the use of: 1) an integrated single-chip digital-friendly design; b) hardware-aware algorithmic optimization; and c) memory- and power-optimized accelerators, ultra-low power is achieved while maintaining high accuracy for speech recognition tasks. The 65-nm implementation achieves 18.3-μW worst case power consumption or 10.6-μW power for typical real-time scenarios, 10$\times$ below state of the art (SoA).**

***Index Terms*— Keyword spotting (KWS), machine learning (ML) hardware, speaker verification (SV), speech recognition.**

I. Introduction

AUTOMATIC speech recognition (ASR) has become widely present in many mobile devices, such as wearables or cellphones. More and more, embedded applications demand speech-triggered devices to be always-ON, always listening to the environment, in order to process commands spoken by the user. Popular commercial voice assistants, such as Siri or Alexa, use one or several keywords for the activation of the natural language query system, allowing to reduce expensive cloud accesses while providing always-ON operability. The increased use of such interfaces is coupled with the emergence of more accurate and complex machine learning (ML) models, capable of providing higher accuracy for better user satisfaction [2]. However, the energy constraints associated with embedded systems are in conflict with the use of advanced state-of-the-art (SoA) ML algorithms, such as computationally complex neural networks [3], random forests [4], or support vector machines (SVMs) [5].

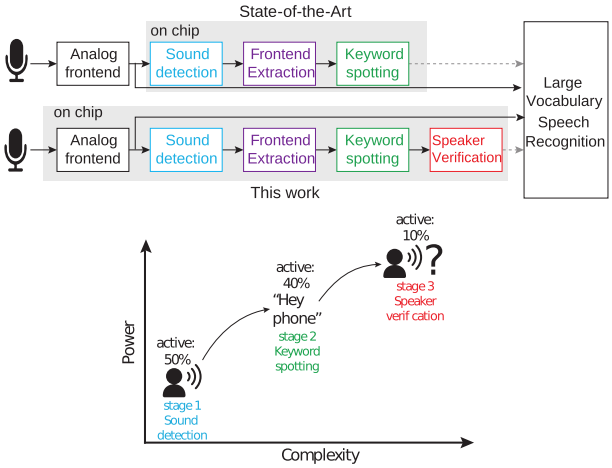

Fig. 1. (Top) Staged functionality for a speech-triggered wake-up system. (Bottom) Hierarchical system functionality of the presented wake-up engine. In this example, sound activation is assumed to be active 50% of the time, KWS 40%, and SV 10%.

Recent hardware developments, such as [1] and [6], tackled the problem of embedded voice command recognition through the design of ASICs that drastically improve energy efficiency compared to general-purpose solutions. These systems integrate wake-up sound detection (SD), feature extraction (FEx), and ML accelerators to detect speech commands [Fig. 1 (top)]. This kind of approach attains very good accuracy on standard KWS data sets at a power consumption of about hundreds of microwatts. However, these solutions present some drawbacks. First, they lack efficient always-ON implementations required to maintain sufficient battery lifetime in energy-constrained devices. Assuming a typical SR44 button cell battery with a total capacity of 190mAh/1.55 V [energy budget for the system on chip (SoC) ignoring extra board circuitry] and a continuous operation time of 24 months (e.g., battery lifespan for an on-wall control panel), the maximum allowed power consumption for the chip would correspond to 17 μW; in contrast with the 100 μW attained by the referred solutions. Second, these systems do not support speaker selectivity, which means the device is not customized for the target user and triggers on anyone's voice. Finally, SOA implementations require off-chip analog circuitry and storage to operate, since they lack an

Manuscript received August 20, 2019; revised November 9, 2019 and December 23, 2019; accepted January 9, 2020. Date of publication February 3, 2020; date of current version March 26, 2020. This article was approved by Guest Editor Ken Takeuchi. This work was supported by the Research Foundation-Flanders (FWO) and the EU European Research Council (ERC) through the Program RE-SENSE under Grant 715037. *(Corresponding author: Juan Sebastian P. Giraldo.)*

Juan Sebastian P. Giraldo and Marian Verhelst are with the Department of Electrical Engineering, KU Leuven ICTS, 3001 Heverlee, Belgium (e-mail: giraldo@esat.kuleuven.be).

Steven Lauwereins is with Televic Rail, 8870 Izegem, Belgium.

Komail Badami is with CSEM Zurich, 8005 Zurich, Switzerland.

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSSC.2020.2968800

on-chip analog frontend (AFE) and sufficient on-chip memory, diminishing their autonomy.

In this article, we propose Vocell, an integrated mixed-signal SoC for keyword spotting (KWS) and speaker verification (SV) that attains ultra-low power consumption while maintaining state-of-the-art recognition accuracy. For this purpose, the concept of hierarchical detection is exploited [Fig. 1 (bottom)]. A set of tasks with increasing complexity (SD, KWS, and SV) are cascaded, allowing the activation of posterior stages by previous steps in the pipeline. The pipeline is optimized by taking into consideration the uneven distribution of input examples: Easy-to-process speech segments (e.g., background noise) are common, and are, hence, handled by the first efficient stages of the hierarchy (e.g., SD). More complex stages for KWS and SV are gradually activated in a data-driven way, significantly reducing the average power consumption. The design of such a hierarchical system presents challenges at three levels of abstraction: 1) algorithm; 2) architecture; and 3) circuit level. At the architecture level, the system complexity increases, as more customized hardware modules (e.g., FEx and ML engines) have to be integrated (as shown in Section IV) increasing implementation and verification time compared with a non-hierarchical system. Finally, at the circuit level, effective adaptive power management techniques, such as clock gating, must be efficiently implemented to avoid unnecessary power consumption and reducing latency wake up.

To this end, this article presents the following contributions.

1) Development of a self-contained SoC for KWS and SV, integrating all the necessary stages from analog preprocessing to labeling (Section II).
2) Hardware-aware algorithmic optimization for each subtask and their combined interplay (Section III).
3) Power and memory-optimized hardware accelerators for FEx, SD, and ML classification (Section IV).
4) Hierarchical execution of the processing tasks, reducing average power consumption through selective module activation, proven by measurements (Section V).

II. Speech Hierarchy

A wake-up speech processing system for voice command recognition is composed of a pipeline of stages, as described in Fig. 1. In Sections II-A–II-E, each one of the stages is discussed along with the algorithms/techniques chosen for the Vocell system.

A. Analog Frontend

The AFE receives the raw speech signal from an external source, such as a passive or active microphone and subsequently amplifies, filters unwanted frequency bands, and digitizes it for posterior processing. State-of-the-art AFEs operating from reduced supply voltage, such as 0.6 V, suffer from the following shortcomings: 1) limited distortion performance due to inoperable cascode structures thus limiting the open-loop gain in the frontend amplifier and 2) performance loss with technology scaling as the intrinsic gain of the amplifier $g_m r_o$ worsens with the technology scaling. The Vocell SoC overcomes aforementioned limitations by using time-domain processing in the AFE, which relies on the *voltage-to-time-to-voltage* conversion as opposed to traditional *voltage-to-current-to-voltage* to achieve a large open-loop gain for the frontend amplifier [7].

B. Sound Detection

Due to the presence of long silence intervals for many real-time applications, the use of SD allows avoiding the execution of expensive posterior processing stages, saving significant amounts of energy. For this reason, this stage must be highly discriminative while consuming low power. Several approaches have been applied to this task in the SoA, including energy calculation [8], spectral analysis [9], and ML models [10]. In this article, energy-based SD is chosen due to the limited amount of computations required and the simplicity of its implementation, which reduces the overhead of the always-on operation. Yet, a careful balancing of miss detects and false alarms is crucial, as covered in Section III-A.

C. Feature Extraction

The FEx block receives samples from the AFE and generates a set of vectors that compress the redundant information present in the speech signal and, which are used by the ML classifiers. Mel frequency cepstral coefficients (MFCCs) are used extensively as features for speech processing applications due to their high reliability as encoders of the human voice spectrum. Generally, every 16 ms, a 32-ms window of consecutive audio samples is processed to compute a feature vector of MFCCs (between 13 and 20 coefficients) [11]. Its processing pipeline is composed of: 1) an fast Fourier transform (FFT) for spectral analysis; 2) mel filtering using triangular filters; 3) Logarithmic compression; and 4) a discrete Fourier transform (DFT) over the intermediate values [11]. Due to its complex dataflow, its implementation using general-purpose compute architectures presents a large overhead in terms of power consumption [12]. This creates a need for hardware acceleration, yet without giving up all flexibility in the feature parameters.

D. Keyword Spotting

Due to the temporal nature of speech processing, the use of RNNs for KWS, and specifically long short-term memories (LSTMs), have shown very good results in terms of accuracy and computational efficiency [13]. The LSTM's compact model size and reduced number of computations per input feature vector bring a favorable balance of achievable accuracy and computational complexity. In typical speech processing networks, one or multiple layers of LSTM neurons are followed by fully connected layers for data projection. For real-time operation, every feature vector is processed sequentially, generating an output probability of the target keyword per time-window. Recent research [14] has shown it is possible to obtain accurate and compact models using small-scale LSTMs with one or two LSTM layers, each having less than 64 neurons.

E. Speaker Verification

Several algorithms have been proposed for SV, such as Gaussian mixture model (GMM) [15], SVM [16], and i-vector [17] approaches. Between them, GMMs show the lowest computational load while maintaining high accuracy for typical recognition contexts. A GMM is a probabilistic model generated by a sum of multivariate Gaussians, which allows computing the probability $P(\mathbf{f} \mid \mathbf{\Theta})$ that a feature vector f with a dimension D belongs to a specific class depending on the set of trained parameters Θ

$$P(\mathbf{f} \mid \mathbf{\Theta}) = \sum_{m=1}^{n_{gauss}} \left[\frac{w_m}{(2\pi)^{D/2} \prod_{d=1}^{D} \sigma_{m,d}} \cdot \exp\left[-\sum_{d=1}^{D} \frac{(f_d - \mu_{m,d})^2}{2\sigma_{m,d}^2} \right] \right]. \quad (1)$$

With f_d being the dth value of feature vector $\mathbf{f}$ while $\mu_{m,d}$ and $\sigma_{m,d}$ are the mean and standard deviation of the dth dimension of the mth Gaussian. Equation (1) shows the simplified equation for Gaussians with diagonal covariance matrices. Classical optimization techniques, such as expectation maximization (EM), are used for training the GMM model parameters [18]. For SV, a universal background model (UBM) is trained with several speakers, which allows to model the average user. A total of $n_{\text{gauss}} = 256$–2048 Gaussians are typically necessary, using a feature space of 20 MFCCs and their first and second order derivatives ($D = 60$). Subsequently, the UBM is used as initial seed for maximum *a posteriori* adaptation of second, speaker-specific, GMM, using the target speaker utterances as training data.

During inference, each feature vector is processed by the UBM and the target speaker GMM, resulting in the output probability for both models. The difference Λ between the output of the two GMMs is used to decide if the current speech segment corresponds to the target speaker (Fig. 2)

$$\Lambda(\mathbf{F}) = \ln(P(\mathbf{f} \mid \mathbf{\Theta_{speaker}})) - \ln(P(\mathbf{f} \mid \mathbf{\Theta_{UBM}})) \quad (2)$$

where Θ_{speaker} and Θ_{UBM} are the trained GMM model parameters for the target speaker GMM and UBM, respectively. In order to obtain an accurate classification, the output log-probabilities $\ln(P(\mathbf{f} \mid \mathbf{\Theta}_*))$ are averaged over a sequence of feature vectors (about 500 ms), before making the decision Λ. The target speaker is detected when Λ is larger than a configurable uncertainty threshold th.

Yet, the number of Gaussian parameters, the need for exponentiation for Gaussian computation, and the consequent floating-point sums generate serious challenges for the efficient deployment of GMMs on embedded platforms. Using a naive approach with $n_{\text{gauss}} = 256$ Gaussians, $D = 60$ dimensions per feature vector, and 8-b precision, the inference would demand about 8 mega-operations per second (MOPS) and 3.8 MB/s of memory bandwidth, making execution infeasible on many low-power devices. In this article, several optimizations are carried out in the algorithmic and micro-architecture domain in order to improve data reuse and hardware economy, as discussed in Section III.

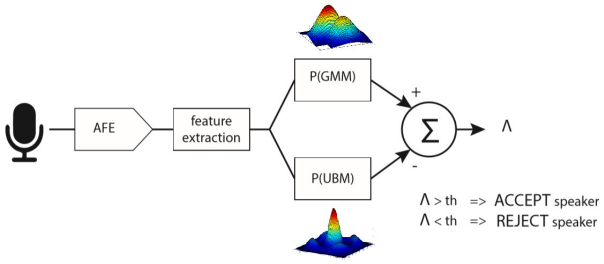

Fig. 2. Inference using the GMM-UBM SV. If $\Lambda >$ th, with th a configurable uncertainty threshold, target speaker is detected. Otherwise the hypothesis is rejected.

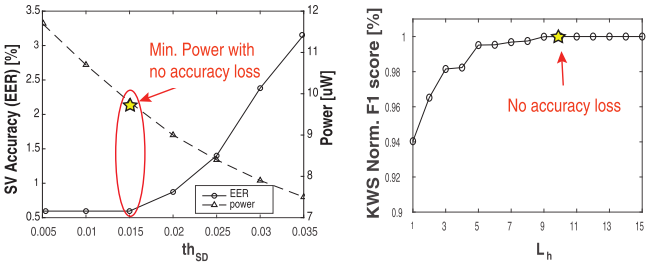

Fig. 3. (Left) Impact of SD threshold th_{SD} on SV accuracy and power consumption. (Right) Impact of hangover length L_h over KWS accuracy.

III. Hardware-Aware Algorithmic Optimizations

To reduce the computational overhead of the algorithms used in the wake-up system, hardware-aware algorithmic optimizations are carried out, exploiting algebraic reordering, approximate computing, and parameter tuning. To maintain high-task accuracy, the impact of each design decision is analyzed on standard speech data sets.

A. Sound Detection

Energy-based SD computes the energy of every incoming speech frame E through the sum of absolute values of the digitized signal: $\sum_{i=1}^{n_w} |x_i|$. If the result is higher than a configurable threshold th_{SD}, an SD signal is asserted. The hangover length L_h, which refers to the configurable amount of frames classified as sound after a sound is detected, must be optimized in order not to cut sound during soft speech segments. th_{SD} and L_h were carefully tuned to minimize system wake up, while avoiding impact on end-to-end KWS and SV accuracy (Fig. 3).

B. Feature Extraction

The computationally most expensive stage of the MFCC pipeline is the DFT phase accounting for 72% of the total computations (i.e., sums and multiplications) per window [Fig. 4 (left)]. Yet, the DCT transformation is based on similar compute kernels as the DFT transformation. This allows reusing optimized hardware resources for the DCT/DFT implementation, saving area, and consequently leakage energy.

1) Real-Point DFT: For this article, an N-point complex DFT is mapped on a single-standard radix-2 fixed point DFT block, also called a butterfly stage. The final result is obtained

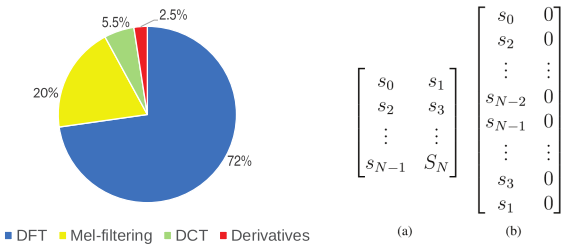

Fig. 4. (Left) MFCC Computation Breakdown in terms of total number of sums and multiplications. (Right) Input sample ordering for (a) real-point DFT and (b) DCT.

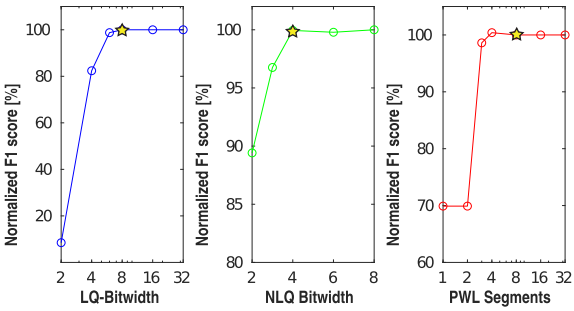

Fig. 5. Accuracy Impact of hardware-aware algorithmic optimizations for a 64-neurons LSTM trained on TIMIT data set. (Left) normalized F1 score in function of operand bitwidth. (Center) normalized F1 score in function of encoded bitwidth. (Right) normalized F1 Score in function of number of linear segments for piecewise linear approximation.

after $N\log2(N)$ steps, calculating two intermediate results per iteration k for each level $i \in [1; \log2(N)]$, with twiddle factor

$$W_N^{k2^{i-1}} = \exp^{-j\frac{2\pi 2^{i-1}k}{N}}. \tag{3}$$

With $k \in [0; N/2^i]$ and $\alpha \in [0; 2^{(i-1)}]$. To reduce the number of computations and memory accesses, the real-point DFT is computed as a complex DFT of half the size $(N/2)$ with two changes: 1) the input samples are restructured in memory such that the even samples are the real input and the odd samples are the imaginary input of the complex DFT [Fig. 4(a)]. 2) A final correction step, (4), is performed. This simplification reduces the number of butterfly computations from N to $N/2$ for each of the $\log_2\ N$ stages, effectively reducing the computations and memory accesses with a factor two

$$\chi_k^{\log_2 N} = \frac{1}{2}\left[\left(\chi_k^{\log_2 \frac{N}{2}} + \chi_{\frac{N}{2}-k}^{*\log_2 \frac{N}{2}}\right) - \left(\chi_k^{\log_2 \frac{N}{2}} - \chi_{\frac{N}{2}-k}^{*\log_2 \frac{N}{2}}\right)\left(jW_N^{kN/2}\right)\right] \tag{4}$$

with χ^* the complex conjugate of χ.

2) DCT: It is possible to compute a DCT via a DFT transformation, through three main steps: 1) reshuffling of the input; 2) computation of a complex DFT; and 3) a final correction step. In the first step, the input data are reshuffled so that all even samples are successively stored in the first half of the memory. The odd samples are in reverse order stored in the second half of the memory, as shown in Fig. 4(b). In the correction step (step 3) the final DCT is extracted from the complex DFT by applying the following transformation to the intermediate result:

$$X_k = \chi_k^{\log_2 N} \exp^{-j\frac{2\pi k}{N}} \frac{2}{\sqrt{2N}} \qquad \forall k \neq 0$$
$$X_k = \chi_k^{\log_2 N} \exp^{-j\frac{2\pi k}{N}} \frac{\sqrt{2}}{\sqrt{2N}} \qquad \forall k = 0. \tag{5}$$

C. LSTM

The HW-algorithmic optimizations for KWS stage target to find the most HW-efficient model, capable of achieving satisfactory KWS accuracy. To this end, accuracy will be benchmarked on a set of KWS tasks from different data sets; namely, Texas Instruments and Massachusetts Institute of Technology Acoustic-Phonetic Continuous Speech Corpus (TIMIT), Google speech command data set (GSCD), and TIDIGITS with a target vocabulary of 4, 12, and 10 classes, respectively. All networks were trained using the Keras framework, dividing the data into training, validation, and testing with a distribution 80–10–10. Accuracy was selected as a performance metric for TIDIGITS and GSCD since the examples are evenly distributed between classes. On the other hand, for TIMIT, F1 score was used since the number of positive examples was significantly higher than the negative instances. Four different algorithmic optimizations were carried out, which will be explained in Sections III-C.1–III-C.4.

1) Fixed Point Analysis: The impact of fixed point neural network computation [19] was evaluated with the objective of finding the minimal bitwidth without performance losses compared with a floating-point implementation. Using quantization post-training, different operand bitwidths were assessed. As observed in Fig. 5 (left), using 8 bits for all weights and activations guarantees the baseline accuracy whereas a smaller bitwidth produces undesirable accuracy loss.

2) Nonlinear Quantization: Deep compression is an algorithmic technique that allows encoding the neural network weights into a reduced word length codebook through clustering, reducing the model memory size [20]. Fig. 5 (center) shows the impact of nonlinear quantization (NLQ) for different encoded bitwidths. Sixteen weight clusters (4-bit weight encoding) prove to be sufficient to represent the 8-bit decoded weights. The required on-line weight decoding from 4- to 8-bit words is implemented via lookup tables (LUTs) and allows to maintain the baseline accuracy while reducing memory bandwidth by $2\times$.

3) Piecewise Linear Approximation: The nonlinear functions used for LSTM networks (tanh and sigmoid) demand high computational complexity if rigorous exactness is required. Nonetheless, the natural resilience of neural networks can be exploited using a piecewise linear approximation for the nonlinear functions. Based on this, the impact of the number of linear segments was assessed for the KWS problem, as observed in Fig. 5 (right), eight linear segments show to be sufficient to maintain baseline accuracy. The cutoff points of the linear segments are stored in a lookup table for both tanh and sigmoid, and at run-time linear interpolation between the

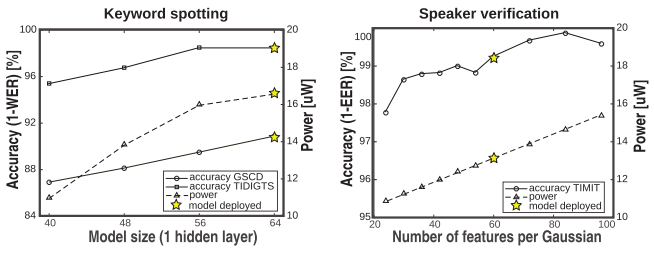

Fig. 6. Accuracy/power tradeoff for (Left) KWS and (Right) SV.

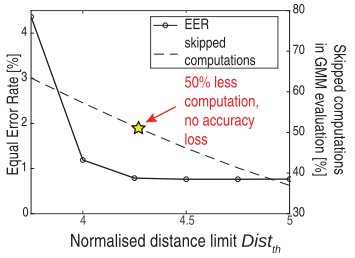

Fig. 7. Effect of distance threshold Dist_{th} on the SV accuracy and total skipped computations for GMM inference.

segment, corners allow efficiently computing the approximated nonlinear functions.

4) Model Size Impact: Models with a varying number of LSTM hidden units were trained in order to evaluate the accuracy/complexity tradeoff. As will be shown with measurements later [Fig. 6 (left)] accuracy on par with SoA deep learning KWS systems [1], [21] can be achieved by one LSTM layer with 64 neurons, requiring 26 kB of model memory (8 bits per weight). To maintain flexibility, the chip is implemented to support any model comprised of LSTM and fully-connected (FC) layers, with less than 32k model parameters.

D. GMM

Likewise, for the GMM algorithm, several HW-optimization techniques were assessed against their accuracy implication at the task level. To this end, a binary detection problem is defined over the TIMIT data set, classifying the input speech as the target speaker or not. The UBM is trained on 168 speakers, and the speaker-specific GMM on a single speaker. For both, each speaker has eight audio files in the training set, and one in the test set. The accuracy is measured as the equal error rate (EER), which is the point where the number of false positives is equal to the number of false negatives.

1) Mathematically Equivalent Computational Optimizations: In order to reduce the computational complexity of (1) and fitting the dataflow requirements to digital hardware, a couple of techniques were applied. First, base 2 was used instead of the natural exponent, rewriting (1) as

$$P(\mathbf{f} \mid \mathbf{\Theta}) = \sum_{m=1}^{n_{\text{gauss}}} \left[\frac{w_m}{(2\pi)^{D/2} \prod_{d=1}^{D} \sigma_d} \cdot 2^{\hat{}} \left(-\sum_{d=1}^{D} \frac{(f_d - \mu_{m,d})^2}{2 \ln 2 \sigma_{m,d}^2} \right) \right]. \quad (6)$$

This allows removing the need for computing natural exponents which are difficult to deploy in digital hardware. Second of all, a simple reordering of last equation puts the exponentiation as the last step before the summation over all the Gaussians

$$P(\mathbf{f} \mid \mathbf{\Theta}) = \sum_{m=1}^{n_{\text{gauss}}} \left[2^{\hat{}} \left(\log_2 \left(\frac{w_m}{(2\pi)^{D/2} \prod_{d=1}^{D} \sigma_{m,d}} \right) - \sum_{d=1}^{D} \left(\frac{f_d - \mu_{m,d}}{\sqrt{2 \ln 2} \sigma_{m,d}} \right)^2 \right) \right]. \quad (7)$$

Moving most part of the operations to the fixed-point log-domain reduces the need for floating point calculations. In addition, factors of (7) can now be precomputed and stored in memory, reducing the amount of online calculations. Instead of storing the Gaussian parameters $\mu_{m,d}$, $\sigma_{m,d}$, and w_m in the model memory, $\mu_{m,d}$, $(1/((2 \ln 2)^{1/2} \sigma_{m,d}))$, and $\log_2(w_m/((2\pi)^{D/2} \prod_{d=1}^{D} \sigma_{m,d}))$ are stored.

2) Approximate Computational Optimizations: Computational load can be reduced further by using approximate computing techniques: the feature vectors that are "far" from the center of a Gaussian contribute only marginally to the final GMM computation. More specifically, the probability of a value belonging to a specific Gaussian decrease exponentially with the distance to the mean. For example, for a 1-D feature vector with a distance 3σ from the center of a 1-D Gaussian, its probability of belonging to this Gaussian is less than 0.3%. Based on these observations, one can use, in each dimension d, the σ_d^2 normalized distance of the feature vector f_d to the Gaussian's center μ_d, as a metric to discard non-relevant Gaussians

$$\text{Dist}_d = \frac{f_d - \mu_d}{\sqrt{2 \ln(2)} \sigma_d^2}. \quad (8)$$

When in any dimension Dist_d exceeds a programmable value Dist_{th}, the probability of the Gaussian under consideration is too small to meaningfully contribute to the overall GMM. Therefore, the computation of this probability can safely be skipped. Since computing the distance Dist_d is part of the computations required for evaluating a Gaussian (7), this intermediate result is used to discard non-relevant Gaussians. More specifically, as soon as the distance Dist_d along any dimension d of the Gaussian surpasses Dist_{th}, the remainder of the computations of that specific Gaussian is aborted. Using a lower Dist_{th} means reducing the number of Gaussians computed at a potential cost in terms of accuracy. Fig. 7 shows the impact of this threshold Dist_{th} on the SV task accuracy, and the resulting computational savings. For this article, $\text{Dist}_{\text{th}} = 4.25$ was used, reducing 50% of the computations without having an impact on SV accuracy.

IV. IC Architecture

A. High-Level Architecture

The complete chip architecture for Vocell is shown in Fig. 8. The AFE receives the incoming analog signal from an external microphone and digitizes the sample thanks to a high-linear amplifier and an SAR ADC. The values obtained from the AFE are passed through the FEx Unit, which generates mel-frequency cepstral coefficients (MFCCs) and their respective

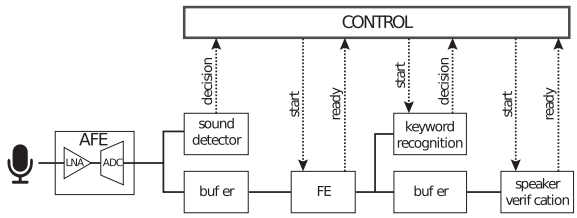

Fig. 8. IC architecture.

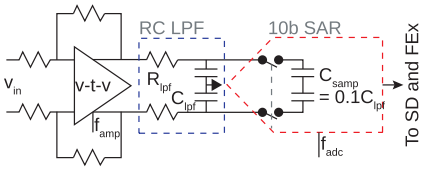

Fig. 9. AFE architecture.

derivatives. This feature vector is used by the two ML accelerators, the GMM and LSTM accelerators. Every module is configurable via dedicated registers allowing to tune several execution parameters depending on the performance and energy demands (Section III). The micro-architecture of each module is presented in Sections IV-B–IV-G.

B. Analog Frontend

The AFE integrated into the SoC is shown in Fig. 9. The AFE consists of a highly linear time-domain amplifier whose output is digitized by a highly power-efficient 10-b oversampling SAR ADC. The AFE is discussed in detail in [7] and is summarized here. The time-domain amplifier uses a dynamic comparator to translate the voltage domain input signal from the sensor to time-domain variations using pulse-density modulation-based encoding. This time-domain information is converted back into the voltage domain using a charge pump, as shown in Fig. 9. This enables the time-domain amplifier to achieve a large open-loop gain whose value is inversely proportional to the operating frequency of the dynamic comparator. The amplifier output is digitized with a 10-b 8$\times$ oversampling SAR ADC [22]. Since the AFE design uses highly dynamic building blocks and the amplifier precision depends on the operating frequency of the dynamic comparator, the AFE power-precision performance potentially improves with technology scaling.

C. Sound Detector

The micro-architecture for SD is shown in Fig. 10. As described in Section III, the energy calculation is based on the sum of the magnitude of each data sample in a frame. Comparing this value with the respective threshold evaluates the incoming sound on the usefulness of its information content. Since in standard speech recognition tasks there is an overlap of half a window between consecutive processing windows, the average energy of each half window is saved in registers D and consequently summed together, reusing the intermediate results for the next computation.

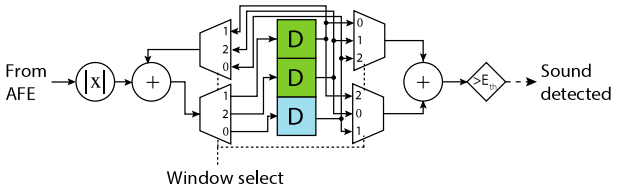

Fig. 10. Sound detector architecture. If window select is equal to 0, the third register D (blue) accumulates the energy of half window; whereas the first and second registers (green) are summed together.

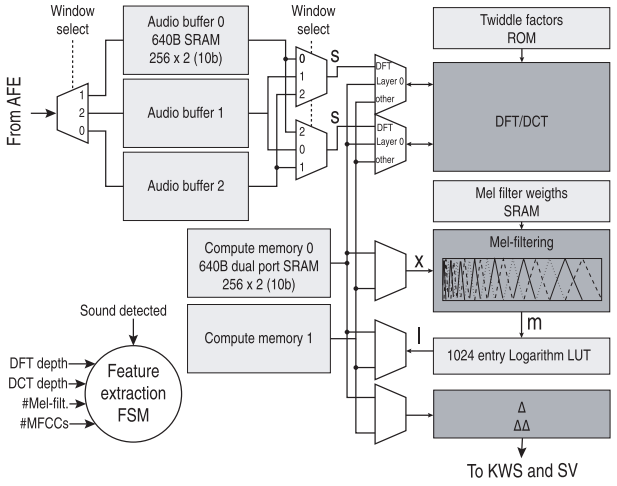

Fig. 11. FEx architecture.

D. Feature Extraction

Upon SD, the FEx is woken up. After this, the FEx module computes 20 MFCCs with their respective delta and delta–deltas at programmable center frequencies. The DFT-size, mel filters, and DCT size are parametrizable via configuration registers. Fig. 11 shows the micro-architecture of the engine. Customized hardware is designed for each one of the MFCC stages with the DFT stage based upon the design shown in [1], which was further optimized for low power.

1) Audio Buffering: Three audio buffers are implemented to interface with the ADC due to the overlap between audio windows. When one buffer is being filled with new data, the other two are used for computing the current DFT. The capacity of each buffer is configurable up to 512 10-b samples.

2) DFT: A single butterfly stage [Fig. 12 (top)] was implemented in fixed-point hardware allowing to compute two intermediate results per cycle. A dedicated FSM controls the dataflow depending on the DFT/DCT configuration parameters. In the first step, two values are read from each one of the input audio buffers, and the result is written to two dual-port SRAM computer memories. For the following even cycles, two data points are read from the first dual-port SRAM and the twiddle factor is retrieved from the twiddle ROM, to perform a butterfly computation. At the same time, the butterfly results of the previous clock cycle are saved to the second dual-port SRAM. For odd cycles, the two input values are retrieved from the second dual-port SRAM for the butterfly computation, while previous cycle results are saved to the first dual-port SRAM. In order to compute a 512-point complex DFT, 2048 cycles are required. As mentioned earlier, all dimensions

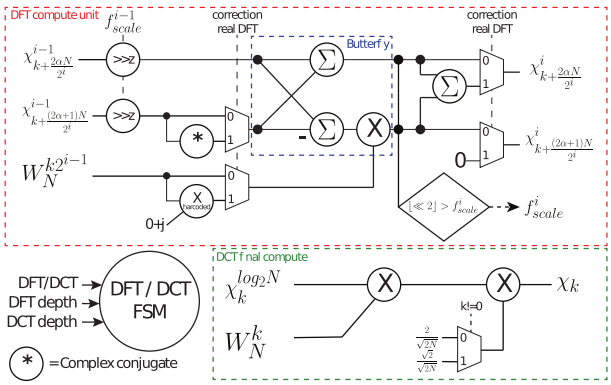

Fig. 12. DFT/DCT dataflow. (Top) Butterfly stage for DFT computation. During correction step [(4)] *correction real DFT* is set to 1. (Bottom) Final DCT computation.

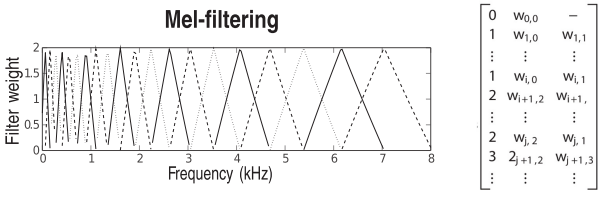

Fig. 13. (Left): Triangular mel filters for frequencies up to 8 kHz. (Right): Storage format for the mel-filter bank weights.

are configurable up to a 512-point complex DFT, or a 1024-point real DFT. To avoid computation overflow, each cycle the outputs are evaluated on their magnitude. When they exceed $|0.5|$, the inputs in the next cycle are shifted down such that no value exceeds such maximum value.

As explained in Section III, the computation of a real point DFT requires a correction step (4). This encompasses the conjugation of one of the χ^{i-1} operands, multiplication of the twiddle factor with j and summation of the two results χ^i. The multiplication with j is hard-coded by negating the imaginary part and swapping the real and imaginary part.

3) Mel Filtering and Logarithm: After the completion of the DFT, mel filtering and dynamic range compression through logarithm scaling are performed. The mel filters are triangular filters, evenly spaced below 1 kHz and mel spaced above it [Fig. 13, (left)]. It supports a maximum of 32 mel filters. The special characteristics of this kind of filter are exploited in order to reduce the memory footprint of the filter coefficients. Since each DFT output value contributes only up to two filters, a special memory format is proposed [Fig. 13, (right)]: The memory contains one row for each DFT output value, composed of three parts: highest filter index, weight 0 and weight 1. The first item refers to the highest Mel filter index to which the specific DFT value contributes. Weight 0 and Weight 1 are the filter weights for the even and odd filter band to which the DFT value contributes. These optimizations allow saving M filter bands into 3M words, with $5b$ and $12b$ for the filter index and the weights respectively. This reduces mel-filter parameter storage by $400\times$, compared to a weight matrix of 1024×32. This matrix would be necessary if each of the 32 mel filters could act upon the 1024 point DFT output. A logarithm operation works over the mel filtering output, implemented through a lookup table. Using the 10 b of the mel filtering output as the address of a 1024-entries LUT, a CORDIC implementation is avoided.

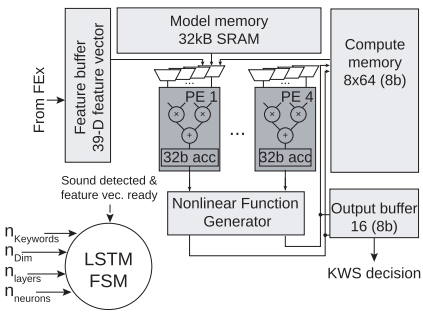

Fig. 14. LSTM architecture.

4) DCT + Time Derivatives: In order to reuse available hardware, the DFT execution engine is exploited to compute the DCT as explained in Section III-B2. To perform the final transformation of a DCT from a complex DFT, a dedicated circuit [Fig. 12 (bottom)] implements the multiplications detailed in 5. To compute the first and second order time derivatives, all intermediate values are stored. The first order time derivative is $9\times$ samples long, with filter response: $[-1\ -1\ -1\ -1\ 0\ 1\ 1\ 1\ 1]$; and the second order time derivative is $3\times$ samples long, with filter response: $[-1\ 0\ 1]$.

The required dimensionality for KWS and SV is 13 and 20 MFCC values, respectively. For KWS, this means only the first 13 MFCC values are selected out of the 32 MFCC values computed. For SV, the last 24 MFCC values are pairwise averaged to 12 new MFCC values. That is MFCC value 8 and 9 are combined to form MFCC value 8 for SV. The same combining is performed for time derivatives.

E. LSTM Accelerator

The LSTM accelerator works over every incoming frame produced by the FEx sequentially Fig. 14. It has support for LSTM networks with 1 or 2 hidden layers with up to 64 neurons per layer and 1 or 2 FC layers. The model weights are fully stored in a 32-kB SRAM, storing 8-bits linear format or 4-bits nonlinear encoding. In case NLQ is used, an LUT is leveraged to decode the compressed weights before they are sent to the processing array. Each one of the LSTM gate kernels is mapped to one of the four processing elements. The data from the model weights, the input feature vector, and the hidden state are routed to each PE in order to compute the necessary dot products. After finishing all the matrix-vector multiplications, the accumulation outputs are passed through an activation function generator which computes the nonlinear functions tanh and sigmoid through an eight-segment LUT-based function approximation. The same PE array is reused for computing the elementwise multiplications of the LSTM algorithm. More details of the accelerator can be found in [23].

F. GMM Accelerator

The GMM engine (Fig. 15) allows computing the log-likelihood ratio between a customized GMM and an UBM to recognize a target speaker (2). In order to reduce the memory bandwidth, parallelism at the input vector level is applied.

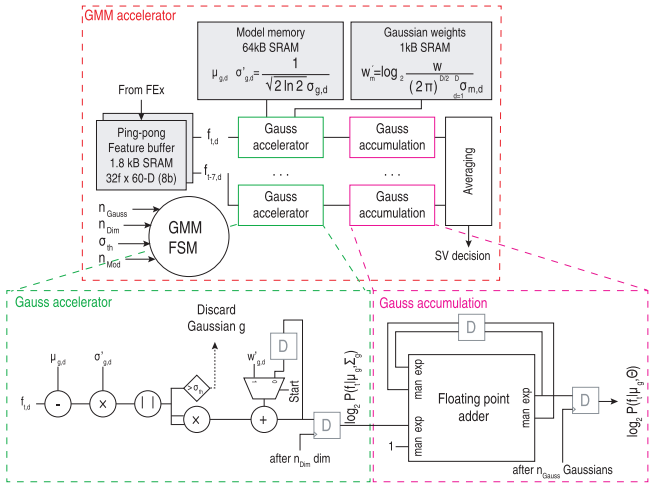

Fig. 15. GMM microarchitecture (top) GMM high-level organization (bottom) Gaussian accelerator and accumulation.

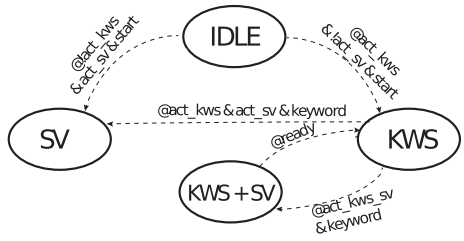

Fig. 16. Configurable master control FSM.

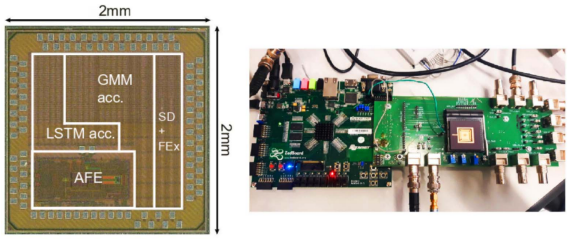

Fig. 17. Chip photograph and measurement setup.

Eight feature vectors are computed concurrently, reducing the memory accesses of the model parameters by 8× due to data reuse. Since standard SV needs at least segments of 500 ms or more to compute a reliable output, the latency constraint due to batching does not affect the overall system performance. Furthermore, all the model weights are saved on-chip in a 65-kB SRAM for a maximum of 512 60-dim Gaussians, avoiding the need for external memory accesses.

Fig. 15 (bottom) shows the inner accelerator that computes the log-probability of a Gaussian for a specific feature vector. At the beginning of a Gaussian evaluation, the first dimension of the feature vector is retrieved from memory with the precomputed Gaussian parameters. The accelerator performs $D-1$ iterations by sequentially loading from memory the dimension d of the current feature vector, and the necessary Gaussian parameters. The model parameters are reused in all eight Gaussian accelerators. In every iteration, each accelerator evaluates the distance of the current feature dimension to the Gaussian-mean dimension $((f_{m,d}-\mu_{m,d})/((2\ln 2)^{1/2}\sigma_{m,d}))$. In case the distance is greater than the programmable threshold Dist_{th}, the remaining computations of this accelerator are aborted, saving dynamic power. Moreover, when all eight parallel vectors discard their Gaussian, the computation jumps to the next Gaussian, also reducing the number of memory accesses. This leads to 20% savings in a number of accesses, while maintaining the baseline accuracy.

When all accelerators finish their Gaussian, the log-probabilities are saved in the log Prob registers. This result is then used as the exponent of a floating-point value with mantissa 1, which is accumulated with the contribution of all the previous completed Gaussians. Since the floating-point adder is used only once every D cycles, the unit is input and clock gated. When all Gaussians are evaluated, the probability of the GMM for feature vector is available at the output of the adder, and the output can be thresholded for the SV decision.

G. Control Unit

A programmable control unit allows activating the different execution modules depending on the desired hierarchy of tasks (Fig. 16). The system starts in *IDLE* state where only the AFE and SD are active. After the detection of sound, a signal *start* is asserted, making the FSM to transition to the states *KWS* or *SV* depending on the configuration registers *act-kws* and *act-sv*. If both registers are asserted, the system will execute first KWS and, after a command is detected (*keyword* = 1), SV. In addition, the register *act-kws-sv* allows running first KWS and, when a command is spotted, both accelerators concurrently. When SV inference is finished, the signal *ready* is set to 1, allowing the FSM to transition to the KWS state again for the next KWS.

V. Chip Implementation and Evaluation

The mixed-signal SoC Vocell was implemented in TSMC 65-nm CMOS, with a die shot shown in Fig. 17, measuring 2 mm × 2 mm with a core size of 1.6 mm × 1.6 mm. The IC possesses two separate power domains, corresponding to the logic and SRAM power supplies. One global clock, and its clock tree distribution, is used throughout the whole chip. In order to reduce the power consumption of idle hardware, clock gating is applied to each one of the accelerators when such modules are not active within the hierarchy.

A. Test Procedures

The measurement setup, shown in Fig. 17, connects the chip to the external digital and analog supplies and an FPGA ZedBoard through SPI and dedicated digital pins. The board feeds digital inputs to the IC and reads outputs and intermediate results generated by the SoC.

B. Full System Performance

Fig. 18 shows the system operating at logic voltages from 0.6 V (250 kHz) to 0.8 V (8 MHz) at SRAM voltage of 0.8 V (Fig. 18). First, the average power, which corresponds to the mean power for real-time operation, is measured in the different system execution states. To run the 16-keywords task and a 256 Gaussian/model SV task in real time, a clock frequency of 250 kHz is selected, with input audio sampled at

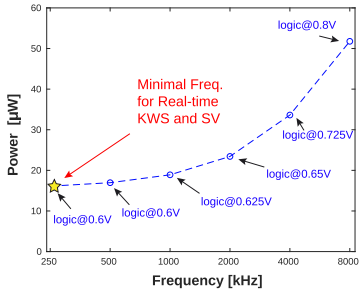

Fig. 18. Voltage-frequency scaling for full system active. SRAM voltage fixed at 0.8 V.

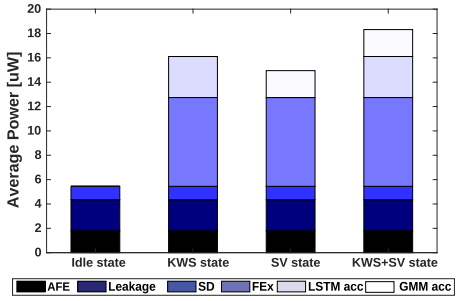

Fig. 19. Power breakdown for continuous operation in each state of the control unit FSM.

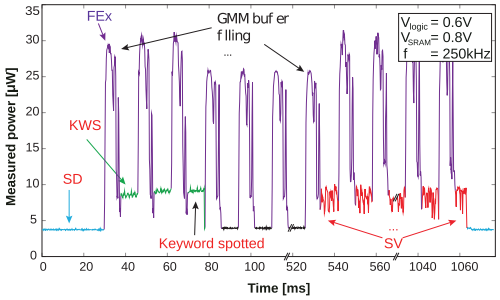

Fig. 20. Instantaneous system power trace for KWS triggering SV.

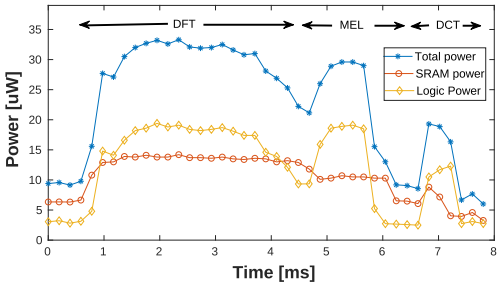

Fig. 21. FEx instantaneous power trace, highlighting the main computational stages: DFT calculation, Mel filtering, and DCT.

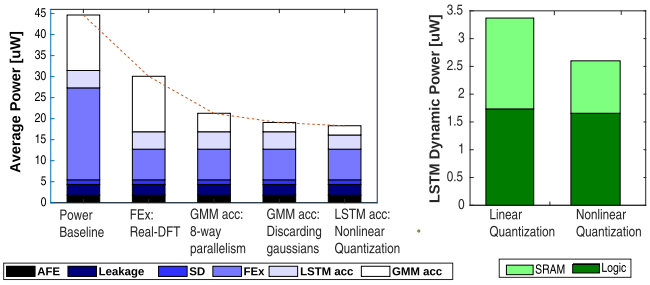

Fig. 22. (Left) Average power consumption for KWS+SV, showing the impact of the introduced optimizations. Each point corresponds to the average power after incrementally applying the corresponding technique. (Right) Dynamic power consumption of LSTM accelerator for LQ (8 bits) and NLQ (4 bits) for an LSTM network with 64 neurons.

16 kHz, and logic voltage set at 0.6 V. As shown in the power breakdown shown in Fig. 19, the power consumption strongly depends on the current execution state, as coarse clock gating freezes idle accelerators. Just 5.45 μW is consumed in idle mode mainly by leakage and the continuous operation of the AFE and SD. If only KWS or SV mode is active, 16.11 μW and resp. 14.95 μW are consumed. The FEx is the main source of energy dissipation for those states accounting for almost 50%. The most-processing use case with concurrent KWS and SV consumes 18.4 μW, which corresponds to the worst case power consumption, as all accelerators (SD, FE, GMM acc, and LSTM acc) are active at the same time.

Fig. 20 shows an example of a power trace when the full hierarchy SD-FEx-KWS-SV is active, with KWS triggering SV. The referred graph shows the instantaneous system power in function of time. As observed, after a sound is detected the FEx and the LSTM accelerator are activated. Likewise, when a keyword is spotted, the FEx buffer for the GMM is filled 500 ms and consequently used for GMM inference. The inference is finished after additional 500 ms when operating at the real-time clock frequency of 250 kHz (shown here), or after a shorter period of time using a higher frequency (e.g., 15 ms at 8 MHz.). The spiky power trace observed during SV stems from the preemptive abortion of Gaussians when their distance is higher than the preset threshold. FEx is responsible for a significant amount of instantaneous power consumption in the form of localized pulses. A zoom-in measured power trace of the FEx is shown in Fig. 21, where the stages DFT, Mel filtering, and DCT are indicated.

In order to evaluate the system impact of the introduced algorithmic and microarchitecture optimization techniques applied in this SoC, Fig. 22, left, shows the expected full-system power consumption (running KWS and SV concurrently) after the incremental addition of each one of

TABLE I

REAL-TIME POWER CONSUMPTION USING HIERARCHICAL EXECUTION FOR DIFFERENT USER SCENARIOS

Scenario	1 stage	2 stages	3 stages
always-on sensor	18.3μW	6.73μW	6.5μW
voice-assistant	18.3μW	11.88μW	10.6μW
push-to-talk	18.3μW	14μW	12.17μW

such design improvements. Specifically, the graph shows the baseline system power consumption without any optimization and after the step-by-step application of: 1) real-DFT for FEx; 2) eight-way parallelism for GMM acc; 3) discarding of Gaussians for GMM acc; and 4) NLQ for LSTM acc. As observed, since FEx is the most consuming module, the use of a real DFT (which reduces computation and memory accesses by 2$\times$ for FEx) has a big impact on the total power consumption. Though the system power impact of NLQ is relatively small, its use allows decreasing the dynamic power of the LSTM accelerator by 20% thanks to 50% less SRAM memory accesses, as shown in Fig. 22 (right).

TABLE II
STATE-OF-THE-ART COMPARISON

Reference	ISSCC'2017 [1]	VLSI'2018 [6]	ISSCC'2017 [24]	TVLSI'2014 [25]	This work
Technology	65nm	28nm	40nm	90nm	65nm
Area	13.17mm^2	1.29mm^2	7.1mm^2	19.53mm^2	2.56mm^2
Voltage	0.6V-1.2V	0.57V-0.9V	0.62V-0.9V	0.9V	0.6V-1.2V
AFE	NA	NA	NA	NA	✓
SD	✓	✓	NA	NA	✓
FE	✓	✓	✓	✓	✓
KWS	✓	✓	✓	NA	✓
SV	NA	NA	NA	✓	✓
Latency	Real-time	0.5ms-25ms	7ms	8ms(per feature vector)	KWS: 0.5-16ms(250kHz-8MHz) SV:515ms-1s(250kHz-8MHz)
Accelerators	VAD, MFCC DNN+HMM	VAD, MFCC BNN	FFT, DNN	LPC, SVM	VAD, MFCC LSTM, GMM
Stages	2 stages	2 stages	1 stage	1 stage	3 stages
KWS Accuracy	98.35% (TIDIGITS) 96.88%(WSJ)	96.11% (TIDIGITS)	NA	Not Reported	98.50% (TIDIGITS) 90.87%(GSCD)
SV Accuracy	NA	NA	NA	92.49% (NIST SRE)	99.5%(TIMIT)
Real-time Power	172μW@3MHz	141μW@2.5MHz	288μW@3.9MHz	8.12mW@100MHz	10.6μW@250kHz

C. Accuracy/Power Trade-Off for KWS and SV

To verify and optimize the tradeoff between model complexity and power consumption for the hardware platform further, the impact of model size was assessed for the KWS and SV tasks using actual system power measurements. As shown in Fig. 6 (left), there is a clear tradeoff between accuracy and system power consumption, which can be tuned easily to the user's need. Likewise, for SV [Fig. 6 (right)], if the model complexity of the GMM is increased, by using a higher number of feature dimensions, the total accuracy $(1 - \mathrm{EER})$ improves. On the other hand, measured power consumption increases linearly with the number of dimensions processed.

D. Hierarchical Execution

To assess the impact of the hierarchical detection execution using varying number of stages on the average power consumption, three scenarios with unbalanced input data statistics were analyzed: *always*-ON-*sensor*, *voice-assistant* and *push-to-talk*. In the first case, an *always*-ON-*sensor* device must always be active, which means long background noise intervals (SD 90%, KWS 9%, and SV 1% on-time). In the *voice-assistant* example, the quantity of speech segments is increased due to higher activation by the user (SD 50%, KWS 40%, and SV 10%). Finally, in the last *push-to-talk* scenario the user only activates the system after specific events, such as pressing a button, making the presence of speech more common (SD 33%, KWS 33%, and SV 33%). In Table I, the real-time power consumption of each one of the scenarios is presented for one stage (full system always ON), two stages (SD triggering joint KWS and SV activation), and three stages (SD activating KWS, and KWS triggering SV). Due to the varying input data statistics, the power consumption can be modulated between 6.5 and 18.3 μW (worst case) depending on the surrounding acoustic environment. In particular, for example, the *voice-assistant* use case, using three stages, it is possible to achieve 10.6 μW or about 45% power savings due to the selective activation of the different digital accelerators.

As noted, the use of hierarchical execution in conjunction with module-level clock gating allows effective power consumption scale. Prospectively, orthogonal power management techniques, such as power gating might be applied in order to reduce leakage. The use of a flash memory storage, where the system configuration and the ML parameters are saved, would allow to power gate each accelerator, including the internal computation memories. Due to the expensive writing from non-volatiles memories and the time-consuming wake-up processes after power gating, the selectivity of each stage needs to be carefully tuned.

E. Comparison With SoA

Table II compares this article with previous SoA voice command recognition ASICs. While providing high accuracy in standard data sets, the present SoC reduces power consumption by 10$\times$ in relation with previous accelerators [1], [6], [24]. Few previous ASICs for SV are present in the literature; [25] presents a design in 90 nm using SVMs and LPC as frontend extraction. However, only simulation results are presented. Vocell is the first ASIC that adds SV to the standard voice command recognition system, improving the quality of the service at reduced power consumption.

VI. CONCLUSION

This article presents Vocell, a speech-triggered wake-up SoC for SV and KWS, attaining high accuracy for the mentioned tasks while consuming only 18.3 μW for worst case scenario. The use of a single-chip solution, memory and power-optimized accelerators, and hardware-aware algorithmic tuning, allows reducing significantly the energy required for real-time operation.

ACKNOWLEDGMENT

The authors would like to thank the MIT team of A. Chandrakasan and M. Price for sharing their DFT design [1].

REFERENCES

[1] M. Price, J. Glass, and A. P. Chandrakasan, "14.4 A scalable speech recognizer with deep-neural-network acoustic models and voice-activated power gating," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2017, pp. 244–245.

[2] N. D. Lane and P. Georgiev, "Can deep learning revolutionize mobile sensing?" in *Proc. 16th Int. Workshop Mobile Comput. Syst. Appl. (HotMobile)*, 2015, pp. 117–122.
[3] Z. Chen, Y. Qian, and K. Yu, "Sequence discriminative training for deep learning based acoustic keyword spotting," *Speech Commun.*, vol. 102, pp. 100–111, Sep. 2018.
[4] Y. Su, F. Jelinek, and S. Khudanpur, "Large-scale random forest language models for speech recognition," in *Proc. 8th Annu. Conf. Int. Speech Commun. Assoc.*, 2007, pp. 1–4.
[5] R. D. Kumar, A. B. Ganesh, and S. Kala, "Speaker identification system using Gaussian mixture model and support vector machines (GMM-SVM) under noisy conditions," *Indian J. Sci. Technol.*, vol. 9, p. 93870, May 2016.
[6] S. Yin *et al.*, "A 141 UW, 2.46 PJ/neuron binarized convolutional neural network based self-learning speech recognition processor in 28 nm CMOS," in *Proc. IEEE Symp. VLSI Circuits*, Jun. 2018, pp. 139–140.
[7] K. Badami, K. D. Murthy, P. Harpe, and M. Verhelst, "A 0.6 V 54 dB SNR analog frontend with 0.18% THD for low power sensory applications in 65 nm CMOS," in *Proc. IEEE Symp. VLSI Circuits*, Jun. 2018, pp. 241–242.
[8] M. Vacher, D. Istrate, J.-F. Serignat, and N. Gac, "Detection and speech/sound segmentation in a smart room environment," in *Proc. IEEE Trends Speech Technol., 3rd Int. Conf. Speech Technol. Hum.-Comput. Dialogue*, Cluj-Napoca, Romania, 2005. [Online]. Available: http://hal.archives-ouvertes.fr/hal-01092602
[9] K. M. H. Badami, S. Lauwereins, W. Meert, and M. Verhelst, "A 90 nm CMOS, 6 μW power-proportional acoustic sensing frontend for voice activity detection," *IEEE J. Solid-State Circuits*, vol. 51, no. 1, pp. 291–302, Jan. 2015, doi: 10.1109/JSSC.2015.2487276.
[10] A. Dufaux, L. Besacier, M. Ansorge, and F. Pellandini, "Automatic sound detection and recognition for noisy environment," in *Proc. 10th Eur. Signal Process. Conf.*, Sep. 2000, pp. 1–4.
[11] M. Shahnawaz, E. Plebani, I. Guaneri, D. Pau, and M. Marcon, "Studying the effects of feature extraction settings on the accuracy and memory requirements of neural networks for keyword spotting," in *Proc. IEEE 8th Int. Conf. Consum. Electron.-Berlin (ICCE-Berlin)*, Sep. 2018, pp. 1–6.
[12] M. Yuan, T. Lee, P. Ching, and Y. Zhu, "Speech recognition on DSP: Issues on computational efficiency and performance analysis," *Microprocessors Microsyst.*, vol. 30, no. 3, pp. 155–164, May 2006.
[13] M. Sun *et al.*, "Max-pooling loss training of long short-term memory networks for small-footprint keyword spotting," in *Proc. IEEE Spoken Lang. Technol. Workshop (SLT)*, Dec. 2016, pp. 474–480.
[14] P. Baljekar, J. F. Lehman, and R. Singh, "Online word-spotting in continuous speech with recurrent neural networks," in *Proc. IEEE Spoken Lang. Technol. Workshop (SLT)*, Dec. 2014, pp. 536–541.
[15] D. A. Reynolds, "An overview of automatic speaker recognition technology," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, vol. 4, May 2002, p. IV-4072.
[16] N. Dehak *et al.*, "Support vector machines and Joint Factor Analysis for speaker verification," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, Apr. 2009, pp. 4237–4240.
[17] N. Dehak, P. J. Kenny, R. Dehak, P. Dumouchel, and P. Ouellet, "Front-end factor analysis for speaker verification," *IEEE Trans. Audio, Speech, Language Process.*, vol. 19, no. 4, pp. 788–798, May 2011.
[18] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer, 2006.
[19] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 6869–6898, 2017.
[20] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*. [Online]. Available: https://arxiv.org/abs/1510.00149
[21] J. Fernandez-Marques, W. T. Vincent, S. Bhattachara, and N. D. Lane "BinaryCmd: Keyword spotting with deterministic binary basis," in *Proc. Conf. Syst. Mach. Learn. (SysML)*, Stanford, CA, USA, 2018.
[22] P. Harpe, H. Gao, R. Van Dommele, E. Cantatore, and A. Van Roermund, "21.2 A 3nW signal-acquisition IC integrating an amplifier with 2.1 NEF and a 1.5 fJ/conv-step ADC," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2015, pp. 1–3.
[23] J. Giraldo and M. Verhelst, "Laika: A 5uW programmable LSTM accelerator for always-on keyword spotting in 65 nm CMOS," in *Proc. IEEE 44th Eur. Solid State Circuits Conf. (ESSCIRC)*, Sep. 2018, pp. 166–169.
[24] S. Bang *et al.*, "14.7 A 288 μW programmable deep-learning processor with 270 KB on-chip weight storage using non-uniform memory hierarchy for mobile intelligence," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2017, pp. 250–251.
[25] J.-C. Wang, L.-X. Lian, Y.-Y. Lin, and J.-H. Zhao, "VLSI design for SVM-based speaker verification system," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 7, pp. 1355–1359, Jul. 2015.

Juan Sebastian P. Giraldo was born in Bogota, Colombia, in 1989. He received the B.S. degree in electrical engineering from the Universidad de los Andes, Bogota, in 2012, and the M.S. degree in microelectronics from Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil, in 2016. He is currently pursuing the Ph.D. degree in digital acceleration of deep learning algorithms for voice command recognition with the Department of Electrical Engineering-Microelectronics and Sensors (ESAT-MICAS) Laboratories, KU Leuven, Leuven, Belgium.

In 2016, he joined as a Research Assistant with ESAT-MICAS. His current research interests include machine learning, speech recognition, and low-power digital design.

Steven Lauwereins (Student Member, IEEE) was born in Leuven, Belgium, in 1990. He received the B.S. and M.S. degrees in electrical engineering from Katholieke Universiteit Leuven (KU Leuven), Leuven, in 2011 and 2013, respectively.

In 2013, he joined the Department of Electrical Engineering-Microelectronics and Sensors (ESAT-MICAS) Laboratories, KU Leuven, as a Research Assistant with a focus on implementing machine learning methods at the circuit level to optimize power consumption in sensor interfaces. He is currently a Research Lead with Televic Rail, Izegem, Belgium. He has authored over 15 conferences and journal publications.

Komail Badami (Student Member, IEEE) received the Ph.D. degree from Microelectronics and Sensors (MICAS) Laboratories, Katholieke Universiteit Leuven, Belgium, in 2019.

He is currently a Research and Development Engineer with CSEM Zurich, Zürich, Switzerland, where he is involved in analog mixed-signal designs for power-efficient information extraction from sensory systems.

Dr. Badami was a recipient of the EXPERTS III Ph.D. Grant in 2013 and the Solid-State Circuits Society Pre-Doctoral Award in 2018. He serves as a Reviewer for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS: EXPRESS BRIEFS.

Marian Verhelst (Senior Member, IEEE) received the Ph.D. degree from Katholieke Universiteit Leuven (KU Leuven), Leuven, Belgium, in 2008.

She was a Visiting Scholar with Berkeley Wireless Research Center, University of California at Berkeley, Berkeley, CA, USA, in 2005. She was a Research Scientist with Intel Laboratories, Hillsboro, OR, USA, from 2008 to 2011. She is currently an Associate Professor with Microelectronics and Sensors (MICAS) Laboratories, Department of Electrical Engineering, KU Leuven. Her research focuses on embedded machine learning, hardware accelerators, self-adaptive circuits and systems, sensor fusion, and low-power edge processing.

Dr. Marian was a member of the Young Academy of Belgium and the STEM advisory committee to the Flemish Government. She is a member of the DATE and ISSCC executive committees and a TPC member DATE and ESSCIRC. She was the Laureate of the Royal Academy of Belgium in 2016. She is the Solid-State Circuits Society (JSSC) Distinguished Lecturer. She currently holds a prestigious European Research Council (ERC) Starting Grant from the European Union. She is the TPC Co-Chair of AICAS2020 and tinyML2020. She was an Associate Editor of Transactions on VLSI Systems (TVLSI), Transactions on Circuits and Systems (TCAS)-II, and the JOURNAL OF SOLID-STATE CIRCUITS.