A 16-nm SoC for Noise-Robust Speech and NLP Edge AI Inference With Bayesian Sound Source Separation and Attention-Based DNNs

Thierry Tambe, *Member, IEEE*, En-Yu Yang, Glenn G. Ko, *Member, IEEE*, Yuji Chai, Coleman Hooper, Marco Donato, *Member, IEEE*, Paul N. Whatmough, *Member, IEEE*, Alexander M. Rush, David Brooks, *Fellow, IEEE*, and Gu-Yeon Wei, *Senior Member, IEEE*

***Abstract*— The proliferation of personal artificial intelligence (AI) -assistant technologies with speech-based conversational AI interfaces is driving the exponential growth in the consumer Internet of Things (IoT) market. As these technologies are being applied to keyword spotting (KWS), automatic speech recognition (ASR), natural language processing (NLP), and text-to-speech (TTS) applications, it is of paramount importance that they provide uncompromising performance for context learning in long sequences, which is a key benefit of the attention mechanism, and that they work seamlessly in polyphonic environments. In this work, we present a 25-mm^2 system-on-chip (SoC) in 16-nm FinFET technology, codenamed SM6, which executes end-to-end speech-enhancing attention-based ASR and NLP workloads. The SoC includes: 1) FlexASR, a highly reconfigurable NLP inference processor optimized for whole-model acceleration of bidirectional attention-based sequence-to-sequence (seq2seq) deep neural networks (DNNs); 2) a Markov random field source separation engine (MSSE), a probabilistic graphical model accelerator for unsupervised inference via Gibbs sampling, used for sound source separation; 3) a dual-core Arm Cortex A53 CPU cluster, which provides on-demand single Instruction/multiple data (SIMD) fast fourier transform (FFT) processing and performs various application logic (e.g., expectation–maximization (EM) algorithm and 8-bit floating-point (FP8) quantization); and 4) an always-ON M0 subsystem for audio detection and power management. Measurement results demonstrate the efficiency ranges of 2.6–7.8 TFLOPs/W and 4.33–17.6 Gsamples/s/W for FlexASR and MSSE, respectively; MSSE denoising performance allowing 6× smaller ASR model to be stored on-chip with negligible accuracy loss; and 2.24-mJ energy consumption while achieving real-time throughput, end-to-end, and per-frame ASR latencies of 18 ms.**

***Index Terms*— Attention mechanism, Gibbs sampling, hardware accelerators, Internet of Things (IoT), natural language processing (NLP), recurrent neural networks (RNNs), sound source separation, speech recognition, system-on-chip (SoC).**

I. Introduction

AI-RELATED workloads are increasingly shifting to the edge, spurred by the unabated growth of raw sensor data [46]. This exploding volume of information has become a dominant driver for mobile and the Internet of Things (IoT) throughout all segments, from consumer to industrial and automotive markets. Intelligence at the edge is also gaining greater interest as it can provide marked advantages over cloud computing in terms of energy efficiency, latency, security/privacy, and autonomy [22], [51].

Conversational artificial intelligence (AI) interfaces using automatic speech recognition (ASR) or keyword spotting (KWS) commands are the main human-to-machine communication channel for a multitude of small form factor IoT devices. Recently published ASR and KWS chips [10], [11], [30], [50] operate in pristine acoustic conditions, raising questions about their performance and viability in a more acoustically adverse environment containing multiple noise sources. Furthermore, the underlying recognition algorithms in several of these works are context-blind deep neural networks (DNNs) implemented on small-vocabulary tasks. However, for long-sequence transduction on large-vocabulary tasks (e.g., >100k words), the attention mechanism [3] provides superior context learning by allowing neural networks to emphasize the most relevant tokens of information when making inference predictions. Enabling bidirectional functionality [6] in sequence-to-sequence (seq2seq) recurrent neural networks (RNNs) further improves inference accuracy of long input sequences (e.g., >10 words). This neural class of attention-based RNN models is commonly known as listen-attend-spell (LAS) models [5] with wide adoption in ASR [15], [16], [29], text-to-speech [36], neural machine translation [47], and text summarization [32] applications. In this work, we describe SM6, a system-on-chip

Manuscript received January 4, 2022; revised March 30, 2022 and May 18, 2022; accepted May 24, 2022. This article was approved by Associate Editor Meng-Fan Chang. This work was supported in part by the Center for Applications Driving Architectures (ADA), one of six centers of JUMP, a Semiconductor Research Corporation (SRC) Program co-sponsored by DARPA; in part by the DARPA DSSoC program; in part by NSF under Award 1704834 and Award 1718160; in part by Intel Corporation; and in part by Arm Inc. *(Corresponding author: Thierry Tambe.)*

Thierry Tambe, En-Yu Yang, Glenn G. Ko, Yuji Chai, Coleman Hooper, David Brooks, and Gu-Yeon Wei are with the School of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138 USA (e-mail: ttambe@g.harvard.edu; enyu_yang@g.harvard.edu; gko@g.harvard.edu; yuc927@g.harvard.edu; chooper@college.harvard.edu; dbrooks@g.harvard.edu; gywei@g.harvard.edu).

Marco Donato is with the Department of Electrical and Computer Engineering, Tufts University, Medford, MA 02155 USA (e-mail: marco.donato@tufts.edu).

Paul N. Whatmough is with Arm Research, Boston, MA 02451 USA, and also with the School of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138 USA (e-mail: pwhatmough@eecs.harvard.edu).

Alexander M. Rush is with the Department of Computer Science, Cornell University, New York, NY 10044 USA (e-mail: arush@cornell.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JSSC.2022.3179303.

Digital Object Identifier 10.1109/JSSC.2022.3179303

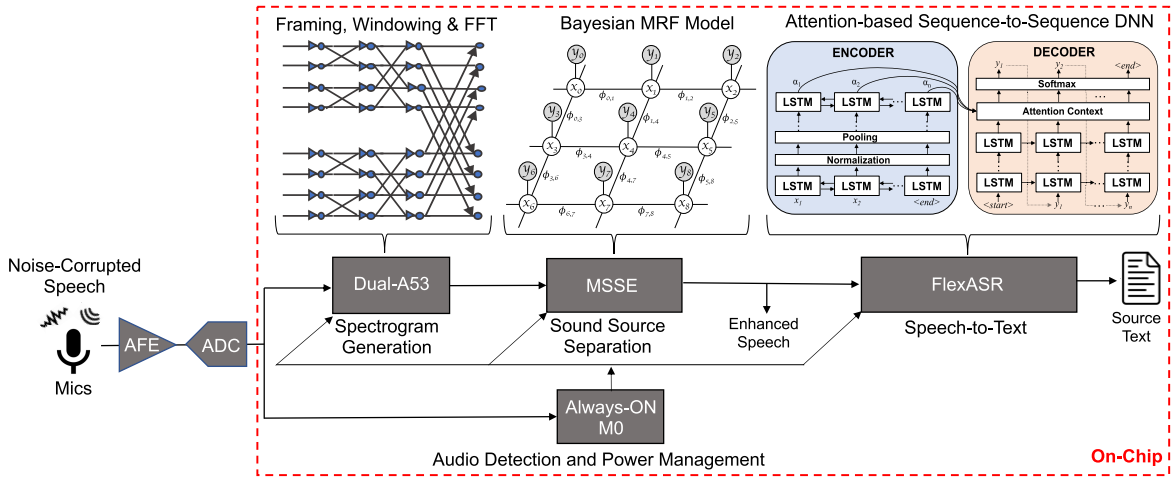


Fig. 1. Inference pipeline executed on SM6. Upon detecting audio, the M0 subsystem wakes up the accelerators and A53 produces spectrogram features that get denoised by the MSSE engine. Then, from the enhanced speech, FlexASR accelerates attention-based ASR workloads.

(SoC) for edge/the IoT devices, executing end-to-end speech-enhancing attention-based ASR workloads (Fig. 1). The proposed hardware–software co-design solution demonstrates a tight coupling between special function accelerators and CPU processing. Notably, the mobile-class A53 CPU performs feature extraction tasks and other critical accelerator supporting tasks on its dual-issue datapath. The Markov random field source separation engine (MSSE) runs sound source separation on the post-processed spectrogram frames in an unsupervised fashion. Then, FlexASR accelerates a bidirectional LAS model for noise-robust ASR. To save power during intermittent inference jobs, the always-ON M0 autonomously monitors incoming audio amplitudes and subsequently boots the A53, MSSE, and FlexASR when the signal magnitude exceeds a threshold.

High-performance ASR systems are often trained with datasets exhibiting adverse acoustic conditions. In such systems, noise robustness is partly paid with larger ASR model sizes that may stretch the on-chip memory capacity of an IoT SoC while at the same time worsening latency and energy metrics. In SM6, by virtue of having MSSE sound source separation preceding the speech-to-text computation, up to 6× smaller and iso-accurate LAS models can be fully stored in FlexASR scratchpads—promoting higher energy efficiency without compromising accuracy.

This article, therefore, makes the following contributions.

1) *Energy-Efficient Acceleration of Attention-Based Bidirectional RNNs in FlexASR:* We describe FlexASR processing element (PE) architecture utilizing adaptive floating-point datapaths for performing quantized DNN computations with greater accuracy. We further describe its multi-function global buffer GB) unit, which efficiently accelerates the attention mechanism among other specialized compute units (e.g., layer normalization and pooling).
2) *Unsupervised Sound Source Separation in MSSE:* We make the case for using probabilistic Bayesian models and solving them via accelerated Gibbs sampling inside MSSE in order to enhance noise-corrupted speech signals. By virtue of preceding the speech-to-text computation, MSSE allows FlexASR to fully store and infer significantly smaller size, iso-accurate ASR models trained on single-source clean datasets.
3) *End-to-End Performance Demonstrating Real-Time Throughput:* SM6 achieves end-to-end per-frame ASR latencies well below the 32-ms spectrogram frame length while consuming nominally 2.24 mJ per inference.

The proposed chip was first presented in [17] and [18]. This article significantly expands on the architectural design and measurement details of the system and its various computing components. This article is organized as follows. Section II provides an overview of the main machine learning workloads executed on the test chip. The SoC architecture and the various agile design logistics used to implement the test chip are presented in Section III. The FlexASR accelerator and the MSSE are detailed in Sections IV and V, respectively. The test chip measurement results are presented in Section VI. Finally, concluding remarks are drawn in Section VII.

II. Background

In this section, we offer an overview of the machine learning models behind the main workloads accelerated in the inference pipeline (Fig. 1), namely, sound source separation and ASR.

A. Bayesian Inference on Markov Random Field

A pairwise Markov random field (MRF) with four-connected neighbors is a type of Bayesian model commonly used for labeling problems. Fig. 2 shows the factorized distribution on a four-connected pairwise MRF. Given a node $i \in V$, the set of all nodes, the relationship between the label on the node x_i and the observed data y_i is represented as potential ϕ_i. The edge between nodes i and j represents the

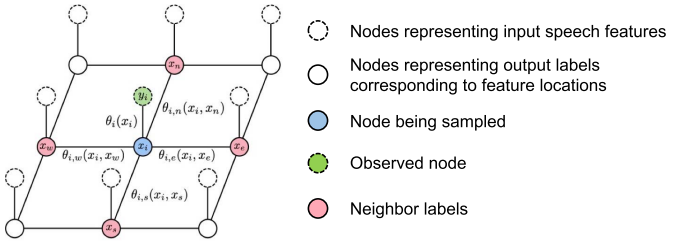


Fig. 2. Bayesian MRF model highlighting sampled, observed, and neighbor nodes.

affinity between the neighboring nodes using potential $\phi_{i,j}$. The posterior distribution of MRF is the product of all node and edge factors

$$P(x, y) = \frac{1}{Z} \prod_{i \in V} \phi_i(x_i, y_i) \prod_{i,j \in E} \phi_{i,j}(x_i, x_j) \tag{1}$$

where Z is called a partition function or a normalizing constant; it is the sum of all possible values of ϕ. Following Gibbs distribution formulation, the probability distribution in (1) is proportional to the sum of energy functions, which is equivalent to taking the negative log of (1):

$$E(x, y) = \sum_{i \in v} \theta_i(x_i, y_i) + \sum_{i,j \in \varepsilon} \theta_{i,j}(x_i, x_j) \tag{2}$$

where $\theta_i(x_i, y_i)$ and $\theta_{i,j}(x_i, x_j)$ are energy functions often referred to as data cost and smoothness cost, respectively [35], [37]. They are labeled next to their corresponding edges in Fig. 2. To find the maximizing assignment of x in (1), we use the Gibbs sampling, which is executed by MSSE, to perform maximum *a posteriori* (MAP) inference to find labels x that minimize the energy function. The Gibbs sampler is constructed by iteratively sampling each variable in the MRF given all the neighboring variables. Since each variable's energy function depends on its neighbor's label, updating each variable's label will slightly improve its neighbor's probability distribution as well. After enough iteration, all variables' labels reach convergence, which would minimize the total energy function and maximize their posterior probability.

B. LAS seq2seq Models

The seq2seq deep learning models [8] have generated impressive results in many sequence transduction tasks, such as speech recognition [8], machine translation [26], question answering [14], and image captioning [49]. For greater contextual performance, it is standard to augment the decoding process of seq2seq networks with an attention mechanism [3], which allows the network to pay attention to the most relevant parts of the source sequence during each decoding time step.

Fig. 3 shows a typical attention-based seq2seq network known as LAS [5], which is adopted in this work for ASR inferencing. The encoder stage contains unidirectional or bidirectional vanilla-RNN, GRU, or long short-term memory (LSTM) stacks sandwiched between normalization and

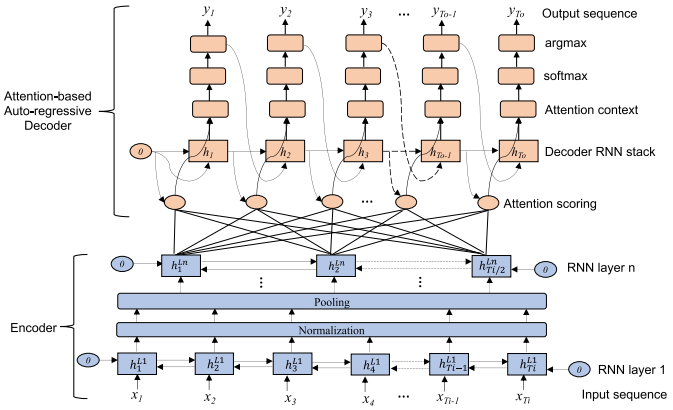


Fig. 3. LAS seq2seq network highlighting the encoder stage (in red) and the attention-based decoding stage (in blue). For each output time step, all the final encoder hidden states are attended and scored.

TABLE I
COMPUTATIONS ACCELERATED IN FLEXASR

Neural Transformation	Vanilla RNN GRU LSTM Linear
RNN Sequence	Forward-Only Bidirectional
Pooling	Mean Maximum
Normalization	Layer Normalization
Attention	Soft attention mechanism [26]

pooling layers. At each output time step, the decoder stage estimates the saliency weight of each output hidden state coming out of the final encoder layer via the attention mechanism [3]. The latter is often called "soft" attention given that the final encoder hidden state sequence acts as a soft-addressable memory [9] whose words are weighted to compute the context vector after a pass through the decoder RNN stack. Assuming a greedy search, the most likely prediction is computed by taking the Argmax of the Softmax output probabilities. We note that this Softmax operation is useful only during training for gradient approximation that can be skipped during inference of the LAS model in FlexASR. The decoder computes the next time steps in an auto-regressive manner.

The main computation kernels in LAS models (i.e., RNN and linear layers, attention, layer normalization, and pooling as shown in Table I) are efficiently accelerated inside FlexASR (discussed in Section IV).

III. SoC ARCHITECTURE AND IMPLEMENTATION

This section describes the SM6 SoC architecture and the hardware–software design and verification methodology employed during the test chip implementation. The main computing components of the SoC (Fig. 4) are: 1) FlexASR; 2) MSSE; 3) dual-core Arm Cortex-A53 CPUs; and 4) an always-ON M0 subsystem—interconnected by 128-bit AXI and 32-bit AHB buses. A 1-MB SRAM buffer is provisioned at the top level in order to store the intermediate pre- and post-processed data [25] of the inference pipeline (Fig. 1). The SoC is also equipped with various off-chip interfaces required for DRAM access via field-programmable gate array (FPGA) and for debug.

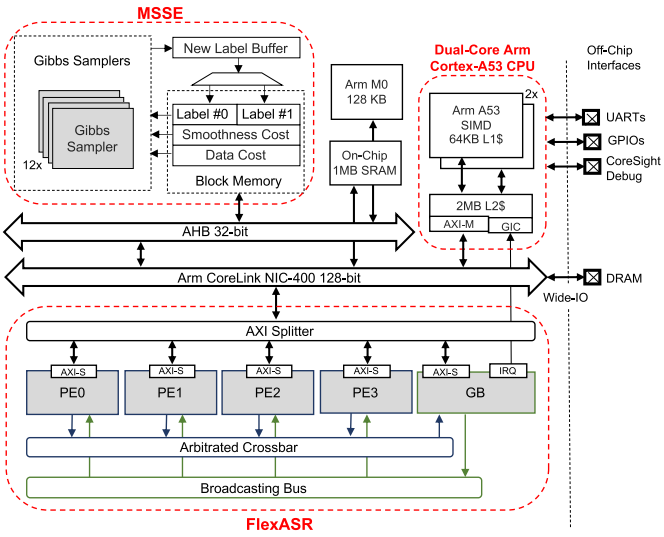


Fig. 4. Block diagram of the SM6 SoC, highlighting main components.

A. FlexASR

It is designed to support the key computational kernels and features seen in LAS seq2seq networks (Fig. 3) as shown in Table I while also allowing spatial and temporal configurations. As shown in Fig. 4, FlexASR consists of four PEs and a multi-function GB unit. The communication between GB and PEs is performed via custom-built channel links. A centralized arbiter is used to referee the stream of PE partial results, which will be aggregated by the GB. Once the full activation has been collected, the GB will then broadcast it back to the PEs for the next time step computation. Each PE and GB is interfaced with an AXI-Slave port. An interrupt (IRQ) channel originating from the GB to the A53 cluster is implemented to indicate the successful completion of the instructed task.

Section IV further expands on the FlexASR architecture, detailing its PE, GB, and tiling mechanisms.

B. MRF Source Separation Engine

Over the last decade, there has been extensive research on the design of ML accelerators [1], [7], [12], [13], [28], [31], [39], [39] to solve supervised learning problems. In contrast, unsupervised Bayesian models can be effective in solving problems relying on unlabeled data expressing various degrees of information uncertainty [19], [20]. Unfortunately, Bayesian inference workloads do not efficiently scale on traditional CPUs and GPUs, therefore requiring specialized hardware.

In this work, we accelerate Gibbs sampling operations on MRFs for the purpose of denoising noise-corrupted speech or enhancing a particular acoustic source in an environment with multiple sound sources. The unsupervised Bayesian algorithm excels in a more dynamic environment such as when sources are moving with respect to the microphones [18], which can potentially create problematic corner cases for supervised methods where it is necessary to cover all scenarios with training data. This specialized computation is accelerated in the MSSE further discussed in Section V.

MSSE, which contains 12 parallel Gibbs samplers (Fig. 4), is similar to PGMA [20], a general-purpose Bayesian inference accelerator. However, we further customized and optimized it specifically for sound source separation workloads by enabling only binary label support. This customization resulted in fewer pipelines and 2× speedup over PGMA (Section VI).

C. Dual-Core Arm Cortex-A53

The inference of the speech-enhancing pipeline (Fig. 1) effects numerous dynamic data exchanges between the CPU and the accelerators. SM6 integrates two A53 CPU cores [42], proven in high-performance embedded and mobile SoCs, for the following versatile purposes.

1) *Feature Extraction Tasks:* Framing, windowing, and 1024-pt fast fourier transform (FFT) tasks, required to synthesize the overlapping sequence of speech spectrograms, are vectorized using Ne10 single Instruction/multiple data (SIMD) instructions [45].
2) *Accelerator Programming:* The AXI-Master port of the A53 issues instruction set architecture (ISA) instructions to FlexASR and MSSE to configure the nature, shape, and size of their workloads.
3) Expectation–maximization (EM) algorithm, which is a supplemental step of the Gibbs sampling process during sound source separation [33].
4) *8-bit Floating-Point (FP8) Quantization:* As FlexASR PEs work on FP8 operands, the 32-bit fixed-point outputs from MSSE need to be converted and scaled down to lower bit precision.
5) *Label Mask Convolution:* A53 convolves the binary label mask from MSSE with the original spectrogram in order to extract the clean speech.
6) Other miscellaneous tasks that include IRQ handling.

D. Design and Verification Methodology

In order to develop the SoC in an agile manner while minimizing tapeout risks, we adopted the chip design and infrastructure methodology first outlined in [24]. Specifically, leveraging the CHIPKIT scaffold [48] and various ARM collaterals (e.g., A53 and M0 soft IPs, Arm Socrates for generating the NIC-400 interconnect) allowed us to focus on the main differentiating features of the SoC. One such differentiation was in the hardware–software co-design of FlexASR.

FlexASR was designed using object-oriented high-level synthesis (HLS) for fast SystemC-to-RTL prototyping [17]. In order to evaluate the bit-level correctness of the hardware on a realistic speech-to-text workload, we developed a design and verification flow, which closes the loop between the software modeling and the backend hardware implementation being abstracted within the HLS environment. Considering the software ML framework (e.g., PyTorch and TensorFlow) to be golden, the HLS environment allowed us to quickly make hardware tweaks and ECOs until the hardware and software DNN activations returned matching numerical results, the post-HLS verification is functionally correct, and post-HLS PPA results are satisfactory. This agility is made possible by the higher level of abstraction imposed by the HLS flow.

The SystemC source code description of FlexASR is now publicly available [43].

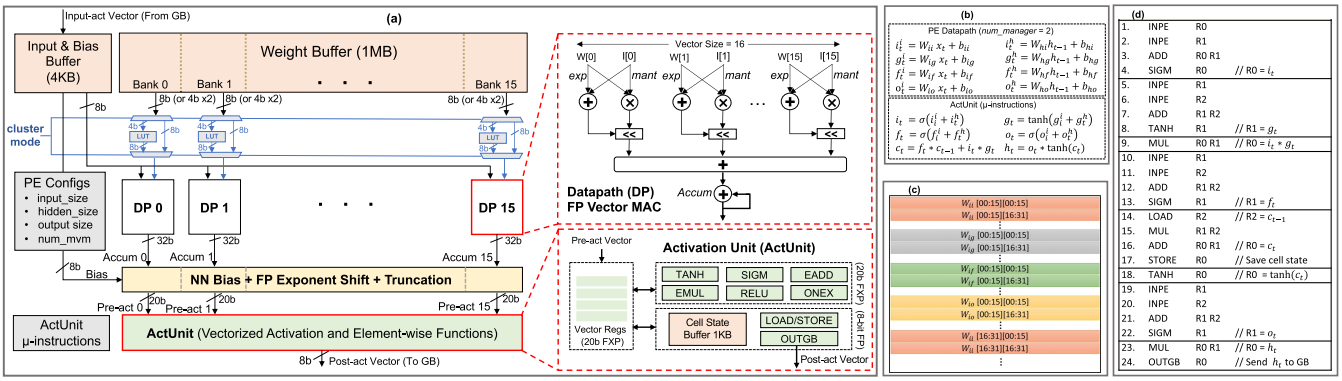


Fig. 5. (a) FlexASR PE highlighting its FP vector MAC and ActUnit. For (b) LSTM example, we adopt (c) a custom interleaved tensor tiling in the weight buffer for (d) hazard-free vector operations in the ActUnit.

IV. FLEXASR

In this section, we provide details of the FlexASR architecture to accelerate LAS models. We can categorize the seq2seq computations into two main parts: 1) RNN computations for each time step, which mainly involve matrix-vector multiplications (MVMs) with fixed weights and dynamic activations, and 2) auxiliary operations such as attention, normalization, and pooling, which involve activations across time steps.

For the first case, four PEs with 16 lanes of vector multiply-and-accumulates (MACs) are provisioned to efficiently parallelize MVMs. The idea of a weight stationary [7], [34] dataflow is adopted to divide the workload of RNN computations and minimize the data movement of weights, given that the RNN workload tends to be memory-bound. Therefore, each PE will initially store fractions of the weight matrix in their respective weight buffer, and during computation, the GB and PEs exchange input and output activations. The second case is handled by the GB unit, which houses the input and output activations and contains several specialized functional units to handle across-time-step seq2seq computations such as normalization, pooling, and attention.

A vector size of 16 is applied to every part of the FlexASR design, that is to say, the operations in the PE or GB are always effected on 16-element vectors or involve multiplication between a 16×16 matrix and a 16-element vector. Larger vector sizes produce higher accelerator throughput at the expense of reduced granularity for the RNN hidden state size. Therefore, the size of the RNN hidden state programmed in FlexASR is constrained to be a factor of 16—although one may zero-pad a non-compliant tensor shape in software prior to acceleration with FlexASR. Our choice of tile size is also influenced by the design of memory instances and the 128-bit AXI format. For example, a tile of input/weight has $16 * 8\text{-bit} = 128\text{-bits}$, which motivates a memory bank design with a data width of 128 bit per entry such that an AXI operation can access the full row in this memory bank.

A. Processing Element

The PE [Fig. 5(a)] is the computational workhorse of FlexASR during RNN, LSTM, GRU, or linear layer computations. It contains a 1-MB 16-bank weight buffer and a 4-kB input and bias buffer for storing the MVM operands in FP8 precision. FlexASR FP8 format is E4M3 (i.e., 1-bit sign, 4-bit exponent, and 3-bit mantissa) without support for denormals. This FP8 format yielded the best accuracy outcomes after performing a search on the optimal exponent bit width to satisfy the dynamic range requirements of LAS models. The PE also provides alternative support for weight clustering implemented using lookup tables (LUTs) mapping 4-bit weight indexes to their 8-bit centers. This enables $2\times$ storage compression in the PE weight buffer.

Each weight buffer bank has a read port that feeds into a floating-point vector MAC that provides scalability along a vector dimension of 16 (similar to the PE architecture in [52]).

Therefore, each PE instantiates 16 vector MACs in total (i.e., 256 MACs/cycles), to perform MVMs between an FP8 weight vector and an FP8 activation vector.

To boost the dynamic range and accuracy of quantized RNN computations, the 32-bit fixed-point accumulated sum is dynamically shifted, at a per-layer granularity, by an exponential bias, expbias. The latter is extracted from the maximum absolute value in the layer's weight matrix and stored in PE registers. This allows resilient and near-FP32 accuracy at FP8 precision on seq2seq models exhibiting wide parameter distribution [41]. After layer-wise adaptive floating-point exponent shift, the partial sums are then post-processed by the PE activation unit (ActUnit).

The ActUnit performs a sequence of vector operations (Table II) to compute the necessary activation functions

TABLE II
VECTOR OPERATIONS SUPPORTED IN THE FLEXASR ACTUNIT

Command	Description
ADD A2 A1	A2 = A2 + A1
MUL A2 A1	A2 = A2 * A1
SIGM A2	A2 = Sigmoid (A2)
TANH A2	A2 = Tanh (A2)
RELU A2	A2 = ReLU (A2)
ONEX A2	A2 = 1 – A2
COPY A2 A1	A2 = A1
INPE A2	Get accumulation result from PECore into A2
STORE A2	Store cell state from A2 into ActUnit Buffer
LOAD A2	Load cell state from ActUnit Buffer into register A2
OUTGB A2	Send A2 to Global Buffer

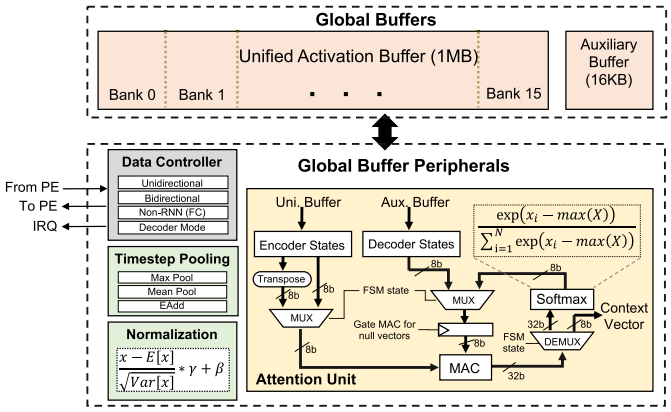


Fig. 6. Macro-architecture of the FlexASR multi-function GB unit.

(e.g., sigmoid, tanh, and ReLU) and the element-wise addition (EADD) and multiplication of matrix-vector products coming out of the truncation unit. Fig. 5(c) shows the tiling convention in the multi-bank weight buffer and the ensuing sequence of ActUnit commands required to fully compute LSTM kernels without encountering logical hazards.

B. Multi-Function GB Unit

The FlexASR GB unit collects and unifies, across time steps, the partial RNN output states that the PEs compute. Once the partial RNN outputs for a time step are fully aggregated, the GB then broadcasts the complete activation back to each PE for the next time step computation. Moreover, the GB is augmented with auxiliary processing units that compute the attention mechanism, mean and max pooling, and layer normalization, all of which are commonly invoked in modern seq2seq natural language processing (NLP) networks. We note that the normalization and the softmax operations used during the attention calculation contain several serial operations (e.g., running average); therefore, DNN accelerators often offload these computations to a nearby CPU due to the lack of parallelism opportunities. We propose to compute them within the confines of FlexASR in order to reduce CPU-accelerator inter-layer activation exchanges, thereby avoiding undue latencies during the end-to-end inference of the seq2seq model.

Fig. 6 shows the macro-architecture of the FlexASR GB. A 1-MB 16-banks unified buffer is used to store the partial RNN hidden states computed by the PEs across time steps. This capacity is large enough to fully store thousands of activation time steps at any given time, corresponding to more than 200 words of speech, and allow inference of large-vocabulary applications requiring nuance and context understanding, especially in long-sequence transductions.

While the unified buffer has a single write port, each bank has its own independent read port. A 16-kB auxiliary buffer is used to house the attention intermediate states and the learnable normalization parameters. Load/store accesses to the 1-MB and 16-kB buffers are controlled by a GB manager, which responds to the requests from the PEs and the auxiliary processing modules. The latter is composed of the following.

1) The RNN control unit which orchestrates the sequencing of the configured RNN flow mode (i.e., uni-directional, bidirectional, and seq2seq decoder mode) between the PEs and GB units. For this purpose, the RNN control module uses the configured number of time steps and the RNN hidden state size to track its job progress.
2) The attention mechanism unit which computes the soft attention mechanism [26] for each decoding time step. During this phase, encoder and decoder states are read from the GB unified and auxiliary buffers, respectively, before MAC operations generate scores processed by a SoftMax unit to produce the attention weights. To prevent numerical instability, the SoftMax is computed by subtracting the max score in the numerator and denominator. The attention context vector is then obtained by multiplying the attention weights with the transposed encoder states. Algorithm 1 details the step-by-step vectorized computation of the attention unit.
3) The layer reduction unit which can be configured to perform mean or maximum pooling on the RNN activations, as well as EADD of the forward and backward time steps during the bidirectional mode. Notably, *Concat*, *sum*, and *average* merge modes used during bidirectional RNNs are supported by striping forward and backward time steps across alternate banks in the GB unified activation buffer. For the sum or average merge modes, the GB layer reduction module performs EADD or averaging on concatenated activations.
4) The normalization unit which computes layer normalization [2] on the RNN activations in order to speed up the training process. During the seq2seq inference, a hidden state activation is normalized as

$$X_{\text{norm}} = \frac{X - E[X]}{\sqrt{\text{Var}[X]}} * \gamma + \beta \tag{3}$$

where γ and β are learnable parameters obtained after training and stored in the GB auxiliary buffer. The normalization module first computes the mean, $E[X]$, by running average over the number of hidden states, and then evaluates the variance, $\text{Var}[X]$, as: $E[X^2] - E[X]^2$. This process gets repeated for all the needed time steps.

Finally, we note here that the attention, pooling, and normalization datapaths vectorize their computations with a vector size of 16, which accelerates sequential operations.

V. MRF Sound Source Separation Engine

The sound source separation mechanism is similar to the approach used in [21] whereby, from the interaural level difference (ILD) of the input spectrograms, a binary MRF is constructed and then solved using the Gibbs sampling, which is a popular Bayesian Markov chain Monte Carlo (MCMC) inference method. The latter is an iterative process that computes a set of labels that minimize a cost function describing the conditional distribution for each label. Consequently, the mean and variance for each of the sources are updated repeatedly during the Gibbs sampling inference until convergence. In Fig. 7, we observe that starting from a mixture of sound sources, the denoising performance improves with the number

Algorithm 1 Computation Steps of the Attention Unit

Input: Encoder Matrix M, Decoder Vector v
Output: Attention Context Vector A
N_T := encoder time steps in tiles
N_D := decoder size in tiles
$max = -\inf$
// 1st *MV Mult*
for $i = 0$ *to* $N_T - 1$ **do**
  $accum := 0$
  for $j = 0$ *to* $N_D - 1$ **do**
    $W := M_{[16i:16i+15][16j:16j+15]}$
    $v := v_{[16j:16j+15]}$
    $accum += W * v$
  store $accum$ to auxiliary buffer
  // Get maximum at the same time of 1st stage
  if $max < max(accum)$ **then**
    $max = max(accum)$
  // Denote output of 1st *MV Mult* as X
  $X_{[16i:16i+15]} = accum$
// Softmax step 1: SRAM read on X
$sum_{exp} = 0$
for $i = 0$ *to* $N_T - 1$ **do**
  $sum_{exp} += sum(exp(X_{[16i:16i+15]} - max)$
// Softmax step 2: SRAM read/write to get result X'
for $i = 0$ *to* $N_T - 1$ **do**
  $X'_{[16i:16i+15]} = (X_{[16i:16i+15]} - max)/sum_{exp}$
// 2nd *MV Mult*
for $i = 0$ *to* $N_D - 1$ **do**
  $accum := 0$
  for $j = 0$ *to* $N_T - 1$ **do**
    $W := M^T_{[16i:16i+15][16j:16j+15]}$
    $v := X'_{[16j:16j+15]}$
    $accum += W * v$
  store $accum$ to GB auxiliary buffer
  // Context vector, A, is output of 2nd *MV Mult*
  $A_{[16i:16i+15]} = accum$

of Gibbs sampling iterations. Notably, at 120 iterations, the mean square error (MSE) with respect to the baseline spectrogram is only 2%.

Formally, Bayesian sound source separation can be divided into two phases. In the first phase, Gibbs sampling is performed for a specified number of T iterations for N nodes in the MRF. Then, in the second phase, the MRF distribution is updated by computing improved Gaussian parameters through EM. MSSE is designed to accelerate the Gibbs sampling process via parallelization, while the EM step is handled by the A53 cores. MSSE contains programmable registers to configure the width and height of the MRF and the number of Gibbs sampling iterations, T, to execute inside the hardware.

The ILD matrix is computed from the difference between the audio recorded at the two microphones. As shown in Fig. 8, MSSE takes in the ILD matrix and assumes

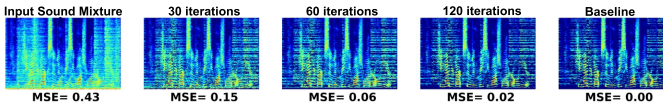


Fig. 7. Unsupervised sound source separation process showing improved spectrogram quality following successive Gibbs sampling iterations.

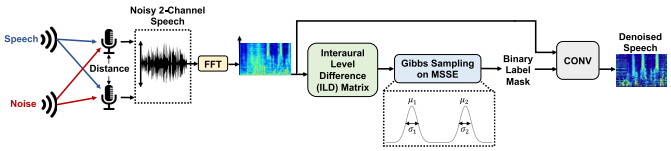


Fig. 8. Sound source separation is performed via Gibbs sampling inference on the ILD from the speech and noise sources.

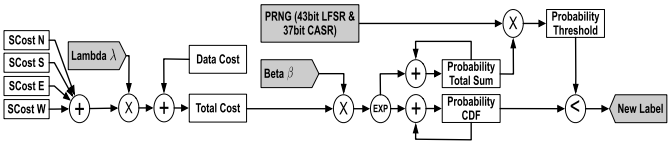


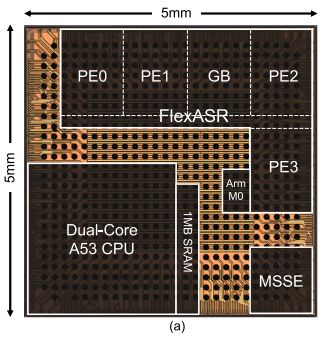
Fig. 9. Datapath of the Gibbs sampler in MSSE.

that the probabilities of observing ILD values follow two source-specific Gaussian distributions and assigns binary labels to minimize a cost function based on how likely the observed ILD values are to have come from either source. The final output labels classify each time–frequency component as being from one of the two separated output channels. The binary mask output is used to isolate the time–frequency components corresponding to each source, and the desired clean audio is then identified as the output channel with greater magnitude. This denoised audio is then passed to the FlexASR inference pipeline.

Prior to running Gibbs sampling in MSSE, the MRF is split into multiple tiles such that each MRF node update step is handled by one of the 12 provisioned Gibbs samplers. Fig. 9 shows the 32-bit fixed-point datapath of the Gibbs sampler. First, the sampler receives the smoothness cost (i.e., degree of label difference between neighboring nodes) and the data cost (i.e., mixture of Gaussian distributions) of an MRF node in order to generate probabilities for every binary label value. The smoothness cost is usually pre-computed and could be stored in a read-only memory (ROM), while the data cost is computed based on the input audio clip. Then, the Gibbs sampler samples a new label based on the computed probability distribution. To sample a new label, the probability distribution for each label and their cumulative sums are stored in a first in, first out (FIFO). A uniform random number, produced from a pseudo-random number generator (PRNG), scales the final cumulative sum (total sum of all probabilities) in order to generate a probability threshold that is compared against each of the cumulative sums in the FIFO. This comparison is repeated until a sum larger than the probability threshold is found—at which point the corresponding label mask is pushed out of the sampler and stored in a dedicated new label buffer.

VI. Measurement Results

An annotated photograph of the 25-mm^2 SM6 die is shown in Fig. 10(a). The SM6 die was implemented in the TSMC



(b)

Technology	TSMC 16nm FFC
Die Area	25mm^2
Total SRAM	9.8MB
Clock Domains	6
Power Domains	5
Supply Voltage	0.55 – 1V
Packaging	Flip-chip BGA-672

(c)

	Dual-A53	MSSE	FlexASR
Workloads	FFT, Application Logic	Markov Random Field	Attention-based seq2seq models
Data Type	FP64	FxP32	FP8
Area	6.21mm^2	1.31mm^2	8.84mm^2
SRAM	2.41MB	0.103MB	5.03MB
Voltage	0.55 - 1V	0.55 - 1V	0.55 - 1V
Frequency	354 – 775MHz	287 – 651MHz	130 – 573MHz
Power @ Fmax/0.8V	50.4mW	42.2mW	214mW
Power Eff.	-	4.33 – 17.6 Gsamples/s/W	2.6 – 7.8 TFLOPS/W

Fig. 10. (a) Annotated die photograph of the 25-mm^2 SM6 test chip. Summary of (b) SoC and (c) its main compute clusters.

16-nm FinFET technology and flip-chip bonded into a 672-pin ball grid array (BGA) package [Fig. 10(b)]. In order to orchestrate various energy efficiency ranges, six clock domains capable of outputting fine-grained frequencies and five power domains with a 0.55–1.0-V functional operation range are servicing the main compute clusters and other on-chip endpoints. Fig. 10(c) lists the technical specifications of the main compute clusters. Notably, at 0.8-V nominal voltage, the A53 cores, MSSE, and FlexASR dissipate 50.4, 42.2, and 214 mW, respectively, at their maximum operating frequencies. With the compute clusters inactive, SM6 has a standby power of ∼4 mW, as the M0 remains active to sense the GPIO pins for audio detection.

To compare SM6 against commodity edge platforms, we evaluated speech-to-text LAS models and Gibbs sampling on an Nvidia TX2 mobile GPU, a Xilinx ZCU102 FPGA, and the integrated dual-core A53 CPU. TX2 results were obtained from CUDA implementations on the GPU module in order to reap the benefits of parallelization. The ASIC RTL of FlexASR and MSSE was programmed on the ZCU102 platform for evaluating FPGA performance. The Ne10 [45] and eigen [44] libraries were used to vectorize supporting ASR and Gibbs sampling kernels on the A53 SIMD units.

We conducted the following application-level measurements at room temperature using typical silicon.

A. Per-Layer Characterization

We begin by characterizing the processing times and energy dissipation of the main SM6 compute clusters while running individual seq2seq layers and Gibbs sampling iterations (Fig. 11). We can make the following observations.

1) FlexASR provides significant speedup gains while accelerating seq2seq layers—with attention, LSTM, and GRU RNNs showing greater benefits. Notably, the 160-time steps bidirectional LSTM (BILSTM) exhibits higher processing speedup over CPU, GPU, and FPGA compared to the unidirectional LSTM scenario—due to FlexASR striping forward and backward activations in alternate banks in its GB unit. In addition, even though the normalization and pooling operations are very serial in nature, by specializing their datapaths within the confines of the accelerator and thereby avoiding accelerator-CPU activation exchange, FlexASR still outperforms all other platforms.
2) MSSE achieves appreciable latency reductions over the commercial edge platforms—demonstrating the need for specialized Gibbs sampling accelerated computing as the A53 cores, TX2 GPU, and ZCU102 are 1577×, 7×, and 4× slower than MSSE, respectively. Moreover, as MSSE was optimized for Bayesian inference with binary labels as opposed to the general-purpose PGMA accelerator [19], [20] (supporting up to 64 labels), Gibbs sampling runs twice as fast on MSSE.
3) Both FlexASR and MSSE generate orders-of-magnitude smaller energy consumption compared to the commercial edge devices. This is particularly striking during Gibbs sampling as the A53 cores, TX2 GPU, and ZCU102 produce 1969×, 134×, and 446× larger energy dissipation, respectively, compared to MSSE. Furthermore, although the FPGA generally executes seq2seq kernels faster than the dual-core A53 and TX2 GPU, its power consumption envelope is significant enough to make it the least energy-efficient platform for several workloads (e.g., BILSTM, GRU, and pooling).

Finally, we note that the RNN (BILSTM, LSTM, or GRU) and linear workloads, which are computed in the FlexASR PEs, achieve near 100% MAC utilization. However, at the overall end-to-end ASR workload level, PE utilization is about 71%, given that the PEs become idle during the computation of normalization, pooling, and attention that account for 19%, 6%, and 4%, respectively, of a representative ASR workload.[1]

B. End-to-End Characterization

To demonstrate the accuracy and end-to-end performance benefits of the proposed speech-enhancing pipeline, we compare our approach (Scenario D) with three other common inference scenarios using the LibriSpeech dataset [27], as shown in Fig. 12. Scenario A emulates a clean environment in which the speaker's voice is the single audio source. Scenario B mixes the speaker's voice with another human voice source in a simulated room environment for a signal-to-noise ratio (SNR) of 0.90 dB. Finally, Scenario C (*Noisy + Big*) adopts a much larger ASR model (22 versus 3.5 MB used in Scenarios A, B, and D) trained with a noise-corrupted LibriSpeech dataset in order to learn from noisy inputs. The *Noisy + Big* model was gradually sized up, by increasing the hidden state dimension, until its word error rate (WER) is much closer to Scenario A in noisy cases. The ASR LAS model adopted in Scenarios A, B, and D consists of four BILSTM RNN stacks in the encoder with 512 cells and a 256-cell unidirectional LSTM RNN in the decoder. The LAS model in Scenario D was scaled up to 1024 cells in each of the four BILSTM RNN stacks in the encoder and 1024 cells in its decoder LSTM RNN unit. The following observations are made.

1) By denoising incoming audio signals and preceding the speech-to-text inference, MSSE allows FlexASR to compute significantly smaller size (up to 6× smaller), iso-accurate ASR models trained on widely available single-source clean datasets [Fig. 12

[1]The ASR model used in Scenarios A, B, and D from Fig. 12.

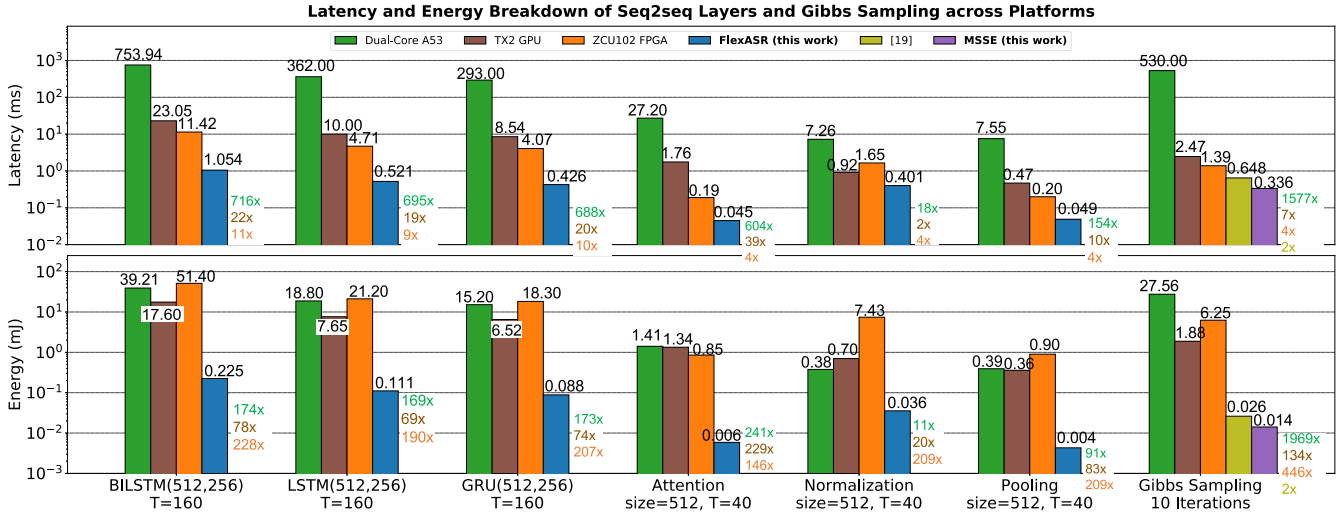


Fig. 11. Breakdown of latency (top row) and energy (bottom row) for individual seq2seq layers running on FlexASR and Gibbs sampling running on MSSE, compared to running on different commercial platforms. Here, FlexASR, MSSE, and the A53 cores are running at frequencies of 440, 533, and 715 MHz, respectively, at 0.8 V. Accelerators' throughput can be found in Table III.

(top left)]—obviating the very inefficient strategy of scaling up the DNN model size (Scenario C) in order to achieve noise robustness. Furthermore, the proposed ASR pipeline delivers 3× accuracy improvement over the unseparated noise case (Scenario B) and is within 1% of the clean input baseline case (Scenario A). We note that in Scenario D, MSSE executes 150 Gibbs sampling iterations, improving speech quality by up to 7.3-dB signal-to-distortion ration (SDR).

2) The proposed pipeline achieves 4.3× lower end-to-end per-frame latency (bottom left) and 7× energy improvement (bottom right) compared to the similarly accurate case in Scenario C, which requires significant off-chip data movements because the weights of the upsized ASR model cannot fully fit in FlexASR PE scratchpads. Notably, SM6 achieves latency per frame of 18.4 ms while dissipating 2.24 mJ of energy during the end-to-end speech-enhancing ASR inference.
3) Due to the use of RNNs, the inference computation is mainly memory-bound. Therefore, it can be observed that the latency and energy costs of memory transfers in the bigger ASR model (i.e., Noisy + Big in Scenario C) are much higher compared to the leaner ASR model used in Scenarios A, B, and D. For example, memory transfers in Scenario C account for 36% of the end-to-end latency versus only 6% in our proposed pipeline whose RNN model is 6× smaller.
4) Fig. 12 (top right) shows that despite substantial energy expenditures, the commercial edge platforms fail to provide real-time performance as their per-frame latencies exceed the 32-ms frame length.

C. CPU Characterization

As specified in Section III-C, the integrated A53 cores perform various important tasks during the inference of the speech-enhancing pipeline, which includes front-end feature extraction, FP8 quantization in preparation to speech-to-text acceleration in FlexASR, and the EM algorithm.

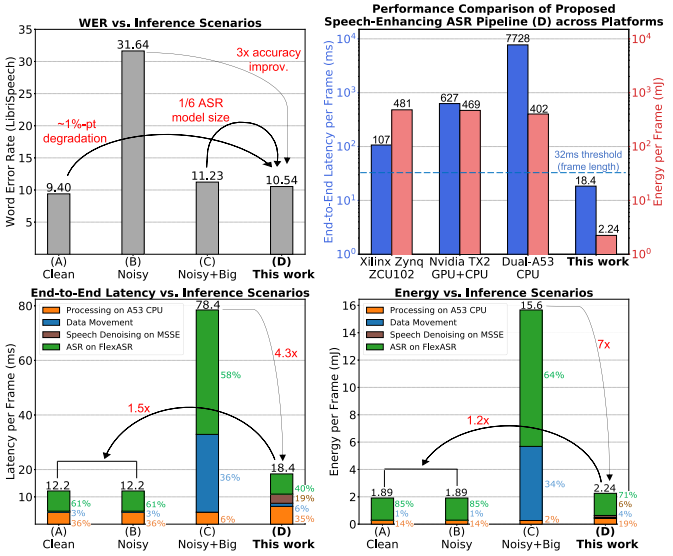


Fig. 12. End-to-end measurement results for ASR inference with (A) clean input audio, (B) noisy input audio, (C) noisy input audio using 6× larger ASR model, and (D) this work—noisy input audio with Bayesian sound source separation denoising. WER performance (top left), end-to-end per-frame latency (bottom left), energy (bottom right), and cross-platform comparisons (top right) are shown.

Fig. 13 shows the runtime breakdown of running these supporting tasks on the dual-core A53 CPU. Notably, the A53 spends the most time working on FP8 quantization (33.2%), EM (29.7%), and FFT (25.9%). FP8 quantization is necessary due to the difference in data type between MSSE (FxP32) and FlexASR (FP8) as Gibbs sampling is known to require significantly more precision for robust operation and fast convergence [4].

Spectrogram framing and windowing and the process of extracting clean labels after Gibbs sampling account for 6.1%

TABLE III
COMPARISON WITH RELATED WORK

	Giraldo et al. [10]	Guo et al. [11]	Yin et al. [50]	Lee et al. [23]	This work		
Technology	65 nm	65 nm	28 nm	7 nm	16 nm		
Core Dimension (mm^2)	2.6	6.2	1.3	19.6	21.8		
Application	KWS	KWS	ASR	Mixed-Precision Training	Speech Denoising, ASR		
Algorithm	RNN	RNN	CNN	CNN, FC, RNN	MRF, Attention-based RNN		
On-Chip Denoising	No	No	No	No	Yes (7.3 dB SDR)		
Dataset (Vocabulary Size)	GSCD (30 words)	Smart Home (11 words)	TIMIT (6k words)	-	LibriSpeech (200k words)		
Total SRAM (MB)	0.105	0.018	0.052	8.0	9.80		
Latency per Frame (ms)	16	0.127	0.5 – 25	-	15 – 45		
Accelerator	LSTM Acc.	RNN Engine	BCNN Unit	FPU	Dual-A53	MSSE	FlexASR
Data Type	4b/8b FxP	1b FxP	1b FxP	HFP8, FP16, INT4, INT2	FP64	FxP32	FP8
F_{MAX} (MHz)	8	75	50	1600	1000	651	573
Power at F_{MAX} (mW)	0.01 @ 0.8V	26 @ 0.9V	2.85 @ 0.9V		50 @ 0.8V	42 @ 0.8V	214 @ 0.8V
Workload Efficiency	-	4096 MACs/cycles	1500 MACs/cycles	8192 MACs/cycles	-	-	1024 MACs/cycles
Peak Throughput	-	0.614 TOPS	0.15 TOPS	26.2 TFLOPS *	5.90 GOPS	0.372 GSamples/s	1.17 TFLOPS
Peak Energy Efficiency	-	11.7 TOPS/W	90 TOPS/W	1.95 TFLOPS/W *	58.7 GOPS/W	17.6 GSamples/s/W	7.8 TFLOPS/W
Peak Area Efficiency	-	0.156 TOPS/mm^2	0.115 TOPS/mm^2	1.34 TFLOPS/mm^2 *	0.95 GOPS/mm^2	0.284 GSamples/s/mm^2	0.13 TFLOPS/mm^2

*: reported for HFP8 operation.

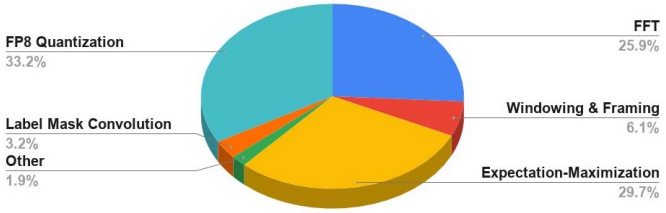


Fig. 13. Breakdown of the processing times resulting from running various supporting workloads on the dual-core A53 CPU during the end-to-end speech-enhancing ASR pipeline.

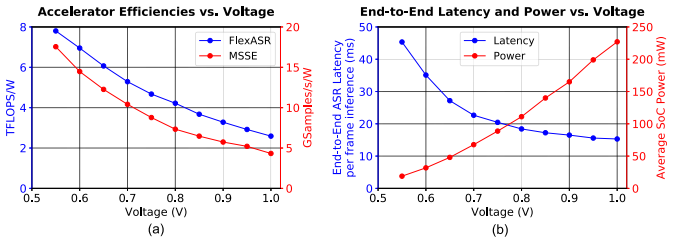


Fig. 14. Impact of voltage scaling on (a) accelerators' power efficiency and (b) end-to-end ASR latency and SoC power envelope.

and 3.2% of the CPU workloads, respectively. The CPU spends the remaining tasks (1.9%) on interrupt handling, instruction dispatch, and other minor miscellaneous application logic.

D. Voltage Scaling

To evaluate the functional efficiency range of the SoC, all the power domains are uniformly scaled from 1.0 down to 0.55 V, while the various compute clusters (FlexASR, MSSE, and dual-A53) are clocked at their respective maximum frequencies. Fig. 14(a) shows that voltage/frequency scaling on FlexASR and MSSE produces the efficiency ranges of 2.6–7.8 TFLOPs/W and 4.33–17.6 GSamples/s/W, respectively. The per-frame, end-to-end latency varies from 45 to 15 ms as the SoC voltage scales from 0.55 to 1.0 V while consuming 19–227 mW on average [Fig. 14(b)]. At nominal 0.8 V, the average per-frame SoC power is 111 mW.

E. Comparison With Previous Work

Table III provides a qualitative and quantitative comparison with some recent related work [10], [11], [23], [50] published in silicon to date. First, the proposed work is the only solution supporting an on-chip denoising solution prior to ASR or KWS, enabling computation of highly accurate, yet leaner, ASR models fully stored on-chip, thereby preventing costly DRAM accesses. Second, this is the first work to demonstrate specialized on-chip support of context-understanding attention-based seq2seq RNNs for large-vocabulary and long-sequence ASR workloads. Third, despite executing sound source separation prior to speech-to-text, this work shows competitive end-to-end per-frame ASR latency compared to prior work. Fourth, this work demonstrates 4× higher FP8 energy efficiency than a recent FP8 implementation in 7 nm [23].

VII. CONCLUSION

This article presents a 25-mm^2 SoC in 16-nm FinFET, which executes end-to-end speech-enhancing attention-based seq2seq NLP workloads. The SoC, codenamed SM6, contains two custom accelerators: 1) MSSE, a probabilistic graphical model accelerator for MRF-based speech denoising, and 2) FlexASR, a reconfigurable processor with a rich ISA for accelerating ASR via attention-based bidirectional RNNs. Feature extraction is performed in the integrated dual-A53 cores, and an always-ON M0 serves as audio detection and power management. The specialized accelerators provide orders-of-magnitude greater latency and energy gains over popular commercial edge platforms. At nominal voltage, the test chip consumes 2.24-mJ per frame while achieving end-to-end latency of 18 ms—enabling real-time

throughput. MSSE speech denoising allows on-chip storage of significantly smaller LAS ASR models in FlexASR scratchpads—promoting higher energy efficiency during the speech-enhancing pipeline without compromising inference accuracy.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

REFERENCES

[1] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," in *Proc. 43rd Int. Symp. Comput. Architecture*, 2016, pp. 1–13.

[2] J. Lei Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," 2016, *arXiv:1607.06450*.

[3] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proc. 3rd Int. Conf. Learn. Represent. (ICLR)*, 2015, pp. 1–15.

[4] Y. Chai, G. G. Ko, W.-T. M. Ting, L. Bailey, D. Brooks, and G.-Y. Wei, "CoopMC: Algorithm-architecture co-optimization for Markov chain Monte Carlo accelerators," in *Proc. IEEE Int. Symp. High-Perform. Comput. Architecture (HPCA)*, Apr. 2022, pp. 38–52.

[5] W. Chan, N. Jaitly, Q. Le, and O. Vinyals, "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Mar. 2016, pp. 4960–4964.

[6] W. Chan and I. Lane, "Deep recurrent neural networks for acoustic modelling," 2015, *arXiv:1504.01482*.

[7] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *Proc. ACM/IEEE Int. Symp. Comput. Architecture (ISCA)*, Jun. 2016, pp. 367–379.

[8] J. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, "Attention-based models for speech recognition," in *Proc. Adv. Neural Inf. Process. Syst., Annu. Conf. Neural Inf. Process. Syst.*, vol. 28, Montreal, QC, Canada, Dec. 2015, pp. 577–585.

[9] J. Dean, D. Patterson, and C. Young, "A new golden age in computer architecture: Empowering the machine-learning revolution," *IEEE Micro*, vol. 38, no. 2, pp. 21–29, Mar. 2018.

[10] J. S. P. Giraldo, S. Lauwereins, K. Badami, and M. Verhelst, "Vocell: A 65-nm speech-triggered wake-up SoC for 10-μw keyword spotting and speaker verification," *IEEE J. Solid-State Circuits*, vol. 55, no. 4, pp. 868–878, Apr. 2020.

[11] R. Guo *et al.*, "A 5.1 pJ/neuron 127.3 μs/inference RNN-based speech recognition processor using 16 computing-in-memory SRAM macros in 65 nm CMOS," in *Proc. Symp. VLSI Circuits*, Jun. 2019, pp. C120–C121.

[12] U. Gupta *et al.*, "MASR: A modular accelerator for sparse RNNs," in *Proc. 28th Int. Conf. Parallel Architectures Compilation Techn. (PACT)*, Sep. 2019, pp. 1–14.

[13] S. Han *et al.*, "EIE: Efficient inference engine on compressed deep neural network," *SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 243–254, Jun. 2016.

[14] K. M. Hermann *et al.*, "Teaching machines to read and comprehend," in *Proc. 28th Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 1–9.

[15] R. Hsiao, D. Can, T. Ng, R. Travadi, and A. Ghoshal, "Online automatic speech recognition with listen, attend and spell model," *IEEE Signal Process. Lett.*, vol. 27, pp. 1889–1893, 2020.

[16] K. Irie, R. Prabhavalkar, A. Kannan, A. Bruguier, D. Rybach, and P. Nguyen, "On the choice of modeling unit for sequence-to-sequence speech recognition," 2019, *arXiv:1902.01955*.

[17] B. Khailany *et al.*, "A modular digital VLSI flow for high-productivity SoC design," in *Proc. 55th Annu. Design Autom. Conf.*, Jun. 2018, pp. 72:1–72:6. [Online]. Available: http://doi.acm.org/10.1145/3195970.3199846

[18] M. Kim, P. Smaragdis, G. G. Ko, and R. A. Rutenbar, "Stereophonic spectrogram segmentation using Markov random fields," in *Proc. IEEE Int. Workshop Mach. Learn. Signal Process.*, Sep. 2012, pp. 1–6.

[19] G. Ko *et al.*, "A scalable Bayesian inference accelerator for unsupervised learning," in *Proc. IEEE Hot Chips 32 Symp. (HCS)*, Aug. 2020, pp. 1–27. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/HCS49909.2020.9220686

[20] G. G. Ko *et al.*, "A 3 mm^2 programmable Bayesian inference accelerator for unsupervised machine perception using parallel Gibbs sampling in 16 nm," in *Proc. IEEE Symp. VLSI Circuits*, Jun. 2020, pp. 1–2.

[21] G. G. Ko, Y. Chai, R. A. Rutenbar, D. Brooks, and G.-Y. Wei, "Accelerating Bayesian inference on structured graphs using parallel Gibbs sampling," in *Proc. 29th Int. Conf. Field Program. Log. Appl. (FPL)*, Sep. 2019, pp. 159–165.

[22] S. Kodali, P. Hansen, N. Mulholland, P. Whatmough, D. Brooks, and G.-Y. Wei, "Applications of deep neural networks for ultra low power IoT," in *Proc. IEEE Int. Conf. Comput. Design (ICCD)*, Nov. 2017, pp. 589–592.

[23] S. K. Lee *et al.*, "A 7-nm four-core mixed-precision AI chip with 26.2-TFLOPS hybrid-FP8 training, 104.9-TOPS INT4 inference, and workload-aware throttling," *IEEE J. Solid-State Circuits*, vol. 57, no. 1, pp. 182–197, Jan. 2022.

[24] S. K. Lee, P. N. Whatmough, M. Donato, G. G. Ko, D. Brooks, and G.-Y. Wei, "SMIV: A 16-nm 25-mm^2 SoC for IoT with arm Cortex-A53, eFPGA, and coherent accelerators," *IEEE J. Solid-State Circuits*, vol. 57, no. 2, pp. 639–650, Feb. 2022.

[25] H. Li, M. Bhargava, P. N. Whatmough, and H.-S.-P. Wong, "On-chip memory technology design space explorations for mobile deep neural network accelerators," in *Proc. 56th ACM/IEEE Annu. Design Autom. Conf. (DAC)*, Jun. 2019, pp. 1–6.

[26] T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process.* Lisbon, Portugal: Association for Computational Linguistics, Sep. 2015, pp. 1412–1421. [Online]. Available: https://www.aclweb.org/anthology/D15-1166

[27] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: An ASR corpus based on public domain audio books," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Apr. 2015, pp. 5206–5210.

[28] A. Parashar *et al.*, "SCNN: An accelerator for compressed-sparse convolutional neural networks," in *Proc. 44th Annu. Int. Symp. Comput. Architecture*. New York, NY, USA: Association for Computing Machinery, Jun. 2017, pp. 27–40.

[29] D. S. Park *et al.*, "SpecAugment: A simple data augmentation method for automatic speech recognition," in *Proc. Interspeech*, Sep. 2019, pp. 2613–2617.

[30] M. Price, J. Glass, and A. P. Chandrakasan, "A low-power speech recognizer and voice activity detector using deep neural networks," *IEEE J. Solid-State Circuits*, vol. 53, no. 1, pp. 66–75, Jan. 2018.

[31] B. Reagen *et al.*, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Architecture (ISCA)*, Jun. 2016, pp. 267–278.

[32] A. M. Rush, S. Chopra, and J. Weston, "A neural attention model for abstractive sentence summarization," in *Proc. EMNLP*, 2015, pp. 379–389.

[33] S. Sahu and G. Roberts, "On convergence of the EM algorithm and the Gibbs sampler," *Statist. Comput.*, vol. 9, pp. 55–64, Apr. 1999.

[34] A. Samajdar, J. M. Joseph, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "A systematic methodology for characterizing scalability of DNN accelerators using SCALE-sim," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Aug. 2020, pp. 58–68.

[35] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *Int. J. Comput. Vis.*, vol. 47, nos. 1–3, pp. 7–42, Apr. 2002, doi: 10.1023/A:1014573219977.

[36] J. M. R. Sotelo *et al.*, "Char2wav: End-to-end speech synthesis," in *Proc. ICLR*, 2017, pp. 1–6.

[37] R. Szeliski *et al.*, "A comparative study of energy minimization methods for Markov random fields with smoothness-based priors," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 6, pp. 1068–1080, Jun. 2008.

[38] T. Tambe *et al.*, "SM6: A 16 nm system-on-chip for accurate and noise-robust attention-based NLP applications," in *Proc. IEEE Hot Chips 33 Symp. (HCS)*, Aug. 2021, pp. 1–13.

[39] T. Tambe *et al.*, "EdgeBERT: Sentence-level energy optimizations for latency-aware multi-task NLP inference," in *Proc. 54th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*. New York, NY, USA: Association for Computing Machinery, Oct. 2021, pp. 830–844, doi: 10.1145/3466752.3480095.

[40] T. Tambe *et al.*, "A 25 mm^2 SoC for IoT devices with 18 ms noise-robust speech-to-text latency via Bayesian speech denoising and attention-based sequence-to-sequence DNN speech recognition in 16nm FinFET," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2021, pp. 158–160.
[41] T. Tambe *et al.*, "Algorithm-hardware co-design of adaptive floating-point encodings for resilient deep learning inference," in *Proc. 57th ACM/IEEE Design Autom. Conf. (DAC)*, Jul. 2020, pp. 1–6.
[42] *Arm Developer, Cortex-A53*. Accessed: Jan. 1, 2022. [Online]. Available: https://developer.arm.com/ip-products/processors/cortex-a/cortex-a53
[43] Harvard Architecture, Circuits, and Compilers. *FlexASR: A Reconfigurable Hardware Accelerator for Attention-Based Seq-to-Seq Networks*. Accessed: Jan. 1, 2022. [Online]. Available: https://github.com/harvard-acc/FlexASR
[44] Libeigen. *Eigen*. Accessed: Jan. 1, 2022. [Online]. Available: https://gitlab.com/libeigen/eigen
[45] Ne10 Library. *Project Ne10*. Accessed: Jan. 1, 2022. [Online]. Available: https://projectne10.github.io/Ne10/
[46] Semiconductor Research Corporation. (2020). *The Decadal Plan for Semiconductors*. [Online]. Available: https://www.src.org/about/decadal-plan/
[47] R. J. Weiss, J. Chorowski, N. Jaitly, Y. Wu, and Z. Chen, "Sequence-to-sequence models can directly translate foreign speech," in *Proc. Interspeech*, Aug. 2017, pp. 2625–2629.
[48] P. N. Whatmough, M. Donato, G. G. Ko, S. K. Lee, D. Brooks, and G.-Y. Wei, "CHIPKIT: An agile, reusable open-source framework for rapid test chip development," *IEEE Micro*, vol. 40, no. 4, pp. 32–40, Jul. 2020.
[49] K. Xu *et al.*, "Show, attend and tell: Neural image caption generation with visual attention," in *Proc. ICML*, 2015, pp. 2048–2057.
[50] 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.
[51] Y. Zhu, M. Mattina, and P. N. Whatmough, "Mobile machine learning hardware at ARM: A systems-on-chip (SoC) perspective," in *Proc. 1st Conf. Syst. Mach. Learn.*, 2018, pp. 1–3.
[52] B. Zimmer *et al.*, "A 0.32–128 TOPS, scalable multi-chip-module-based deep neural network inference accelerator with ground-referenced signaling in 16 nm," *IEEE J. Solid-State Circuits*, vol. 55, no. 4, pp. 920–932, Apr. 2020.

Thierry Tambe (Member, IEEE) received the B.S. and M.Eng. degrees in electrical engineering from Texas A&M University, College Station, TX, USA, in 2010 and 2012, respectively. He is currently pursuing the Ph.D. degree in electrical engineering with Harvard University, Cambridge, MA, USA.

From 2012 to 2017, he was an Engineer with Intel Corporation, Hillsboro, OR, USA, where he worked on various analog/mixed-signal architectures for high-bandwidth memory and peripheral interfaces on Xeon and Xeon-Phi HPC systems-on-chip (SoCs). His current research interests focus on designing energy-efficient and high-performance algorithms, and hardware accelerators and systems for machine learning applications.

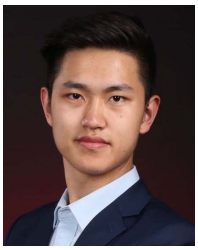
Mr. Tambe was a recipient of the NVIDIA Graduate Ph.D. Fellowship in 2021.

En-Yu Yang received the B.S. degree in electrical engineering from National Tsing Hua University, Hsinchu, Taiwan, in 2018, and the M.S. degree in computer science from Harvard University, Cambridge, MA, USA, in 2020, where he is currently pursuing the Ph.D. degree in computer science.

His research focuses on specialized architecture and hardware design for machine learning applications.

Glenn G. Ko (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical and computer engineering from the University of Illinois at Urbana–Champaign, Urbana, IL, USA, in 2004, 2006, and 2017, respectively.

He was previously with Samsung Electronics, Suwon, South Korea, where he worked on mobile application processor system-on-chip. He also spent time at Qualcomm, San Diego, CA, USA, and the IBM T. J. Watson Research Center, Yorktown Heights, NY, USA, working on machine learning accelerator architectures and deep learning kernels. He is currently a Research Associate with the Department of Electrical Engineering and Computer Science, School of Engineering and Applied Sciences (SEAS), Harvard University, Cambridge, MA, USA. He is also the CEO of Stochastic, Inc., Cambridge. His research interests include machine learning algorithms, computer architecture, and integrated circuits.



Yuji Chai received the B.S. degree in electrical engineering and engineering physics from the University of Illinois at Urbana–Champaign, Urbana, IL, USA, in 2019. He is currently pursuing the Ph.D. degree in computer science with Harvard University, Cambridge, MA, USA.

He was an Intern Research Engineer with Arm Research, Boston, MA, USA. He is the Co-Founder of Stochastic, Inc., Cambridge. His research interests include architecture and algorithm co-design, learned performance modeling, and machine learning algorithms.

Coleman Hooper is currently pursuing the B.S. degree in electrical engineering with Harvard University, Cambridge, MA, USA.

He was an Intern Digital Design Engineer with Boréas Technologies, Bromont, QC, Canada. He has worked as a Research Assistant with the Harvard Architecture, Circuits, and Compilers Group, Harvard University. His research interests are in developing model compression techniques and in hardware–software co-design for efficient edge deployment of machine learning models.

Marco Donato (Member, IEEE) received the B.S. and M.S. degrees in electrical engineering from the Università di Roma La Sapienza, Rome, Italy, in 2008 and 2010, respectively, and the Ph.D. degree in electrical sciences and computer engineering from Brown University, Providence, RI, USA, in 2016.

From 2017 to 2020, he was a Post-Doctoral Research Associate with the John A. Paulson School of Engineering and Applied Sciences, Harvard University, Cambridge, MA, USA. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, Tufts University, Medford, MA, USA. His research interests include novel design methodologies targeting energy-efficient and reliable circuits and architectures for emerging computing paradigms.

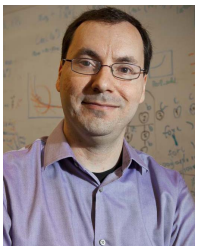
Paul N. Whatmough (Member, IEEE) received the B.Eng. degree (Hons.) from Lancaster University, Lancaster, U.K., the M.Sc. degree (Hons.) from the University of Bristol, Bristol, U.K., and the Ph.D. degree from University College London, London, U.K., in 2003, 2004, and 2012, respectively.

From 2005 to 2008, he was with Philips/NXP Research Labs, Redhill, U.K., researching hardware architecture and signal processing for software-defined radio. From 2008 to 2015, he was with the Silicon Research and Development Group, ARM Ltd., Cambridge, U.K., working on topics, including DSP hardware accelerators, variation tolerance, and system-on-chip (SoC) supply voltage noise. From 2015 to 2017, he was a Research Associate with Harvard University, Cambridge, MA, USA. He currently leads research on hardware for machine learning at Arm Research, Boston, MA, USA, and is a part-time Associate with the School of Engineering and Applied Sciences, Harvard University. He has coauthored the book *Deep Learning for Computer Architects* (Morgan & Claypool, 2017).

Dr. Whatmough is a member of the Institution of Engineering and Technology (IET). He was a recipient of the IET Student Project Award in 2003, the IEEE Communications Chapter Award in 2004, and multiple best paper awards. He has served on the technical program committees of numerous conferences in the fields of solid-state circuits, computer architecture, and machine learning.

Alexander M. Rush received the B.A. degree in computer science from Harvard University, Cambridge, MA, USA, in 2007, and the Ph.D. degree in computer science from the Massachusetts Institute of Technology, Cambridge, in 2014.

He was with Facebook artificial intelligence (AI) Research, New York, NY, USA, where he worked on natural language generation. He is currently an Associate Professor of computer science with the Cornell Ann S. Bowers College of Computing and Information Science, New York. His research is in the intersection of natural language processing, deep learning, and structured prediction with applications in text generation and efficient inference.



David Brooks (Fellow, IEEE) received the B.S. degree in electrical engineering from the University of Southern California, Los Angeles, CA, USA, in 1997, and the M.A. and Ph.D. degrees in electrical engineering from Princeton University, Princeton, NJ, USA, in 1999 and 2001, respectively.

He is currently the Haley Family Professor of computer science with the School of Engineering and Applied Sciences, Harvard University, Cambridge, MA, USA. His current research interests include resilient and power-efficient computer hardware and software design for high-performance and embedded systems.

Dr. Brooks was a recipient of several honors and awards, including the ACM Maurice Wilkes Award and ISCA Influential Paper Award.

Gu-Yeon Wei (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, USA, in 1994, 1997, and 2001, respectively.

He is currently a Robert and Suzanne Case Professor of electrical engineering and computer science with the Paulson School of Engineering and Applied Sciences (SEAS), Harvard University, Cambridge, MA, USA. His research interests span multiple layers of a computing system: mixed-signal integrated circuits, computer architecture, and design tools for efficient hardware. His research efforts focus on identifying synergistic opportunities across these layers to develop energy-efficient solutions for a broad range of systems from flapping-wing microrobots to machine learning hardware for the Internet of Things (IoT) devices to large-scale servers.