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

J.S.P Giraldo, Steven Lauwereins, Komail Badami and Marian Verhelst
ESAT-MICAS - KU Leuven, Kasteelpark Arenberg 10, B-3001 Leuven, Belgium

***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 be always active, power consumption is one of their primary design metrics. This work 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 a) 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 18.3μW peak power consumption or 10.6μW average power for typical real-time scenarios, 10x below state-of-the-art.**

I. Introduction

Automatic Speech Recognition (ASR) has become widely present in many mobile devices such as wearables or cell-phones. 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 [1]. However, the energy constraints associated with embedded systems are in conflict with the use of advanced SotA ML algorithms such as computationally complex neural networks [2], random forests [3] or support vector machines [4].

Recent hardware developments such as [5] 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), as well as ML accelerators to detect speech commands (Fig. 1-*top*). This kind of approach attains very good accuracy on standard KWS datasets at a power consumption of about hundreds of microwatts. However, these solutions present some drawbacks: First, they lack efficient always-on implementations, consuming well below 100μW, required to maintain sufficient battery lifetime in energy constrained devices. Secondly, 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, SotA implementations require off-chip analog circuitry and storage

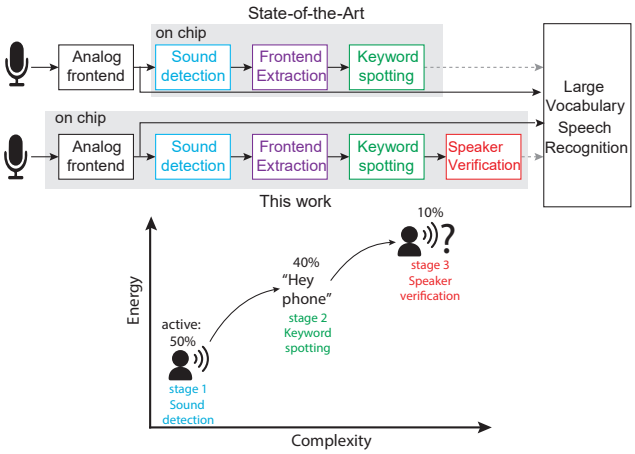

Fig. 1. *Top)* Staged Functionality for a speech-triggered wake-up system *Bottom)* Hierarchical system functionality of the presented wake-up engine.

to operate, since they lack an on-chip Analog Frontend (AFE) and sufficient on-chip memory, diminishing their autonomy.

In this work, we propose Vocell, an integrated mixed-signal SoC for Keyword Spotting and Speaker Verification 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 sound detection). More complex stages for KWS and SV are gradually activated in a datadriven way, significantly reducing the average energy cost per inference.

To this end, this paper present the following contributions:

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

II. The speech hierarchy

A wake-up speech processing system for voice command recognition is composed of a pipeline of stages as described in figure 1. In the following subsections each one of the tasks is discussed along with the algorithms/techniques chosen for the Vocell system.

A. Analog Frontend

The analog frontend (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 0.6V suffer from the following shortcomings: (a) limited distortion performance due to inoperable cascode structures thus limiting the open loop gain in the frontend amplifier (b) 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]. The amplifier output is then digitized with a 10b 8X oversampling SAR ADC after which the digitized signal is decimated in the the digital back end to compute the necessary features.

B. Sound Detection

Due to the presence of long silence intervals for many real-time applications, the use of SD allows to avoid 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 SotA, including energy calculation [8], spectral analysis [9], and ML models [10]. In this work, 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 reducing the input dimensionality for the ML classifiers. Mel Frecuency 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 16ms, a 32ms 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 a) an FFT for spectral analysis b) Mel-filtering using triangular filters c) Logarithmic compression and d) a 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

Recently, deep learning models have increasingly been applied to the field of keyword spotting, replacing the classical two-stage GMM/HMM by a single-stage feedforward neural network or RNN [13]. Due to the temporal nature of speech processing, the use of RNNs, and specifically LSTMs, have shown very good results in terms of accuracy and computational efficiency [14]. RNN's compact model size and reduced amount of computations per input feature vector bring a favorable balance of achievable accuracy and computational complexity. LSTMs are recurrent neural networks that consist of 4 kernels, also called 'gates', each operating over a hidden state vector h_{t-1} and the current input feature vector x_t at the step t [15]:

$$f_t = \sigma_s(W_f x_t + U_f h_{t-1} + b_f) \tag{1}$$

$$i_t = \sigma_s(W_i x_t + U_i h_{t-1} + b_i) \tag{2}$$

$$o_t = \sigma_s(W_o x_t + U_o h_{t-1} + b_o) \tag{3}$$

$$g_t = \sigma_h(W_g x_t + U_g h_{t-1} + b_g) \tag{4}$$

$$c_t = f_t \circ c_{t-1} + i_t \circ g_t \tag{5}$$

$$h_t = o_t \circ \sigma_h(c_t) \tag{6}$$

With W_*, U_* and b_* being trained parameters for each gate, and σ_* nonlinear activation functions sigmoid σ_s and tanh σ_h. To compute the next memory vector c_t and hidden vector h_t, the result of the gates is used by element-wise multiplications marked by $\circ$ in (5)-(6). 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 [16] has shown it is possible to obtain accurate and compact models using small-scale LSTMs with 1 or 2 LSTM layers, each having less that 64 neurons.

E. Speaker Verification

Several algorithms have been proposed for speaker verification such as GMM [17], SVM [18] and i-vector [19] approaches. Between them, GMMs show the lowest computational load while maintaining high accuracy for typical recognition contexts. A GMM (Gaussian Mixture Model) is a probabilistic model generated by a sum of multivariate Gaussians, which allows to compute 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.

$$\mathrm{P(f} \mid \mathbf{\Theta}) = \sum_{m=1}^{n_{gauss}} \left[\frac{w_m}{(2\pi)^{D/2} \prod_{d=1}^{D} \sigma_{m,d}} \cdot \exp\left[-\sum_{d=1}^{D} \frac{(f_d - \mu_{m,d})^2}{2\sigma_{m,d}^2} \right] \right] \tag{7}$$

With f_d being the d'th value of feature vector $\mathbf{f}$ while $\mu_{m,d}$ and $\sigma_{m,d}$ are the mean and standard deviation of the d'the dimension of the m'th Gaussian. Eq. (7) shows the simplified equation for Gaussians with diagonal covariance

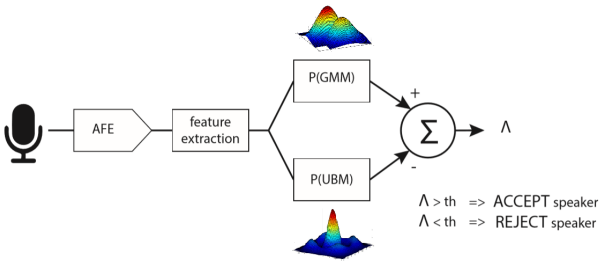

Fig. 2. Inference using the GMM-UBM speaker verification. If $\Lambda > th$, with th a configurable uncertainty threshold, target speaker is detected. Otherwise the hypothesis is rejected.

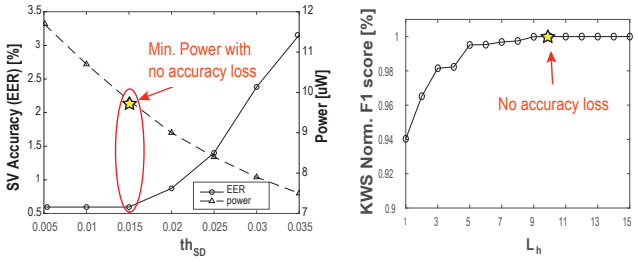

Fig. 3. *Left*) Impact of SD threshold th_{SD} on SV accuracy and power consumption; *Right*) Impact of hangover length L_h over KWS accuracy.

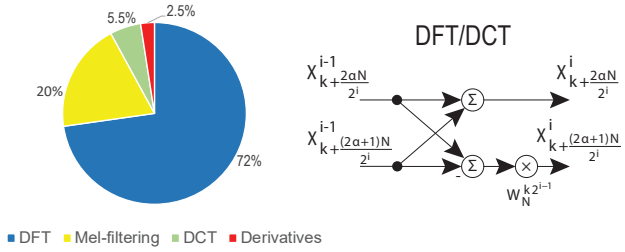

Fig. 4. *Left)* MFCC Computation Breakdown *Right)* DFT Butterfly stage

matrices. Classical optimization techniques such as Expectation Maximization (EM) are used for training the GMM model parameters [20]. For SV, a universal background model (UBM) is trained with several speakers, which allows to model the average user. A total of $n_{gauss} = 256$ to 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 Aposteriori (MAP) 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 2 GMMs is used to decide if the current speech segment corresponds to the target speaker (Fig. 2):

$$\Lambda(\mathbf{F}) = \ln\left(P(\mathbf{f} \mid \mathbf{\Theta_{speaker}})\right) - \ln\left(P(\mathbf{f} \mid \mathbf{\Theta_{UBM}})\right) \quad (8)$$

Where $\Theta_{speaker}$ 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, as well as the consequent floating point sums, generate serious challenges for the efficient deployment of GMMs on embedded platforms. Using a naive approach with $n_{gauss} = 256$ Gaussians, $D = 60$ dimensions per feature vector, and 8b precision, inference would demand about 8MOPS and 3.8MB/s of memory bandwidth, making execution infeasible on many low power devices. In this work, several optimizations are carried out in the algorithmic and micro-architecture domain so as to improve data reuse and hardware economy, as discussed in Section III.

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 as well as parameter tuning. To maintain high task accuracy, the impact of each design decision is analyzed on standard speech datasets.

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:

$$E(\text{window}) = \sum_{i=1}^{n_w} |x_i| \quad (9)$$

If the result is higher than a configurable threshold th_{SD}, a sound detection signal is asserted. The hangover length L_h, which refers to the configurable amount of frames classified as sound after sound is detected, must be optimized in order to not 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 per window (Figure 4-*left*). Yet, the DCT transformation is based on similar compute kernels as the DFT transformation. This allows to reuse optimized hardware resources for the DCT/DFT implementation, saving area and consequently leakage energy.

1) real-point DFT: For this work, an N-point complex DFT is mapped on a single standard radix-2 fixed point DFT block, also called a butterfly stage (Fig. 4-*right*). The final result is obtained after $log2(N)$ steps, calculating 2 intermediate results per iteration i =1..$log2(N)$, with twiddle factor:

$$W_N^{k2^{i-1}} = \exp^{-j\frac{2\pi 2^{i-1}k}{N}} \quad (10)$$

With $k \in [0; N/2^i[$ and $\alpha \in [0; 2^(i-1)]$.

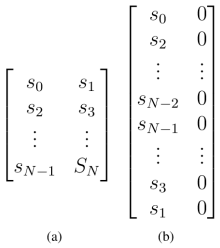

Fig. 5. Input sample ordering for (a) a real-point DFT and (b) a DCT.

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. 5a). 2) A final correction step, equation 11, 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.

$$\chi_k^i = \frac{1}{2}\left[\left(\chi_k^{\log_2 \frac{N}{2}} + \chi_{\frac{N}{2}-k}^{*\log_2 \frac{N}{2}}\right) - j\left(\chi_k^{\log_2 \frac{N}{2}} - \chi_{\frac{N}{2}-k}^{*\log_2 \frac{N}{2}}\right) W_N^{k2^{i-1}}\right] \quad (11)$$

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 is 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 figure 5b. In the correction step (step 3) the final DCT is extracted from the complex DFT by applying the following transformation to the intermediate result:

$$\begin{aligned} X_k &= \chi_k^{\log_2 N} \exp^{-j\frac{2\pi k}{N}} \frac{2}{\sqrt{2N}} && \forall k \neq 0 \\ X_k &= \chi_k^{\log_2 N} \exp^{-j\frac{2\pi k}{N}} \frac{\sqrt{2}}{\sqrt{2N}} && \forall k = 0 \end{aligned} \quad (12)$$

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 datasets, as shown in table I. 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 performance metric for TIDIGITS and Google Speech Command Dataset (GSCD) since the examples are evenly distributed between classes. On the other hand, for TIMIT, F1 score was used since the amount of positive examples was significantly higher than the negative instances. Four different algorithmic optimizations were carried out, which will be explained in the following subsections.

TABLE I
TASKS FOR KWS BENCHMARKING

Dataset	Target Vocab.	Speakers	Training Size	Metric
TIMIT	4	630	25200 utterances	F1 score
GSCD	12	2500	65000 utterances	Accuracy
TIDIGTS	10	300	25000 utterances	Accuracy

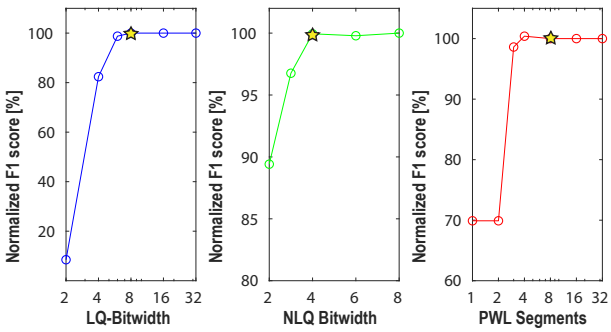

Fig. 6. Accuracy Impact of hardware-aware algorithmic optimizations for a 64-nodes LSTM trained on TIMIT dataset. *left*) Normalized F1 Score in function of Operand Bitwidth *center*) Normalized F1 Score in function of Encoded Bitwidth *right*) Normalized F1 Score in function of Number of Linear Segments for piecewise linear approximation.

1) Fixed Point Analysis: The impact of fixed point neural network computation [21] was evaluated with the objective of finding the minimal bitwidth without performance losses compared to a floating-point implementation. Using quantization post-training, different operand bit-widths were assessed. As observed in Fig. 6-*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 to encode the neural network weights into a reduced wordlength codebook through clustering, reducing the model memory size [22]. Since weight clustering often produces accuracy losses, retraining epochs are necessary. Fig. 6-*center* shows the impact of Nonlinear Quantization (NLQ) for different encoded bitwidths. 16 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-bit to 8-bit words is implemented via Look Up Tables (LUTs), and allows to maintain the baseline accuracy while reducing memory bandwidth by 2x.

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 piecewise linear approximation (PWL) 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. 6-*right*, 8 linear segments show to be sufficient to maintain baseline accuracy. The cut-off points of the linear segments are stored in a look-up table for both tanh and sigmoid, and at run-time linear interpolation between the segment corners allows to efficiently compute the

approximated non-linear functions.

4) Model Size Impact: Models with varying number of LSTM hidden units were trained in order to evaluate the accuracy/complexity trade-off. As will be shown with measurements later (Fig. 21-*left*) accuracy on par with SotA deep learning KWS systems [5][23] can be achieved by 1 LSTM layer with 64 cells, requiring 26kB of model memory (8 bits per weight). To maintain flexibility, the chip is implemented to support any model comprised of LSTM and FC layers, with less than 32k model parameters.

D. GMM

Also for the GMM algorithm, several HW-optimization techniques were assessed against their accuracy implication at task level. To this end, a binary detection problem is defined over the TIMIT dataset, 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 8 audio files in the training set, and 1 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 optimisations: In order to reduce the computational complexity of eq. (7) 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 eq. (7) as:

$$\mathrm{P(f \mid \Theta)} = \sum_{m=1}^{n_{gauss}} \left[\frac{w_m}{(2\pi)^{D/2} \prod_{d=1}^{D} \sigma_d} \cdot 2\hat{} \left(- \sum_{d=1}^{D} \frac{(f_d - \mu_{m,d})^2}{2 \ln 2 \sigma_{m,d}^2} \right) \right] \quad (13)$$

This allows to remove 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:

$$\mathrm{P(f \mid \Theta)} = \sum_{m=1}^{n_{gauss}} \left[2\hat{} \left(\log_2 \left(\frac{w_m}{(2\pi)^{D/2} \prod_{d=1}^{D} \sigma_{m,d}} \right) - \sum_{d=1}^{D} \left(\frac{f_d - \mu_{m,d}}{\sqrt{2 \ln 2} \sigma_{m,d}} 2 \right) \right) \right] \quad (14)$$

Moving most part of the operations to the fixed-point log-domain reduces the need for floating point calculations. Additionally, factors of equation 14 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, $\mu_{m,d}$, $\frac{1}{\sqrt{2 \ln 2} \sigma_{m,d}}$ and $\log_2 \left(\frac{w_m}{(2\pi)^{D/2} \prod_{d=1}^{D} \sigma_{m,d}} \right)$ are stored.

2) Approximate computational optimisations: 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 decreases 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-dimensional 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:

$$Dist_d = \frac{f_d - \mu_d}{\sqrt{2ln(2)\sigma_d^2}} \quad (15)$$

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 (eq. 14), 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. Figure 7 shows the impact of this threshold $Dist_{th}$ on the SV task accuracy, as well as the resulting computational savings. For this work, $Dist_{th} = 4.25$ was used, reducing 50% of the computations without having an impact on SV accuracy.

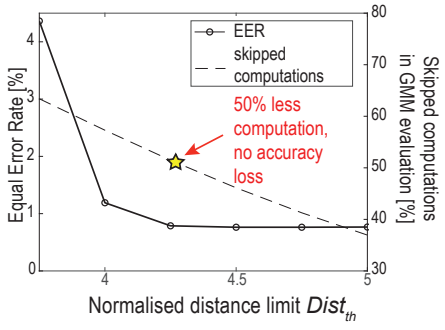

Fig. 7. Effect of distance threshold $Dist_{th}$ on the SV accuracy and total skipped computations for GMM inference.

IV. IC Architecture

A. High-level Architecture

The complete chip architecture for Vocell is depicted in Figure 8. The Analog Frontend receives the incoming analog signal from an external microphone and digitizes the sample thanks to a high linear amplifier and a SAR ADC. The values obtained from the AFE are passed through the FEx Unit which generates Mel Frequency Cepstral Coefficients (MFCCs) and their respective 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 the following subsections.

B. Analog Frontend

The Analog Frontend (AFE) integrated in the SoC is shown in figure 9. The AFE consists of a highly linear time domain amplifier whose output is digitized by a highly power efficient

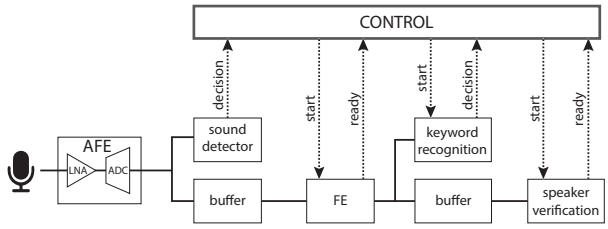

Fig. 8. IC Architecture

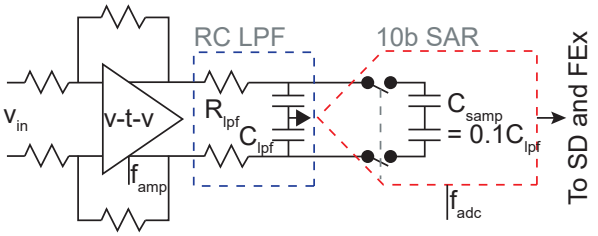

Fig. 9. AFE architecture and specifications.

10b 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 illustrated in the Figure 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 10b 8X oversampling SAR ADC [24]. 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. The targeted AFE specifications are shown in table II.

C. Sound Detector

The micro-architecture for SD is depicted in figure 10. As described in the previous section, 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.

TABLE II
TARGETED SPECIFICATIONS FOR THE 0.6V ANALOG FRONT END (AMPLIFIER + ADC)

Characteristic	Value
Gain	34 dB
Bandwidth	4k Hz
RMS Input referred noise	$\leq 10\mu$V
Distortion at 1kHz and 75% swing	$\leq 0.2\%$

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. Figure 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 [5] 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 10b samples.

2) DFT: A single butterfly stage (Fig. 4), was implemented in digital hardware allowing to compute 2 intermediate results per cycle. 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 compute 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,

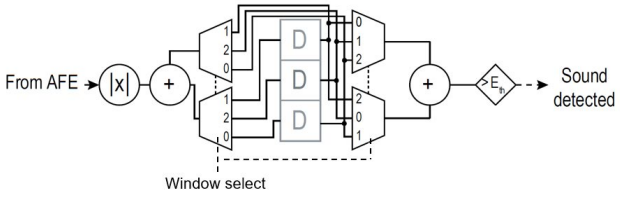

Fig. 10. Sound Detector Architecture

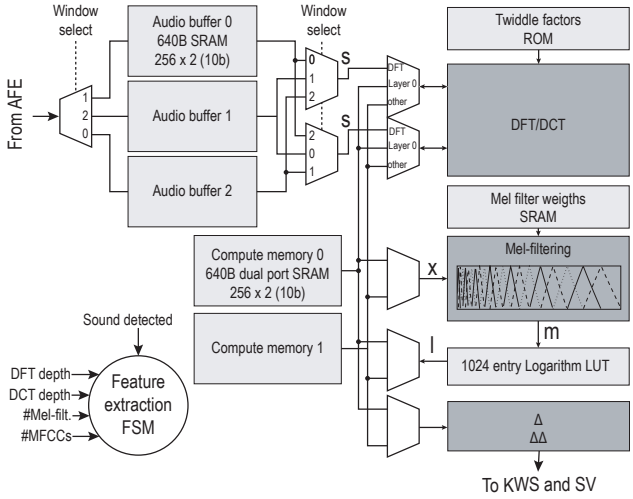

Fig. 11. Feature extraction Architecture

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 are configurable up to a 512-point complex DFT, or a 1024-point real DFT.

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 1kHz and mel-spaced above it (Fig. 12-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 2 filters, a special memory format is proposed (Fig. 12-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 allows to save 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 400x, compared to a weight matrix of 1024x32. 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 look-up table. Using the 10b of the mel filtering output as the address of a 1024-entries LUT, a CORDIC implementation is avoided.

4) DCT + time derivatives: In order to reuse available hardware, the DFT execution engine is exploited to compute the DCT transformation as explained in Section III-B2. To compute the first and second order time derivatives, all intermediate values are stored. The first order time derivative is 9 time samples long, with filter response: [-1 -1 -1 -1 0 1 1 1 1]; and the second order time derivative is 3 time 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. I.e. MFCC value 8 and 9 are combined to form MFCC value 8 for SV. The same combining is performed for the time derivatives.

E. LSTM accelerator

The LSTM accelerator works over every incoming frame produced by the FEx sequentially. 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 32kB SRAM, storing 8 bits linear format or 4 bits nonlinear encoding. In case NLQ is used, a LUT is leveraged to decode the compressed weights, before they are sent to

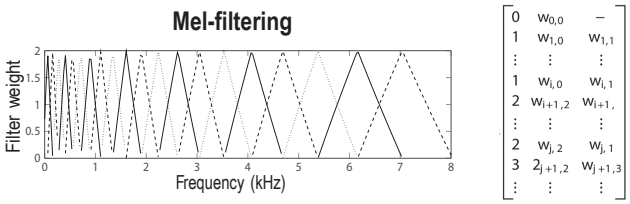

Fig. 12. *left*) Triangular mel filters for frequencies up to 8kHz *right*) Storage format for the mel-filter bank weights

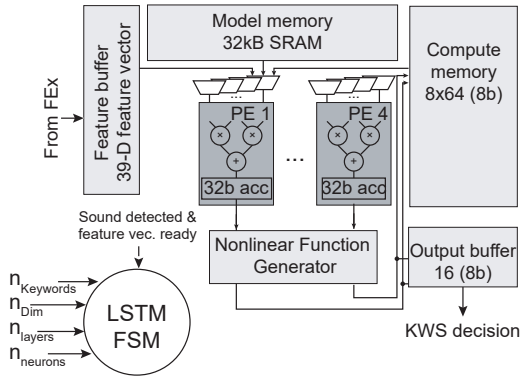

Fig. 13. LSTM Architecture

the processing array. Each one of the LSTM gate kernels (equations (1)-(4)) is mapped to one of the 4 processing elements. The data from the model weights, the input feature vector and the hidden state is 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 8-segment LUT-based function approximation. The same PE array is reused for computing the element-wise multiplications of the LSTM algorithm (equations (5)-(6)). More details of the accelerator can be found in [25].

F. GMM accelerator

The GMM engine (figure 14) allows to compute the log-likelihood ratio between a customized GMM and an UBM to recognize a target speaker (eq. 8). In order to reduce the memory bandwidth, parallelism at the input vector level is applied. 8 feature vectors are computed concurrently reducing the memory accesses by 8x. 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 65kB SRAM for a maximum of 512 60-Dim Gaussians, avoiding the need of external memory accesses.

Figure 14-*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

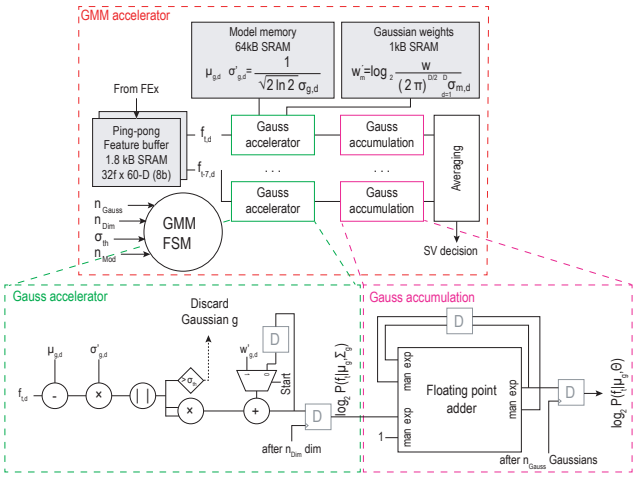

Fig. 14. GMM microarchitecture*Top)* GMM high-level organization*Bottom)* Gaussian Accelerator and Accumulation

$D-1$ 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 8 gaussian accelerators. In every iteration, each accelerator evaluates the distance of the current feature dimension to the gaussian-mean dimension:

$$\frac{f_{m,d}-\mu_{m,d}}{\sqrt{2\ln 2}\sigma_{m,d}} > Dist_{th} \tag{16}$$

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 8 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 number of accesses,while maintaining the baseline accuracy.

When all accelerators finish their gaussian, the log-probabilities are saved in the logProb 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 to activate the different execution modules depending on the desired hierarchy of tasks (Fig. 15). 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 keyword spotting and, after a command is detected (*keyword*=1), speaker verification. Additionally, the

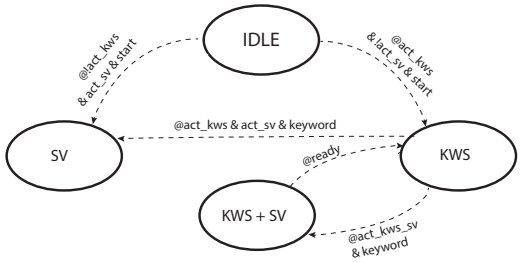

Fig. 15. Configurable master control FSM.

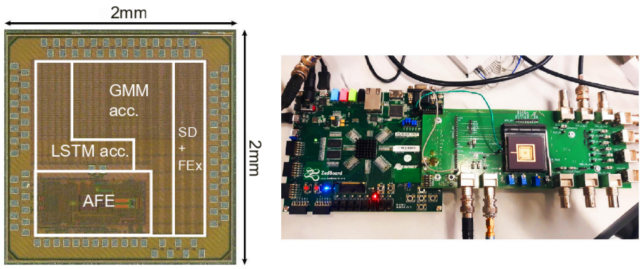

Fig. 16. Chip photo and measurement setup.

register *act-kws-sv* allows to run first keyword spotting and, when a command is spotted, both accelerators concurrently. When speaker verification inference is finished, the signal *ready* is set to 1, allowing the FSM to transition to the KWS state again for the next keyword spotting.

V. Chip Implementation and Evaluation

The mixed-signal SoC Vocell was implemented in TSMC 65 nm CMOS, with a die shot shown in Figure 16, measuring 2mmx2mm with a core size of 1.6mmx1.6mm.

A. Test procedures

The measurement setup, shown in figure 16, connects the chip to the external digital and analog supplies, as well as 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. 17 shows the system operating at logic voltages from 0.6 V (250 kHz) to 0.8 (8 MHz) at a constant SRAM voltage of 0.8 V (Fig 17). First, the system power is measured in the different system execution states. To run the 16-keywords task and a 256 Gaussian/model speaker verification task in real time, a clock frequency of 250kHz is selected, with input audio sampled at 16kHz, and logic voltage set at 0.6 V. As shown in the power breakdown depicted in figure 20, the power consumption strongly depends on the current execution state, as coarse clock gating freezes idle accelerators. Just 5.45μW are consumed in idle mode mainly by leakage and the continuous operation of the AFE and SD. If only KWS or SV mode are active, 16.11μW, resp. 14.95μW are consumed. The FEx is the main source of energy dissipation for those

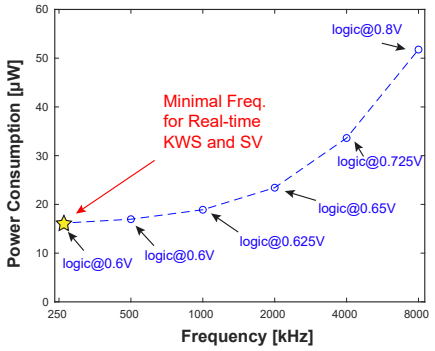

Fig. 17. Voltage-frequency scaling for full system active. SRAM voltage fixed at 0.8V.

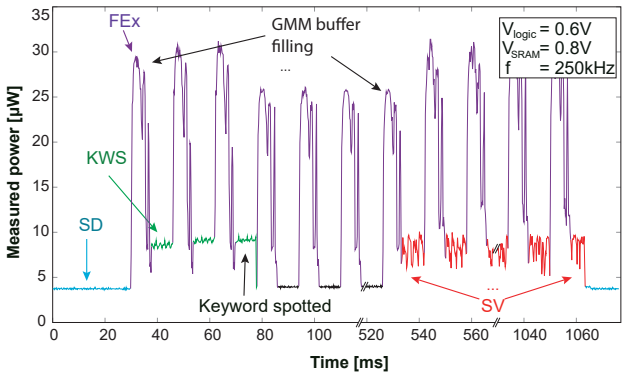

Fig. 18. Full system Power trace.

states accounting for almost 50%. The intensive use case with concurrent KWS and SV consumes 18.4μW, as all accelerators (SD, FE, GMM acc, LSTM acc) are active at the same time.

Figure 18 shows an example of a power trace when the full hierarchy SD-FEx-KWS-SV is active, with KWS triggering SV. 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 and consequently used for GMM inference. The spiky power trace observed during speaker verification stems from the preemptive abortion of gaussians when their distance is higher than the preset threshold. FEx is responsible for a significant amount of power consumption in form of localized pulses. A zoom-in measured power trace of the FEx is shown in figure 19 where the stages DFT, Mel-filtering and DCT are indicated.

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

To verify and further optimize the trade-off between model complexity and power consumption for the hardware platform, the impact of model size was assessed for the KWS and SV tasks using actual system power measurements. As shown in figure 21-*left*, there is a clear trade-off between accuracy and system power consumption, which can be tuned easily to the user's need. The impact of NLQ on the measured dynamic power consumption of the LSTM accelerator is shown in figure 22. The use of NLQ, reduces the memory accesses by almost

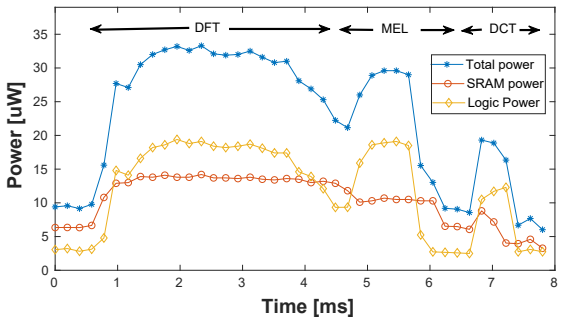

Fig. 19. FEx power trace, highlighting the main computational stages: DFT calculation, Mel-filtering and DCT.

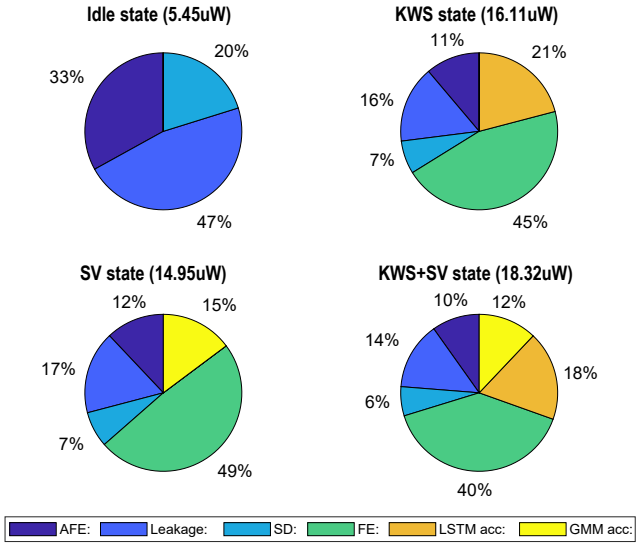

Fig. 20. Power breakdown for continuous operation in each state of the Control Unit FSM

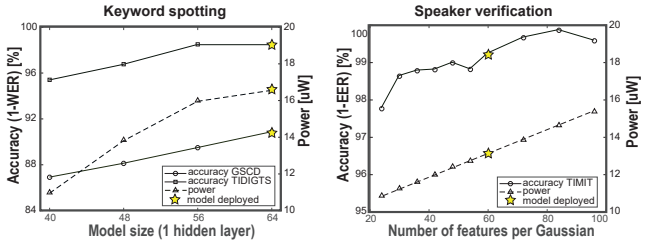

Fig. 21. Accuracy/power trade-off for *Left)* KWS and *Right)* SV.

50% and hence the SRAM dynamic power of the accelerator. Likewise for SV (Fig. 21-*right*), if the model complexity of the GMM is increased, by using a higher number of feature dimensions, the total accuracy $(1 - EER)$ 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 energy effi-

TABLE III
STATE-OF-THE-ART COMPARISON

Reference	ISSCC'2017 [5]	VLSI'2018 [6]	ISSCC'2017 [26]	TVLSI'2014 [27]	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)	16ms (KWS) 0.5s(SV)
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	18.3μW@250kHz

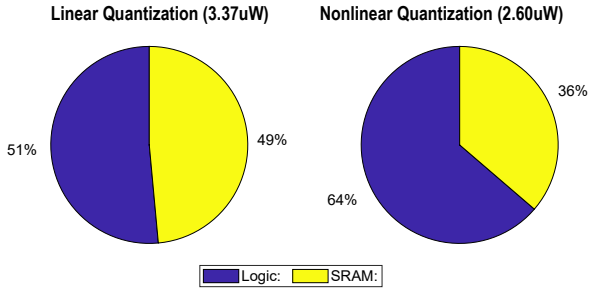

Fig. 22. Dynamic Power consumption of LSTM accelerator for *left)* Linear Quantization (8 bits) and *right)* Nonlinear Quantization (4 bits) for a LSTM network with 64 neurons.

ciency, 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 be always active, which means long background noise intervals (SD 90%, KWS 9%, 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%, SV 10%). Finally, in the last *push-to-talk* scenario the user only activates the system after specific events like pressing a button making the presence of speech more common (SD 33%, KWS 33%, SV 33%). In table IV, the real-time power consumption of each one of the scenarios is presented for 1 stage (Full system always on), 2 stages (SD triggering joint KWS and SV activation) and 3 stages (SD activating KWS, KWS triggering SV). Due to the varying input data statistics, the power consumption can be modulated between 6.5μW and 18.3μW depending on the surrounding environment. In this way, for the *voice-assistant* setting, using 3 stages it is possible to achieve about 45% power savings due to the selective activation of the different digital accelerators.

TABLE IV
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

E. Comparison with SoA

Table V-B compares this work with previous SoA voice command recognition ASICs. While providing high accuracy in standard datasets, the present SoC reduces power consumption by 10x in relation with previous accelerators [5][6][26]. Few previous ASICs for SV are present in the literature; [27] presents a design in 90nm using SVMs and LPC as frontend extraction. However, only simulations results are presented. Vocell is the first ASIC that adds speaker verification to the standard voice command recognition system, improving the quality of the service provided by the platform at a reduced power consumption.

VI. CONCLUSION

This work presents Vocell, a speech-trigered wake-up SoC for Speaker Verification and Keyword Spotting, attaining high accuracy for the mentioned tasks while consuming only 18.3μW for worst-case scenario. The use of a single-chip solution, memory and power optimized accelerators as well as hardware-aware algorithmic tuning, allows to reduce significantly the energy required for real-time operation.

VII. ACKNOWLEDGEMENTS

This work was supported by the Research Foundation - Flanders (FWO) and the EU ERC starting grant RE-SENSE. We thank the MIT team of A. Chandrakasan and M. Price for sharing their DFT design [5].

REFERENCES

[1] N. D. Lane and P. Georgiev, "Can deep learning revolutionize mobile sensing?" in *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*. ACM, 2015, pp. 117–122.

[2] Z. Chen, Y. Qian, and K. Yu, "Sequence discriminative training for deep learning based acoustic keyword spotting," *Speech Communication*, vol. 102, pp. 100–111, 2018.

[3] Y. Su, F. Jelinek, and S. Khudanpur, "Large-scale random forest language models for speech recognition," in *Eighth Annual Conference of the International Speech Communication Association*, 2007.

[4] 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 Journal of Science and Technology*, vol. 9, no. 93870, 2016.

[5] 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 *2017 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2017, pp. 244–245.

[6] S. Yin, P. Ouyang, S. Zheng, D. Song, X. Li, L. Liu, and S. Wei, "A 141 uw, 2.46 pj/neuron binarized convolutional neural network based self-learning speech recognition processor in 28nm cmos," in *2018 IEEE Symposium on VLSI Circuits*. IEEE, 2018, pp. 139–140.

[7] K. Badami, K. D. Murthy, P. Harpe, and M. Verhelst, "A 0.6 v 54db snr analog frontend with 0.18% thd for low power sensory applications in 65nm cmos," in *2018 IEEE Symposium on VLSI Circuits*. IEEE, 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," 2005.

[9] K. M. Badami, S. Lauwereins, W. Meert, and M. Verhelst, "A 90 nm cmos, 6μw power-proportional acoustic sensing frontend for voice activity detection," *IEEE Journal of Solid-State Circuits*, vol. 51, no. 1, pp. 291–302, 2015.

[10] A. Dufaux, L. Besacier, M. Ansorge, and F. Pellandini, "Automatic sound detection and recognition for noisy environment," in *2000 10th European Signal Processing Conference*. IEEE, 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 *2018 IEEE 8th International Conference on Consumer Electronics-Berlin (ICCE-Berlin)*. IEEE, 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 and Microsystems*, vol. 30, no. 3, pp. 155–164, 2006.

[13] G. Chen, C. Parada, and G. Heigold, "Small-footprint keyword spotting using deep neural networks," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2014, pp. 4087–4091.

[14] M. Sun, A. Raju, G. Tucker, S. Panchapagesan, G. Fu, A. Mandal, S. Matsoukas, N. Strom, and S. Vitaladevuni, "Max-pooling loss training of long short-term memory networks for small-footprint keyword spotting," in *2016 IEEE Spoken Language Technology Workshop (SLT)*. IEEE, 2016, pp. 474–480.

[15] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 10, pp. 2222–2232, 2016.

[16] P. Baljekar, J. F. Lehman, and R. Singh, "Online word-spotting in continuous speech with recurrent neural networks," in *2014 IEEE Spoken Language Technology Workshop (SLT)*, Dec 2014, pp. 536–541.

[17] D. A. Reynolds, "An overview of automatic speaker recognition technology," in *2002 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 4. IEEE, 2002, pp. IV–4072.

[18] N. Dehak, P. Kenny, R. Dehak, O. Glembek, P. Dumouchel, L. Burget, V. Hubeika, and F. Castaldo, "Support vector machines and joint factor analysis for speaker verification," in *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2009, pp. 4237–4240.

[19] N. Dehak, P. J. Kenny, R. Dehak, P. Dumouchel, and P. Ouellet, "Front-end factor analysis for speaker verification," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 19, no. 4, pp. 788–798, 2010.

[20] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[21] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[22] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[23] J. Fernández-Marqués, W.-S. T. Vincent, S. Bhattachara, and N. D. Lane, "Binarycmd: Keyword spotting with deterministic binary basis," 2018.

[24] 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 *2015 IEEE International Solid-State Circuits Conference-(ISSCC) Digest of Technical Papers*. IEEE, 2015, pp. 1–3.

[25] J. S. Giraldo and M. Verhelst, "Laika: A 5uw programmable lstm accelerator for always-on keyword spotting in 65nm cmos," in *ESSCIRC 2018-IEEE 44th European Solid State Circuits Conference (ESSCIRC)*. IEEE, 2018, pp. 166–169.

[26] S. Bang, J. Wang, Z. Li, C. Gao, Y. Kim, Q. Dong, Y.-P. Chen, L. Fick, X. Sun, R. Dreslinski *et al.*, "14.7 a 288μw programmable deep-learning processor with 270kb on-chip weight storage using non-uniform memory hierarchy for mobile intelligence," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2017, pp. 250–251.

[27] J.-C. Wang, L.-X. Lian, Y.-Y. Lin, and J.-H. Zhao, "Vlsi design for svm-based speaker verification system," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 7, pp. 1355–1359, 2014.