# Accelerating LSTM-based High-Rate Dynamic System Models

Ehsan Kabir*, Daniel Coble$, Joud N. Satme$, Austin R.J. Downey$, Jason D. Bakos†,
David Andrews*, Miaoqing Huang*
*Department of Computer Science and Computer Engineering, University of Arkansas, USA
$Department of Mechanical Engineering, University of South Carolina, USA
†Department of Computer Science and Engineering, University of South Carolina, USA
{ekabir, dandrews, mqhuang}@uark.edu, {dncoble, jsatme}@email.sc.edu, austindowney@sc.edu, jbakos@cse.sc.edu

***Abstract*—In this paper, we evaluate the use of a trained Long Short-Term Memory (LSTM) network as a surrogate for a Euler–Bernoulli beam model, and then we describe and characterize an FPGA-based deployment of the model for use in real-time structural health monitoring applications. The focus of our efforts is the DROPBEAR (Dynamic Reproduction of Projectiles in Ballistic Environments for Advanced Research) dataset, which was generated as a benchmark for the study of real-time structural modeling applications. The purpose of DROPBEAR is to evaluate models that take vibration data as input and give the initial conditions of the cantilever beam on which the measurements were taken as output. DROPBEAR is meant to serve an exemplar for emerging high-rate "active structures" that can be actively controlled with feedback latencies of less than one microsecond. Although the Euler–Bernoulli beam model is a well-known solution to this modeling problem, its computational cost is prohibitive for the time scales of interest. It has been previously shown that a properly structured LSTM network can achieve comparable accuracy with less workload, but achieving sub-microsecond model latency remains a challenge. Our approach is to deploy the LSTM optimized specifically for latency on FPGA. We designed the model using both high-level synthesis (HLS) and hardware description language (HDL). The lowest latency of 1.42 $\mu$S and the highest throughput of 7.87 Gops/s were achieved on Alveo U55C platform for HDL design.**

***Index Terms*—FPGA, LSTM, High-rate dynamics, Flexible, High-Level Synthesis, Hardware Description Language, RTL.**

## I. Introduction

Long Short-Term Memory (LSTM) neural networks are capable of capturing dependencies for long sequential or temporal data in applications such as speech recognition, natural language processing, image captioning, scene analysis, etc. [1], [2]. In earlier works, such networks have been shown to be effective in utilizing time-series data to infer the state of a structure in a high-rate dynamic environment [3], [4]. The primary goal of this study is to develop a hardware-based LSTM model to enable ultra-low latency state estimation applications [5]. High-rate dynamic systems refer to environments in which structures are subjected to impact loading that results in accelerations greater than 100 g for time periods of less than 100 ms [6]. Such systems are civil structures exposed to blasts, space infrastructures prone to debris strikes, and aerial vehicles capable of supersonic flight [7], [8]. Systems exposed to such environments require rapid response, from event detection to decision-making, in the sub-millisecond or microsecond scale to ensure safe and reliable operations [4], [9], [10].

DROPBEAR data [5], [11] was used to come up with a suitable three-layer LSTM architecture that is trained off-line in software using logs recorded from the physical DROPBEAR apparatus. Our model showed an acceptable accuracy in terms of signal-to-noise ratio and desired latency on a real-time operating system (RTOS). However, the RTOS is unable to exploit the available parallelism available within LSTMs. On the other hand, FPGAs can accelerate inference with low power consumption using pipelined and parallel processing elements and are therefore suitable for LSTM implementation. Thus, a custom accelerator of the trained LSTM model was designed using both HLS and HDL in this paper. Furthermore, an implementation on various FPGA platforms was carried out for performance enhancement and comparison.
*The contributions of this paper are:*

- Design of an LSTM accelerator framework using high-level synthesis (HLS) that meets the real-time requirements set by high-rate applications. Results show that outermost loop pipelining generates a more efficient hardware design than outermost loop unrolling of the algorithm.
- An alternative approach to the accelerator design using hardware description language (HDL) to improve performance. Results show that HDL provides the flexibility to choose the level of parallelism based on the available resources and timing requirements which not possible with the HLS-based approach.
- An investigation into model deployment on several FPGA platforms from Xilinx to determine the best-performing configuration given the application. We targeted datacenter platforms such as Xilinx Alveo U55c and VC707, and an embedded platform, ZCU104. We found that the additional resources available in the U55C were unnecessary for the size of the deployed model. Nonetheless, the U55C's superior resources allowed maximum level of parallelism. Both ZCU104 and U55C boards achieve latency lower than VC707 because they achieve better frequency. U55C achieved the highest

frequency of all, but its latency is lower than that of ZCU104 within the same level of parallelism.

## II. Model Selection

The LSTM model is chosen for a high-rate dynamic system to predict the real-time response within microsecond latency. The experimental data known as DROPBEAR was generated by Joyce et al. [12]. The test setup and data are described in [4], [13]. Various LSTM models were trained on Python with Tensorflow and Keras to find a suitable model that meets the RTOS requirement of 500 $\mu$s in a real-time device consisting of a cRIO-9035 with a 1.33 GHz dual-core Intel Atom (E3825) manufactured by NI. The number of units per LSTM layer was varied from 8 to 40 units, each layer having the same number of units for simplicity. The number of layers was swept from 1 to 3. Fig. 1 shows a large variance in the measurement of signal-to-noise ratio (SNR) as we vary units per layer, though the SNR improves with increased number of layers. The 3-layer configuration with 15 units/layer is chosen in this paper for FPGA implementation as it has the highest SNR. The model has 16 input features sourced from the input signal uniformly sampled across the previous timestep and produces an output state prediction every 500 $\mu$s on RTOS.

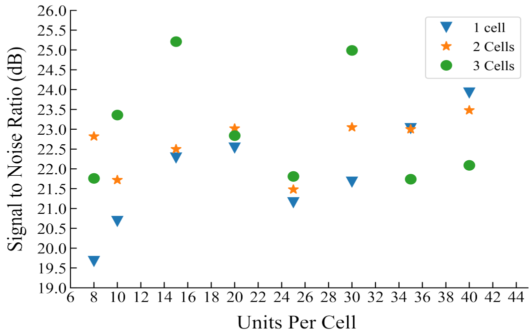

Fig. 1. $\text{SNR}_{\text{dB}}$ values of models with different numbers of cells and units

## III. Related work

LSTMs have been deployed in previous works using HLS design. For example, work in [14] used HLS pragmas such as loop unrolling and pipelining to build a real-life speech recognition system using an LSTM model, whereas [15] implemented a real-time aircraft anomaly detection system using small scale LSTM on FPGA. A multilayer LSTM accelerator template was developed using the HLS tool for detecting gravitational waves which is a time-series data produced from LIGO detectors [16]. Low-power LSTM accelerators are built using pipeline and parallel algorithms in HLS [17]. Some works dealt with real-time response applications like electrical fault detection or heart rate monitoring with LSTM on FPGA after offline construction of a suitable LSTM architecture on python [18], [19]. Some HDL-based LSTM accelerators such as a human activity monitoring system described in [20], provide flexibility of reconfiguration by adding parameterized features. The same group also detected artifacts from EEG signals by an LSTM model on FPGA at a low power and low frequency in [21]. Another research for healthcare applications reports an energy-efficient and high throughput real-time human action detection system [22], where both HLS and HDL-based RTL modules are combined for the whole system design. Comparative studies between HLS and HDL-based designs have been done in the past for applications other than neural networks [23], [24]. Here, we compared the custom LSTM designs in both HDL and HLS formats.

## IV. High Level Synthesis Implementation

This section describes the high-level synthesis design technique. The core of the accelerator is designed in C++ language on Vitis HLS 2022.2.1 tool.

There are two main units in the accelerator architecture - the matrix-vector operations (MVO) unit and the element-wise operations (EVO) unit. HLS design did not have any function defined for them. As a result, the exported RTL did not have separate RTL instances for them. The LSTM gates within the MVO unit are separately defined as functions with unique arguments, allowing for the generation of independent parallel RTL modules. Instead of any distinct functions, the EVO unit's operations are expressed as distinct for-loops. Fig. 2 gives a high-level overview of the HLS implementation of the LSTM accelerator. Each gate can perform multiplication and accumulation (MAC) operations either in parallel or sequentially depending on the number of BRAMs. Since each of the LSTM network layers function in succession, the gate modules are reused for operations in various layers. Each gate module contains two for-loops. One loop iterates over the hidden units. This loop contains two other distinct for-loops, one for multiplication and the other for accumulation operations. The other loop inside the function executes the summing operation with the bias over each hidden unit before executing the activation function. The main arguments of the gate functions are the inputs, hidden states, input weights, recurrent weights, and bias vectors. The array

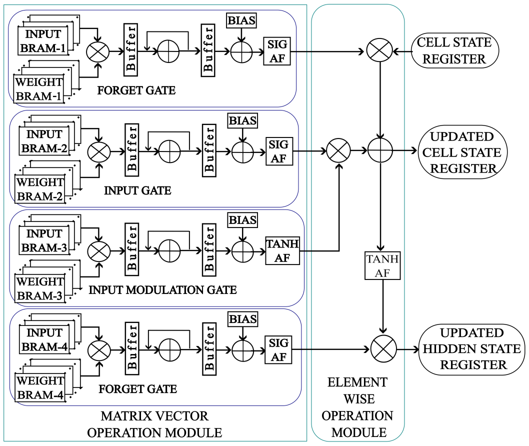

Fig. 2. LSTM Operations for High Level Synthesis Design.

partition pragma entirely partitions the vectors of the inputs, hidden states, and biases to create registers and permit parallel access because the size of these vectors is small. However, the vectors for input and recurrent weights are represented as BRAMs to store a large number of elements. Depending on the size of the LSTM network and the compiling capacity of the synthesis tool, they can be partially or entirely partitioned to generate multiple BRAMs. Large array partitioning slows down the compilation flow and synthesis process may even stop. For an easier representation of the multiplication and accumulation operations of LSTM on HLS, the inputs and hidden state vectors, as well as the input and recurrent weight vectors, are concatenated. For fully pipelined operations, pipeline pragmas were applied on the outer loops of the functions, which fully unrolled the internal loops, but the operations were not fully parallel because of the BRAMs. The number of DSPs used for multiplication appears to depend on the size of the concatenated vector of inputs and hidden states. However, they do not start computation at the same clock cycle even being allocated simultaneously which is a limitation of HLS. To increase the utilization of DSPs for parallel multiplications, the outermost loop can be unrolled by some factors depending on the available resources. The EVO unit contains several for loops, but all loop operations are pipelined, and no loop was unrolled.

## V. Register Transfer Level Implementation

Since RTL provides added flexibility as compared to HLS, we developed a Verilog implementation of the LSTM and synthesized it with Vivado 2022.2.1 tool. Fig. 3 shows the connections of modules in the RTL implementation.

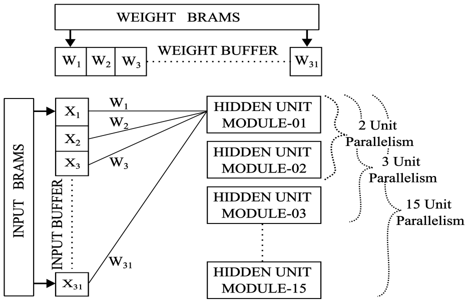

Fig. 3. LSTM Operations for HDL Design

The gate modules and the MVO module are not defined separately in the HDL design. It aided in reducing connections between modules that consume extra LUTs. For parallel execution, the hidden units within each gate are defined as modules and instantiated multiple times. This module contains some descriptions of logic operations as well as the instances of other modules such as a multiplier, adder, and activation functions. The number of this module to instantiate for each LSTM gate at the top module is configurable. It indicates a total number of parallel operations which is shown by unit parallelism in Fig. 3. We were able to increase the parallel operations with parallel DSPs in this manner, which was not possible in HLS. HDL design required parallel DSPs for the EVO unit. The weights are stored in separate BRAMs in each hidden unit. As parallel DSPs require input data simultaneously, the number of BRAM instances grows in proportion to the number of hidden unit instances. The weights stored in the BRAMs are first transferred to the registers (w1, w2,...,w31 in Fig. 3) to facilitate parallel data access. As the number of DSPs was increased, performance improved dramatically over the HLS design. Because of the heavy usage of DSPs, the design becomes crowded, preventing high-frequency operation. LUT usage rises so that correct data gets multiplexed to the DSPs. As the size of this LSTM is tiny, the number of concatenated inputs and hidden states were kept constant. Only flexibility over hidden units was demonstrated, but the same flexibility may be extended to inputs as well.

## VI. Overall System

Fig. 4 shows the complete system design for running the LSTM model on different FPGA platforms such as VC707 (Virtex-7 XC7VX485TFFG1761-2), ZCU104 (Zynq UltraScale+XCZU7EV-2FFVC1156 MPSoC) and U55C (UltraScale+XCU55C-FSVH2892-2L-E) for our experiments. The overall system was designed on Vivado 2022.1.2 design suite. It contains a custom IP block for the LSTM accelerator (LA), which can either be exported from HLS or be built directly with HDL. The LA has internal BRAMs as shown in Fig. 2. However, the system design has some external block ram generator modules for storing inputs, weights, and outputs. These inputs and weights are fetched from external DDR3 DRAM or High Bandwidth Memory (HBM). The outputs are returned to DRAM or HBM. Because of the limited local memory, the software executing on the CPUs is also saved on DRAM or HBM.

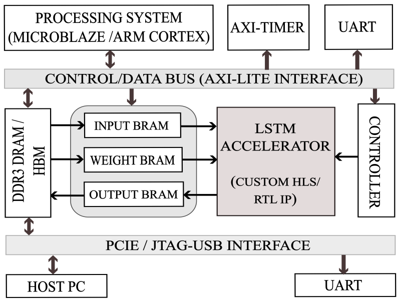

Fig. 4. Complete System Design

Both VC707 and ZCU104 boards have onboard DRAM memory, while the Alveo U55C contains HBMs. The DRAM of VC707 is connected to the programmable logic (PL) side. It can communicate with the MicroBlaze ($\mu$B) softcore processing system (PS) using a memory interface generator (MIG) connected by AXI-lite interface. The $\mu$B is configured for the maximum frequency operation [25]. On ZCU104, we access the onboard DRAM through the ARM-based Multiprocessor System-on-Chip (MPSoC) subsystem. We used the same ARM processor to run our LSTM model and check the performance only in PS. On U55c, the $\mu$B can access the HBMs. All the boards are connected to the same HOST PC with USB-JTAG interface. U55C was connected with a PCIe 3.0×4 interface of another host which is a server with Intel Xeon CPU E5-2603 v4 @1.70GHz, so that it receives enough power to run and gets reset signal after power on. This host can communicate with other IPs except the PS using the DMA/Bridge Subsystem for PCI Express IP [26] in the system. PS uses AXI-TIMER to measure the latency which includes the time between the start and the stop signal from the custom IP module. The host connected with JTAG cable displays the results on the terminal using the UARTlite interface.

## VII. Results and Evaluation

On three separate platforms with varying levels of parallelism, results for 32-bit (FP-32), 16-bit (FP-16), and 8-bit (FP-8) fixed point precision were obtained. Then they were compared in terms of maximum frequency (Fmax), resource utilization, latency, throughput, and normalized throughput [27].

Fmax is reported for the system design in Fig. 4, which includes all IPs, while resource usage is only reported for the accelerator. This is because the accelerator is identical across all platforms, but the overall system design differs among them. For a fair comparison, the bit lengths of the inputs, outputs, weights, and intermediate values are maintained the same in both HLS and HDL designs. The findings for HLS design reveal that for the same pragmas such as loop unroll, loop pipeline, and array partition, the number of BRAM and DSP is different in different platforms and bit precision. HLS tends to optimize itself regardless of the pragmas in different platforms. Hence, array partition was done with different factors on different platforms so that the number of DSPs remained the same. Despite the reduction in resource usage and increase in frequency for FP-8, the model did not automatically utilize the freed-up resources to decrease the delay. The improvement in frequency resulted in a minor reduction in latency. To further reduce latency for FP-8, the design must be updated again. The results in Table-I compares the effect of pipelining and unrolling the outermost loop inside each gate. Unrolling the outermost loop entirely or partially did not enhance performance significantly, even though resource use, such as DSP, increased by 8×.

TABLE I
HLS Loop Optimization

| HLS Designs | Platform & Precision | DSP | Maximum Frequency (MHz) | Latency ($\mu$S) |
|---|---|---|---|---|
| Loop Unroll | Virtex 7 | 1852 | 166 | 6.12 |
| Loop Pipeline | Fixed-16 | 224 | 250 | 6.54 |

On the other hand, the number of DSP can be controlled in HDL design. Parallelism is increased by increasing DSP blocks, however doing so leads to congestion in the routing system, which reduces overall frequency and occasionally results in no routing at all. As a result, it is important to carefully manage the amount of parallelism to avoid drastically decreasing frequency or going over the resource limit. Although the increment of DSP causes a reduction of frequency, the performance gets better than that of HLS designs. The number of DSP depends on the number of hidden units in HDL. In order to prevent the frequency from declining and resources from being overused, we reduced the number of parallelism as the bit width increased up to 32. The results in Table-III and Table-IV also indicate that after 32-bit precision, HLS design starts performing better than the HDL design. It is because DSPs are heavily utilized in HDL design resulting in frequency decay. Furthermore, as precision rises, more DSPs for MAC are utilized, causing a resource overflow. As a result, HDL is unable to maximize parallelism for FP-32 without sacrificing frequency. Full parallelism can be achieved for our LSTM model up to 16-bit precision in all the FPGA platforms except ZCU104 which exceeds available DSPs if more than 2 unit parallelism is applied. With full parallelism (15 units for our model), U55C achieves the lowest latency of all as shown in Table-II. Thus, increasing the parallelism improved performance more than the HLS design in spite of the frequency drop.

TABLE II
Effects of Parallelism on HDL Design

| Platform | Bit Precision | LUT (%) | DSP (%) | Highest Level of Parallelism | Fmax (MHz) | Latency ($\mu$S) |
|---|---|---|---|---|---|---|
| Virtex 7 | FP-32 | 28 | 69 | 4 Units | 142 | 5.78 |
| | FP-16 | 39 | 72 | 15 Units | 166 | 2.06 |
| U55C | FP-32 | 11 | 38 | 8 Units | 150 | 2.38 |
| | FP-16 | 9 | 22 | 15 Units | 250 | 1.42 |

For HLS design of FP-8, DSPs were only employed for the activation functions because DSPs is not used below 10-bit precision. Although we compelled the use of DSPs for our multipliers in HDL design by employing Verilog attributes, their proper sharing could not be obtained which would have reduced consumption. Only the LUT and FF consumption was decreased by low bit precision. Thus, FP-8 will be useful for bigger models. One important improvement with FP-8 is achieving high frequency, and it helped reduce the latency to some extent. The total number of operations was determined for our LSTM model from which the throughput (Giga operations/second [GOPS]) was computed [27]. For a fair comparison between HLS and HDL-based design, normalized throughput both with respect to the LUTs (GOPS/LUT) and DSPs (GOPS/DSP) was calculated. Same parameters are also used for a fair comparison with different LSTM models in other works. HDL design has the flexibility to increase the resources to maximize throughput. Thus, throughput is higher than the HLS design in all the platforms. However, the (GOPS/LUT) and (GOPS/DSP) are higher in HLS design because it consumes fewer resources. As HLS automates the majority of optimization procedures, there is little scope to increase the resources to decrease latency. Table-III reports all data related to the HLS design on different platforms. ZCU104 achieves the lowest latency, the highest GOPS, and the highest GOPS/LUT and GOPS/DSP for all precision.

TABLE III
RESULTS FOR HIGH-LEVEL SYNTHESIS DESIGN

| Platform | Bit Precision | LUT | FF | BRAM 36k | DSP | Fmax (MHz) | Latency ($\mu$S) | Throughput (GOPS) | GOPS/ LUT | GOPS/ DSP |
|---|---|---|---|---|---|---|---|---|---|---|
| Virtex 7 | FP-32 | 70380 (23%) | 86579 (14%) | 41.5 (4%) | 712 (25%) | 210 | 8.75 | 1.28 | 18.19 | 1.80 |
| | FP-16 | 30532 (10%) | 36186 (6%) | 22 (2%) | 224 (8%) | 213 | 7.4 | 1.51 | 49.46 | 6.74 |
| | FP-8 | 26889 (9%) | 20683 (3%) | 0 (0%) | 30 (1%) | 235 | 6.36 | 1.76 | 65.45 | 58.67 |
| ZCU104 | FP-32 | 78850 (34%) | 94936 (21%) | 17.5 (16%) | 712 (41%) | 305 | 3.74 | 2.99 | 37.92 | 4.20 |
| | FP-16 | 36458 (16%) | 39326 (9%) | 10 (3%) | 224 (13%) | 350 | 2.92 | 3.83 | 105.05 | 17.10 |
| | FP-8 | 23575 (10%) | 21590 (5%) | 0 (0%) | 15 (1%) | 400 | 2.83 | 3.95 | 167.55 | 263.33 |
| U55C | FP-32 | 64930 (5%) | 80191 (3%) | 29.5 (1%) | 711 (8%) | 362 | 6.86 | 1.63 | 25.10 | 2.29 |
| | FP-16 | 25346 (2%) | 31136 (1%) | 16 (1%) | 224 (2%) | 375 | 4.72 | 2.36 | 93.42 | 10.57 |
| | FP-8 | 23899 (2%) | 17422 (1%) | 0 (0%) | 15 (0.2%) | 380 | 4.65 | 2.4 | 100 | 160.00 |

TABLE IV
RESULTS FOR HARDWARE DESCRIPTION LANGUAGE SYNTHESIS DESIGN

| Platform | Bit Precision | LUT (%) | FF (%) | BRAM 36k (%) | DSP (%) | Fmax (MHz) | Latency ($\mu$S) | Throughput (GOPS) | GOPS/ LUT | GOPS/ DSP |
|---|---|---|---|---|---|---|---|---|---|---|
| Virtex 7 | FP-32 | 17 | 16 | 1 | 43 | 150 | 11.48 | 0.97 | 19.34 | 0.81 |
| | FP-16 | 22 | 23 | 5 | 41 | 166 | 3.71 | 3.01 | 45.19 | 2.64 |
| | FP-8 | 13 | 12 | 5 | 35 | 200 | 3.10 | 3.61 | 95.06 | 3.64 |
| ZCU104 | FP-32 | 22 | 21 | 4 | 69 | 230 | 7.11 | 1.57 | 31.62 | 1.31 |
| | FP-16 | 30 | 29 | 15 | 66 | 250 | 2.14 | 5.21 | 76.69 | 4.56 |
| | FP-8 | 16 | 16 | 15 | 57 | 300 | 1.72 | 6.50 | 171.61 | 6.55 |
| U55C | FP-32 | 4 | 4 | 1 | 13 | 250 | 6.826 | 1.64 | 6.83 | 1.37 |
| | FP-16 | 5 | 5 | 2 | 13 | 256 | 2.492 | 4.48 | 2.49 | 3.92 |
| | FP-8 | 3 | 3 | 2 | 11 | 300 | 2.108 | 5.30 | 2.11 | 5.34 |

Table-IV reports all data related to the HDL design on different platforms for 2 unit parallelism. The utilization of LA is the same regardless of the platforms. ZCU104 shows the best performance among other platforms for HDL design also. The utilization is higher than HLS design, so latency was reduced by 1.34×. GOPS/LUT is close to HLS design, but GOPS/DSP is much lower because HDL design mainly reduces latency by parallel operations of DSPs. Table-V compares our LA with other LAs on FPGA. Our HDL designs here are for the highest level of parallelism achieved by the platforms for FP-16. Since the models are not the same, all the performance parameters such as frequency, latency, throughput, and normalized throughput are measured for a fair comparison. We achieved the lowest latency of 1.42 $\mu$S and the highest GOPS of 7.87 with the HDL design on U55C board running at 250 MHz frequency. Among all of our HDL designs, it exploits full parallelism and gives the highest GOPS/LUT and GOPS/DSP. Work [28] achieved latency closest to ours on VC707 at 140 MHz, but its GOPS is 1.73× lower. The LA in [29] got 1.5× more GOPS/LUT but 3.5× less GOPS than ours because it consumed fewer LUTs. The HDL design in [30] has the highest GOPS/DSP meaning it uses fewer DSPs. While its GOPS is comparable to ours, our slowest HDL design on ZCU104 and our slowest HLS design on VC707 are, respectively, 3.78× and 1.26× faster than this. The HLS design performed better on ZCU104 with the lowest latency and the highest GOPS of 2.92 $\mu$S and 3.83 respectively. Since the design consumes fewer resources, the GOPS/LUT and GOPS/DSP are higher than that of the HDL design. Both our HDL and HLS designs are respectively 280× and 136× faster than the ARM Core CPU running at 1.2GHz frequency.

TABLE V
COMPARISON WITH OTHER LSTM ACCELERATORS

| Work | Platform | Method | Fmax (MHz) | Latency ($\mu$S) | Throughput (GOPS) | GOPS/ (LUT*1000) | GOPS/ (DSP*1000000) |
|---|---|---|---|---|---|---|---|
| [14] | VC707 | HLS | 150 | 390 | 7.26 | 38.23 | 6.17 |
| [15] | VC707 | HLS | 150 | 4.3 | 13.45 | 47 | 7.77 |
| [16] | U250 | HLS | 300 | 0.867 | 17.2 | – | 1.9 |
| [17] | Zynq-7020 | HLS | 118 | 18760 | 0.00977 | 1.14 | 0.143 |
| [20] | Artix-7 | HDL | 160 | 800 | 0.631 | – | – |
| [21] | Artix-7 | HDL | 53 | 1240 | 0.055 | 56 | 13.75 |
| [29] | XC7Z030 | HDL | 100 | – | 2.26 | 98.1 | – |
| [28] | VC707 | HDL | 140 | 2.05 | 4.535 | 31.2 | 5.06 |
| [30] | XC7Z020 | HDL | 164 | 9.3 | 7.51 | – | 192 |
| [31] | ZC7020 | – | 142 | 932 | 1.049 | 16.96 | – |
| This Work | ARM Cortex A53 | Embedded C | 1200 | 398 | 0.028 | – | – |
| | U55C | HDL | 250 | 1.42 | 7.87 | 65.67 | 3.9 |
| | ZCU104 | | 215 | 2.46 | 4.56 | 67 | 3.99 |
| | VC707 | | 166 | 2.06 | 5.37 | 45.5 | 2.67 |
| | U55C | HLS | 375 | 4.72 | 2.36 | 93.42 | 10.57 |
| | ZCU104 | | 350 | 2.92 | 3.83 | 105 | 17 |
| | VC707 | | 213 | 7.40 | 1.51 | 49.45 | 6.7 |

## VIII. CONCLUSION

In this research, we demonstrated a custom LSTM accelerator on FPGA created using both high-level synthesis (HLS) and hardware description language (HDL). For a use case involving high-rate time series data that were dynamically generated by simulating a ballistic environment, we created a new three-layer LSTM model. To address the demands for real-time reaction, the hardware accelerator for this model was subsequently constructed on an FPGA. Even with the use of certain directives, the HLS design process produces an unmanageable circuit despite being quick and simple to modify. On the other hand, while the HDL design process is lengthy, it can still result in the intended circuit, and we were able to manage the degree of parallelism. We contrasted the two designs' performance, utility, and adaptability. Then, we analyzed the differences between the performance, utilization, and flexibility of the two design strategies. The ZCU104 platform uses the outermost loop pipelining pragma to provide the lowest latency for HLS design. The outermost loop unrolling pragma can use more resources (DSP) in HLS, but it did not achieve latency that was lower than the outermost loop pipelining pragma. High resource usage may be enabled for the HDL design. As a result, we could set up the U55C so that it can fully parallelize our LSTM model, which has the highest DSP usage. It had the lowest latency at full parallelism as a consequence. Yet, ZCU104 also outperformed U55C in HDL design at the same amount of parallelism. Our HLS and HDL designs are significantly faster than the CPU, according to experimental findings. In terms of latency, throughput, frequency, or normalized throughput, the findings further demonstrate that our approach is better than the majority of current LSTM accelerators on FPGA.

# References

[1] T. Mikolov, M. Karafiát, L. Burget, J. H. ernocký, and S. Khudanpur, “Recurrent neural network based language model,” in *Interspeech*, 2010.

[2] W. Byeon, T. M. Breuel, F. Raue, and M. Liwicki, “Scene labeling with lstm recurrent neural networks,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 3547–3555.

[3] M. Nelson, S. Laflamme, C. Hu, A. G. Moura, J. Hong, A. Downey, P. Lander, Y. Wang, E. Blasch, and J. Dodson, “Generated datasets from dynamic reproduction of projectiles in ballistic environments for advanced research (DROPBEAR) testbed,” *IOP SciNotes*, vol. 3, no. 4, p. 044401, nov 2022.

[4] *Progress Towards Data-Driven High-Rate Structural State Estimation on Edge Computing Devices*, ser. International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, vol. Volume 10: 34th Conference on Mechanical Vibration and Sound (VIB), 08 2022, v010T10A017. [Online]. Available: https://doi.org/10.1115/DETC2022-90118

[5] A. Downey, J. Hong, J. Dodson, M. Carroll, and J. Scheppegrell, “Millisecond model updating for structures experiencing unmodeled high-rate dynamic events,” *Mechanical Systems and Signal Processing*, vol. 138, p. 106551, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0888327019307721

[6] J. Hong, S. Laflamme, J. Dodson, and B. Joyce, “Introduction to state estimation of high-rate system dynamics,” *Sensors*, vol. 18, no. 2, p. 217, jan 2018.

[7] J. Hong, S. Laflamme, and J. Dodson, “Study of input space for state estimation of high-rate dynamics,” *Structural Control and Health Monitoring*, vol. 25, no. 6, p. e2159, 2018.

[8] M. Nelson, V. Barzegar, S. Laflamme, C. Hu, A. R. Downey, J. D. Bakos, A. Thelen, and J. Dodson, “Multi-step ahead state estimation with hybrid algorithm for high-rate dynamic systems,” *Mechanical Systems and Signal Processing*, vol. 182, p. 109536, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0888327022006379

[9] *Microsecond State Monitoring of Nonlinear Time-Varying Dynamic Systems*, ser. Smart Materials, Adaptive Structures and Intelligent Systems, vol. Volume 2: Modeling, Simulation and Control of Adaptive Systems; Integrated System Design and Implementation; Structural Health Monitoring, 09 2017, v002T05A013. [Online]. Available: https://doi.org/10.1115/SMASIS2017-3999

[10] J. Dodson, A. Downey, S. Laflamme, M. D. Todd, A. G. Moura, Y. Wang, Z. Mao, P. Avitabile, and E. Blasch, “High-rate structural health monitoring and prognostics: An overview,” in *Data Science in Engineering, Volume 9*, R. Madarshahian and F. Hemez, Eds. Cham: Springer International Publishing, 2022, pp. 213–217.

[11] High-Rate-SHM-Working-Group, “Acceleration-vs-roller-displacement dataset for dropbear.” [Online]. Available: https://github.com/High-Rate-SHM-Working-Group/Dataset-2-DROPBEAR-Acceleration-vs-Roller-Displacement

[12] B. Joyce, J. Dodson, S. Laflamme, and J. Hong, “An experimental test bed for developing high-rate structural health monitoring methods,” *Shock and Vibration*, vol. 2018, 2018.

[13] A. Panahi, E. Kabir, A. Downey, D. Andrews, M. Huang, and J. D. Bakos, “High-rate machine learning for forecasting time-series signals,” in *2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2022, pp. 1–9.

[14] Y. Guan, Z. Yuan, G. Sun, and J. Cong, “Fpga-based accelerator for long short-term memory recurrent neural networks,” in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2017, pp. 629–634.

[15] Z. Sun, Y. Zhu, Y. Zheng, H. Wu, Z. Cao, P. Xiong, J. Hou, T. Huang, and Z. Que, “Fpga acceleration of lstm based on data for test flight,” in *2018 IEEE International Conference on Smart Cloud (SmartCloud)*, 2018, pp. 1–6.

[16] Z. Que, E. Wang, U. Marikar, E. Moreno, J. Ngadiuba, H. Javed, B. Borzyszkowski, T. Aarrestad, V. Loncar, S. Summers, M. Pierini, P. Y. Cheung, and W. Luk, “Accelerating recurrent neural networks for gravitational wave experiments,” in *2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2021, pp. 117–124.

[17] U. Yoshimura, T. Inoue, A. Tsuchiya, and K. Kishine, “Implementation of Low-Energy LSTM with Parallel and Pipelined Algorithm in Small-Scale FPGA,” in *2021 International Conference on Electronics, Information, and Communication (ICEIC)*. Jeju, Korea (South): IEEE, Jan. 2021, pp. 1–4. [Online]. Available: https://ieeexplore.ieee.org/document/9369806/

[18] Q. Liu, T. Liang, Z. Huang, and V. Dinavahi, “Real-time fpga-based hardware neural network for fault detection and isolation in more electric aircraft,” *IEEE Access*, vol. 7, pp. 159 831–159 841, 2019.

[19] L. G. Rocha, M. Liu, D. Biswas, B.-E. Verhoef, S. Bampi, C. H. Kim, C. Van Hoof, M. Konijnenburg, M. Verhelst, and N. V. Helleputte, “Real-time hr estimation from wrist ppg using binary lstms,” in *2019 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, 2019, pp. 1–4.

[20] A. N. Mazumder, H.-A. Rashid, and T. Mohsenin, “An Energy-Efficient Low Power LSTM Processor for Human Activity Monitoring,” in *2020 IEEE 33rd International System-on-Chip Conference (SOCC)*. Las Vegas, NV, USA: IEEE, Sep. 2020, pp. 54–59. [Online]. Available: https://ieeexplore.ieee.org/document/9524796/

[21] N. K. Manjunath, H. Paneliya, M. Hosseini, D. Hairston, and T. Mohsenin, “A Low-Power LSTM Processor for Multi-Channel Brain EEG Artifact Detection.”

[22] J. Yin, J. Han, R. Xie, C. Wang, X. Duan, Y. Rong, X. Zeng, and J. Tao, “MC-LSTM: Real-Time 3D Human Action Detection System for Intelligent Healthcare Applications,” *IEEE Transactions on Biomedical Circuits and Systems*, vol. 15, no. 2, pp. 259–269, Apr. 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9373938/

[23] R. Millón, E. Frati, and E. Rucci, “A comparative study between hls and hdl on soc for image processing applications,” *arXiv preprint arXiv:2012.08320*, 2020.

[24] H. S. Lee and J. W. Jeon, “Comparison between hls and hdl image processing in fpgas,” in *2020 IEEE International Conference on Consumer Electronics - Asia (ICCE-Asia)*, 2020, pp. 1–2.

[25] “MicroBlaze Processor Reference Guide,” 2022. [Online]. Available: https://docs.xilinx.com/v/u/en-US/ug984-vivado-microblaze-ref

[26] “Introduction • DMA/Bridge Subsystem for PCI Express Product Guide (PG195) • Reader • Documentation Portal.” [Online]. Available: https://docs.xilinx.com/r/en-US/pg195-pcie-dma

[27] E. Kabir, A. Poudel, Z. Aklah, M. Huang, and D. Andrews, “A runtime programmable accelerator for convolutional and multilayer perceptron neural networks on fpga,” in *Applied Reconfigurable Computing. Architectures, Tools, and Applications: 18th International Symposium, ARC 2022, Virtual Event, September 19–20, 2022, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2022, p. 32–46.

[28] J. C. Ferreira and J. Fonseca, “An FPGA implementation of a long short-term memory neural network,” in *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. Cancun, Mexico: IEEE, Nov. 2016, pp. 1–8. [Online]. Available: http://ieeexplore.ieee.org/document/7857151/

[29] E. Azari and S. Vrudhula, “An Energy-Efficient Reconfigurable LSTM Accelerator for Natural Language Processing,” in *2019 IEEE International Conference on Big Data (Big Data)*, Dec. 2019, pp. 4450–4459.

[30] E. Bank-Tavakoli, S. A. Ghasemzadeh, M. Kamal, A. Afzali-Kusha, and M. Pedram, “POLAR: A Pipelined/Overlapped FPGA-Based LSTM Accelerator,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 3, pp. 838–842, Mar. 2020. [Online]. Available: https://ieeexplore.ieee.org/document/8889770/

[31] A. X. M. Chang and E. Culurciello, “Hardware accelerators for recurrent neural networks on FPGA,” in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. Baltimore, MD, USA: IEEE, May 2017, pp. 1–4. [Online]. Available: http://ieeexplore.ieee.org/document/8050816/