Low-Area and Low-Power VLSI Architectures for Long Short-Term Memory Networks

Mohammed A. Alhartomi, *Member, IEEE*, Mohd Tasleem Khan, Saeed Alzahrani, *Member, IEEE*, Ahmed Alzahmi, *Member, IEEE*, Rafi Ahamed Shaik, *Senior Member, IEEE*, Jinti Hazarika, *Graduate Student Member, IEEE*, Ruwaybih Alsulami, *Member, IEEE*, Abdulaziz Alotaibi, and Meshal Al-Harthis

***Abstract*— Long short-term memory (LSTM) networks are extensively used in various sequential learning tasks, including speech recognition. Their significance in real-world applications has prompted the demand for cost-effective and power-efficient designs. This paper introduces LSTM architectures based on distributed arithmetic (DA), utilizing circulant and block-circulant matrix-vector multiplications (MVMs) for network compression. The quantized weights-oriented approach for training circulant and block-circulant matrices is considered. By formulating fixed-point circulant/block-circulant MVMs, we explore the impact of kernel size on accuracy. Our DA-based approach employs shared full and partial methods of add-store/store-add followed by a select unit to realize an MVM. It is then coupled with a multi-partial strategy to reduce complexity for larger kernel sizes. Further complexity reduction is achieved by optimizing decoders of multiple select units. Pipelining in add-store enhances speed at the expense of a few pipelined registers. The results of the field-programmable gate array showcase the superiority of our proposed architectures based on the partial store-add method, delivering reductions of 98.71% in DSP slices, 33.59% in slice look-up tables, 13.43% in flip-flops, and 29.76% in power compared to the state-of-the-art.**

***Index Terms*— Inner products, long-short term memory, matrix-vector multiplication, recurrent neural networks, VLSI.**

I. Introduction

THE surge in data demand drives ongoing evolution in artificial intelligence and machine learning. This evolution spans neural network enhancements-algorithms, architectures, and real-world applications. Notably, deep neural networks transform these domains, mimicking human-level performance, particularly in tasks like speech and image recognition [1]. This mimicry draws from the brain's neural interconnections and learning processes. Rosenblatt's *perceptron* concept initially delineated how neurons learn [2].

In Fig. 1(a), a single-layer perceptron includes reception, association, and response cells. Reception cells capture input, associate cells weigh features, and response cells activate based on conditions. Single/multi-layer perceptrons cannot learn input weights. This led to recurrent neural networks (RNNs) with internal memory for weight training [3]. Usually, the RNNs compute output using input and previous output, as shown in Fig. 1(b). Many RNNs suffer vanishing gradients, disjoint connection issues, and struggle with long-term info retention [4].

The RNN issues have been addressed by long short-term memory (LSTM) networks by employing multiple gates [3], [4]. Each gate can be viewed as a small neural network. These networks find widespread use in various sequential tasks, including pattern recognition in temporal data sequences [5], speech recognition [6], electrocardiogram classification [7], machine translation [8], and more. Because of the multiple gates, LSTM incurs significant computational costs, stemming from numerous matrix-vector multiplications (MVMs) and element-wise multiplications (EWMs). Furthermore, it demands substantial memory to store network parameters. Therefore, it is desirable for implemented LSTM networks to occupy low chip-area and consume low-power.

Various compression schemes, including those presented in [9], [10], [11], [12], [13], [14], [15], [16], and [17], aim to minimize memory needs. For instance, [10] accelerated sparse MVM using weight-sharing within a compact network model. In the study by Guo et al. [11], it was shown that utilizing fixed-point representation preserves model accuracy while allowing for minimal memory usage. Effective pruning variants for embedding RNNs entail structured pruning [16] and top-k pruning [15]. Leveraging low-displacement rank matrices, such as circulant matrices for MVM presents a potential approach to trim redundant parameters [17]. This strategy effectively reduces memory usage by curbing redundant network parameter storage.

Extensive research, as discussed above, has been focused on reducing memory requirements in LSTM networks. However, the main obstacle related to very large-scale integration (VLSI)

Manuscript received 29 May 2023; revised 24 August 2023 and 24 October 2023; accepted 28 October 2023. Date of publication 6 November 2023; date of current version 29 December 2023. This work was supported by the Deanship of Scientific Research, University of Tabuk, through Research, under Grant S-1443-0195. This article was recommended by Guest Editor J. Eshraghian. *(Corresponding author: Mohammed A. Alhartomi.)*

Mohammed A. Alhartomi, Saeed Alzahrani, Ahmed Alzahmi, Abdulaziz Alotaibi, and Meshal Al-Harthis are with the Department of Electrical Engineering, University of Tabuk, Tabuk 71491, Saudi Arabia (e-mail: malhartomi@ut.edu.sa).

Mohd Tasleem Khan is with the Department of Electronics, IIT Dhanbad, Dhanbad 826004, India.

Rafi Ahamed Shaik and Jinti Hazarika are with the Department of Electronics and Electrical Engineering, IIT Guwahati, Guwahati 781039, India.

Ruwaybih Alsulami is with the Department of Electrical Engineering, Umm Al-Qura University Makkah, Mecca 24382, Saudi Arabia.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JETCAS.2023.3330428.

Digital Object Identifier 10.1109/JETCAS.2023.3330428

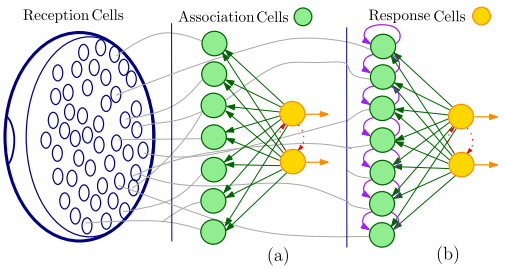

Fig. 1. Visualization of a single-layer (a) Perceptron, and (b) RNN.

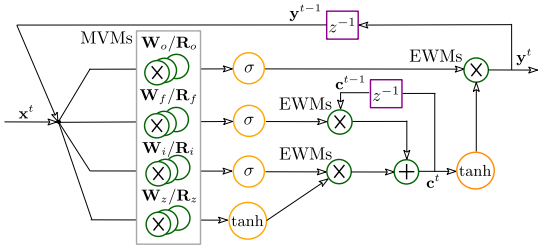

Fig. 2. A typical layer in an LSTM network.

parameters, such as area and power, stems from the computational cost associated with intricate MVMs, particularly for FPGA implementation. Recently, the research is focused on minimizing LSTM computations for FPGA implementation [9], [10], [11], [12], [14], [17], [18], [19], [20], [21]. This shift is motivated by the recognition that FPGAs offer a relatively more efficient platform for accelerating LSTM inference. Furthermore, FPGA provides the flexibility of hardware architecture customization and the utilization of multiple on-chip digital signal processor (DSP) blocks. This adaptation is driven by the need for low-latency inference and the pursuit of competitive energy efficiency [22]. For instance, [12] reduced LSTM computational costs by capitalizing on sparsity within network parameters, while [21] utilized spatio-temporal sparsity in LSTM for FPGA acceleration.

Distributed arithmetic (DA), a highly efficient bit-serial method for inner product (IP) realization [23], [24], [25], [26], [27], [28], typically involves a look-up table (LUT) followed by a shift-accumulate (SA) unit. While various DA strategies have been proposed [17], [25], this paper focuses on addressing limitations in existing methods. Notably, [25], [26], [28] employed offset-binary coding (OBC) for LUT implementation, reducing computational costs. The work presented in [17] utilizes an OBC-based approach that involves the parallelized adders and multiplexers for LUT contents, which caused a notable increase in hardware costs. Fine-grained pipelining was introduced, accompanied by a higher register count, to mitigate critical path delays. However, the significant number of multipliers in EWMs remained a resource-intensive aspect. These observations motivate the introduction of optimized, low-cost DA-based architectures for LSTM implementation on FPGA. The key contributions of this work are as follows.

- Training of a compressed fixed-point LSTM network is presented, followed by its formulation using DA;
- A new approach is introduced for reducing the LUT cost, utilizing both full and partial methods;
- Four architectures for circulant/block-circulant MVMs based on LUT structures are proposed.
- The MVM cost is further reduced through the sharing of LUT contents across multiple select units. An optimized select unit is also introduced;
- Performance evaluation of the proposed architectures on FPGA with different design parameters is carried out;

The rest of the paper is organized as follows: Section II presents the development of the LSTM model based on structure compression, training in quantized scenarios, and DA formulation. Section III discusses the proposed fixed-point DA-based designs and optimization of LSTM inference. Section IV discusses the implementation results of the proposed designs for performance evaluation and comparison with state-of-the-art FPGA-based LSTM accelerators. Conclusions are provided in Section V.

II. Development of Compressed LSTM Model

Consider a typical layer in the LSTM network, as illustrated in Fig. 2. It comprises a cell, a block gate, an input gate, an output gate, and a forget gate. The cell performs two tasks: firstly, it recalls the value at a time instant t $(0 \leq t \leq T-1)$, and then regulates the information flow, where T is the target. The subsequent set of equations describes the functionality of an LSTM unit:

$$\text{Input Gate:} \quad \mathbf{i}^t = \sigma(\mathbf{W}_i \mathbf{x}^t + \mathbf{R}_i \mathbf{y}^{t-1} + \mathbf{b}_i), \tag{1}$$

$$\text{Forget Gate:} \quad \mathbf{f}^t = \sigma(\mathbf{W}_f \mathbf{x}^t + \mathbf{R}_f \mathbf{y}^{t-1} + \mathbf{b}_f), \tag{2}$$

$$\text{Output Gate:} \quad \mathbf{o}^t = \sigma(\mathbf{W}_o \mathbf{x}^t + \mathbf{R}_o \mathbf{y}^{t-1} + \mathbf{b}_o), \tag{3}$$

$$\text{Block Gate:} \quad \mathbf{z}^t = \tanh(\mathbf{W}_z \mathbf{x}^t + \mathbf{R}_z \mathbf{y}^{t-1} + \mathbf{b}_z), \tag{4}$$

$$\text{Memory Cell:} \quad \mathbf{c}^t = \mathbf{z}^t \odot \mathbf{i}^t + \mathbf{c}^{t-1} \odot \mathbf{f}^t, \tag{5}$$

$$\text{Hidden State:} \quad \mathbf{y}^t = \tanh(\mathbf{c}^t) \odot \mathbf{o}^t, \tag{6}$$

where the matrices $\mathbf{W}_s$ and $\mathbf{R}_s$ $(s = i, f, o, z)$ represent the input weights and recurrent connections, respectively. The vector $\mathbf{b}_s$ corresponds to the bias terms, and $\odot$ represents an EWM. The activation functions $\sigma(\cdot)$ and $\tanh(\cdot)$ are, respectively, given by

$$\sigma(x) = \frac{1}{1+e^{-x}} \quad \text{and} \quad \tanh(x) = \frac{1-e^{-x}}{1+e^{-x}} \tag{7}$$

where x is any input. The input gate and the forget gate are responsible for deciding how much of the incoming information should be stored in memory and how much of the past information should be discarded, respectively. The hidden state serves both as the internal state and its final output. The output gate regulates the amount of sequential information to be produced as output, while the block gate can be viewed as the extracted information from the input.

Consider an $N_1 \times N_2$ LSTM layer with N_1 and N_2 are the sizes of input and output vectors, respectively. We possess $\mathbf{x}^t \in \mathbb{R}^{N_1 \times 1}$; $\mathbf{y}^t, \mathbf{b}_s, \mathbf{c}^t \in \mathbb{R}^{N_2 \times 1}$; $\mathbf{R}_s \in \mathbb{R}^{N_2 \times N_2}$; and $\mathbf{W}_s \in \mathbb{R}^{N_2 \times N_1}$ $(s = i, f, o, z)$. The combined count of parameters depicted by the LSTM in (1)–(4) amounts to $4(N_1 N_2 + N_2^2 + N_2)$. When $N_1 = N_2 = N$, the computational and spatial complexities of

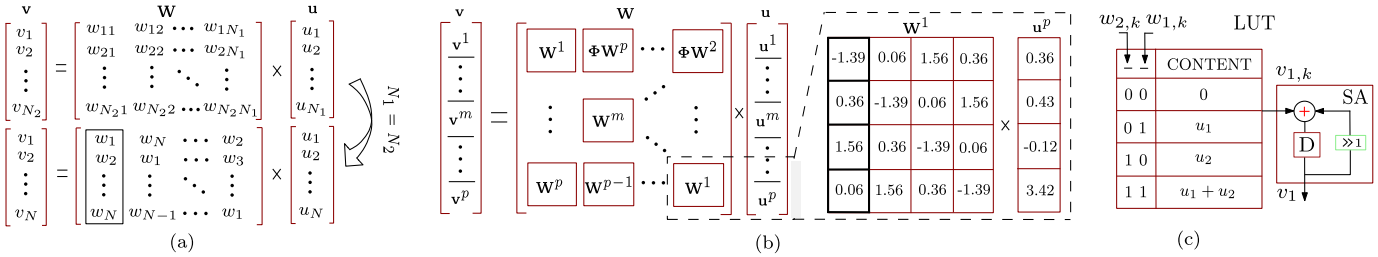

Fig. 3. (a) Mapping of a normal MVM to a circulant MVM, where black box indicates the elements of the primitive vector. (b) Block-circulant MVM with an example of sub-MVM for $q = 4$ and $N = 4p$. (c) A typical DA-based IP computation for $q = 2$ and $N = 2$.

LSTM both scale at $O(N^2)$. As a result, the parameter count becomes $4(N \times N + N^2 + N) = 8N^2 + 4N$.

A. Structure Compression

As stated, LSTM primarily consists of EWMs and MVMs. Overall, matrix-related operations tend to dominate computational complexity. Hence, a suitable network compression technique must be utilized to trim redundant parameters, rectify structural irregularities, minimize model size and memory usage.

1) Circulant Matrices: A circulant matrix falls into the category of Toeplitz-like matrices. It is commonly square, which implies $N_1 = N_2$. A typical mapping of the normal matrices to circulant matrices is shown in Fig. 3(a). By constraining the matrix to be circulant, the space complexity can be lowered from $O(N^2)$ to $O(N)$, resulting in a noteworthy reduction in the number of LSTM parameters [29]. In some cases $N_1 \neq N_2$, adjustments are needed to achieve circulant matrices. When $N_2 > N_1$, extend the input vector by adding $N_2 - N_1$ zeros; when $N_2 < N_1$, keep only the first N_2 elements in the multiplication result. In addition to the benefit of reduced space requirements, employing circulant matrices as weight matrices can also result in the reduction of computational complexity for both training and inference [30].

Any circulant matrix $\mathbf{W}$ is defined by primitive vector $\mathbf{w}_1 = [w_1, w_2, w_3, \ldots, w_N]^T$ corresponding to its first column. It can be noticed that each row (or column) is a cyclic shift of the row (or column) immediately preceding it. So, any column vector of $\mathbf{W}$ is given by $\mathbf{w}_i = [w_{(1-i)\text{mod}(N)+1}, w_{(2-i)\text{mod}(N)+1}, \ldots, w_{(N-i)\text{mod}(N)+1}]^T$, with $1 \leq i \leq N$, is the column index of a circulant matrix. Thus, MVMs in (1)–(4) with circulant matrices can be conceptualized as an IP array

$$v_j = \mathbf{w}^j \mathbf{u} = \sum_{i=1}^{N} w_{i'} u_i \text{ with } i' = (j - i)\text{mod}(N) + 1 \quad (8)$$

where $\mathbf{u} = [u_1, u_2, u_3, \ldots, u_N]^T$ is the input vector, $\mathbf{w}^j$ is the jth-row of $\mathbf{W}$ and v_j is the jth element of the output vector $\mathbf{v}$, with $1 \leq j \leq N$, is the row index of a circulant matrix.

Empirical findings in [31] revealed that the parameters corresponding to the block gate are sensitive to numerical errors during the training. In contrast, the parameters corresponding to the input, forget, and output gates are relatively resilient to such errors. Exploiting this insight, we employ circulant matrices exclusively for input, forget, and output gate parameter matrices, effectively reducing required memory space. Modifications to the extensively employed backpropagation through time (BPTT) algorithm facilitate batch training of circulant-like parameter matrices [13], [30], [32], [33]. Under these scenarios, with $N_1 = N_2 = N$, the resulting parameter count becomes $2 \times N^2 + 6 \times N + 4 \times N = 2N^2 + 10N$. In contrast, the original network model possesses $8N^2 + 4N$ parameters. Thus, the network is compressed by

$$(8N^2 + 4N)/(2N^2 + 10N) \approx 4, \quad \text{when } N \text{ is large} \quad (9)$$

2) Block-Circulant Matrices: In most applications, MVM order in the LSTM network is often high, posing constraints on-chip area and power consumption. The block-circulant formulation of circulant matrices offers a solution to this challenge [17]. Essentially, it splits a $N \times N$ circulant matrix into smaller $q \times q$ circulant matrices, with $q = N/p$ representing the kernel size. As a result, the number of smaller circulant MVMs becomes p^2, evenly distributed across rows and columns, as shown in Fig. 3(b). By utilizing block-circulant matrices, the model complexity is effectively reduced to $O(pq^2 \log q)$ [30], [34]. Similar to circulant MVM, each sub-MVM of the block-circulant variant can be obtained through a primitive vector, as illustrated through an example in Fig. 3(b). In general, p primitive vectors are derived by splitting the original primitive vector $\mathbf{w}_1$, such that $\mathbf{w}_1^1, \mathbf{w}_1^2, \ldots, \mathbf{w}_1^p$, where $\mathbf{w}_1^m = [w_n, w_{n+q}, w_{n+2q}, \ldots, w_{n+(m-1)q}]$, with $1 \leq m \leq p$ and $1 \leq n \leq q$. Similarly, the input vector $\mathbf{u}$ and output vector $\mathbf{v}$ are split into p input vectors $\mathbf{u}^m$ and p output vectors $\mathbf{v}^m$ respectively. As depicted in Fig. 3(b), Φ represents another $q \times q$ circulant matrix whose elements are either 0 or 1 to rotate the sub-matrices $\mathbf{W}^m$ [17]. It can be observed from Fig. 3(b) that the partial products generated from $\mathbf{u}^m$ are shared among p sub-circulant MVMs. Consequently, an array of IPs corresponding to the first row can be written as

$$\mathbf{v}^1 = \mathbf{W}^1 \mathbf{u}^1 + \Phi \mathbf{W}^p \mathbf{u}^2 + \ldots + \Phi \mathbf{W}^2 \mathbf{u}^p \quad (10)$$

All the IPs corresponding to the sub-circulant MVMs can be computed in parallel; thus, the full MVM requires only p clock cycles.

B. Training Strategy and Results

The considered training strategy with quantized weight alongside the BPTT algorithm is illustrated in Fig. 4. It is a two-stage approach. In the first stage, network parameters are randomly initialized and trained, incorporating circulant matrices and forward approximation. The second stage

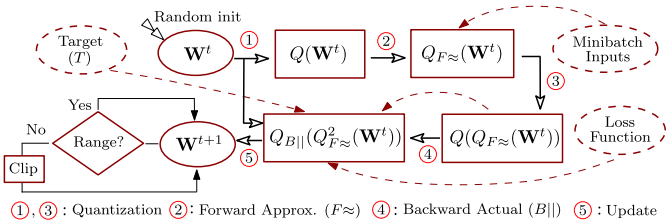

Fig. 4. Flowchart to describe a quantized-weight oriented training approach.

involves retraining the model with additional quantization methods [35]. Note that random initialization may affect the network's prediction accuracy [36]; however, employing an appropriate retraining strategy can resolve this issue. The nonlinear activation functions given in (7) are approximated using piece-wise linear functions during the forward process, identical to [37]. In training, the network employs piece-wise linear functions for forwarding, while actual nonlinear functions continue to accumulate gradients during the backward process.

Intermediate values are computed using the forward approximation $Q_{F\approx}(\mathbf{W}^t)$ during the forward pass with quantized weights $Q(\mathbf{W}^t)$ and piece-wise linear functions, where $Q(\cdot)$ function is defined similar to [35]. These values are then quantized themselves. Backpropagation calculates gradients of loss function L, according to $\partial L/\partial \mathbf{W}^t = Q_{B||}(Q^2_{F\approx}(\mathbf{W}^t))$ using the inputs indicated by the dashed lines in Fig. 4 and original nonlinear functions, where superscript 2 over $Q(\cdot)$ means two-step quantization. Note that minibatch inputs are chosen in line with kernel size for training and evaluation purposes. The initial weights, $\mathbf{W}^t$, receive updates based on the summed gradients $\partial L/\partial \mathbf{W}^t$. However, if the gradients are too small, direct weight quantization during backpropagation could nullify their impact in subsequent steps. This can impede network training efficiency. Optionally, a slightly enhanced outcome can be achieved by clipping the updated weights, $\mathbf{W}^{t+1}$, to align within the quantized weight range. During training, storing both original and quantized weights requires more memory. An option is on-the-fly quantization computation instead of fixed memory allocation, which may cause extra calculations and a slight increase in training time.

To assess the performance of the proposed training strategy for the compressed LSTM, we opted for a speech recognition task utilizing the TIMIT dataset [38]. This dataset comprises recordings from 630 speakers across eight American English dialects, with each speaker reading ten phonetically dense sentences. Additionally, we employ a larger speech recognition dataset with approximately 500 hours of training, 50 hours of validation, and 5 hours of testing. The input speech features correspond to those in [12]. In the context of speech recognition tasks [30], [34], we measure performance using the phone error rate (PER), where a lower PER indicates greater prediction accuracy.

In practice, we randomly pair data within a batch for model training. Both training and evaluation employ a batch size equal to the kernel size. The kernel size determines the assessment of similarity; while it accelerates stochastic gradient convergence, it does so at the expense of accuracy.

TABLE I
TRAINING RESULTS OF COMPRESSED LSTM MODEL ON TIMIT DATASET FOR 256 × 256 LAYER

Label	$\sigma(\cdot)$, $\tanh(\cdot)$	$\mathbf{W}_s$, $\mathbf{R}_s$	$\mathbf{c}^t$, $\mathbf{y}^t$	$\mathbf{i}^t$, $\mathbf{f}^t$ $\mathbf{o}^t$, $\mathbf{z}^t$	$\mathbf{b}_i$, $\mathbf{b}_f$ $\mathbf{b}_o$, $\mathbf{b}_z$	PER	CR
(a)†	Eq. (7)	FLP32	FLP32	FLP32	FLP32	26.18%	–
(b)†	[37]	FLP32	FLP32	FLP32	FLP32	28.79%	–
(c)‡	[37]	FP8	FP8	FP8	FP8	22.98%	93.53%
(d)‡	[37]	FP8	FP8	FP8	FP8	25.42%	95.08%
q^*	1	2	4	8	16	32	64
PER	22.98	23.03	23.22	23.64	24.87	27.35	31.30
ΔPER	0.00	0.05	0.19	0.42	1.23	2.48	3.95

The bit width the parameters for block gate $\mathbf{z}^t$ is considered same as other gates, except for (d), which kept its bit width as FP6. PER: Phone-error rate, CR: Compression ratio, †: Random initialization, ‡: Retrained from (b), FLP32: 32-bit floating point, FP8/FP6: 8-bit/6-bit fixed point. * Retrained from (c) for different q with baseline as $q = 1$, where ΔPER is relative reduction in PER with respect to q.

Table I presents results for the 256 × 256 layer, covering the designated cases labeled as follows:

(a) 32-bit floating point (FLP32) circulant-matrices based parameters, actual non-linear sigmoid/tangent functions (7), and random initialization.

(b) FLP32 circulant-matrices based parameters, approximate non-linear sigmoid/tangent functions [37] and random initialization.

(c) 8-bit fixed-point (FP8) circulant-matrices based parameters, approximate non-linear sigmoid/tangent functions [37] and re-trained from (b).

(d) FP8 circulant-matrices based parameters for $\mathbf{i}^t$/ $\mathbf{f}^t$/$\mathbf{o}^t$ gates and FP6 for $\mathbf{z}^t$, approximate non-linear sigmoid/tangent functions [37] and re-trained from (b).

From these cases, it is observed that random initialization with circulant matrices-based parameters always results in poor prediction accuracy, regardless of whether actual or approximate non-linear sigmoid/tangent functions are employed. However, retraining with FP8 quantization using a pre-trained model (assumed to be randomly initialized) consistently improves prediction accuracy. For example, the prediction accuracy in case (c) is enhanced when the model is re-trained from case (b). As mentioned, the block gates are sensitive to numerical errors during training [31]. To validate this, we considered case (d) with FP6 circulant matrices based parameters for the block gate. It is found that the prediction accuracy in terms of PER has degraded. Thus, we considered case (c) as the baseline with FP8 circulant matrices based parameters for all the gates and analyzed the impact of choosing q on prediction accuracy.

The second section of Table I illustrates the influence of the kernel size ($1 \leq q \leq 64$) on the PER. Notably, when $q = 1$, no compression is achieved as it mirrors circulant matrices. Higher values of q lead to an increase compression [30], but the choice of q affects accuracy. For smaller q, the impact on PER is marginal; however, for larger q, the degradation can be significant. For example, a compressed LSTM with $q = 2$ shows nearly equivalent PER to $q = 1$, but experiences a relative reduction of 3.95 from $q = 32$ to $q = 64$. While using block-circulant matrices can be advantageous for compression, it might compromise accuracy, particularly for larger kernel sizes. Notably, degradation is not substantial until $q = 32$, making it a relatively reasonable

TABLE II
MEANING OF ACRONYMS USED TO DESCRIBE LSTM ARCHITECTURES

Acronym	Meaning	Acronym	Meaning
RAM	Random Access M/m	STF	Sigmoid-Tangent Function
LUT	Look-up Table	FASS	Full Add Store Select
SA	Shift-Accumulate	FSAS	Full Store Add Select
CS	Circular Shift	PASS	Partial Add Store Select
LU	LUT Enumeration	PSAS	Partial Store Add Select
AU	Accumulation Unit	CSA	Carry-Save Adder Based SA
AS	Accumulator Stage	CPD	Critical Path Delay
PISO	Parallel-in Serial-out	SOP	Sum-of-Product

choice for designing compressed LSTMs that fulfill other requirements like throughput. To address accuracy concerns, limiting $q \leq 32$ is a potential solution, albeit at the expense of increased clock cycles needed for computations in block-circulant MVMs. Furthermore, as shown in [39], compressed LSTMs utilizing block circulant matrices are proven to be asymptotically approach the performance of regular LSTMs. This suggests that compressed LSTM models demonstrate adequate resilience when integrating this matrix type into speech recognition tasks.

C. Formulation of Compressed LSTM Inference Using DA

As mentioned, all sub-circulant MVMs $\mathbf{W}^{m'}\mathbf{u}^m$ in (10) are calculated concurrently, allowing the circulant MVM to finish within p clock cycles, where $m' = (m-p)\text{mod}(p)+1$ and $1 \leq m \leq p$. In compact form, one may re-write (10) as

$$\mathbf{v}^1 = \mathbf{W}^1\mathbf{u}^1 + \sum_{m=2}^{p} \tilde{\mathbf{W}}^{m'}\mathbf{u}^m \text{ s.t. } \tilde{\mathbf{W}}^{m'} = \Phi\mathbf{W}^{m'} \tag{11}$$

Given the pre-execution of weight rotation within sub-circulant matrices $\mathbf{W}^m$ with Φ, particularly the upper triangular region of the block-circulant matrix $\mathbf{W}$, as shown in Fig. 3(b). This would allow to re-write (11) in scalar form as

$$\mathbf{v}^1 = \sum_{m=1}^{p} \mathbf{W}^{m'}\mathbf{u}^m = \sum_{m=1}^{p} v_n^{1m} \tag{12}$$

where v_n^{1m} is the nth sub-IP in sub-circulant MVM, $\mathbf{W}^{m'}\mathbf{u}^m$, with $1 \leq n \leq q$, is the row index of a sub-circulant matrix. Similar to (8), one may express v_n^{1m} as

$$v_n^{1m} = \mathbf{w}^{m'n}\mathbf{u}^m = \sum_{l=1}^{q} w_{l'}^{m'} u_l^m \tag{13}$$

where $l' = (n-l)\text{mod}(q)+1$, $\mathbf{w}^{m'n} = [w_{l'}, w_{l'+q}, w_{l'+2q}, \ldots, w_{l'+(m'-1)q}]$ with $w_{l'}^{m'} = w_{l'+(m'-1)q}$, $\mathbf{u}^m = [u_l, u_{l+q}, u_{l+2q}, \ldots, u_{l+(m-1)q}]$ with $u_l^m = u_{l+(m-1)q}$, and $1 \leq l \leq q$, is the column index of a sub-circulant matrix. For tractability, we provide specific cases to enhance understanding of (13).

- Block-circulant matrices reduce to a circulant matrix when $q = N$ and $p = 1$. This implies (13) represents a more generic form of (8).
- Each row of a circulant MVM corresponds to an IP. The weights are circular variants of one another, and the inputs are shared among all the IPs.

For conceptual clarity, we express the IPs of a circulant matrix as defined in (8), using DA. This approach can also be readily applied to the sub-IPs of sub-circulant MVMs within a block-circulant MVM.

Consider the row elements $w_{i'}$ of a circulant matrix $\mathbf{W}$ shown in Fig. 3(a), where $i' = (j-i)\text{mod}(N)+1$ and $1 \leq i, j \leq N$. In B-bit two's complement (TC) form, they can be given as

$$w_{i'} = -w_{i',B-1} + \sum_{k=0}^{B-2} w_{i',k}2^{-k}, \tag{14}$$

where $w_{i',k}$ is the kth-bit of $w_{i'}$. Substituting (14) into (8) and interchanging the order of summation, we obtain

$$v_j = -\sum_{i=1}^{N} u_i w_{i',B-1} + \sum_{k=0}^{B-2}\sum_{i=1}^{N} u_i w_{i',k}2^{-k} \tag{15}$$

Define

$$v_{i',k} = \sigma_k \sum_{i=1}^{N} u_i w_{i',k} \tag{16}$$

where $\sigma_k = (-1)^{\lfloor \frac{k}{B-1} \rfloor}$ denotes the sign of kth-bit in TC representation of $w_{i'}$. Substituting (16) into (15), we get

$$v_j = \sum_{k=0}^{B-1} v_{i',k}2^{-k} \tag{17}$$

Referring to (16), each $w_{i',k}$ falls within the range of $[0, 1]$, which corresponds to 2^N potential combinations for $v_{i',k}$. These combinations can be pre-computed and stored in a LUT of size 2^N. For instance, when $N = 2$, $v_{i',k}$ corresponds to 4 combinations, which are stored in a LUT as depicted in Fig. 3(c). The successive shift-and-accumulation of LUT contents for B clock cycles through a SA unit yields the result v_j. Similarly, the remaining IPs can be computed using DA to obtain the circulant MVM. The DA formulation for circulant MVM can be extended to encompass block-circulant MVM using (13).

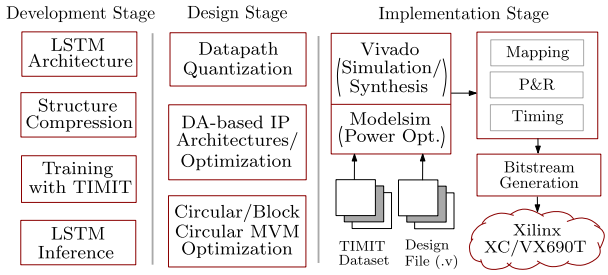

Fig. 5. Complete framework for the considered LSTM network.

III. DESIGNS OF COMPRESSED LSTM NETWORK

The considered framework for FPGA-based hardware acceleration of compressed LSTM network is illustrated in Fig. 5. It comprises of three stages, namely, development stage, design stage, and implementation stage. In the previous section, the LSTM model was developed using compressed circulant/block-circulant matrices [29], and then undergoes training utilizing the TIMIT dataset [38]. After training, the LSTM inference was formulated using DA. Additionally, the FP datapath quantization of LSTM inference needs to be considered with the quantized trained weight matrices. Subsequently, the design and optimization of the proposed architectures are carried out, aiming at FPGA-based hardware implementation based on the synthesis procedure shown in Fig. 5.

The system-level diagram of the compressed LSTM network based on the design stage of the considered framework

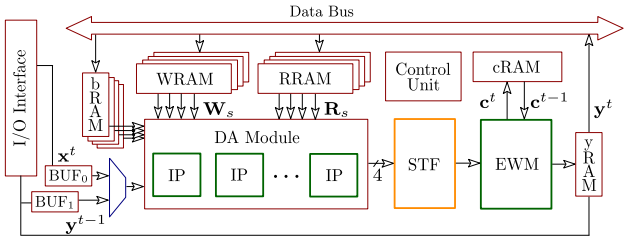

Fig. 6. System-level diagram of DA-based compressed LSTM network.

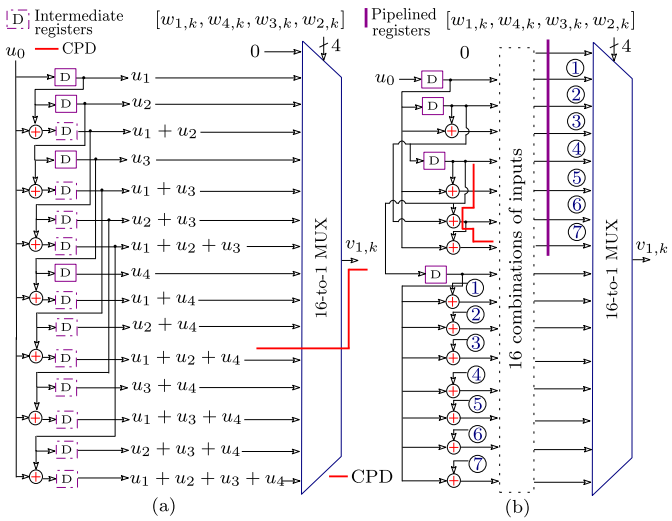

Fig. 7. LUT architectures for $N = 4$, (a) FASS and (b) FSAS methods.

is shown in Fig. 6. The MVMs are computed by the DA module, whereas the activation functions are calculated by sigmoid-tangent function (STF) units, and EWMs are computed by EWM units. In addition, it consists of an input/output interface unit that manages communication between the chip's external environment and the on-chip storage system through several buffers. The storage system comprises memory banks in the form of random access memory (RAM) such as bias RAMs (bRAMs), weights/recurrent connections RAMs (WRAMs/RRAMs), cell RAM (cRAM) and output RAM (yRAM) are used to store the bias terms, input weights/recurrent weights, cell information, and hidden states respectively. A ping-pong buffer (BUF) comprising two sub-buffers, BUF_0 and BUF_1 that work alternately for inputs and intermediate activations. The DA module computes arrays of MVMs for gates, each containing N IPs with LUT and SA units. DA module outputs to STF units for gates $\mathbf{i}^t$, $\mathbf{f}^t$, $\mathbf{o}^t$, and $\mathbf{z}^t$. The previous candidate cell $\mathbf{c}^{t-1}$ is retrieved from cRAM, processed by EWM units as per equation (5), and stored back. The multiplexer's select line and control logic needed in different components is generated by the control unit.

A. Datapath Quantization of DA-Based LSTM Network

In the DA formulation, we assume that inputs weights are already quantized to a target B bit width. However, practical LSTM accelerator implementation requires a more thorough analysis of FP considerations to efficiently execute with reduced FP precision. In an LSTM layer, obtaining an input vector $\mathbf{x}^t$ and a recurrent vector $\mathbf{y}^{t-1}$ is crucial for operation. The input vector $\mathbf{x}^t$ can come from another layer within the LSTM accelerator or an external source. For each gate, a minimum and maximum values of $\mathbf{W}_s/\mathbf{R}_s$ and $\mathbf{x}^t/\mathbf{y}^{t-1}$ products need to be determined using DA. However, this requires the execution of a trained LSTM accelerator with a set of inputs to produce the outputs e.g., to receive the utterances in speech recognition.

Quantization converts each floating-point element of input weight matrix $\mathbf{W}_s$ and recurrent weight matrix $\mathbf{R}_s$ to a B-bit width using a scale factor S. This scaling depends on whether quantized FP values are unsigned or signed integers. Given a representation of signed integers with B bits, S can be written as follows

$$S = (2^B - 1)/2\Delta \tag{18}$$

where Δ is the range of input data a, calculated as $\Delta = a_{\max} - a_{\min}$, with $a_{\max}$ and $a_{\min}$ are maximum and minimum values of a, respectively. To mitigate rounding errors during quantization of $\mathbf{W}_s$ and $\mathbf{R}_s$ elements, suitable representation of $S_a \cdot a_{\min}$ on the integer scale is needed. For hardware implementation, asymmetric quantization can be employed [40], shifting the range to $[a_{\min}, a_{\max}]$. The following subsections discuss the DA module for efficient implementation of circulant/block-circulant MVM.

B. DA-Based IP Architectures

In Fig. 3(c), the output of IP is produced after B clock cycles from SA unit, requiring B LUT-read operations using weight bit slices $w_{i',k}$ $(0 \leq k \leq B-1)$ as addresses for the LUT. This configuration presents two key challenges: the LUT size grows exponentially with N causing access delays, and considerable carry propagation occurs during successive shift-and-accumulation over B clock cycles. To overcome the speed limitations of SA units, we employ a carry-save adder based SA (CSA) unit [25]. In the following, various LUT architectures are proposed to alleviate access delay and complexity concerns.

1) LUT Architectures and Optimization: Traditional LUTs, typically constructed from static random access memory (SRAM), face performance limitations due to access delays, especially for larger N values. However, smaller N values can lead to faster access times and reduced LUT sizes, as shown in Fig. 3(c). Addressing this challenge for larger N values involves utilizing block-circulant matrices with a suitable choice on q. In these scenarios, LUT sizes exhibit exponential growth with q, where $q < N$ and $p > 1$. Although this mitigates the exponential dependency on N, it increases the computation time in terms of p number of clock cycles, subsequently decreasing the throughput of LSTM networks.

To tackle these issues, alternative LUT realization methods are necessary. One approach entails implementing LUT contents using sets of registers, adders, and multiplexers. Two primary methods are employed, depending on whether intermediate registers are used to store the LUT contents or not. The method involving intermediate registers is referred to

TABLE III
ADOPTION OF PIPELINING IN FASS VS. FSAS

CC	IPA	FASS	FSAS
1	$u_0 \rightarrow u_1$	LU$(u_1, 0, 0, 0)$	LU$(0, 0, 0)$
2	$u_1 \rightarrow u_2$	LU$(u_1, u_2, 0, 0)$	LU$(u_1, 0, 0)$
3	$u_2 \rightarrow u_3$	LU$(u_1, u_2, u_3, 0)$	LU$(u_1, u_2, 0)$
4	$u_3 \rightarrow u_4$	LU(u_1, u_2, u_3, u_4)	LU(u_1, u_2, u_3)
5	$u_4 \rightarrow u_5$	–	$D\{\text{LU}(u_1, u_2, u_3)\}+u_4$

CC: clock cycle, IPA: Input assignment, LU: LUT enumeration. $D\{\cdot\}$: delay operator.

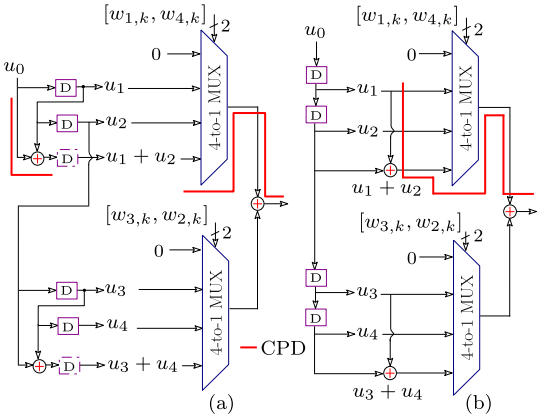

Fig. 8. LUT architectures for $N = 4$, (a) PASS and (b) PSAS methods.

as the full add-store-select (FASS) method, while the method without intermediate registers is known as the full store-add-select (FSAS) method. The term 'full' denotes the complete summation of LUT contents as defined in (16).

LUT architectures based on the FASS and FSAS methods are shown in Fig. 7(a) and Fig. 7(b), respectively. The FASS method generates LUT contents using a combination of adders and registers, followed by a multiplexer as the select unit. In contrast, the FSAS method forgoes intermediate registers but employs more adders. For any N, implementing an IP using the FASS method requires $(2^N - 1)$ registers, $(2^{N-1} - 1)$ adders, and a 2^N-to-1 multiplexer (or $(2^N - 1)$ 2-to-1 multiplexers). While implementing an IP using the FSAS method necessitates N registers, $(3 \cdot 2^{N-2} - 1)$ adders, and a 2^N-to-1 multiplexer.

The FASS method utilizes intermediate registers that inherently create a pipelined LUT structure without introducing clock cycle latency, as demonstrated in Fig. 7(a). In this approach, the output of the adder is instantly stored in intermediate registers, resulting in LUT content generation over 4 clock cycles for $N = 4$, as shown in the LUT enumeration (LU) in Table III.

On the other hand, the FSAS method lacks intermediate registers and relies solely on sample registers. To mitigate the critical path delay (CPD), we introduce pipelined registers in the initial half of the LUT contents for explanatory purposes. This arrangement corresponds to three input samples in the LU, as illustrated in Table III. However, the introduction of these pipelined registers creates a clock latency, which may not be suitable for applications with low latency requirements [41]. Additionally, the CPD of the FSAS method lacks the balance observed in the FASS method. A comparison of pipelining

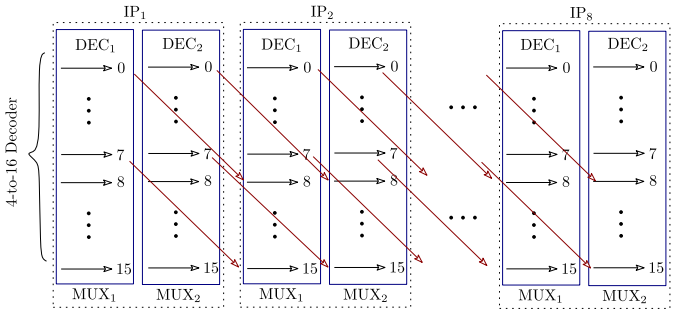

Fig. 9. Flowgraph illustrates the sharing of decoders for $N = 8$ and $P = 4$.

adoption between the FASS and FSAS methods is provided in Table II. Consequently, pipelining is applied to the FASS method, while the FSAS method is employed without pipelining.

While the access delay issue is addressed, the hardware complexities in both the FASS and FSAS methods increase exponentially with N. This challenge can be mitigated by expressing (16) through two summations:

$$v_{i',k} = \sigma_k \left(\sum_{i=1}^{P} u_i w_{i',k} + \sum_{i=P+1}^{N} u_i w_{i',k} \right) \tag{19}$$

where the first summation deals with the initial P weights and inputs for IP calculation at bit-level, while the second summation corresponds to an IP calculation of the remaining $(N - P)$ weights and inputs. This decomposition is 'partial' as it does not consider full summation, unlike (16). The corresponding implementation approaches are referred to as partial add-store-select (PASS) and partial store-add-select (PSAS) methods.

Within the PASS method, LUT complexities tied to the first summation encompass $(2^P - 1)$ registers, $(2^{P-1} - 1)$ adders, and a 2^P-to-1 multiplexer. While LUT complexities for the second summation involve $(2^{N-P} - 1)$ registers, $(2^{N-P-1} - 1)$ adders, and a 2^{N-P}-to-1 multiplexer. Additionally, an explicit adder is required to sum up the outputs from the sub-LUTs corresponding to the two summations. Consequently, the sub-LUT complexities encompass $(2^{N-P} + 2^P - 2)$ registers, $(2^{N-P-1} + 2^{P-1} - 2)$ adders, and $(2^{N-P} + 2^P - 2)$ 2-to-1 multiplexers. A similar analysis applies to the PSAS method.

Importantly, a smaller value of P reduces the complexity of the first LUT but increases the complexity of the second LUT. To determine the optimal sub-LUT complexities, differentiation of their combined complexities involving registers, adders, and multiplexers with respect to P is necessary. Upon simplification, both the PASS and PSAS methods exhibit optimized complexities at $P = N/2$. This indicates that sub-LUTs related to an IP would share the same size, as depicted in (19). Hence, the simplified LUT complexities are calculated as $2 \cdot (2^{N/2} - 1)$ registers, $2 \cdot (2^{N/2-1} - 1)$ adders, and $2 \cdot (2^{N/2})$-to-1 multiplexers. In line with the FASS and FSAS methods, individual inputs are initially stored in a set of sample registers. Subsequently, partial products are computed via adders and are either routed through intermediate registers or not. They are then selected using a multiplexer, with the select lines corresponding to weight bit-slices. The respective

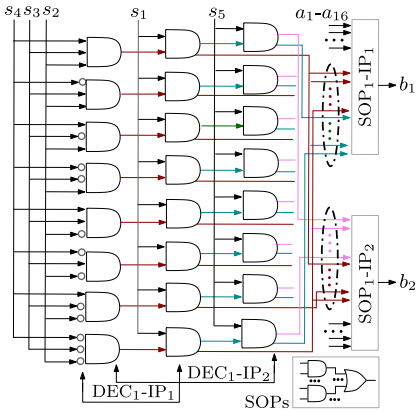

Fig. 10. Gate-level design of reformulated decoder for $N = 2P = 8$.

architectures based on the PASS and PSAS methods are depicted in Fig. 8(a) and Fig. 8(b), respectively.

When compared to the full methods, implementing the proposed architectures could result in reduced LUT resource usage. For instance, the total LUT words that could be saved for $N = 8$ amount to, $1 - (2 \cdot 2^{N/2} \cdot N/2^N \cdot N) = 87.5\%$. Notably, the CPD for the PASS method is shorter, as the LUT contents are first computed and then passed through registers before being selected through two smaller multiplexers.

2) Optimization of Select Units: In the previous subsection, we explored optimizing LUTs by considering registers, adders, and multiplexers. However, there is even more room for architecture optimization, particularly within the select units. This possibility arises from the circular nature of the select lines, resulting in redundancies across multiple select units.

For example, let's take the FASS method with $N = 4$, employing a 16-to-1 multiplexer as the select unit, denoted as s_4, s_3, s_2, s_1. In the case of the remaining rows in an MVM, which corresponds to three separate 16-to-1 multiplexers, the select lines would be s_1, s_4, s_3, s_2, s_2, s_1, s_4, s_3, and s_3, s_2, s_1, s_4. This clearly opens up the potential for sharing sum-of-product (SOP) expressions among these different select units. However, in the context of partial methods, achieving this is not as straightforward due to the division of the LUT into two sub-LUTs. This concept is further elaborated below.

The Boolean expressions of select units also enable the exploitation of redundancies in partial methods. Let's take the example of $N = 8$, which corresponds to 8 IPs within the MVM, designated as IP$_{1-8}$. In each IP, we encounter two 16-to-1 multiplexers, referred to as MUX$_1$ and MUX$_2$, serving as the select units. MUX$_1$ is linked to inputs a_1–a_{16} and employs select lines s_4, s_3, s_2, s_1. On the other hand, MUX$_2$ also uses the same inputs, but with distinct select lines s_8, s_7, s_6, s_5. Consequently, these multiplexers produce outputs b_1 and b_2 through SOP expressions. It is important to note that, due to the LUT decomposition, the select lines for these multiplexers do not share circularly shifted versions of each other. Mathematically, the Boolean expressions corresponding to b_1 and b_2 can be generally expressed as:

$$b_1 = \phi(\cdot)a_{r_1} \text{ and } r_1 = \sum_{l=0}^{N/2-1} s_{l+1}2^l \tag{20}$$

$$b_2 = \phi(\cdot)a_{r_2} \text{ and } r_2 = \sum_{l=N/2}^{N-1} s_{l+1}2^{l-N/2} \tag{21}$$

where $\phi(\cdot)$ maps any variable g using the following relation

$$\phi(g) = \begin{cases} 1, & \text{if } g = r_i \ (i = 1, 2, \ldots, N) \\ 0 & \text{otherwise} \end{cases}$$

From (20) and (21), it is evident that $N/2 - 1$ common select lines for the MUXs can be shared. In each successive sub-IP, a new select line is introduced while an old one is discarded. The flow graph in Fig. 9 illustrates the binary combinations of select lines between consecutive sub-IPs. These combinations serve as outputs from the decoder corresponding to the select lines. Notably, the first half of the binary combination from the previous sub-IP aligns with the latter half of the binary combination from the next sub-IP. Consequently, it is possible to reformulate the decoder design, accommodating $2^{N/2} - 1$ binary combinations. While redundant decoder outputs can be generated once, uncommon select lines need explicit generation. The remaining decoder outputs for each sub-IP are produced through an array of $2^{N/2-1}$ AND gates.

The reformulated decoder for $N = 8$ and $N = 4$ is shown in Fig. 10. The common select lines s_4, s_3, s_2 for MUX$_1$ in sub-IP$_1$ and sub-IP$_2$ are generated through a 3-input AND gate array. The output of these gates is then directed to a 2-input AND gate array using select line s_1 of MUX$_1$ in sub-IP$_1$, as well as select line s_5 of MUX$_1$ in sub-IP$_2$. These components collaboratively construct an optimized select unit through a SOP expression.

Compared to a direct approach, the reformulated decoder implementation significantly reduces the number of required AND gates. The saved number of AND gates is computed as $\{2^{N/2-1}(N/2+1)\}/\{2^{N/2}(N/2-1)\}$.

3) Proposed Circulant MVM Architecture: The proposed circulant MVM architectures based on full and partial methods are depicted in Fig. 11. To clarify, we focus on the circulant MVM architecture for $N = 4$ using the PASS method. The pipelining is applied in the FASS and PASS methods as indicated by brown-colored boxes, while absent in the FSAS and PSAS methods. For $N = 4$, the circulant MVM structure consists of four IPs within the DA module, sharing a unit of split LUTs, referred to as PASS$_1$ and PASS$_2$ in Fig 11. This optimization leverages shared sub-LUTs for IP$_1$, resulting in significant resource savings during MVM realization.

The selection process employs eight distinct $2^{N/2}$-to-1 multiplexers, with select lines derived from bit-slices of circular weights through four parallel-in serial-out (PISO) registers. PISOs and CSAs work concurrently every B clock cycles. Between IP modules, three circular shifts (CSs) composed of shift registers facilitate weight rotation. These CS units ensure correct circular shifts. Subsequently, an array of four CSAs generates circulant MVM output, denoted as v_j. Lastly, an array of accumulator units (AUs) adds bias terms and recurrent MVMs. For clarity, we call the combined CSA and AU stages as the accumulation stage (AS).

As per (1)–(4), the DA module computes MVM by either processing inputs $\mathbf{x}^t$ or previous outputs $\mathbf{y}^{t-1}$ with parameter

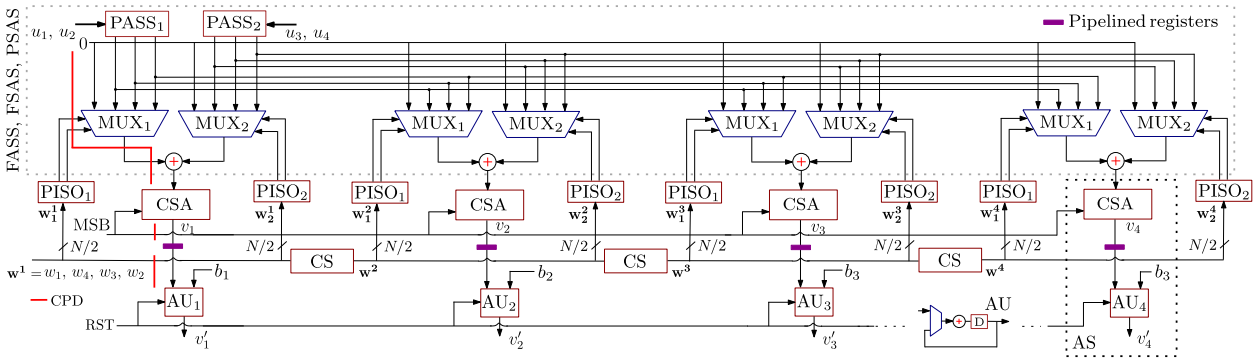

Fig. 11. Circulant MVM architectures for $N = 4$ based on full/partial methods. Pipelined registers are present for FASS/PASS.

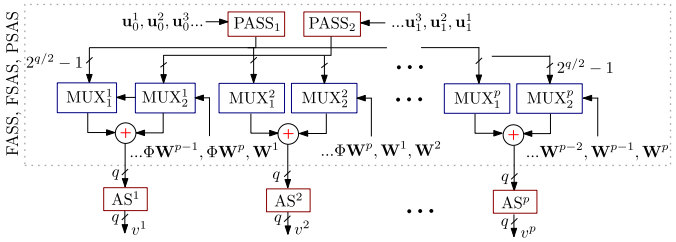

Fig. 12. Block-circulant MVM architectures based on full/partial methods.

matrices $\mathbf{W}s$ or $\mathbf{R}s$ ($s = z, i, f, o$) respectively. These outcomes are then combined with their respective bias terms $\mathbf{b}_s$, according to

$$\mathbf{v}_j = \begin{cases} \mathbf{v}_j + \mathbf{t}^j\mathbf{z} & \text{if RST} = 0 \\ \mathbf{v}_j + \mathbf{b}_s & \text{otherwise} \end{cases} \tag{22}$$

where $\mathbf{t}, \mathbf{z} = \mathbf{w}, \mathbf{x}$ or $\mathbf{r}, \mathbf{y}$. Subsequently, they are passed through activation functions within the STF unit, generating gate outputs $\mathbf{s}^t$ and the candidate memory $\mathbf{c}^t$.

4) Block Circulant MVM Architecture: As mentioned in Section II.C, the computation of a sub-circulant MVMs can be performed concurrently. As a result, the entire block-circulant MVM can be executed using both full and partial methods in p clock cycles. In the case of full methods, the inputs $\mathbf{u}^m$ and the row weight vector $\mathbf{w}^{m'n}$ undergo processing. Conversely, in the partial methods, the inputs $\mathbf{u}^m$ and the corresponding row weight vector $\mathbf{w}^{m'n}$ are split into two halves and then processed independently, similar to the one shown in Fig. 8.

The block circulant MVM constructed using the PASS method encompasses p CS units, each with $(q-1)$ registers, p PISO units, each having q registers, p select units with $2\cdot(2^{q/2}-1)$ 2-to-1 multiplexers, p sub-LUTs, each containing $2\cdot(2^{q/2-1}-1)$ adders and $2\cdot(2^{q/2}-1)$ registers, p adders to sum up sub-LUT outputs, an array of p CSA units, each comprising q CSA subunits with B CSFAs and $2B$ registers (at the bit level) followed by a carry propagate adder. Finally, there is an array of p AUs, each containing q AUs equipped with an adder, a register, and a 2-to-1 multiplexer.

Fig. 12 illustrates a generic architecture for the proposed block-circulant MVM applicable to any values of p and q. Unlike (19), the number of combinations required for LUT generation varies with q, which is evident in Fig. 12 where the sub-LUT size changes with q. Notably, as q increases, the sub-LUT sizes and associated complexities also increase. To address this, the approach of dividing q into smaller blocks, represented as $q = p_s \times q_s$ through multi-partial decompositions, can be utilized. Consequently, p sub-LUTs, employing (19), can be expressed as follows:

$$v_{i,k} = \sigma_k\left(\sum_{r=0}^{p_s-1}\left(\sum_{i=rq_s}^{Pq_s-1} u_i w_{i',k} + \sum_{i=Pq_s}^{(r+1)q_s-1} u_i w_{i',k}\right)\right) \tag{23}$$

Evidently, the dimensions of sub-LUTs, which depend on q, can be aligned with distinct functions denoted as q_s. This becomes particularly valuable when dealing with the need for a larger kernel sizes in the design of an LSTM layer.

IV. Implementation Results and Discussion

A. Computational Complexities of DA-Based LSTMs

For clarity in the discussion, the existing DA-based design in [17] is referred to as DA-LSTM. Fig. 13(a)–(c) illustrates the hardware complexities of both proposed and DA-LSTM designs, taking into account adders, registers, and multiplexers for $q \in \{2, 4, 8\}$ and $N \in \{128, 256\}$. Circulant/block-circulant matrices have been leveraged by the DA-based different architectures to streamline memory access and hardware intricacies. However, DA-LSTM adopted the OBC scheme for hardware implementation, while the proposed architectures exclusively employed the TC scheme for both full and partial methods.

It is evident from Fig. 13(a)–(c) that all the proposed architectures have notably reduced hardware complexities for LSTM in comparison to DA-LSTM. For instance, the FASS and FSAS methods curtail adder and register costs, albeit with an approximately two-fold rise in 2-to-1 multiplexer count. Yet, these differences are significantly mitigated in the PASS and PSAS methods, particularly for larger kernel sizes. This improvement stems from the sharing of two sub-LUTs across multiple small select units. Further hardware expense reduction is achieved through SOP optimization within select units and decoder reformulation.

Unlike DA-LSTM, the growth of pipelined registers in the FASS and PASS methods is not exponential with the kernel size q, given their inherent pipelining through intermediate registers. For example, when contrasted with DA-LSTM, the proposed architecture based on the FASS method for $N = 128$ and $q = 8$, yields 83.70% fewer adders and

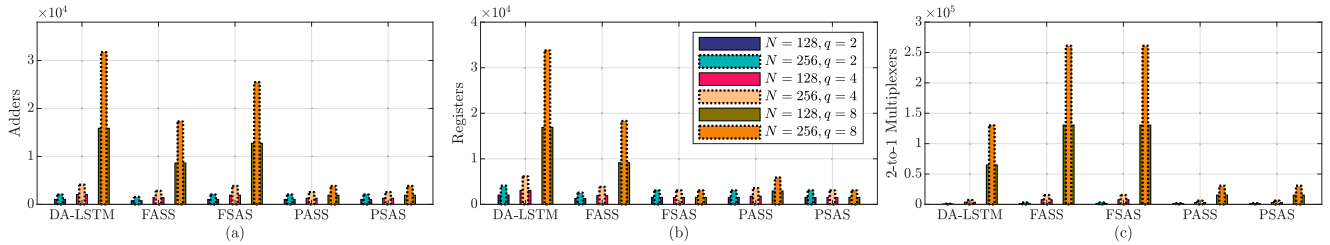

Fig. 13. Illustration of hardware complexities of different DA-based LSTM architectures for $q \in \{2, 4, 8\}$ in terms of (a) Adders, (b) Registers, and (c) 2-to-1 Multiplexers, with $N = 128$ (□) and $N = 256$ (⬚).

TABLE IV
CRITICAL PATH DELAY OF DIFFERENT DA-BASED LSTM DESIGNS

Design	Critical Path Delay
[17]†	$\max[BT_0, T_{BA}]; T_0 = (q-1)T_{MX} + 2T_X + T_D + T_{FA}$
FASS‡	$\max[BT_1, T_{BA}]; T_1 = qT_{MX} + T_X + T_D + T_{FA}$
FSAS‡	$\max[BT_2, T_{BA}]; T_2 = qT_{MX} + (q-1)T_{MX} + T_X + T_D + T_{FA}$
PASS‡	$\max[BT_3, T_{BA}]; T_3 = (q/2)T_{MX} + T_A + T_X + T_D + T_{FA}$
PSAS‡	$\max[BT_4, T_{BA}]; T_4 = (q/2)T_{MX} + (q/2+1)T_A + T_X + T_D + T_{FA}$

†: employed OBC, ‡: employed TC and $T_{BA} = 2T_M + 2T_{STF} + T_A$ with T_{STF}, T_A, T_{MX}, T_D, T_{FA} and T_X, are the computational delays of sigmoid/tangent hyperbolic function, adder, 2-to-1 multiplexer, register, full-adder and XOR gate respectively.

84.62% fewer registers. Similarly, based on the FSAS method, it offers 24.62% fewer adders and a 10× reduction in registers. The PASS method boasts 7.27× fewer adders, 4.73× fewer registers, and 3.23× fewer 2-to-1 multiplexers. Lastly, the PSAS method yields 7.27× fewer adders, 10× fewer registers, and 3.23× fewer 2-to-1 multiplexers as compared to DA-LSTM.

To ensure a fair comparison, CPD is evaluated for each design. DA-LSTM used fine-grained pipelining to minimize CPD to a single adder delay. In contrast, proposed FASS/PASS architectures achieve the same CPD using intermediate registers, avoiding fine-grained pipelining, as illustrated in Table IV. However, FSAS/PSAS architectures, unlike FASS/PASS, are non-pipelined due to their unsuitability for pipelining.

All designs exhibit two CPD components: one for MVM and another for activation functions and EWMs. The CPD corresponding to MVM is different for all the designs, while the CPD belongs to activation functions and EWMs are same for all the designs. For instance, CPD for the LUT architecture for MVM computation based on the PASS method in Fig. 8(a) is $(q/2)T_{MX} + T_A$, exceeding adder delay between intermediate registers. Additionally, the CSA unit has CPD $T_X + T_{FA} + T_D$, with T_{MX}, T_X, T_{FA}, and T_D representing computational delays of 2-to-1 multiplexers, XOR gate, full adder, and flip-flop respectively. Sign inversion via XOR before CSA introduces T_X delay, as illustrated in Table IV. The remaining CPD component includes computational delays of sigmoid/tangent activation functions ($2T_{STF}$), two EWMs ($2T_M$), and one adder (T_A) as denoted by T_{BA}. Note the first CPD component involves a factor of B due to bit-serial DA computations. It reduces sample rate by B clock cycles, however by employing a $B\times$ faster clock for the CSA unit can overcome this problem [25].

B. Design Considerations

The implementation methodology of the proposed architectures on a Xilinx Virtex XC/VX690T FPGA is depicted in Fig. 5. All the architectures, coded in Verilog HDL (.v) format, including trained weights, are imported into Xilinx Vivado, resulting in the translation into a netlist in Xilinx netlist format (.xnf). The functional verification is then executed using the TIMIT dataset within a simulator. The mapping allocates the netlist to available FPGA resources, followed by the place and route (PnR) process, which fixes locations and leverages routing resources.

For accurate power consumption analysis, the Xilinx Power Analyzer is employed. This tool configures clock frequencies of FPGA resources to measure power consumption at the PnR level. Timing details and clock frequencies are integrated into a power constraint file. Simulated data from ModelSim (in.vcd and .saif formats) is imported to provide toggling information on nets. Coupled with interconnection delay details, this facilitates more accurate netlist refinement through timing analysis (back-annotation). Ultimately, a bitstream file is generated for each proposed architecture, enabling configuration data transfer to the XC/VX690T FPGA device.

Since the proposed architectures rely on DA, modifications are needed for calculating throughput in the block-circulant formulation [13]. The SA unit requires B clock cycles to process each sample more rapidly. Thus, the throughput (TP) is formulated as follows:

$$\text{TP} = (N \times f_{\text{clk}})/(p^2 \times B) = (q \times f_{\text{clk}})/(p \times B) \qquad (24)$$

where $q = N/p$. The DA-based LSTM architecture maintains consistent TP across varying selections of p and B within specific N and f_{clk} settings. For instance, considering an LSTM model with $N = 128$ and a clock rate of 200 MHz, the TP remains at 6.4 Gsps whether employing $p = 2$ and $B = 1$, or $p = 1$ and $B = 4$. This adaptability is valuable for hardware designers seeking to balance performance with different p and B configurations.

In our assessment, we consider model sizes of $N \in \{128, 256\}$, chosen to match the data, given the lack of specific guidelines for N determination. In contrast, the kernel size $q \in \{2, 4, 8\}$ can be selected based on desired accuracy, as outlined in Table I. The wordlength B for parameters/inputs is determined considering throughput and quantization performance. We opt for $B = 8$ for both inputs and weights, in alignment with (18), ensuring reasonable accuracy and performance [15]. However, a higher B can be considered at the expense of increased implementation cost.

FPGA-based LSTM designs adhere to constraints imposed by logic resources, including slice LUTs (SLUTs)/flip-flops (FFs) and DSPs, with block RAMs (BRAMs) serving as

memory resources. FPGA DSPs, being costly and limited, primarily support LSTM computations involving multipliers. Since the proposed architectures handle MVMs and EWMs, FPGA-based EWMs are realized through DSP slices. Similar to (10), EWMs in (5) and (6) can be computed in blocks. For example, the $3q$ EWMs can be shared and processed in p clock cycles. Therefore, the usage of DSP slices can be significantly reduced, promoting an efficient FPGA implementation. The proposed architectures, devoid of MVM multipliers, exclusively utilize DSP slices for EWMs. Fixed-point adders, multiplexers, and registers within the architectures are implemented using slice LUTs/FFs, while parameters of compressed LSTMs are stored in BRAMs.

C. Performance Evaluation

1) Maximum Clock Frequency and Throughput: In this section, we delve into the achievable clock frequency and throughput of our proposed architectures. Importantly, each architecture maintains a safe margin from timing violations in terms of its maximum clock frequency. Yet, with an increasing q, the maximum clock frequency experiences a decline due to the heightened complexity of the DA module for larger q, as per the CPD estimation listed in Table IV. This decrease is more prominent for larger q values, illustrated by distinct markers in Fig. 14. This trend is particularly evident in FASS, FSAS, PASS, and PSAS methods, where the computational delay of adders/multiplexers dominates the CPD. This effect is more pronounced in non-pipelined methods such as FSAS and PSAS. For example, the clock frequency of the FSAS method at $N = 128$ with $q = 2$ is 308.59 MHz, whereas at $q = 8$, it drops to 145.83 MHz.

Conversely, pipelining with additional registers reduces the CPD for FASS and PASS methods, resulting in a higher maximum clock frequency for the FASS method compared to the PASS method. Moreover, smaller model sizes exhibit relatively higher clock frequencies, as indicated by the markers and exemplified by the maximum clock frequencies for $N = 128$ and 256 in Fig. 14.

In terms of the considered design parameters, the FASS method achieves the highest speed at $N = 128$ and $q = 2$, while the slowest architecture is based on the FSAS method at $N = 256$ and $q = 8$. Corresponding to the maximum clock frequency of each architecture, peak TP is also depicted in Fig. 14. Evidently, architectures based on FASS/PASS methods exhibit higher throughput than FSAS/PSAS counterparts, benefiting from their pipelined nature. Notably, the throughput of the proposed architectures increases for higher q values and decreases for lower q values. Additionally, for a fixed p, increasing N enhances the throughput, in accordance with (24).

2) SLUT and FF Usage: The hardware utilization of the proposed architectures in terms of SLUTs and FFs is shown in Fig. 15(a) and Fig. 15(b), respectively. The hardware components of each architecture are distributed across multiple SLUTs and FFs. To assess chip area, SLUT and FF usage serves as valuable metrics. While weight storage remains uniform across the proposed architectures, resource utilization

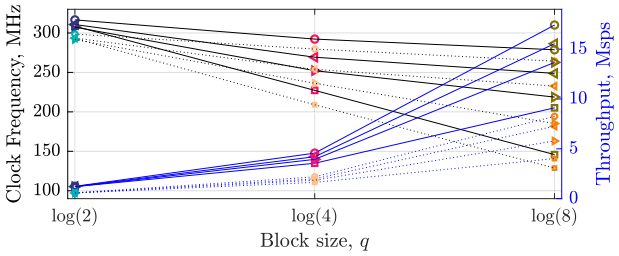

Fig. 14. Variation of the maximum clock frequency and throughput of the proposed architectures for $N = 128$ (solid line) and $N = 256$ (dashed line) with $q \in \{2, 4, 8\}$, FASS (○), FSAS (□), PASS (◁) and PSAS (▷). Note that the colors of the markers are consistent with the legends in Fig. 13(b).

in SLUTs and FFs varies significantly due to computational complexity.

For example, the proposed PASS and PSAS methods display notably lower SLUT and FF usage compared to the FASS and FSAS methods. It may be noted that the SLUT/FF usage increases with higher q values for each architecture. This correlation arises from the influence of kernel size on the complexity of FASS/FSAS LUTs, involving adders, registers, and multiplexers, in contrast to PASS/PSAS LUTs. Reducing this dependency considerably decreases SLUT/FF usage.

Notably, as $q \geq 4$, FASS/FSAS methods may demand considerably more adders, registers, and multiplexers, thereby consuming more SLUT/FF logic resources. To mitigate this, a multi-partial strategy, as outlined in (23), can be adopted. This strategy curbs the exponential SLUT growth by utilizing sub-blocks. For instance, for a kernel size of $q = 8$, an efficient realization could involve $q_s = 4$ and $p_s = 2$, optimizing FPGA logic resource utilization. Fig. 15(a) and Fig. 15(b) also demonstrate that SLUT and FF usage approximately 1.6× with a doubling of q.

3) BRAMs, DSP Slices, and Power Consumption: While circulant/block-circulant matrices compress the LSTM, the simplicity stems from their nature: each matrix value is used once, allowing BRAM-localized values and straightforward address generation. Yet, limited BRAM size constrains overall design. Shrinking matrices significantly cuts area and power. Hence, circulant/block-circulant MVM is embraced, storing only primitive vector weights in BRAMs for sub-circulant MVMs.

All proposed architectures share uniform memory requirements, yet varying hardware element counts contribute to differences in computational costs. Increasing the kernel size allow more compression, as discussed earlier, yields a minor decline in network accuracy, showcased in Table I. This is validated through FPGA implementation as seen in Table V. Despite slight network accuracy reduction with TIMIT dataset, doubling the kernel size results in a two-fold increase in throughput.

DSP slices efficiently handle block EWM operations, each aligning with block circulant MVMs and completing within p clock cycles. DSP slice utilization remains consistent for $N = 128$ and $N = 256$ in Table V, despite halved clock cycles in the former. Efficient block EWM execution, particularly for larger models, preserves DSP slices, notably influencing overall power consumption.

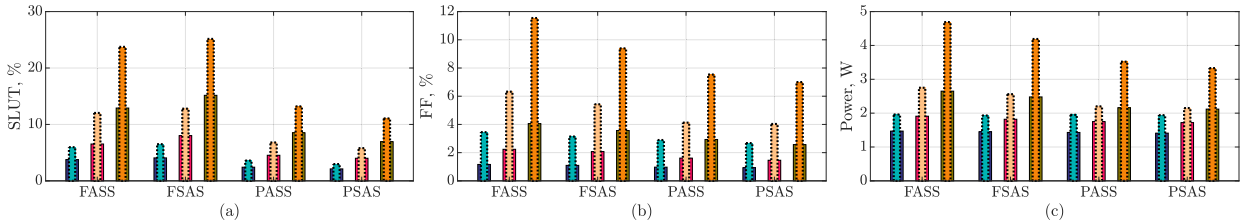

Fig. 15. Illustration of FPGA utilization of the proposed architectures. (a) SLUTs, (b) FFs, and (c) Power consumption. Note that the bar plots hold the same meaning as depicted in Fig. 13(b).

TABLE V
BRAMs AND DSP SLICES USAGE OF THE PROPOSED ARCHITECTURES

Model size (N)	128			256		
kernel size (q)	2	4	8	2	4	8
BRAM (%)	5.13	3.26	2.33	20.99	13.53	9.81
DSP (%)	0.17	0.34	0.68	0.17	0.34	0.68

DSP utilization for $N = 128, 256$ are same due to the usage of same multipliers in EWM for p clock cycles.

Smaller kernel size LSTM models maintain comparable PER performance for accuracy, yet accommodate more parameters in memory buffers. This abundance leads to higher memory access frequency and increased power consumption. Conversely, larger kernel sizes consolidate parameter access within CS units for multiple samples, minimizing memory accesses and subsequent power consumption. Computational elements further impact power consumption; for instance, FASS/FSAS-based architecture incorporates more adders, registers, and multiplexers than PASS/PSAS-based, resulting in higher power consumption. Remarkably, FSAS/PSAS consumes less power than FASS/PASS due to fewer registers with larger kernel sizes. Fig. 15(c) depicts power consumption across proposed architectures, illustrating that doubling kernel sizes does not equate a similar increase in power, particularly at lower kernel sizes due to the aforementioned factors. Notably, this trend diverges from LUT/FF usages.

D. Comparison With Previous Works

For a comprehensive evaluation, we compare the proposed architectures with leading accelerators [12], [14], [15], [18], [19], [20] in Table VI. It covers model sizes, bit-precision, PERs, FPGA platforms, resource usage (DSP slices, BRAMs, slice LUTs, and FFs), clock frequency, multiply-and-accumulate units (MACs), throughput, performance, power consumption, and power efficiency.

ESE [12] employs weight pruning for high-throughput LSTM inference. DeltaRNN [18] exploits temporal sparsity to enhance gated recurrent unit speed. C-LSTM [14] and E-RNN [19] utilize structured matrices with a low-cost FFT during inference. In contrast, E-LSTM [15] and BBS [20] use structured pruning for fine-grained workload balance, setting them apart from ESE. While Spartus [21] exploits spatio-temporal sparsity in LSTM using column-balanced targeted dropout. The design was tested on the TIMIT dataset with $N = 1024$ and a batch size of 1. C-LSTM and DA-LSTM tested with $N = 512$ across various platforms. E-RNN and BBS utilized a Google LSTM with an identical neuron count due to peephole connections and projection layers, resulting in reduced dimensions for both recurrent and output connections. This reduction consequently impacts the number of network parameters.

With access to the XC/VX69V0T FPGA device, we re-synthesized the DA-LSTM at a clock frequency of 200 MHz, ensuring an equitable comparison with the suggested architectures. Furthermore, a kernel $q = 16$ was retained to establish consistency with similar design types [12], [14], [15], [17]. The corresponding results are displayed in an additional column within Table VI. Given that most FPGA-based LSTM accelerators were implemented for $N = 512$ or 1024, it is reasonable to compare the proposed architectures within this scope. As discussed, the proposed architectures are adaptable to various throughput, accuracy, and computational demands, we deliberate on $N = 512$ and 1024 with $q = 16$ and 4 respectively, to discern the results.

The proposed architectures use circulant/block-circulant matrices for network compression in fixed-point quantized scenarios. With the batch size matching the kernel size for training, and random parameter initialization employed, the considered training strategy yields the highest PER, particularly with larger kernel sizes, as listed in Table I. However, it improves the throughput performance (24) and model compression [34]. This is consistent with the results presented in Table VI, where a smaller kernel (with $N = 1024$) yields improved PER, and vice versa. However, there is a trade-off in terms of throughput and BRAM usage. Higher kernels are implemented based on our multi-partial strategy (23) using pq adder trees of depth $\log_2(p_s)$ to reduce the logic utilization. Compared with $N = 512$ and $q = 16$, the logic utilization of the proposed architectures does not double even for $N = 1024$ with $q = 4$, as these architectures do not require adder trees to split higher kernels into smaller ones and allow more optimization of FPGA resources. While a smaller value of q reduces PER, power consumption increases due to the higher number of BRAM accesses and clock cycles. This increase in power consumption is partially compensated by complexity reduction. This means that the proposed approach can be applied to various design parameters depending on requirements.

As evident from Table VI, the proposed architectures exhibit low usages of DSP, SLUT, FF, and power consumption due to their bit-serial nature. Several factors are responsible for the contribution of these outcomes:

- The utilization of compression with circulant/block-circulant matrices eradicates structured irregularities.

TABLE VI
PERFORMANCE COMPARISON OF THE PROPOSED AND STATE-OF-THE-ART LSTM ACCELERATORS ON FPGA

Design	ESE [12]	Del. RNN [18]	[14]	E-RNN [19]	BBS [20]	DA-LSTM [17]		[15]	Spartus [21]	FASS	FSAS	PASS	PSAS
Model size (N)	128†	128†	512†	1024†	1024†	512*		1024†	1024	512*/1024**			
Bit Precision	INT16/12	INT16/16	INT16/16	INT16/16	INT16/16	INT8/8		INT8/8	INT16/8	INT8/8			
PER on TIMIT	20.7	–	24.6	20.3	23.6	–		23.2	21.8±0.3	24.87/23.22			
FPGA Platform	XCKU060	XC/Z100	7V3	XC/VX690T	GX1150	XCKU060	XC/VX690T	SX660	XC7Z100	XC/VX690T			
DSP (%)	54.5	38.0	74.3	79.6	100	55.65	42.67	1.40	25.7	1.33/0.33			
BRAM (%)	87.7	60.6	65.7	65.2	92	–	31.73	32.1	33.0	31.73/50.07			
SLUT (%)	88.6	94.2	58.7	59.4	68	59.75	43.38	87.8	49.2	31.24/48.12	36.05/56.89	24.56/39.93	21.45/32.67
FF (%)	68.3	21.5	46.5	55.3	–	44.10	32.67	15.6	19.5	21.17/30.35	17.61/25.64	16.49/22.72	13.32/16.88
Freq. (MHz)	200	125	200	200	200	100	200	200	200	200/200			
#MACs	32	768	128	128	4096	1536	1536	128	512	48/12			
TP (Msps)	12.8	192	51.2	51.2	1638.4	6.25	12.5	51.2	204.8	12.5/0.39			
Perf. (GOP/s)	78.6	1198.0	714.3	783.1	2432	16.8	25.85	403.3	9447.8	25.85/1.61			
Power (W)	41.0	7.3	23.0	25.0	19.1	–	21.8	15.9	8.4	7.7/9.5	7.4/9.1	5.1/6.5	4.6/5.9
Eff. (GOP/s/W)	1.9	164.1	31.1	31.3	127.4	–	1.54	25.4	1124.7	3.36/0.17	3.49/0.18	5.06/0.25	5.62/0.27

Perf.: Performance, Eff.: Power Efficiency. † : reported results of the existing designs, * : $q = 16$ with $q_s = 4$, $p_s = 4$ exploits multi-partial strategy for efficient implementation of designs on FPGA, as per (23), ** : $q = 4$. Performance is estimated by the formula [42]: $2 \times$ #Parameters$\times$ TP/N. In DA-LSTM [17], EWMs were not realized with block EWMs but rather implemented all on DSP slices. The maximum clock frequency for DA-LSTM [17] with $N = 512$ and $q = 16$ (with $q_s = 4$, $p_s = 4$) is 212.54 MHz, whereas for FASS, FSAS, PASS, and PSAS, these are 208.86 MHz, 139.24 MHz, 198.91 MHz, and 171.43 MHz respectively

- Employing low-cost MVM implementation through DA, utilizing both full and partial methods while sacrificing throughput.
- Sharing of LUT contents among hardware elements for multiple IP operations within MVM.
- Reduced memory accesses and increased parallelism, are advantageous from an FPGA perspective.
- Implementation of block EWM operations to curtail the usage of DSP slices across all proposed architectures.
- Storage of the complete model, inclusive of weight matrices, in on-chip BRAMs, ensuring the proposed architectures are not constrained by memory limitations.

Compared with DA-LSTM, all proposed architectures leverage block EWM operations with minimal multipliers, maintaining performance while utilizing 96.88% DSP resources. Specifically, the FASS, FSAS, PASS, and PSAS-based architectures demonstrate reductions of 27.98%, 16.89%, 43.38%, and 50.55% in SLUTs, and 35.20%, 46.09%, 49.52%, and 59.23% in FFs, respectively. These architectures also experience power consumption reductions of 64.68%, 66.05%, 77.61%, and 78.89%, while achieving enhanced power efficiency improvements of 2.83×, 2.95×, 4.27×, and 4.74×, respectively, compared to DA-LSTM. Moreover, the maximum achievable frequency of DA-LSTM and FASS are almost the same, as listed in the footnote of Table VI. This is because the architecture based on the FASS method is inherently pipelined, whereas DA-LSTM utilizes fine-grained pipelining. Evidently, these findings are consistent with the CPD estimation outlined in Table IV. The Spartus [21] is designed to achieve high throughput performance and efficiency. In contrast, the proposed architectures focus on low-area and low-power LSTM implementation based on DA. For instance, the proposed PSAS-based architectures demonstrate reductions of 98.71% in DSPs, 33.59% in SLUTs, 13.43% in FFs, and 29.76% in power consumption at the expense of BRAM usage, throughput, and performance as compared to Spartus. Furthermore, the primary reason for the notable reduction in DSP slices in the proposed architectures, as compared to Spartus, is the deliberate reduction in the use of MACs in both MVMs and EWMs. It is interesting to note that these gains can be enhanced or reduced for the desired requirements of accuracy, throughput, and computational cost.

V. CONCLUSION

In this paper, we introduced four LSTM architectures using DA, capitalizing on circulant and block-circulant MVMs for network compression. The investigation explored a quantized weights-based methodology for training circulant/block-circulant matrices. Formulating their fixed-point MVMs using DA, we assessed the influence of kernel size on accuracy. Additionally, we presented methods for generating LUT contents, employing full and partial approaches of add-store/store-add, followed by selecting units. This led to more efficient circulant/block-circulant MVM calculations. To manage complexity with larger kernels, a streamlined multi-partial strategy was employed, optimizing decoders for select units. While pipelining in the add-store method boosts processing speed, it slightly increases pipelined registers. Remarkably, FPGA evaluations highlighted the superiority of proposed architectures, yielding significant reductions over [17], [21]. This investigation establishes a foundation for advancing more efficient DA-based implementations tailored for speech recognition applications.

ACKNOWLEDGMENT

The authors extend their gratitude to the associate editor and the reviewers for their valuable insights and suggestions, which have significantly contributed to enhancing the quality of the manuscript.

REFERENCES

[1] K. K. Parhi and N. K. Unnikrishnan, "Brain-inspired computing: Models and architectures," *IEEE Open J. Circuits Syst.*, vol. 1, pp. 185–204, 2020.

[2] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychol. Rev.*, vol. 65, no. 6, pp. 386–408, 1958.

[3] K. Cho et al., "Learning phrase representations using RNN encoder-decoder for statistical machine translation," 2014, *arXiv:1406.1078*.

[4] G. Kornaros, "Hardware-assisted machine learning in resource-constrained IoT environments for security: Review and future prospective," *IEEE Access*, vol. 10, pp. 58603–58622, 2022.

[5] Y. Miao, M. Gowayyed, and F. Metze, "EESEN: End-to-end speech recognition using deep RNN models and WFST-based decoding," in *Proc. IEEE Workshop Autom. Speech Recognit. Understand. (ASRU)*, Dec. 2015, pp. 167–174.

[6] A. Graves, N. Jaitly, and A.-R. Mohamed, "Hybrid speech recognition with deep bidirectional LSTM," in *Proc. IEEE Workshop Autom. Speech Recognit. Understand.*, Dec. 2013, pp. 273–278.

[7] S. Saadatnejad, M. Oveisi, and M. Hashemi, "LSTM-based ECG classification for continuous monitoring on personal wearable devices," *IEEE J. Biomed. Health Informat.*, vol. 24, no. 2, pp. 515–523, Feb. 2020.

[8] S.-Z. Yu, "Explicit duration recurrent networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 7, pp. 3120–3130, Jul. 2022.

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

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

[11] K. Guo et al., "Angel-eye: A complete design flow for mapping CNN onto customized hardware," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2016, pp. 24–29.

[12] S. Han et al., "ESE: Efficient speech recognition engine with sparse LSTM on FPGA," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2017, pp. 75–84.

[13] Z. Wang, J. Lin, and Z. Wang, "Accelerating recurrent neural networks: A memory-efficient approach," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 10, pp. 2763–2775, Oct. 2017.

[14] S. Wang et al., "C-LSTM: Enabling efficient LSTM using structured compression techniques on FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2018, pp. 11–20.

[15] M. Wang, Z. Wang, J. Lu, J. Lin, and Z. Wang, "E-LSTM: An efficient hardware architecture for long short-term memory," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 2, pp. 280–291, Jun. 2019.

[16] S. Wang et al., "Acceleration of LSTM with structured pruning method on FPGA," *IEEE Access*, vol. 7, pp. 62930–62937, 2019.

[17] K. P. Yalamarthy, S. Dhall, M. T. Khan, and R. A. Shaik, "Low-complexity distributed-arithmetic-based pipelined architecture for an LSTM network," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 2, pp. 329–338, Feb. 2020.

[18] C. Gao, D. Neil, E. Ceolini, S.-C. Liu, and T. Delbruck, "DeltaRNN: A power-efficient recurrent neural network accelerator," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2018, pp. 21–30.

[19] Z. Li et al., "E-RNN: Design optimization for efficient recurrent neural networks in FPGAs," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2019, pp. 69–80.

[20] S. Cao et al., "Efficient and effective sparse LSTM on FPGA with bank-balanced sparsity," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2019, pp. 63–72.

[21] 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 Netw. Learn. Syst.*, early access, Jun. 10, 2022, doi: 10.1109/TNNLS.2022.3180209.

[22] Y. Ma, Y. Cao, S. Vrudhula, and J.-S. Seo, "Optimizing the convolution operation to accelerate deep neural networks on FPGA," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 7, pp. 1354–1367, Jul. 2018.

[23] S. A. White, "Applications of distributed arithmetic to digital signal processing: A tutorial review," *IEEE ASSP Mag.*, vol. 6, no. 3, pp. 4–19, Jul. 1989.

[24] S. Y. Park and P. K. Meher, "Low-power, high-throughput, and low-area adaptive FIR filter based on distributed arithmetic," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 60, no. 6, pp. 346–350, Jun. 2013.

[25] M. T. Khan and R. A. Shaik, "Optimal complexity architectures for pipelined distributed arithmetic-based LMS adaptive filter," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 2, pp. 630–642, Feb. 2019.

[26] M. T. Khan, R. A. Shaik, and M. A. Alhartomi, "An efficient scheme for acoustic echo canceller implementation using offset binary coding," *IEEE Trans. Instrum. Meas.*, vol. 71, pp. 1–14, 2022.

[27] M. T. Khan, M. A. Alhartomi, S. Alzahrani, R. A. Shaik, and R. Alsulami, "Two distributed arithmetic based high throughput architectures of non-pipelined LMS adaptive filters," *IEEE Access*, vol. 10, pp. 76693–76706, 2022.

[28] M. T. Khan and R. A. Shaik, "Analysis and implementation of block least mean square adaptive filter using offset binary coding," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2018, pp. 1–5.

[29] Y. Cheng, F. X. Yu, R. S. Feris, S. Kumar, A. Choudhary, and S.-F. Chang, "An exploration of parameter redundancy in deep networks with circulant projections," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 2857–2865.

[30] S. Liao, Z. Li, X. Lin, Q. Qiu, Y. Wang, and B. Yuan, "Energy-efficient, high-performance, highly-compressed deep neural network design using block-circulant matrices," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD)*, Nov. 2017, pp. 458–465.

[31] Z. Lu, V. Sindhwani, and T. N. Sainath, "Learning compact recurrent neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Mar. 2016, pp. 5960–5964.

[32] J.-T. Chien and T.-W. Lu, "Deep recurrent regularization neural network for speech recognition," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Apr. 2015, pp. 4560–4564.

[33] J. Oruh, S. Viriri, and A. Adegun, "Long short-term memory recurrent neural network for automatic speech recognition," *IEEE Access*, vol. 10, pp. 30069–30079, 2022.

[34] Z. Li, S. Wang, C. Ding, Q. Qiu, Y. Wang, and Y. Liang, "Efficient recurrent neural networks using structured matrices in FPGAs," 2018, *arXiv:1803.07661*.

[35] S.-E. Chang et al., "Mix and match: A novel FPGA-centric deep neural network quantization framework," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Feb. 2021, pp. 208–220.

[36] A. N. Jahromi, S. Hashemi, A. Dehghantanha, R. M. Parizi, and K. R. Choo, "An enhanced stacked LSTM method with no random initialization for malware threat hunting in safety and time-critical systems," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 4, no. 5, pp. 630–640, Oct. 2020.

[37] B. Zamanlooy and M. Mirhassani, "Efficient VLSI implementation of neural networks with hyperbolic tangent activation function," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 1, pp. 39–48, Jan. 2014.

[38] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, and D. S. Pallett, "DARPA TIMIT acoustic-phonetic continous speech corpus CD-ROM. NIST speech disc 1-1.1," Nat. Inst. Standards Technol. (NIST), Gaithersburg, MD, USA, NASA STI/Recon Tech. Rep. n, 1993, p. 27403, vol. 93.

[39] L. Zhao, S. Liao, Y. Wang, Z. Li, J. Tang, and B. Yuan, "Theoretical properties for neural networks with weight matrices of low displacement rank," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 4082–4090.

[40] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort, "A white paper on neural network quantization," 2021, *arXiv:2106.08295*.

[41] C. Gao, A. Rios-Navarro, X. Chen, T. Delbruck, and S.-C. Liu, "Edge-DRNN: Enabling low-latency recurrent neural network edge inference," in *Proc. 2nd IEEE Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, Aug. 2020, pp. 41–45.

[42] N. M. Rezk, M. Purnaprajna, T. Nordström, and Z. Ul-Abdin, "Recurrent neural networks: An embedded computing perspective," *IEEE Access*, vol. 8, pp. 57967–57996, 2020.

Mohammed A. Alhartomi (Member, IEEE) received the B.Sc. degree from King Abdulaziz University, Jeddah, Saudi Arabia, the M.Sc. degree in wireless communications systems from Swansea University, Swansea, U.K., and the Ph.D. degree from the University of Leeds, Leeds, U.K., in 2016. He is currently an Associate Professor with the Electrical Engineering Department, University of Tabuk, Saudi Arabia. He is also a Visiting Scholar with the University of Central Florida, where he is working on dense VLC networks. His current research interests include mobile and wireless communications, optical communications, and signal processing.

Mohd Tasleem Khan received the B.Tech. degree in electronics from the Zakir Hussain College of Engineering and Technology, Aligarh Muslim University, Aligarh, India, in 2013, and the Ph.D. degree in VLSI from the Indian Institute of Technology Guwahati, India, in 2019. He was a Principal Engineer with Taiwan Semiconductor Manufacturing Company (TSMC), Hsinchu, Taiwan. He is currently an Assistant Professor with the Department of Electronics Engineering, Indian Institute of Technology Dhanbad, India. His current research interests include the study of algorithms and architectures for VLSI implementation of signal processing, communication, machine learning, and artificial intelligence applications.

Saeed Alzahrani (Member, IEEE) received the B.S. degree in electrical engineering from King Abdulaziz University, Jeddah, Saudi Arabia, the M.S. degree in electrical engineering from the University of Colorado at Colorado Springs, and the Ph.D. degree in electrical engineering from The Ohio State University. He was with the Microelectronics Research Laboratory, University of Colorado at Colorado Springs, from 2012 to 2014, where he was involved in developing tunable ferroelectric-based filters, VCOs, amplifiers, and antennas. From 2014 to 2019, he was with the ElectroScience Laboratory, The Ohio State University, where he focused on developing design techniques for wide TR VCOs at the mm-wave frequency band. He is currently an Assistant Professor with the Electrical Engineering Department, University of Tabuk, Saudi Arabia. His current research interests include addressing design and technological challenges related to RF/millimeter-wave and mixed-signal integrated circuits and systems for emerging technologies, including communication, automotive, and biomedical applications.

Ahmed Alzahmi (Member, IEEE) received the Ph.D. degree in electrical engineering from Southern Methodist University, Dallas, TX, USA. He is currently an Assistant Professor of engineering with the University of Tabuk, Tabuk, Saudi Arabia. His current research interests include the 3-D/2-D memory interface design, RF transceiver design, power delivery networks, clock delivery networks, and analog/mixed-signal integrated circuit design.

Rafi Ahamed Shaik (Senior Member, IEEE) received the B.Tech. and M.Tech. degrees in electronics and communication engineering from Sri Venkateswara University, Tirupati, India, in 1991 and 1993, respectively, and the Ph.D. degree from the Indian Institute of Technology Kharagpur, India, in 2008. He was a Faculty Member with the Deccan College of Engineering and Technology, Hyderabad, India, from 1993 to 1995, and the Bapatla Engineering College, Bapatla, India, from 1995 to 2003. He is currently a Professor with the Department of Electronics and Electrical Engineering, Indian Institute of Technology Guwahati, India. His current research interests include digital and adaptive signal processing, biomedical signal processing, and VLSI signal processing.

Jinti Hazarika (Graduate Student Member, IEEE) received the B.E. degree from the Jorhat Engineering College in 2012 and the M.Tech. degree from the Indian Institute of Technology Roorkee, India, in 2015. She is currently pursuing the Ph.D. degree from the Indian Institute of Technology Guwahati, India. Her current research interests include VLSI DSP algorithms and architectures.

Ruwaybih Alsulami (Member, IEEE) was born in Mecca, Saudi Arabia. He received the B.S. degree in electrical and computer engineering from the University of Colorado at Boulder, Boulder, CO, USA, in 2011, the M.S. degree in electrical engineering from the University of Colorado at Colorado Springs, Colorado Springs, CO, USA, in 2013, and the Ph.D. degree in electrical and computer engineering from The Ohio State University, Columbus, OH, USA, in 2020 and 2021, respectively. He is currently an Assistant Professor of electrical engineering with Umm Al-Qura University, Mecca. His current research interests include circuit design, such as antennas, filters, multiband power amplifiers, PA linearization, and measurements of nonlinear microwave devices and circuits.

Abdulaziz Alotaibi received the B.Eng. degree in electrical engineering from the Florida Institute of Technology, the M.Sc. degree in systems engineering from the Florida Institute of Technology, and the Ph.D. degree in manufacturing engineering from Loughborough University, U.K. He is currently an Assistant Professor with the Industrial Engineering Department, Tabuk University, Saudi Arabia. His current research interests include dynamic optimization, resource consumption minimization, agent-based distributed industrial control, and intelligent manufacturing.

Meshal Al-Harthis is currently pursuing the degree in electrical engineering with the University of Tabuk, with a specialization in circuit design, control systems, and power electronics. His expertise in various programming languages enhanced his capabilities within the scientific field. He devoted part of his undergraduate journey to research and development within the field of electrical engineering.