# ROSETTA: A Resource and Energy-Efficient Inference Processor for Recurrent Neural Networks Based on Programmable Data Formats and Fine Activation Pruning

Jiho Kim, *Student Member, IEEE,* and Tae-Hwan Kim, *Member, IEEE*

**Abstract**—Recurrent neural networks (RNNs) are extensively employed to perform inference based on the temporal features of the input data. However, their computational workload and power consumption involved in inference are prohibitively high in practice, which may be problematic to achieve a high-speed inference in devices with tight limitations in the available silicon resources and power supply. This paper presents an efficient inference processor for RNNs, named ROSETTA. ROSETTA supports multiple data formats programmable for each vector operand to achieve a wide range or high precision with a limited data size. ROSETTA consistently performs every vector operation based on homogeneous processing units with a high utilization rate. Moreover, ROSETTA skips operations and reduces memory accesses to achieve high energy efficiency by pruning the activation elements in a fine-grained manner. Implemented in a low-cost 28 nm field-programmable gate array, ROSETTA exhibits a resource and energy efficiency as high as 2.51 – 1.14 MOP/s/LUT and 434.01 – 113.29 GOP/s/W, respectively, while producing near-floating-point inference results. The resource and energy efficiency of ROSETTA are higher than those of the previous processor implemented in the same device by up to 206.1% and 304.0%, respectively. The functionality has been verified for several RNN models of various types under a fully-integrated inference system.

**Index Terms**—accelerator, recurrent neural networks, inference, field programmable gate array, microarchitecture.

## 1 Introduction

Recurrent neural networks (RNNs) are a class of artificial neural networks with recurrent dataflows formed by feedback connections [1], [2]. As the recurrent dataflows can realize stateful inference considering time dependencies immanent in the data, RNN inference is being employed as a key technology for applications, which primarily necessitate handling time-series data. Such applications include sequence classification [3], language modeling [4], and handwriting recognition [5]. However, in practice, RNN inference usually involves massive computational workload as well as prohibitive power consumption. This hinders efficient implementation of a fast inference process in miniature devices with tight limitations in the available silicon resources and power supply. Hence, we need a resource and energy-efficient processor to perform RNN inference in such devices.

Several researchers have presented efficient RNN inference processors. Guan *et al.* developed an inference processor for long short-term memory (LSTM) [2] using a high-level synthesis technique, where each data element is represented in a floating-point format [6]. Han *et al.* developed an efficient speech recognition engine based on an architecture-aware pruning technique to ensure a high utilization rate [7]. Cao *et al.* presented an efficient pruning technique to exploit the bank-balanced sparsity [8]. Wang *et al.* presented another pruning technique based on the block-circulant matrices and applied a Fourier-transform-based fast circulant convolution to realize efficient computations [9]. Bank-Tavakoli *et al.* presented an efficient architecture to perform the overall flow of LSTM inference in an overlapped manner [10]. Gao *et al.* modified the conventional gated recurrent unit (GRU) [11] to reduce computational workload and memory accesses and developed an efficient inference processor targeting edge-computing devices [12]. Azari *et al.* presented an approximation technique to reduce the multiplication complexity involved in RNN inference [13]. Francesco *et al.* presented a systolic-array-based architecture to achieve a scalability of an LSTM inference processor [14]. Jo *et al.* presented an approximate computing method for LSTM inference operations to realize energy-efficient speech recognition [15]. Deepak *et al.* presented an algorithm-hardware co-optimized memory compression technique to implement an efficient LSTM inference processor [16].

Although each of the aforementioned processors supports only one type of RNN models targeting a specific application, a few processors support various types of RNN models with reconfigurability. Based on the dataflow analysis, Kim *et al.* derived common primitive vector operations involved in RNN inference, developed an instruction-set processor specialized in performing them, and demonstrated the functionality for various RNN models [17]. Fowers *et al.* developed a single-instruction multiple-data (SIMD) processor to perform RNN inference in a large-scale field-programmable gate array (FPGA) and presented its performance for several RNN models [18]. Motivated

- *Jiho Kim and Tae-Hwan Kim are with the School of Electronics and Information Engineering, Korea Aerospace University, 76-3, Hanggongdaehak-ro, Deogyang-gu, Goyang-si, Gyeonggi-do, Republic of Korea, 10540.*
*E-mail: taehwan.kim@kau.kr*

TABLE 1
List of the previous inference processors supporting RNN models.

| Processor | (Demonstrated) Model type(s) | (Demonstrated) Application(s) | Implementation technology [a] |
|---|---|---|---|
| [6] | LSTM | Speech recognition | FPGA |
| [7] | (Peephole) LSTM | Speech recognition | FPGA |
| [8] | (Peephole) LSTM | Language modeling, speech recognition | FPGA |
| [9] | (Peephole/bidirectional) LSTM | Speech recognition | FPGA |
| [10] | LSTM | Sequence classification | FPGA |
| [12] | GRU | Speech recognition, regression | FPGA |
| [13] | LSTM | Language modeling | FPGA |
| [14] | (Peephole) LSTM | Speech recognition | ASIC |
| [15] | LSTM | Speech recognition | ASIC |
| [16] | LSTM | Speech recognition | ASIC |
| [17] | (Bidirectional) GRU, (peephole/bidirectional) LSTM | Language modeling, sequence classification | FPGA |
| [18] | LSTM, GRU, CNN | Speech recognition | FPGA |
| [19] | LSTM, CNN | Image classification, image captioning | ASIC |
| [20] | LSTM, CNN | Image captioning, video detection | FPGA |
| [21] | (Bidirectional) GRU | Machine translation | FPGA |

[a] FPGA and ASIC stand for the field-programmable gate array and application-specific integrated circuit, respectively.

by the fact that RNN models involve the fully-connected layers usually employed to implement the classification in convolutional neural networks (CNN), Shin *et al.* developed an efficient CNN-RNN processor based on the hardware-sharing technique [19]. Zeng *et al.* presented a reconfigurable system to perform CNN-RNN inference for general-purpose applications [20]. Li *et al.* implemented neural machine translation based on bi-directional GRU with attention mechanism using a high-level synthesis technique [21]. Table 1 lists the previous processors mentioned in this section in summary.

This paper presents an efficient RNN inference processor based on programmable data formats and fine activation pruning, named ROSETTA. The contributions of this paper are listed as follows:

- An efficient instruction-set architecture specialized in performing RNN inference has been proposed. The proposed instruction-set architecture supports multiple data formats to realize a wide range or high precision in spite of the small data size.
- A resource-efficient microarchitecture has been proposed. The proposed microarchitecture has been designed to perform every vector operation consistently based on homogeneous processing units (PUs) with a high utilization rate.
- An activation pruning scheme has been proposed, based on which the pruning information can be exploited to skip operations and reduce memory accesses with the aim of achieving a high energy efficiency.
- An RNN inference processor, ROSETTA, has been implemented based on the ideas above in a low-cost 28 nm FPGA. Its functionality has been verified exhaustively for several practical models under a fully-integrated system. The resource and energy efficiency of ROSETTA are 2.51 – 1.14 MOP/s/LUT and 434.01 – 113.29 GOP/s/W, respectively.

The rest of the paper is organized as follows. Section 2 explains the background and motivation of this work. Section 3 presents the architecture of ROSETTA in detail. Section 4 shows the implementation results and evaluates them. Section 5 discusses the novelty of ROSETTA as compared with the previous related work. Finally, Section 6 draws the conclusion.

## 2 Background and Motivation

RNN inference is performed through multiple timesteps, where the dataflow for each timestep has feedback connections through which the activations for the current timestep may affect those in the next timestep. Such recurrent dataflow imposes the states on the models, enabling inference to be performed finding the temporal features of the input. There are various types of RNN models [1], [11], [22], [23], [24]; the dataflows of two popular models are illustrated in Fig. 1. In the figure, the edges represent the activation or parameter vectors. The vectors produced from the gates, such as the states, and those generated inside the gates are called activation vectors [7].

The dataflows vary depending on the model types; however, they can be built of the three kinds of the primitive vector operations [17], namely, the elementwise multiply-accumulate (EMAC), elementwise non-linear activation function (ENOF), and matrix-vector multiply-accumulate (MVMA) operations. In addition, there are data dependencies between some of the vector operations; for example, in the forget gate of LSTM, the ENOF operation is performed for the results from the MVMA operation. Suppose the processor is designed by incorporating multiple different PUs for the vector operations. In that case, some of them may not work due to the data dependencies inherent in the dataflows, leading to a low resource efficiency. Such architectural consideration by the dataflow analysis provides ROSETTA with the motivation to be designed based on homogeneous PUs.

The data distributions in the dataflows of practical models have interesting characteristics. Fig. 1 illustrates the distributions of the data of some activation and parameter

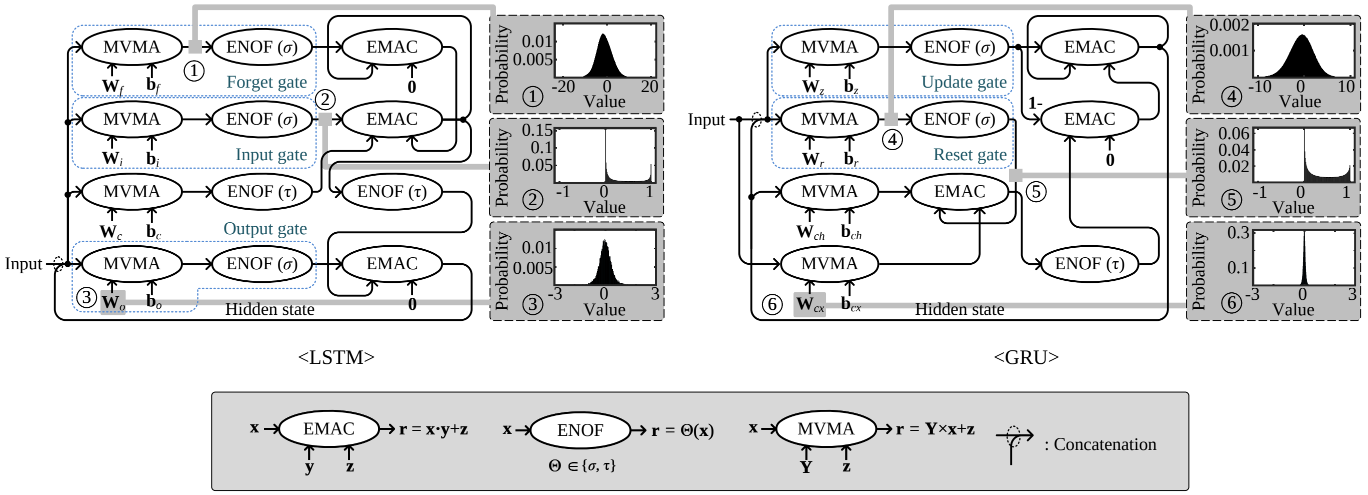


Fig. 1. Inference dataflows of RNN models. The data distributions have been obtained for the practical models to perform word-level Penn Treebank task [25] in 64 timesteps, where the sizes of the input and states are 128 and 64, respectively. The operators $\cdot$ and $\times$ represent the elementwise and general multiplications, respectively. $\sigma$ and $\tau$ represent the elementwise sigmoid and hyperbolic tangent activation functions, respectively. $\mathbf{W}$ and $\mathbf{b}$ denote the weight and bias, respectively, with the subscripts to distinguish the parameters with respect to the gates ($f$: forget gate, $i$: input gate, $c$: cell gate, $o$: output gate, $z$: update gate, $r$: reset gate, $cx$: cell gate for the input, $ch$: cell gate for the state). The dimensions of $\mathbf{W}_{cx}$ and $\mathbf{W}_{ch}$ are $\beta \times \alpha$ and $\beta \times \beta$, respectively, while those of the others are $\beta \times (\alpha + \beta)$, where $\alpha$ and $\beta$ denote the sizes of the input and state(s), respectively.

vectors, which have been collected by performing the practical inference task [25]. First, it can be seen that the data distribution varies considerably depending on the vector. For example, the data of the input activation vectors of the ENOF operations are distributed in a wider range than those of the output vectors from the operations. Hence, the data format for the former ones (e.g., ① and ④ in Fig. 1) needs to be capable of representing the data in a wider range, whereas that for the latter ones (e.g., ② and ⑤ in Fig. 1) needs to be capable of representing the data with a high precision. In addition, the parameter data may have different distributions for the model types (e.g., ③ and ⑥ in Fig. 1). Therefore, it would be inefficient to represent every data using a single format. Second, most of the activation data have small magnitudes. If the activation data with small magnitudes do not significantly affect the inference results, they can be effectively pruned to be excluded in subsequent operations. Such observations regarding the data distributions provide ROSETTA with the motivation to support multiple data formats and activation pruning.

## 3 Proposed Processor: ROSETTA

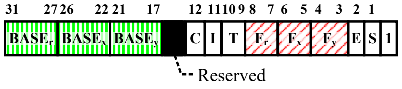
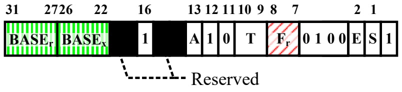
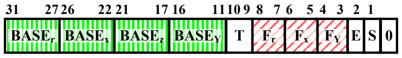
### 3.1 Instruction-Set Architecture Supporting Multiple Data Formats

ROSETTA is an instruction-set processor, for which the instruction set has been devised to efficiently perform vector operations by extending our preliminary work [26]. Table 2 lists the supported instructions, describing their behaviors. Each instruction corresponds to one of the primitive vector operations involved in the dataflows of RNN models, where the result as well as operands can be considered vectors. Three memories are accommodated for vector storages; they are named after the data stored: activation memory (AM), weight memory (WM), and bias memory (BM). As described in Table 2, the vectors are fetched from and written to the memories directly while executing the instructions, for which the addresses are calculated effectively from the bases given by the related fields (the green-striped fields shown in Table 2). The dimensions of the vectors are determined according to the sizes of the input and state(s) of an RNN model, given by the configurable parameters $\alpha$ and $\beta$, respectively. Each instruction has delay slots that can be filled with bubbles if the S field is set. The instruction whose E field is set finalizes the dataflow. ROSETTA executes the vector operations one after another according to the sequence of the instructions stored in the instruction memory (IM). The sequence of the instructions is programmed before performing inference by scheduling the vector operations composing an inference dataflow with the consideration of the data dependencies between them.

ROSETTA supports multiple data formats. The supported data formats are S1.7, S3.5, and U0.8, where S$m,n$ and U$m,n$ stand for the 2's complement and unsigned fixed-point numbers, respectively, consisting of an $m$-bit integer and $n$-bit fraction parts. The size of the integer part differs depending on the data format; however, the data size is made fixed regardless of the format for hardware regularity. The programmability of the data formats is provided per vector operand by the related fields in the instructions (the red-hatched fields shown in Table 2). The data formats need to be selected carefully considering the data distributions in practice (e.g., the data distributions obtained for the training dataset) to realize a wide range or high precision, which might be attained costly by a single data format (e.g., floating-point format).

ROSETTA successfully produces near-floating-point inference results for several practical tasks based on various

TABLE 2
Supported instructions in ROSETTA.

| Instruction | Format | Behavior |
|---|---|---|
| EMAC |  31 27 26 22 21 17 12 11 10 9 8 7 6 5 4 3 2 1<br>BASE$_r$<br>BASE$_x$<br>BASE$_y$<br>C<br>I<br>T<br>F$_r$<br>F$_x$<br>F$_y$<br>E<br>S<br>1<br>Reserved | Performs the EMAC operation shown in Fig. 1: $\mathbf{r} = \mathbf{x} \cdot \mathbf{y} + \mathbf{z}$, where $\mathbf{x}$ is bitwise-inverted before the multiplication if I is set. The two vectors, $\mathbf{x}$ and $\mathbf{y}$, whose dimensions are $\beta$, are read from AM at the addresses given by BASE$_\mathbf{x}$ and BASE$_\mathbf{y}$, respectively. The data formats of the elements of the two vectors are indicated by F$_\mathbf{x}$ and F$_\mathbf{y}$, respectively. The resulting vector, $\mathbf{r}$, is accumulated to the vector in AM at the address given by BASE$_\mathbf{r}$ if C is set; otherwise, it is written there. The data format of each element of the resulting vector is adapted to F$_\mathbf{r}$, considering F$_\mathbf{x}$ and F$_\mathbf{y}$. T can be used to prune the elements in the resulting vector. |
| ENOF |  31 27 26 22 16 13 12 11 10 9 8 7 2 1<br>BASE$_r$<br>BASE$_x$<br>1<br>A<br>1<br>0<br>T<br>F$_r$<br>0 1 0 0<br>E<br>S<br>1<br>Reserved | Performs the ENOF operation shown in Fig. 1: $\mathbf{r} = \Theta(\mathbf{x})$, where $\Theta$ is the elementwise activation function of which type is the sigmoid if A is set; otherwise the hyperbolic tangent. The activation vector, $\mathbf{x}$, whose dimension is $\beta$, is read from AM at the address given by BASE$_\mathbf{x}$. The resulting vector, $\mathbf{r}$, is written to AM at the address given by BASE$_\mathbf{r}$. The data format of each element of the resulting vector is adapted to F$_\mathbf{r}$. T can be used to prune the elements in the resulting vector. |
| MVMA |  31 27 26 22 21 17 16 11 10 9 8 7 6 5 4 3 2 1<br>BASE$_r$<br>BASE$_x$<br>BASE$_z$<br>BASE$_Y$<br>T<br>F$_r$<br>F$_x$<br>F$_Y$<br>E<br>S<br>0 | Performs the MVMA operation shown in Fig. 1: $\mathbf{r} = \mathbf{Y} \times \mathbf{x} + \mathbf{z}$. The weight matrix, $\mathbf{Y}$, whose dimension is $\beta \times (\alpha + \beta)$, is read from WM at the address given by BASE$_\mathbf{Y}$. The activation vector, $\mathbf{x}$, whose dimension is $\alpha + \beta$, is read from AM at the address given by BASE$_\mathbf{x}$. The bias vector, $\mathbf{z}$, whose dimension is $\beta$, is read from BM at the address given by BASE$_\mathbf{z}$. The data formats of the elements of the weight matrix and activation vector are indicated by F$_\mathbf{Y}$ and F$_\mathbf{x}$, respectively. The resulting vector, $\mathbf{r}$, whose dimension is $\beta$, is written to AM at the address given by BASE$_\mathbf{r}$. The data format of each element of the resulting vector is adapted to F$_\mathbf{r}$, considering F$_\mathbf{Y}$ and F$_\mathbf{x}$. T can be used to prune the elements in the resulting vector. |

RNN models by supporting the data formats stated above, as will be shown in Section 4.2. Furthermore, its data size is much smaller than those of the previous studies based on a single data format [8], [9], [17], [30]. It might be beneficial to support more or larger data formats to reduce the difference of the inference results from those of the floating-point data; however, this would increase not only the circuit complexity involved in the datapath but also the memories for the data storages. This is not suitable to achieve a low resource usage and low power.

### 3.2 Resource-Efficient Microarchitecture

ROSETTA has been designed based on a SIMD processing pipeline. Fig. 2 shows the overall architecture.[†] Each instruction is fetched from IM and issued into the pipeline to perform the corresponding vector operation. The pipeline has been designed to perform scalar operations in parallel for 64 lanes of the vectors per cycle. A vector operation may thus be decomposed into the suboperations with the operands of 64 lanes and performed through the pipeline. The pruned activation memory (PAM) is a small memory with the address space equivalent to the address space of AM and it stores the pruning information represented by a single bit for each element of the activation vectors. PAM is first queried to find whether each element of the activation vector operands had been pruned, and then the operands except the pruned ones are read from the other memories. The leading-zero counting unit (LZCU) is used to locate the non-pruned elements. The activation coefficient unit (ACU) finds the coefficients used to evaluate the activation functions as in our previous work [17]. Vector processing unit (VPU) computes out the resulting activation vector, possibly pruning its elements. Finally, the resulting activation vector and its pruning information are written to AM and PAM, respectively.

VPU has been designed with the focus on performing the EMAC operations efficiently. Fig. 3 shows the architecture of VPU. VPU is the array of homogeneous PUs, where each PU performs the operations of multiply-accumulate (MAC), format adaption, and pruning for a vector lane. Each PU is pipelined in two stages. The first stage multiplies the operands and pre-aligns the (additive) operand by shifting it to be accumulated in the next stage. The second stage accumulates the product into the register, where the accumulated result is represented in S8.24 without regard to the data formats of the operands. The format adaptation unit (FAU) adapts the accumulated result to the resulting data format by shifting and saturating it. The shifting part adapts the radix point of the accumulated result to the resulting data format. The saturating part saturates the result to the maximum or minimum value that can be represented by the resulting data format on detecting overflow or underflow conditions, respectively. The shift amounts in the pre-alignment and format adaptation are determined by considering the data formats given by the related fields in the instructions and shared by the PUs. The pruning unit (PRU) calculates the pruning information of the result, which will be delineated in Section 3.3.

ROSETTA performs vector operations consistently on the basis of the EMAC operations. As illustrated in Fig. 4, the ENOF operation is performed by the EMAC operation based on the linear splines with the slopes and offsets obtained by ACU [17]; the MVMA operation is performed by decomposing it into several EMAC operations for the column vectors of the weight matrix. This explains why VPU has been designed to be specialized in performing the EMAC operations by the array of the homogeneous PUs. The previous processors incorporate several heterogeneous PUs for vector processing, each of which is dedicated to

[†] The design of ROSETTA is available at http://abit.ly/rosetta.

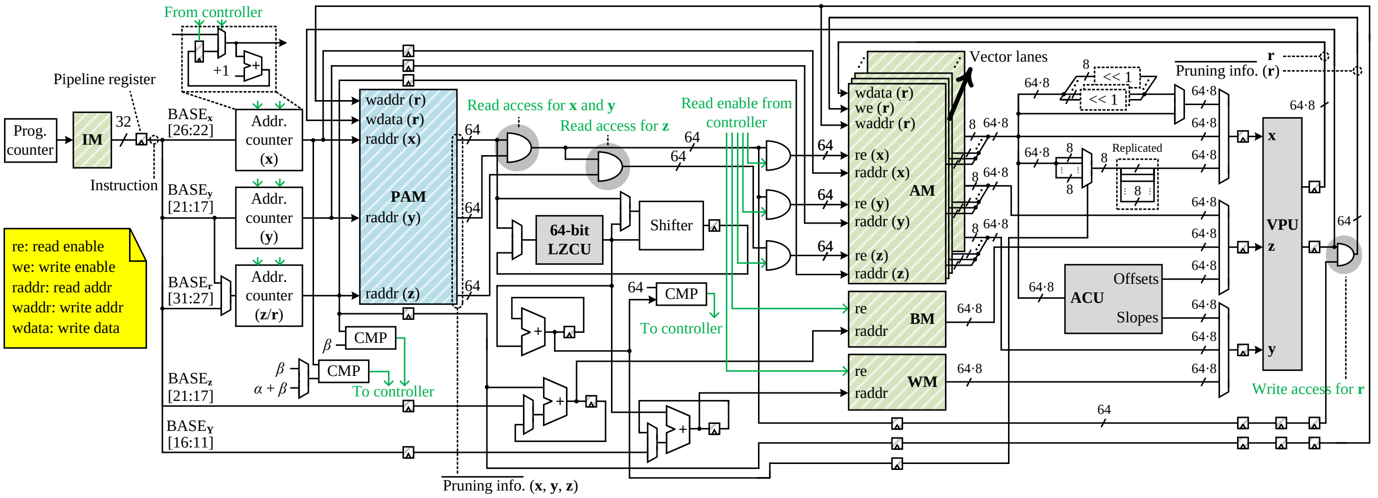


Fig. 2. Overall processing pipeline of ROSETTA, where ≪ represents the bit-wise left shift operation and the brackets with indexes or index ranges specify the parts of the signals. The clock of each memory is gated selectively by the read and/or write enable signals for a low power consumption.

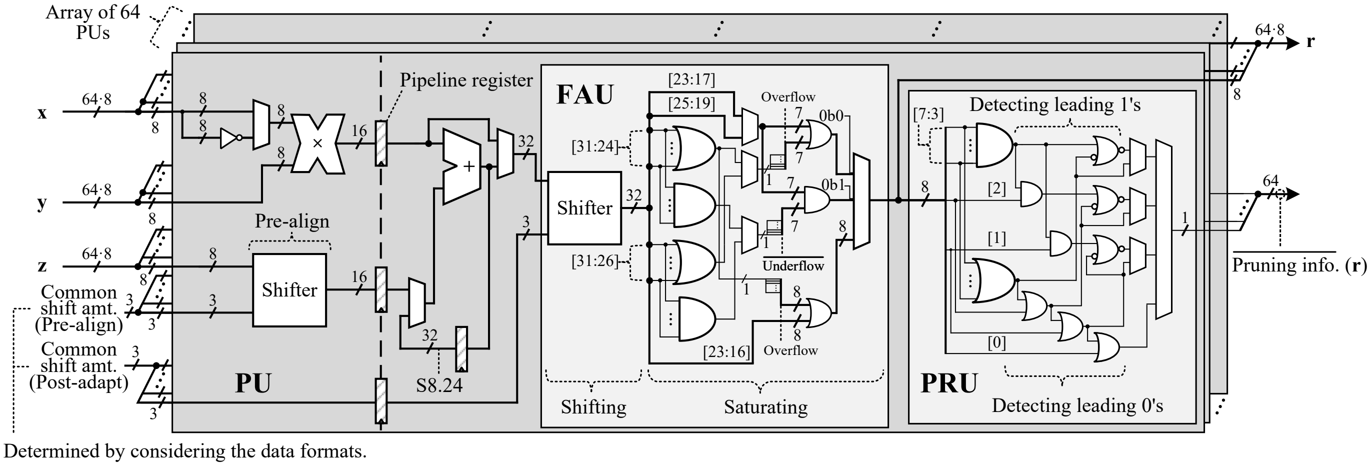


Fig. 3. Microarchitecture of VPU, where 0b is the prefix to stand for a binary number.

a particular operation [7], [8], [12], [13], [27]. This is not efficient in utilizing the hardware resource because some of the PUs sometimes may not be utilized inevitably because of the data dependency in the dataflow through the timesteps. ROSETTA performs every vector operation by fully utilizing the PUs, thereby achieving high resource efficiency. It should be remarked that the homogeneity of the PUs has become feasible as the data are represented in the fixed size irrespective of what they represent.

Additional description of the other miscellaneous components are followed:

- LZCU has been designed based on the divide-and-conquer technique presented in [28]. Let the leading-zero count of an $N$-bit vector $x$ denoted by a $(1+\log n)$-bit vector $l$. $x$ is partitioned into two disjoint sub-vectors of $N/2$ bits, $x_{\mathrm{HI}}$ and $x_{\mathrm{LO}}$, where $x = \{x_{\mathrm{HI}}, x_{\mathrm{LO}}\}$ and $\{\cdot\}$ concatenates the vectors. Given the leading-zero counts of $x_{\mathrm{HI}}$ and $x_{\mathrm{LO}}$, which $l_{\mathrm{HI}}$ and $l_{\mathrm{LO}}$ denote, respectively, $l$ can be obtained recursively as expressed in (1). The recursion is initialized with the base: $l = 1$ for $x = 0\mathrm{b}1$ and $l = 0$ for $x = 0\mathrm{b}0$. Fig. 5 shows the microarchitecture of the $n$-bit LZCU. The leading-zero count of $x$ is calculated with those of the two disjoint parts of $x$, which are calculated by the two sub LZCUs, based on the recursion expressed in (1). LZCU in ROSETTA has been implemented by instantiating the architecture shown in Fig. 5 with $n = 64$.
- ACU is the array of subunits, where each subunit finds the coefficients of the linear spline to evaluate the non-linear activation function for a vector lane. The microarchitecture of ACU is shown in Fig. 6. In a subunit of ACU, let the slope and offset of the spline to evaluate the sigmoid function of $x$ denoted by $\psi_\sigma(x)$ and $\phi_\sigma(x)$, respectively; and those to evaluate the hyperbolic tangent function of $x$ by $\psi_\tau(x)$ and $\phi_\tau(x)$, respectively. ACU looks up the sigmoid tables to find $\psi_\sigma(x)$ and $\phi_\sigma(x)$ with the most significant bits of $x$. ACU has been designed by incorporating only sigmoid tables, which are also utilized to find the spline coefficients to evaluate the hyperbolic tangent function. This design is based on the technique we proposed in [17]; to be specific, $\psi_\tau(x)$ and $\phi_\tau(x)$ can be obtained by $2\psi_\sigma(2x)$

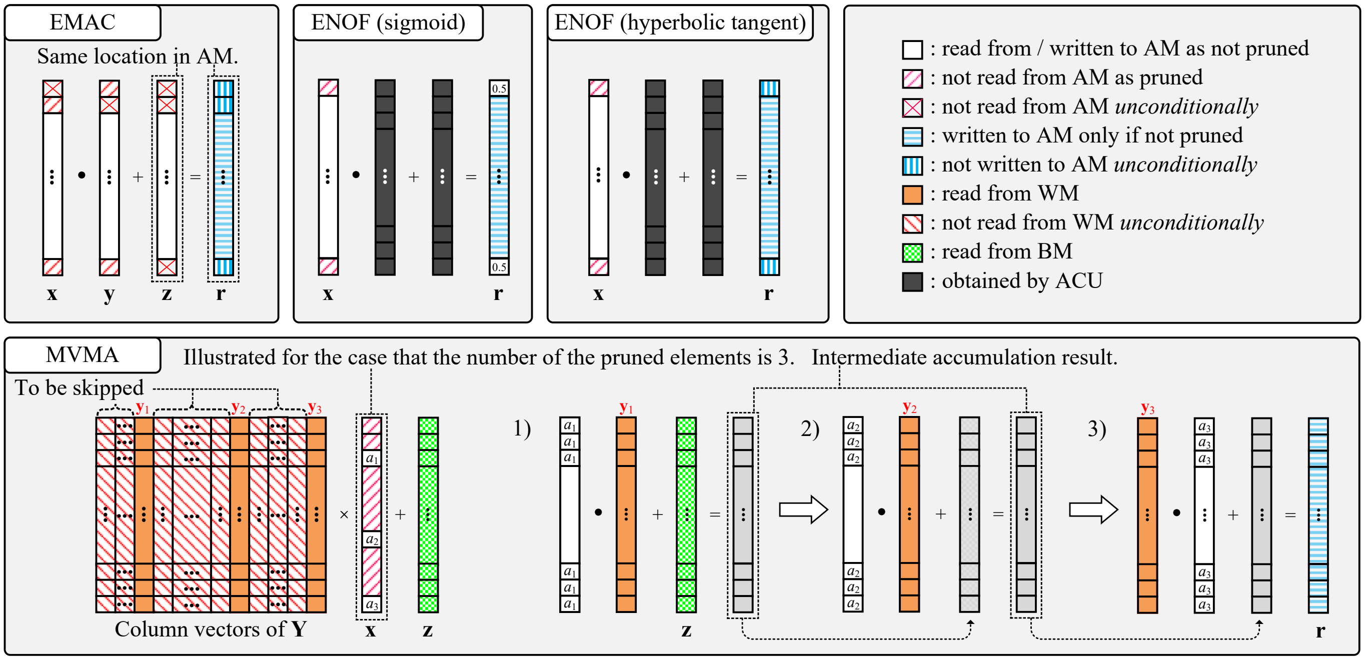

EMAC
Same location in AM.
x
y
z
r
ENOF (sigmoid)
0.5
x
r
ENOF (hyperbolic tangent)
x
r
□ : read from / written to AM as not pruned
▨ : not read from AM as pruned
☒ : not read from AM unconditionally
▤ : written to AM only if not pruned
▥ : not written to AM unconditionally
🟧 : read from WM
☒ : not read from WM unconditionally
🟩 : read from BM
■ : obtained by ACU
MVMA
Illustrated for the case that the number of the pruned elements is 3.
Intermediate accumulation result.
To be skipped
$y_1$
$y_2$
$y_3$
$a_1$
$a_2$
$a_3$
Column vectors of Y
x
z
1)
2)
3)
r

Fig. 4. Mechanisms to perform the vector operations in ROSETTA with the activation vector operand(s) whose pruning information is as shown.

and $2\psi_\sigma(2x) - 1$, respectively, since the hyperbolic function of $x$, $\tau(x)$ can be calculated by $2\sigma(2x) - 1$.

### 3.3 Fine Activation Pruning

ROSETTA may prune the activation elements while performing inference to achieve high energy efficiency. The pruning is performed for the activation vectors with thresholds differently programmable for each vector rather than with one threshold consistently used, so called the fine activation pruning. This section describes how the pruning is performed and details how the pruning information is exploited to skip operations and reduce memory accesses in executing each instruction, showing how a high energy efficiency can be attained.

Each element of the activation vectors is pruned if its magnitude is small. PRU decides if an element of the resulting activation vector will be pruned. More specifically, the element, whose value is $r$, will be pruned if it satisfies the condition:

$$\begin{aligned} r = 0 \quad & \text{for} \quad T = 0, \\ -\epsilon 2^T \le r < \epsilon 2^T \quad & \text{for} \quad T \neq 0, \end{aligned} \tag{2}$$

where $\epsilon$ denotes the unit in the least position for the resulting data format and $T$, which is given by the related field in the instruction (the T fields shown in Table 2), can be 0, 1, 2, or 3. In the pruning condition, $T$ is the parameter to control the pruning amount. When programmed so that $T$ is 0, the pruning amount is least; only zero elements of the activation vectors will be pruned and the inference results are not affected at all. As shown in Fig. 3, PRU has been designed to make such pruning decision independently of the size of the integer part of the data format. Instead of the general comparisons, PRU detects the leading zeros and ones to check the pruning condition by considering whether the data format is signed or not. If an element is determined to be pruned, its value will not be written to AM, nor read from AM subsequently.

In executing each instruction, ROSETTA first reads the pruning information of the operands and exploits it to skip operations and reduce memory accesses, as follows:

- EMAC: For each lane, if at least one element of the multiplicative operands (in $\mathbf{x}$ and $\mathbf{y}$) had been pruned, not only the pruned element but also the others are not read from AM even if they had not been pruned, and the result (in $\mathbf{r}$) is not written to AM, either. The rationale can be explained as follows: since the pruned element of the multiplicative operands has a small magnitude in fact, the product would have a small magnitude, too, affecting the result insignificantly. Fig. 4 illustrates an EMAC operation, where the elements in the first two lanes and the last lane of the vectors ($\mathbf{x}, \mathbf{y}, \mathbf{z}$, and $\mathbf{r}$) are not read from or written to AM as there are at least one element of the multiplicative operands (in $\mathbf{x}$ and $\mathbf{y}$) had been pruned for each of the lanes. AM has been designed based on the multiple banks as shown in Fig. 2, where each bank stores the elements for a certain lane of the vectors, and its access control is made independently of those for other banks. The read and

$$\begin{array}{lll} 0\text{b}10\cdots0 & \text{for} & l_{\text{HI}}\left[\log n - 1\right] = 0\text{b}1 \text{ and } l_{\text{LO}}\left[\log n - 1\right] = 0\text{b}1, \\ \{0\text{b}01, l_{\text{HI}}[\log n - 2:0]\} & \text{for} & l_{\text{HI}}\left[\log n - 1\right] = 0\text{b}1 \text{ and } l_{\text{LO}}\left[\log n - 1\right] = 0\text{b}0, \\ \{0\text{b}00, l_{\text{LO}}[\log n - 2:0]\} & \text{for} & l_{\text{HI}}\left[\log n - 1\right] = 0\text{b}0, \end{array} \tag{1}$$

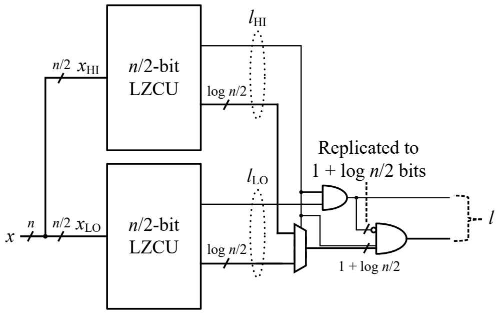


Fig. 5. Microarchitecture of the $n$-bit LZCU.

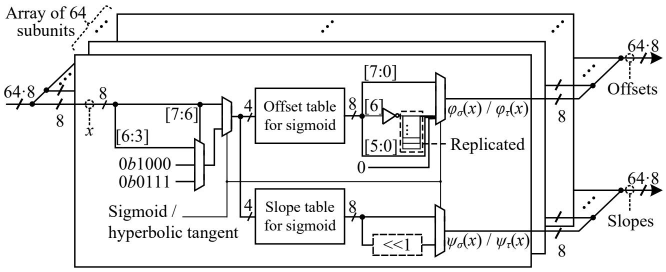


Fig. 6. Microarchitecture of ACU, which has been adapted from that presented in [17] considering the data format.

write enable signals for each bank combine the pruning information (those read from PAM for the read enable signals and that calculated from VPU for the write enable signal) and other control signals as highlighted in Fig. 2.

- ENOF: For each lane, the result (in $\mathbf{r}$) is determined without reading the operand (in $\mathbf{x}$) if the element had been pruned. The results of the hyperbolic tangent and sigmoid activation functions for the pruned element are immediately approximated to $0$ and $0.5$, respectively. The rationale can be explained as follows: the results of the hyperbolic tangent ($\triangleq (e^{2x}-1)/(e^{2x}+1)$) and sigmoid activation functions ($\triangleq 1/(1+e^{-x})$) are $0$ and $0.5$ for $x=0$. If an element has been pruned, its magnitude can be considered small, for which the activation function results can be approximated so as if it were a zero. Fig. 4 illustrates ENOF operations, where the operands in the first lane and the last lane had been pruned and the results are determined without reading the operands from AM.
- MVMA: If some elements of the activation vector operand ($\mathbf{x}$) had been pruned, the accumulations of the vectors scaled by the pruned elements are skipped. The rationale is that the vectors scaled by the pruned elements would not contribute significantly to the accumulation result because the pruned elements have small magnitudes in fact. Fig. 4 illustrates an MVMA operation with the three non-pruned activation elements (in $\mathbf{x}$), $a_1$, $a_2$, and $a_3$, where the vector operations with the other pruned elements are skipped effectively. To skip the operations with the pruned elements, ROSETTA seeks the non-pruned elements by LZCU, which counts the leading zeros of the pruning information as shown in Fig. 2.

ROSETTA has been designed based on the SIMD pipeline, through which 64 scalar operations are performed in parallel in order of querying the pruned information, fetching data, computing, and writing the results, as described in Section 3.2. For simplicity and regularity, the pipeline control regarding data processing is fulfilled not independently for each SIMD lane but commonly for all the SIMD lanes, except for those related to the AM accesses. This is also because ROSETTA does not assume any structured patterns of the pruning information. Therefore, no operations are skipped in executing EMAC and ENOF instructions, though the AM accesses can be reduced by exploiting the pruning information. In executing MVMA instructions, they are performed by being decomposed into several (sub) vector operations scaled by the activation elements, as illustrated in Fig. 4. If there are any pruned activation elements, the entire scalar operations of the corresponding (sub) vector operations are skipped. The related memory accesses of WM and AM are thus eliminated effectively.

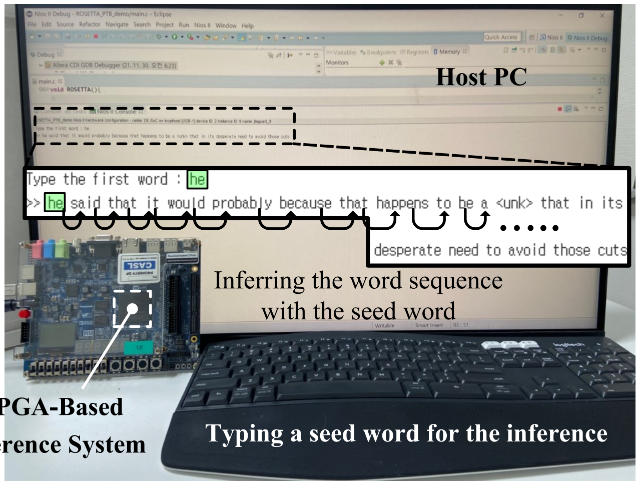
Exploiting the pruning information as described above may improve energy efficiency. The energy efficiency is defined by the inference speed attributed to the unit power consumption. Skipping operations increases the inference speed, and reducing memory accesses lowers the dynamic power consumption. The energy efficiency can be thus improved by the pruning as the additional hardware to perform the pruning has marginal effects on the energy efficiency. This may however affect the inference results if non-zero elements are pruned for $T \neq 0$. Section 4.2 will investigate the energy efficiency and inference performance of ROSETTA thoroughly for several practical tasks with various pruning configurations in order to demonstrate the validity of this technique.

The major additional components required to implement the activation pruning are the PRUs and PAM. Each PRU has been designed based on the low-complexity microarchitecture, which performs the pruning decision by detecting leading-bit patterns as described earlier in this subsection, and PAM has been designed based on the registers as will be described in Section 3.4, which are more efficient in terms of the power consumption than BRAMs. As a result, although the activation pruning brings about considerable amounts in the operation skipping and memory access reduction, its hardware overhead is not substantial: the overall logic resource usage and power consumption have been increased by 5.4% and 7.9%, respectively, to implement the activation pruning.

### 3.4 Prototype Inference System

A prototype RNN inference system has been developed in an FPGA to verify the functionality of ROSETTA by integrating all the essential components required to perform practical tasks. Fig. 7. shows the overall architecture of the inference system. The inference core has been implemented based on the architecture shown in Fig. 2. AM, WM, BM, and IM have been implemented based on the block random-access memories (BRAMs), whereas PAM has been implemented based on the registers. AM has a multi-bank structure in which the access of each bank can be controlled individually. The access router dynamically forwards the data requests to the internal BRAM instances of AM, realizing such a high bandwidth that is required for ROSETTA to work without pipeline stalls. The control and status registers (CSRs) include the registers storing the model dimensions, determined by $\alpha$ and $\beta$. AM, WM, BM, IM, PAM, and CSRs are mapped to the address space of the microcontroller unit (MCU).

MCU fulfills the overall flow by controlling the components of the system as follows. The parameter and instruction data, which are initially stored in the external SDRAM, are loaded into WM, BM, and IM, respectively, for their types; ROSETTA does not directly access the data stored in the external SDRAM while performing inference. For each timestep of the inference tasks, the input is loaded into AM; ROSETTA executes the instructions with the input and state stored in AM, where the state was calculated in the previous timestep. MCU reads the inference results from ROSETTA and transmits the results to the host PC through the JTAG UART at the baud rate of 115200 without parity bits. The communication with the PC occurs only after the inference flow and is not part of the inference flow.


Host PC
Inferring the word sequence
with the seed word
FPGA-Based
Inference System
Typing a seed word for the inference

```
Type the first word : he
>> he said that it would probably because that happens to be a <unk> that in its
desperate need to avoid those cuts
```

Fig. 8. Verification environment setup.

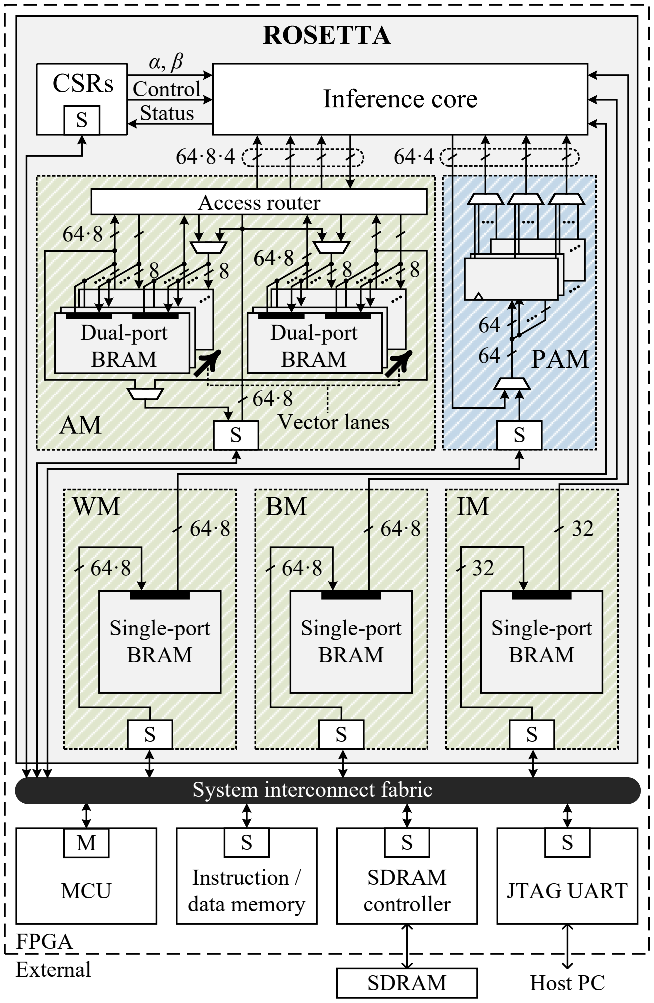


Fig. 7. Overall architecture of the prototype inference system, where S and M stand for a slave and master interface, respectively.

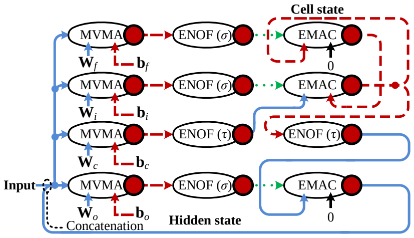
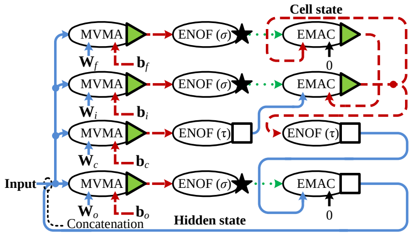
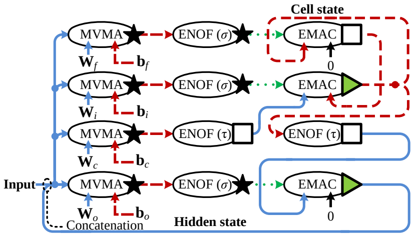
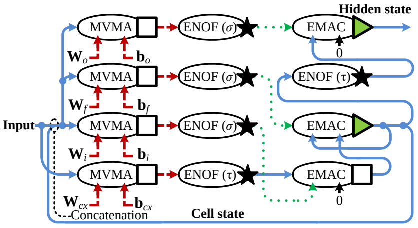
## 4 Results and Evaluation

### 4.1 Implementation Results

The entire RNN inference system with ROSETTA integrated has been implemented by synthesizing it to fit in a low-cost FPGA with tight limitations in the available resources. The synthesis has been conducted using Intel® Quartus® Prime v20.1, targeting Intel® Cyclone®V FPGA, whose part number is 5CSXFC6D6. The entire system has been successfully fitted in the device using the resources of 11.5K adaptive logic modules (ALMs), 10.1K-bit registers, 64 digital signal processing blocks (DSPs), and 1814K-bit BRAMs. The resource usage of ROSETTA itself is 8.6K ALMs, 5.3K-bit registers, 64 DSPs, and 800K-bit BRAMs, where the DSPs have been used to implement the multipliers and accumulators in the PUs, and BRAMs have been used to implement AM, BM, WM, and IM. At the maximum operating frequency, estimated to 124MHz under the slow condition at 85°C with a 1.1 V supply, the theoretical inference speed is 15.9 GOP/s. The theoretical speed has been derived by multiplying the operating frequency by the count of the operations, which the microarchitecture is capable of processing per cycle, counting each of the multiplication and addition operations by 1 OP. It is worth noting that the practical speed may effectively be significantly higher than the theoretical speed by the activation pruning, as will be investigated below.

The functionality of ROSETTA has been verified for practical inference tasks with various RNN models. Tables 3 and 4 list the RNN models used to conduct the functional verification for the two tasks: the word-level Penn Treebank (PTB) task [25] in 64 timesteps and the sequential MNIST task [33] in 16 timesteps. The bi-directional LSTM in Table 4 is composed of multiple layers for the recurrent dataflows in both of the forward and backward directions. Fig. 8 shows the setup of the verification environment, wherein ROSETTA is successfully inferring the word sequence.

TABLE 3
Performance of the word-level Penn Tree Bank task in 64 timesteps.

| Model [a] | Data format & pruning configurations [b] | Op. skip. (%) | Eff. infer. speed (GOP/s) | Mem. acc. reduc. (%) | Pow. cons. (mW) | Energy eff. (GOP/s/W) | Perplexity [c] |
|---|---|---|---|---|---|---|---|
| LSTM<br>($\alpha = 64, \beta = 128$,<br>198.91 KOP/timestep) |  Cell state<br>MVMA<br>ENOF (σ)<br>EMAC<br>$\mathbf{W}_f$<br>$\mathbf{b}_f$<br>0<br>MVMA<br>ENOF (σ)<br>EMAC<br>$\mathbf{W}_i$<br>$\mathbf{b}_i$<br>MVMA<br>ENOF (τ)<br>ENOF (τ)<br>$\mathbf{W}_c$<br>$\mathbf{b}_c$<br>Input<br>MVMA<br>ENOF (σ)<br>EMAC<br>$\mathbf{W}_o$<br>$\mathbf{b}_o$<br>Hidden state<br>0<br>Concatenation | 24.87 | 21.18 | 39.06 | 107.98 | 196.20 | 99.50 (0.85) |
| |  Cell state<br>MVMA<br>ENOF (σ)<br>EMAC<br>$\mathbf{W}_f$<br>$\mathbf{b}_f$<br>0<br>MVMA<br>ENOF (σ)<br>EMAC<br>$\mathbf{W}_i$<br>$\mathbf{b}_i$<br>MVMA<br>ENOF (τ)<br>ENOF (τ)<br>$\mathbf{W}_c$<br>$\mathbf{b}_c$<br>Input<br>MVMA<br>ENOF (σ)<br>EMAC<br>$\mathbf{W}_o$<br>$\mathbf{b}_o$<br>Hidden state<br>0<br>Concatenation | 38.91 | 26.06 | 47.19 | 94.43 | 275.93 | 100.44 (1.79) |
| |  Cell state<br>MVMA<br>ENOF (σ)<br>EMAC<br>$\mathbf{W}_f$<br>$\mathbf{b}_f$<br>0<br>MVMA<br>ENOF (σ)<br>EMAC<br>$\mathbf{W}_i$<br>$\mathbf{b}_i$<br>MVMA<br>ENOF (τ)<br>ENOF (τ)<br>$\mathbf{W}_c$<br>$\mathbf{b}_c$<br>Input<br>MVMA<br>ENOF (σ)<br>EMAC<br>$\mathbf{W}_o$<br>$\mathbf{b}_o$<br>Hidden state<br>0<br>Concatenation | 49.16 | 31.30 | 50.84 | 88.88 | 352.22 | 101.89 (3.24) |
| Peephole LSTM [23]<br>($\alpha = 64, \beta = 128$,<br>166.14 KOP/timestep) |  Hidden state<br>MVMA<br>ENOF (σ)<br>EMAC<br>$\mathbf{W}_o$<br>$\mathbf{b}_o$<br>0<br>MVMA<br>ENOF (σ)<br>ENOF (τ)<br>$\mathbf{W}_f$<br>$\mathbf{b}_f$<br>Input<br>MVMA<br>ENOF (σ)<br>EMAC<br>$\mathbf{W}_i$<br>$\mathbf{b}_i$<br>MVMA<br>ENOF (τ)<br>EMAC<br>$\mathbf{W}_{cx}$<br>$\mathbf{b}_{cx}$<br>0<br>Concatenation<br>Cell state | 49.53 | 31.55 | 48.34 | 91.59 | 344.43 | 103.02 (3.89) |

[a] $\alpha$ and $\beta$ denote the sizes of the input and state(s), respectively.

[b] The blue solid, red dashed, and green dotted lines stand for the data formats of S1.7, S3.5, and U0.8, respectively. The circle, triangle, rectangle, and star symbols attached after the vector operations stand for the pruning with $T = 0, 1, 2,$ and, $3$, respectively. $\sigma$ and $\tau$ are the sigmoid and hyperbolic tangent functions, respectively.

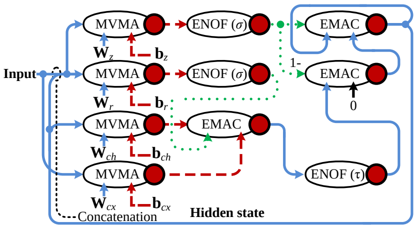
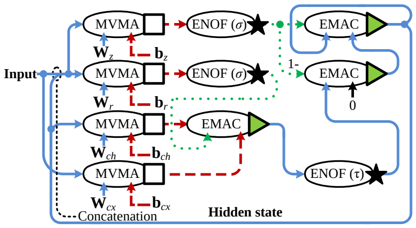
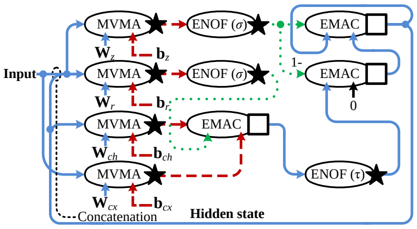
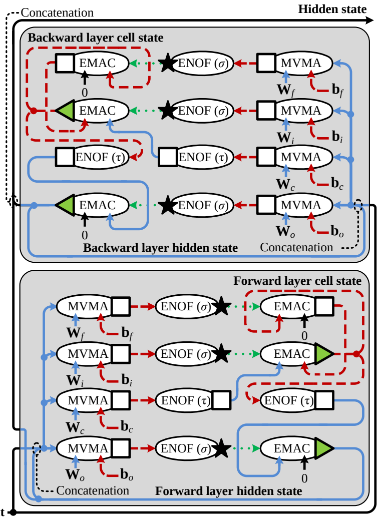
[c] The number inside the parentheses is the absolute difference from the single-precision floating-point result, obtained based on each model without any pruning.

### 4.2 Performance Analysis

The performance of ROSETTA has been analyzed elaborately for the practical inference tasks. The practical counts with respect to the operating cycles and memory accesses have been obtained along with the inference performance by performing inference for the entire test data sets under the *cycle-accurate* simulator. The rates of the operation skipping and memory access reduction have been estimated by the ratios of the practical counts to the referential workload and referential memory access count, respectively. The effective inference speed, defined by how many operations can be processed effectively per second, has been obtained by dividing the referential workload by the inference time calculated by multiplying the practical cycle count and period. Here, the referential workload and referential memory access count can be calculated directly depending on the model type and size (specified by $\alpha$ and $\beta$). The power consumption results with underlines have been estimated for the post-fit design by using Intel® Quartus® Prime Power Analyzer with the switching activity information; while other results without underlines have been obtained by scaling the dynamic power consumed by the memories according to the memory access reduction rates. The detailed analysis of the power consumption will be discussed later in this subsection.

As shown in the tables, the inference performance (i.e., the perplexity for the Penn Treebank (PTB) task and the accuracy for the sequential MNIST task) achieved by ROSETTA does not make a significant difference from that achieved based on the single-precision floating-point data, owing to the programmable data format, in spite of the size of each data element being as small as 8 bits. The first three

TABLE 4
Performance of the sequential MNIST task in 16 timesteps.

| Model [a] | Data format & pruning configurations [b] | Op. skip. (%) | Eff. infer. speed (GOP/s) | Mem. acc. reduc. (%) | Pow. cons. (mW) | Energy eff. (GOP/s/W) | Accuracy [c] (%) |
|---|---|---|---|---|---|---|---|
| GRU<br>($\alpha = 64, \beta = 128$,<br>149.5 KOP/timestep) |  Input<br>MVMA<br>ENOF (σ)<br>EMAC<br>1-<br>0<br>$W_z$ $b_z$ $W_r$ $b_r$ $W_{ch}$ $b_{ch}$ $W_{cx}$ $b_{cx}$<br>ENOF (τ)<br>Concatenation<br>Hidden state | 29.81 | 22.72 | 37.81 | <u>109.33</u> | 207.85 | 98.75 (0.12) |
| |  Input<br>MVMA<br>ENOF (σ)<br>EMAC<br>1-<br>0<br>$W_z$ $b_z$ $W_r$ $b_r$ $W_{ch}$ $b_{ch}$ $W_{cx}$ $b_{cx}$<br>ENOF (τ)<br>Concatenation<br>Hidden state | 46.05 | 29.56 | 49.44 | 89.34 | 330.89 | 97.95 (0.92) |
| |  Input<br>MVMA<br>ENOF (σ)<br>EMAC<br>1-<br>0<br>$W_z$ $b_z$ $W_r$ $b_r$ $W_{ch}$ $b_{ch}$ $W_{cx}$ $b_{cx}$<br>ENOF (τ)<br>Concatenation<br>Hidden state | 54.30 | 34.9 | 54.69 | 80.41 | 434.01 | 97.12 (1.75) |
| Bidirectional LSTM [24]<br>($\alpha = 64, \beta = 64$,<br>133.38 KOP/timestep) |  Concatenation<br>Hidden state<br>Backward layer cell state<br>EMAC<br>ENOF (σ)<br>MVMA<br>ENOF (τ)<br>0<br>$W_f$ $b_f$ $W_i$ $b_i$ $W_c$ $b_c$ $W_o$ $b_o$<br>Backward layer hidden state<br>Concatenation<br>Forward layer cell state<br>Forward layer hidden state<br>Input | 52.56 | 33.06 | 41.70 | 85.52 | 386.55 | 97.81 (0.89) |

[a] $\alpha$ and $\beta$ denote the sizes of the input and state(s), respectively.
[b] The blue solid, red dashed, and green dotted lines stand for the data formats of S1.7, S3.5, and U0.8, respectively. The circle, triangle, rectangle, and star symbols attached after the vector operations stand for the pruning with $T = 0, 1, 2$, and, 3, respectively. $\sigma$ and $\tau$ are the sigmoid and hyperbolic tangent functions, respectively.
[c] The number inside the parentheses is the absolute difference from the single-precision floating-point result, obtained based on each model without any pruning.

rows after the header in each table show different results for different pruning configurations, even based on the same models. If the pruning parameters (i.e. $T$'s for the vector operations) are configured so as to prune more activation elements (e.g., as in the third row after the header), more operations and memory accesses are skipped and reduced, respectively. This results in a superior energy efficiency with an inferior inference performance; contrastingly, the inferior energy efficiency with a superior inference performance is achieved by configuring the parameters for less pruning (e.g., as in the first row after the header). More importantly, ROSETTA provides the programmability to control the pruning amount in a fine-grained manner, by which we can make compromise on energy efficiency with inference performance, taking certain application-specific constraints into account. Moreover, such programmability

TABLE 5
Implementation results of the FPGA-based RNN inference processors.

| Processor | | ROSETTA | [9] [a] | [29] | [30] | [12] | [17] | [31] |
|---|---|---|---|---|---|---|---|---|
| FPGA device (part number) | | Cyclone®V (5CSXFC6D6) | Virtex®7 (XC7VX690T) | Zynq®7000 (XC7Z020) | Zynq®7000 (XC7Z045) | Zynq®7000 (XC7Z007S) | Cyclone®V (5CSXFC6D6) | Zynq®7000 (XC7Z007S) |
| RNN model | | Reconfigurable (programmable dataflow) | LSTM (fixed dataflow) | LSTM (fixed dataflow) | LSTM (fixed dataflow) | Modified GRU (fixed dataflow) | Reconfigurable (programmable dataflow) | LSTM (fixed dataflow) |
| Training methodology | | Not required | Required | Not Required | Not Required | Required | Not required | Required |
| Data format | | Programmable 8-bit | 16-bit | 16-bit | 16-bit | 16-bit activations<br>8-bit weights | 16-bit | 16-bit activations<br>8-bit weights |
| Res. usage | ALM (K) | 8.6 | N.A. | N.A. | N.A. | N.A. | 11.0 | N.A. |
| | LUT (K) | 13.9 [b] | 504.4 | 51.6 | 166.0 | 4.4 | 18.0 [b] | 11.3 |
| | FF (K) | 5.3 | 199.7 | 69.2 | 150.0 | 2.7 | 10.1 | 12.0 |
| | BRAM (Kb) | 800 | 34768 | 6451 | 18630 | 288 | 1620 | 1368 |
| | DSP | 64 | 2675 | 180 | 900 | 9 | 64 | 5 |
| On-chip param. (kilo param.) [c] | | $\leq 98.8$ | $\leq 2173.0$ | $\leq 403.2$ | $\leq 1164.4$ | $\leq 36.0$ | $\leq 98.8$ | $\leq 171.0$ |
| Power cons. (W) [d] | | 0.71 – 0.68 [e]<br>(0.14 – 0.08) | 22.00<br>(N.A.) | 2.29<br>(N.A.) | 10.60<br>(N.A.) | 2.29<br>(0.07) | 0.75<br>(0.14) | 2.30<br>(0.07) |
| Infer. speed (GOP/s) | | 34.90 – 15.90 [e] | 349.60 – 43.70 | 4.25 | 221.00 | 20.20 – 2.00 | 14.83 | 77.30 – 1.00 |
| Res. efficiency (MOP/s/LUT) | | 2.51 – 1.14 [e] | 0.69 – 0.09 | 0.08 | 1.33 | 4.55 – 0.45 | 0.82 | 6.84 – 0.09 |
| Energy efficiency (GOP/s/W) [d] | | 51.77 – 22.55 [e]<br>(434.01 – 113.29) | 15.89 – 1.99<br>(N.A.) | 1.86<br>(N.A.) | 20.84<br>(N.A.) | 8.82 – 0.87<br>(306.06 – 30.30) | 19.77<br>(107.43) | 33.61 – 0.43<br>(1120.29 – 14.49) |

[a] This corresponds to C-LSTM FFT8 in [9].
[b] This result has been estimated to be the combinational ALUT count, as suggested in the official guideline [32].
[c] This metric has been estimated to be the number of the parameters that can be fetched by the processor without involving external memory accesses. The metrics for the processors in [9], [12], [29], [30], [31] have been estimated to be very optimistic by considering the total BRAM usages and data formats since their papers do not show the concrete information of the BRAM usages dedicated to implement the parameter storages.
[d] The results inside the parentheses correspond to the power consumption or energy efficiency of the processor itself, while the results outside the parentheses correspond to those of the entire system.
[e] The highest result (the lowest result for power consumption) has been obtained for the model with the configuration shown in the third row in Table 4, at which the classification accuracy of 97.12 % is achieved for the sequential MNIST task [33]. The lowest result (the highest result for power consumption) has been obtained under the hypothetical worst-case condition where no pruning occurred.

is viable independently for the inference process for each timestep; enabling the pruning amount to be controlled through timesteps dynamically (e.g., controlling the pruning amount based on the results from the preceding timesteps). Such dynamic control based on the fine activation pruning might be studied in further work for the practical inference tasks.

The power consumption is detailed in Fig. 9. It is the breakdown of the power consumption of ROSETTA to perform the sequential MNIST task based on the GRU model with the configurations in the first row after the header in Table 4. As revealed in the figure, the dynamic power consumption of the memories is most dominant in the overall power consumption. The dynamic power consumption of the memories is linearly proportional to the memory access count, which can be reduced effectively by the activation pruning.

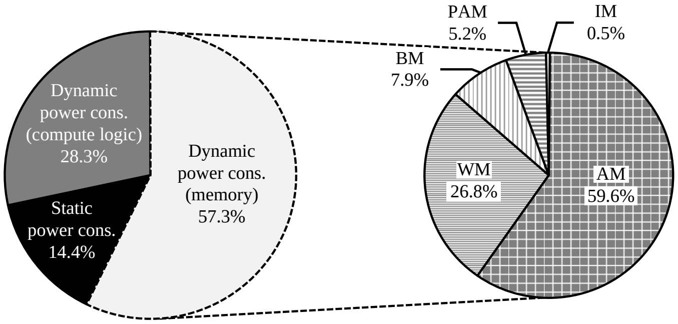


Fig. 9. Power consumption breakdown of ROSETTA for the model with the configurations, which are described in the first row after the header in Table 4.

### 4.3 Evaluation

ROSETTA is compared with the previous state-of-the-art RNN inference processors in terms of implementation results, which are summarized in Table 5. For fair comparison, the RNN inference processors implemented in the FPGA devices of the same technology (28 nm) node have been selected carefully. ROSETTA supports various types of RNN models by providing dataflow reconfigurability with the instructions corresponding to the primitive vector operations, whereas most of the previous processors [9], [12], [29], [30], [31] support limited types of RNN models with fixed dataflows. The data size of ROSETTA is much smaller than those of the previous processors. This makes ROSETTA capable of supporting a relatively larger model even with a smaller BRAM resource usage in case the model parameters are stored in the BRAM-based memories. Implemented based on the same platform, ROSETTA can be directly compared with our previous work [17]: the BRAM usage of ROSETTA has been reduced to about 49.4% by the reduced data size; the inference speed has been effectively increased by up to about 135.3% owing to the activation pruning. Despite such improvements, the logic resource usage is no higher than the previous result due to the efficient circuitry to process the data of the reduced size; the inference results are still maintained to be near-floating-point results.

The resource and energy efficiency of the RNN inference processors are compared in Table 5. The efficiency range has been obtained based on the range of the achievable inference speed. The minimum inference speed is the guaranteed speed achievable regardless of the statistics of the activation and parameter data. The maximum inference speed is that can be achieved with the data-dependent speed-up technique (e.g., activation pruning in ROSETTA, delta thresholding in [12], and model pruning in [9], delta thresholding and model pruning in [31]), which may not be theoretically guaranteed under certain constraints on the inference performance but proven practically for a few of the specific inference tasks.

ROSETTA exhibits a high resource and energy efficiency. Compared to the results of our previous work [17], the resource and energy efficiency are higher by up to 206.1% and 304.0%, respectively. They are also significantly higher than the efficiency of the other previous processors listed in the table, particularly in the guaranteed minimum results. This has been contributed by the efficient microarchitecture designed based on the homogeneous PUs with a high utilization rate as well as the reduced data size, which decreases not only the resource usage but also power consumption. Some of the previous ones show high efficiency in the peak results. However, they rely on the architecture-specific training methodologies [9], [12], [31] to boost the efficiency. The processors presented in [9], [31] are based on the *model pruning* techniques to sparsify the models, which are quite different from the *activation pruning* in ROSETTA. If the model pruning be performed in a structured way as presented in [34] to reduce the model size, ROSETTA might be also benefited from the reduced model to achieve a higher efficiency. Furthermore, some of the previous processors shows absolutely high resource usages, which might not be fit in low-cost resource-limited devices such as what ROSETTA has been implemented in.

## 5 Discussions with Related Work

ROSETTA is based on the instruction set supporting multiple data formats and activation pruning to perform RNN inference efficiently. There are a few of previous RNN inference processors based on the similar approaches. This section presents the related work briefly and discusses the novelty of ROSETTA in contrast.

The previous studies in [18], [35], [36], [37] presented RNN inference processors based on the instruction sets. One of them [36] was designed to support fixed-point data formats without programmability, which might be failing in realizing various dataflows of different data distributions with high resource efficiency; the others were designed to support floating-point data formats [18], [35], [37], inevitably involving a high resource usage. The performance-scalable architecture presented in [38], [39], [40] provides the flexibility of the data width in design time but not the programmability in runtime. ROSETTA has been designed based on the instruction set supporting multiple data formats and activation pruning with programmability. Designed based on the complex-instruction-set-computer principle, various dataflows involved in practical RNN inference can be described conveniently with only a few of instructions. This has been successfully demonstrated for several RNN models of different types targeting practical tasks in Section 4.2.

Several previous studies [7], [8], [9], [41] presented the model pruning techniques that exploit the sparsity in the parameters induced by special training methodologies to achieve efficient inference. The inference processor was designed to achieve high efficiency by pruning the model with the consideration of the architecture [7]. The previous studies presented in [42], [43], [44] are based on the activation pruning, which seems similar to that of ROSETTA. However, in ROSETTA, the activation pruning is programmable in the fine-grained manner differently for each vector operand, so as to achieve a moderate inference performance even with a non-zero pruning, in contrast to the other previous ones that do not support such programmability regarding non-zero pruning. Similarity-based input-skipping RNN inference processors are presented in [12], [15], [31], [42], where the computation overhead is inevitable because the difference between the current input and the input of the previous timestep needs to be computed for each timestep.

There are our previous studies [17], [26] that directly precede this study. In the study presented in [17], we designed an instruction-set architecture by deriving the common primitive vector operations, which the inference dataflows of the RNN models are composed of, intending to support various models for inference. In addition, we devised a multiplication-approximation scheme and a hardware-sharing technique to implement a resource-efficient processor. Motivated by observing the data distributions in practical inference tasks, we presented an RNN inference processor supporting multiple data formats in [26], which is the preliminary work of ROSETTA.

We have completely re-designed the instruction-set architecture as well as microarchitecture, for ROSETTA to achieve a higher resource and energy efficiency. The instruction-set architecture has been re-designed to provide the programmability regarding the multiple data formats[†]

[†]Our previous work presented in [17] supports only a single format (S4.12); whereas ROSETTA supports multiple data formats (S1.7, S3.5, and U0.8) of a smaller size.

and activation pruning. In addition, the instruction-set architecture has been simplified significantly by removing pointer handling operations, so as to reduce the code size for a smaller IM. The microarchitecture has been re-designed based on homogeneous PUs, that are efficiently utilized for every vector operation, supporting the multiple data formats and activation pruning. VPU in our previous work [17] was designed to perform not only EMAC but also the inner-product operation; however, VPU in ROSETTA has been designed to perform the EMAC operation only, based on which every vector operation is performed with a higher resource efficiency. As a result, compared to our previous work, ROSETTA achieves such improvements in terms of the resource and energy efficiency that have been shown in the previous section.

## 6 Conclusion

This paper presents an efficient RNN inference processor, named ROSETTA. ROSETTA is an instruction-set processor that supports per-vector programmable multiple data formats, for attaining a wide range or high precision with a limited data size. ROSETTA consistently performs every vector operation on the basis of the EMAC operation by fully utilizing the PUs to achieve high resource efficiency. ROSETTA prunes the activation elements in a fine-grained manner. The operations and memory accesses related to the pruned elements are skipped and reduced, respectively, to achieve high energy efficiency. A fully-integrated RNN inference system is developed in a low-cost 28 nm FPGA, under which the functionality of ROSETTA has been verified for practical inference tasks based on four different RNN models of different types. The resource and energy efficiency of ROSETTA are 1.14 MOP/s/LUT and 113.29 GOP/s/W, respectively, at the guaranteed minimum. Furthermore, it has been proven in practice that they can be effectively improved to be as high as 2.51 MOP/s/LUT and 434.01 GOP/s/W, respectively.

Although ROSETTA successfully achieves a high resource and energy efficiency, it has some limitations. First, only activation pruning is considered by ROSETTA even though many of the parameters have such small magnitudes that they can be pruned to achieve a higher efficiency [7], [8], [9], [41]. Second, some RNN models that emerged from very recent studies of the neural networks [45] may not be supported efficiently as they rely on the complicated dataflows quite different from those of the conventional models. Further studies may improve ROSETTA by considering such limitations stated above.

## Acknowledgments

This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) [2017-0-00528, The Basic Research Lab for Intelligent Semiconductor Working for the Multi-Band Smart Radar] and the GRRC program of Gyeonggi province [2017-B02, Study on 3D Point Cloud Processing and Application Technology]. The electronic design automation tools were supported by the IC Design Education Center.

## References

[1] J. L. Elman, “Finding structure in time,” *Cognitive Science*, vol. 14, no. 2, pp. 179–211, Mar. 1990.

[2] H. Stepp and S. Jurgen, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[3] J. Jurgovsky, M. Granitzer, K. Ziegler, S. Calabretto, P.-E. Portier, L. He-Guelton, and O. Caelen, “Sequence classification for credit-card fraud detection,” *Expert Systems with Applications*, vol. 100, pp. 234–245, Jun. 2018.

[4] B. Athiwaratkun and J. W. Stokes, “Malware classification with LSTM and GRU language models and a character-level CNN,” in *Proc. IEEE Int’l Conf. Acoustics, Speech & Signal Processing*. IEEE, Mar. 2017, pp. 2482–2486.

[5] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber, “A novel connectionist system for unconstrained handwriting recognition,” *IEEE Trans. Pattern Analysis & Machine Intelligence*, vol. 31, no. 5, pp. 855–868, May. 2008.

[6] Y. Guan, Z. Yuan, G. Sun, and J. Cong, “FPGA-based accelerator for long short-term memory recurrent neural networks,” in *Proc. Asia & South Pacific Design Automation Conf.* IEEE, Jan. 2017, pp. 629–634.

[7] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang *et al.*, “ESE: Efficient speech recognition engine with sparse LSTM on FPGA,” in *ACM/SIGDA Int’l Symp. Field-Programmable Gate Arrays*. ACM, Feb. 2017, pp. 75–84.

[8] S. Cao, C. Zhang, Z. Yao, W. Xiao, L. Nie, D. Zhan, Y. Liu, M. Wu, and L. Zhang, “Efficient and effective sparse LSTM on FPGA with bank-balanced sparsity,” in *ACM/SIGDA Int’l Symp. Field-Programmable Gate Arrays*. ACM, Feb. 2019, pp. 63–72.

[9] S. Wang, Z. Li, C. Ding, B. Yuan, Q. Qiu, Y. Wang, and Y. Liang, “C-LSTM: Enabling efficient LSTM using structured compression techniques on FPGAs,” in *ACM/SIGDA Int’l Symp. Field-Programmable Gate Arrays*. ACM, Feb. 2018, pp. 11–20.

[10] E. Bank-Tavakoli, S. A. Ghasemzadeh, M. Kamal, A. Afzali-Kusha, and M. Pedram, “Polar: A pipelined/overlapped FPGA-based LSTM accelerator,” *IEEE Trans. Very Large Scale Integration Systems*, vol. 28, no. 3, pp. 838–842, Nov. 2019.

[11] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using RNN encoder-decoder for statistical machine translation,” *arXiv preprint arXiv:1406.1078*, Jun. 2014.

[12] C. Gao, A. Rios-Navarro, X. Chen, S.-C. Liu, and T. Delbruck, “EdgeDRNN: Recurrent neural network accelerator for edge inference,” *IEEE Jrnl. Emerging & Selected Topics in Circuits & Systems*, vol. 10, no. 4, pp. 419–432, Dec. 2020.

[13] E. Azari and S. Vrudhula, “An energy-efficient reconfigurable LSTM accelerator for natural language processing,” in *Proc. IEEE Int’l Conf. Big Data*. IEEE, Dec. 2019, pp. 4450–4459.

[14] F. Conti, L. Cavigelli, G. Paulin, I. Susmelj, and L. Benini, “Chipmunk: A systolically scalable 0.9mm$^2$, 3.08 GOP/s/mW @ 1.2 mW accelerator for near-sensor recurrent neural network inference,” in *Proc. IEEE Custom Integrated Circuits Conf.* IEEE, Apr. 2018, pp. 1–4.

[15] J. Jo, J. Kung, and Y. Lee, “Approximate LSTM computing for energy-efficient speech recognition,” *Electronics*, vol. 9, no. 12: 2004, Nov. 2020. [Online]. Available: https://www.mdpi.com/2079-9292/9/12/2004

[16] D. Kadetotad, S. Yin, V. Berisha, C. Chakrabarti, and J.-S. Seo, “An 8.93-TOPS/W LSTM recurrent neural network accelerator featuring hierarchical coarse-grain sparsity for on-device speech recognition,” *IEEE Jrnl. Solid-State Circuits*, vol. 55, no. 7, pp. 1877–1887, Jul. 2020.

[17] J. Kim, J. Kim, and T.-H. Kim, “AERO: A 1.28 MOP/s/LUT reconfigurable inference processor for recurrent neural networks in a resource-limited FPGA,” *Electronics*, vol. 10, no. 11: 1249, May. 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/11/1249

[18] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi *et al.*, “A configurable cloud-scale DNN processor for real-time AI,” in *Proc. of ACM/IEEE Int’l Symp. Computer Architecture*. IEEE, Jun. 2018, pp. 1–14.

[19] D. Shin, J. Lee, J. Lee, and H.-J. Yoo, “DNPU: An 8.1 TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks,” in *Proc. IEEE Int’l Solid-State Circuits Conf.* IEEE, Feb. 2017, pp. 240–241.

[20] S. Zeng, K. Guo, S. Fang, J. Kang, D. Xie, Y. Shan, Y. Wang, and H. Yang, "An efficient reconfigurable framework for general purpose CNN-RNN models on FPGAs," in *Proc. IEEE Int'l Conf. Digital Signal Processing*. IEEE, Nov. 2018, pp. 1–5.

[21] Q. Li, X. Zhang, J. Xiong, W.-m. Hwu, and D. Chen, "Implementing neural machine translation with bi-directional GRU and attention mechanism on FPGAs using HLS," in *Proc. Asia & South Pacific Design Automation Conf.*, Jan. 2019, pp. 693–698.

[22] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A search space odyssey," *IEEE Trans. Neural Networks & Learning Systems*, vol. 28, no. 10, pp. 2222–2232, Jul. 2016.

[23] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, "Learning precise timing with LSTM recurrent networks," *Jrnl. Machine Learning Research*, vol. 3, pp. 115–143, Aug. 2002.

[24] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Trans. Signal Processing*, vol. 45, no. 11, pp. 2673–2681, Nov. 1997.

[25] M. Marcus, B. Santorini, and M. A. Marcinkiewicz, "Building a large annotated corpus of English: The Penn Treebank," *Computational Linguistics*, vol. 19, no. 2, pp. 313–330, Jun. 1993.

[26] J. Kim, K. Park, and T.-H. Kim, "A reconfigurable inference processor for recurrent neural networks based on programmable data format in a resource-limited FPGA," in *Proc. Asia & South Pacific Design Automation Conf.* IEEE, Jan. 2022, pp. 94–95.

[27] A. X. M. Chang, B. Martini, and E. Culurciello, "Recurrent neural networks hardware implementation on FPGA," *arXiv preprint arXiv:1511.05552*, Nov. 2015.

[28] V. G. Oklobdzija, "An algorithmic and novel design of a leading zero detector circuit: comparison with logic synthesis," *IEEE Trans. Very Large Scale Integration Systems*, vol. 2, no. 1, pp. 124–128, Mar. 1994.

[29] X. Zhang, W. Jiang, and J. Hu, "Achieving full parallelism in LSTM via a unified accelerator design," in *IEEE Int'l Conf. Computer Design*. IEEE, Oct. 2020, pp. 469–477.

[30] Z. Que, Y. Zhu, H. Fan, J. Meng, X. Niu, and W. Luk, "Mapping large LSTMs to FPGAs with weight reuse," *Jrnl. Signal Processing Systems*, vol. 92, no. 9, pp. 965–979, Jul. 2020.

[31] C. Gao, T. Delbruck, and S.-C. Liu, "Spartus: A 9.4 TOP/S FPGA-based LSTM accelerator exploiting spatio-temporal sparsity," *IEEE Trans. Neural Networks & Learning Systems*, early access. doi: 10.1109/TNNLS.2022.3180209.

[32] *Xilinx Design Flow for Intel FPGA SoC Users*, UG1192 (v2.2), Xilinx, San Jose, CA, U.S., Feb. 2018.

[33] Q. V. Le, N. Jaitly, and G. E. Hinton, "A simple way to initialize recurrent networks of rectified linear units," *arXiv preprint arXiv:1504.00941*, Mar. 2015.

[34] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proc. of IEEE Int'l Conf. Computer Vision*, Oct. 2017, pp. 1389–1397.

[35] E. Nurvitadhi, D. Kwon, A. Jafari, A. Boutros, J. Sim, P. Tomson, H. Sumbul, G. Chen, P. Knag, R. Kumar *et al.*, "Why compete when you can work together: FPGA-ASIC integration for persistent RNNs," in *IEEE Int'l Symp. Field-Programmable Custom Computing Machines*. IEEE, Apr. 2019, pp. 199–207.

[36] A. Boutros, E. Nurvitadhi, R. Ma, S. Gribok, Z. Zhao, J. C. Hoe, V. Betz, and M. Langhammer, "Beyond peak performance: comparing the real performance of AI-optimized FPGAs and GPUs," in *Int'l Conf. Field-Programmable Technology*. IEEE, Dec. 2020, pp. 10–19.

[37] R. Ma, J.-C. Hsu, T. Tan, E. Nurvitadhi, D. Sheffield, R. Pelt, M. Langhammer, J. Sim, A. Dasu, and D. Chiou, "Specializing FGPU for persistent deep learning," *ACM Trans. Reconfigurable Tech. & Systems*, vol. 14, no. 2, pp. 1–23, Jul. 2021.

[38] V. Rybalkin, C. Sudarshan, C. Weis, J. Lappas, N. Wehn, and L. Cheng, "Efficient hardware architectures for 1D-and MD-LSTM networks," *Jrnl. Signal Processing Systems*, vol. 92, no. 11, pp. 1219–1245, Jul. 2020.

[39] V. Rybalkin, J. Ney, M. K. Tekleyohannes, and N. Wehn, "When massive GPU parallelism ain't enough: A novel hardware architecture of 2D-LSTM neural network," *ACM Trans. Reconfigurable Tech. & Systems*, vol. 15, no. 1, pp. 1–35, Nov. 2021.

[40] Z. Que, H. Nakahara, E. Nurvitadhi, A. Boutros, H. Fan, C. Zeng, J. Meng, K. H. Tsoi, X. Niu, and W. Luk, "Recurrent neural networks with column-wise matrix-vector multiplication on FPGAs," *IEEE Trans. Very Large Scale Integration Systems*, vol. 30, no. 2, pp. 227–237, Dec. 2021.

[41] B. Wu, Z. Wang, K. Chen, C. Yan, and W. Liu, "GBC: An energy-efficient LSTM accelerator with gating units level balanced compression strategy," *IEEE Trans. Circuits & Systems I, Reg. Papers*, vol. 69, no. 9, pp. 3655–3665, Jun. 2022.

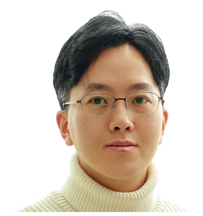
[42] J. Jiang, T. Xiao, J. Xu, D. Wen, L. Gao, and Y. Dou, "A low-latency LSTM accelerator using balanced sparsity based on FPGA," *Microprocessors and Microsystems*, vol. 89, Jan. 2022.

[43] M. Wang, Z. Wang, J. Lu, J. Lin, and Z. Wang, "E-LSTM: An efficient hardware architecture for long short-term memory," *IEEE Jrnl. Emerging & Selected Topics in Circuits & Systems*, vol. 9, no. 2, pp. 280–291, Apr. 2019.

[44] Y. Zheng, H. Yang, Y. Jia, and Z. Huang, "PermLSTM: A high energy-efficiency LSTM accelerator architecture," *Electronics*, vol. 10, no. 8: 882, Mar. 2021.

[45] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Info. Processing Systems*. Neural Information Processing Systems, Dec. 2017, pp. 5998–6008.

**Ji-Ho Kim** Jiho Kim received her B.S. degree in electronics and information engineering from Korea Aerospace University, Gyeonggi-do, Korea, in 2021, where she is currently pursuing the M.S. degree at the same university. Her research interests include low-power and resource-efficient deep-neural-network inference/training processor, reconfigurable architecture, deep learning algorithms, and embedded systems design.



**Tae-Hwan Kim** Tae-Hwan Kim received the B.S. degree in electrical engineering from Yonsei University, Seoul, Korea, in 2005 (with high honors), the M.S. and Ph. D. degree in electrical engineering from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 2007 and 2010, respectively. In 2010, he received "Best Paper Award" for his Ph.D. dissertation. From 2010 to 2011, he worked for Samsung Electronics as a senior research engineer, in charge of R&D on the baseband systems for Wi-Fi. Since Oct. 2011, he has been in the faculty of School of Electronics and Information Engineering in Korea Aerospace University. His current research is focused on the circuits and systems for artificial intelligence and signal processing systems.