A scalable Echo State Networks hardware generator for embedded systems using high-level synthesis

Nan-Sheng Huang, Jan-Matthias Braun, Jørgen Christian Larsen, Poramate Manoonpong
Embodied AI and Neurorobotics Laboratory
Mærsk Mc-Kinney Møller Institute, University of Southern Denmark
Odense M, Denmark
{nan, j-mb, jcla, poma}@mmmi.sdu.dk

Abstract—**Reservoir computing (RC) features with the rich computational dynamics is a kind of powerful machine learning paradigm that is well suited for non-linear time-series prediction and classification problems. However, this impressive performance comes with a cost of complex arithmetic operations and high memory usage that make it significantly challenging to deploy on embedded systems. Solutions based on CPU and/or GPU-based designs, provides flexibility but suffers from a lack of efficiency in terms of power, performance, and area (PPA). Although hardware-accelerated solutions can improve efficiency, it takes longer design cycles and is time-consuming. Furthermore, it may happen that design spec requires run change due to the fact that the network is retrained with the new data set to improve the performance. It leads to extra effort in the redesign of the hardware-accelerated solution. This preliminary work presents the design and implementation of a hardware generator for RC-ESNs (echo state networks) to tackle the problem. The proposed methodology is demonstrated by various offline-trained network parameters and topologies. Compared to existing solutions, the proposed framework provides scalability in agile hardware design.**

Index Terms— Neural Networks; Reservoir Computing; Echo State Networks; Hardware Accelerator; Embedded Systems; High-Level Synthesis

I. Introduction

Machine learning (ML) has within the last 5 years become an important and powerful development within Neural Networks (NNs). However, one of the vital research problems in ML is to predict the future sequence of values from the available past sequence of observed data sets. Among the state-of-the-art algorithms in publications, reservoir computing (RC), which is a brain-inspired paradigm, has been proven a novel and effective framework to tackle the prediction problem[1]. For instance, echo state networks (ESNs), liquid state machines (LSMs) and time delay reservoirs (TDRs) are three main variants of the reservoir algorithm which has been widely exploited by researchers for many kinds of ML applications such as non-linear time-series processing, robotic control, and hard classification tasks[2]. Thus, it is a trending to develop intelligent embedded systems by empowering embedded devices with the capability to process and interpret plenty of noisy and sophisticated measured data sets for further prediction.

Despite its impressive performance, deploying RC on a low-power real-time embedded system is still a challenging task due to computationally expensive operations, recurrence and the random connection structure. On one hand, CPU and GPU-based solution provide flexibility in development but they lack efficiency compared to hardware (HW) accelerated solution in terms of power, performance, and area (PPA)[3]. On the other hand, even though some hardware accelerated architectures have been proposed previously, it still takes significant efforts to implement, even for a developer with an in-depth understanding of digital hardware design. For example, it usually consumes several weeks or even months to design and verify a handcrafted digital Neural Network (NN) implementation from scratch by virtue of Verilog, VHDL or SystemVerilog as design entry. If the developer adopts high-level synthesis (HLS) tools, it can drastically lower the development time[4]. However, even if the HLS employs C/C++ as design entry to generate RTL (register-transfer level) and if the user expects to obtain optimized hardware for performance, power and area requirement, it implies either a highly skilled RTL designer or that the designer has a steep learning curve to cross in order to harness HLS tools for efficient hardware design[5]. Furthermore, algorithm development is always a process of continuous improvement which means it may evolve with new data sets in the future. The retrain of NNs may result in the run change of network parameters and topologies. Thus, enabling ML hardware accelerator in embedded systems is often a practical but daunting task.

In this preliminary work, a scalable RC-ESNs hardware generator for embedded computing is presented. We exploit the methodology of HLS in conjunction with design automation to automatically transform an offline-trained RC-ESNs algorithm into embedded hardware accelerator for FPGA applications. This approach removes the steep learning curve that the developer otherwise would have to tackle. Firstly, the complexity profiling for the RC-ESNs algorithm is conducted to explore the adequate generic building blocks. Secondly, according to the analysis result, a pipelined and parallel layered microarchitecture is presented. Next, by leveraging the capability of design pattern template in HLS C/C++, the scalable hardware generator is proposed to shorten the turn-around time in hardware development. Furthermore, design-space exploration (DSE) can also be applied to the generated hardware module for further PPA optimization in HLS. Finally, a case study applied in the development of BMI (Brain-

Machine Interface) proactive control system is presented to demonstrate the feasibility of the hardware generator. To our best knowledge, this is the first work to develop a hardware generation tool for reservoir computing.

The next section of the paper describes the overview of RC-ESNs algorithm. Related works and design challenges are elaborated in Section III. The preliminary RC-ESNs hardware generator is proposed in Section IV. Section V presents some of the results, analysis, and discussion. The paper is concluded in Section VI with future research works.

II. Overview of RC-ESNs

The ESN model, based on the RC framework, is depicted in Fig. 1. It is composed of three basic layers: one input layer, one hidden (either internal or reservoir) layer and one output layer. In this case study, connections of the input-to-output layer and output-to-reservoir layer are not used temporarily. The functionality of the neurons in the input layer is to transport the input to the next layer. For the connection between input and reservoir layer, it is featured by random and recurrent connections associated with a given sparsity which is one of the parameters in ESNs. The connection of reservoir-to-output layer is fully connected.

The discrete time state dynamics of reservoir neurons is given in the following equations as [6]:

$$x(t+1) = (I-\Lambda)\theta(t) + \Lambda(W_{sys}\theta(t)) + W_{in}v(t) \tag{1}$$

$$y(t) = W_{out}x(t) \tag{2}$$

$$\lambda_i = \frac{1}{T_c}(\frac{1}{1+\rho_i}) \tag{3}$$

$$\theta_i(t) = tanh(a_i x_i(t) + b_i) \tag{4}$$

where x(t) is the vector of dynamic reservoir state activation, v(t) is the vector of time-dependent input, y(t) is the vector of output neurons, $\Lambda = (\lambda_1, \lambda_2, \cdots, \lambda_N)^T$ is the collection of the individual leak decay rates which each reservoir neuron has, ρ_i is the leak control parameter which can be modulated by a global time constant T_c >0, a_i governs the slope of the firing rate curve and b_i is the bias value of individual neuron.

III. Related Works and Design Challenges

To satisfy the stringent requirements for real-time embedded systems, parallel hardware acceleration of NNs, in particular, has become increasingly popular by virtue of good performance and better energy-efficiency compared to CPUs and GPUs [7].

Reference [8] presented the use of stochastic computing to reduce the hardware resources required to implement different arithmetic operations in RC-ESNs, which was the first hardware implementation example of an RC system using classical sigmoid neurons. The major advantage of the stochastic logic design is soft-error tolerance, low-power consumption, and low hardware cost but with the price of long computation time and design time. In [9], a real-time hardware-based FPGA architecture for RC-ESNs as recurrent neural network(RNN) training was designed. The author showed that it was the first time to design the hardware for real-time training of RC-ESNs on FPGA. A folded architecture ESNs processor with online training on FPGA was also presented by [10]. Reference [11] proposed an efficient parallel implementation of RC-ESNs systems by simplifying the synapses and using linear piece-wise activation functions for neurons on an FPGA. The maximum neuron numbers presented in this work is 300.

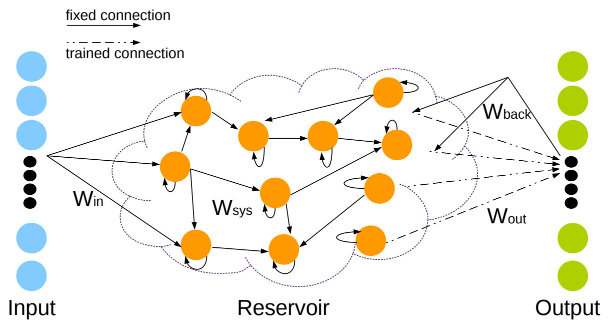

Fig. 1. Block Diagram of RC-ESNs

The general development flow of NNs comprises a training phase and a classification/inference phase. However, the quality of the training phase mainly depends on the quality of input data sets. On one hand, the data sets may be incomplete or problematic in the early development stage. On the other hand, data sets of the corner cases may be collected in the late stage after deployment which is not employed in the previous training phase. Both cases require to have iterations in the training phase, even though the NNs is offline-trained. It may result in the run change of not only the weight values but also the topology and structure of the NNs. Hence, it ends up with the redesign of the hardware accelerator which takes the extra resources, efforts and time for the hardware team. Furthermore, the schedule may slip to miss the milestone. Nevertheless, the aforementioned related works only focus on the efficient hardware implementation of RC-ESNs assuming that the design specification is ready and stable.

Therefore, these design challenges motivate to propose a scalable hardware generator for RC-ESNs in this paper. To the best of our knowledge, this is the first work to address the problem in the hardware design of RC.

IV. The Proposed RC-ESNs Hardware Generator

This section elaborates on the design method of the proposed RC-ESNs hardware generator. The use case of the hardware generator is shown in Fig. 2. Firstly, algorithm developers begin to train NNs after getting training data. After training, the offline-trained parameters of the NNs are provided for further implementation of the hardware accelerator. During

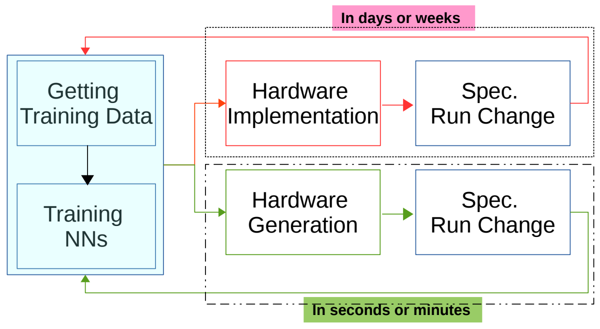

Fig. 2. Various Iteration Flows for Hardware Run Changes

the period of hardware design and testing, the availability of new data sets may trigger the specification run change owing to retraining the NNs. The run change may result in noticable efforts of the hardware team to modify the design including re-verifying. However, when it adopts the method of hardware generation, the run change only requires to trigger the push-button flow instead of manual re-design.

A. Analysis and Design of LEGO-like HW Building Blocks

First of all, when it comes to the data dependence, the data flow of RC-ESNs depends on the output of the previous layer and the output of the recurrent connections within the reservoir layer compared to feedforward neural networks.

Next, on one hand, if we observe the operations of the neurons in the reservoir and output layer, the underlying building blocks are multiply-accumulators (MACs) in addition to other logic primitives. On the other hand, from the (1) to (4), the coarse-grained build blocks are matrix-vector-multiplication (MVM) and tanh in addition to memory access.

For the non-linear activation function tanh, one of the well-known solutions for hardware implementation is to adopt the piece-wise linear (PWL) method in-which different line segments with specific slope and offset are exploited. In this work, the 16-segment PWL model is exploited to approximate the tanh function.

To further reduce the PPA in hardware, we take advantage of the characteristic of sparsity in W_{in} and W_{sys} by transforming the MVM with sparse matrix-vector multiplication (spMVM). spMVM is a kind of irregular algorithm but only stores non-zero values of the weight matrix. Given that the reservoir layer has 300 neurons with a sparsity 80%, considerable operations of MACs and weight memory can be saved to improve the PPA.

B. RC-ESNs HW Microarchitecture

After consolidating the above analysis of the building blocks in RC-ESNs algorithm, a pipelined and parallel hardware architecture is devised as shown in Fig. 3. The blue circle means spMVM block, the red circle is the scalar multiplier and the yellow circle represents the MVM module. In the

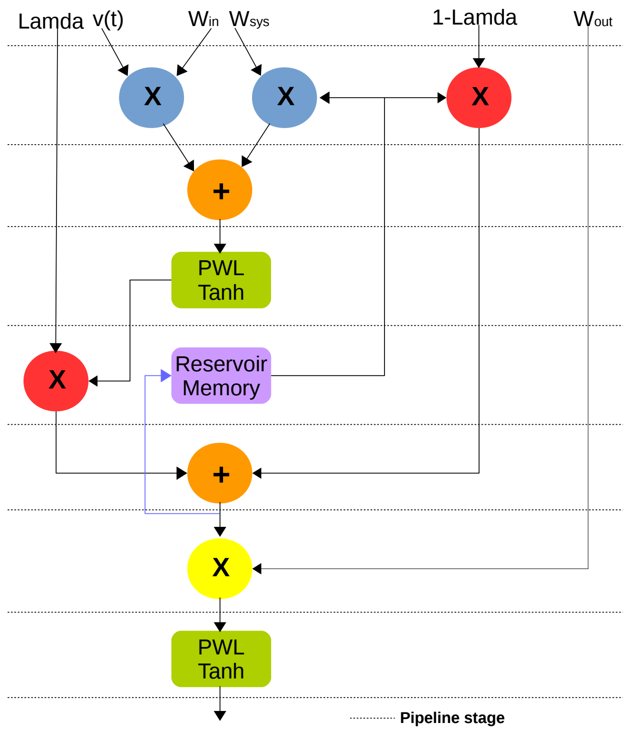

Fig. 3. Pipelined and Parallel Microarchitecture of the RC-ESNs

bottom of the architecture, the output of the adder is restored to the reservoir memory bank to realize the delay behavior of the previous state. The inputs and outputs are implemented as either programmable registers or memory blocks which can be changed on-the-fly.

C. Design Pattern for RC-ESNs Hardware Generator

Design patterns are a kind of solution methodologies to a recurring problem which has broadly popularized in the field of software engineering [12]. The major merits are to achieve good modularity, reusability, and flexibility. However, in [13] further advocated the study of design patterns for reconfigurable hardware computing, attempting to facilitate developers to mitigate the recurring design challenges in reconfigurable hardware design.

By transforming the problem of the frequent run changes of network topology and parameters in the training of NNs into a recurring design problem, the design pattern methodology is adopted as the framework of the hardware generator in this work. Firstly, the HLS C/C++ is used to capture the circuit behavior of the coarse-grained building blocks and RC-ESNs microarchitecture. Meanwhile, the method of design pattern is applied during development and results in the design pattern template library for RC-ESNs. Secondly, a hyperparser is presented to monitor and parse the specification of the updated offline-trained networks to reconfigure the code generation from the hardware design pattern library. Next, the generated

HLS C/C++ codes are forwarded to Xilinx HLS tool for further design-space exploration [14]. Lastly, the corresponding Verilog/VHDL codes are generated in HLS. The framework and the whole user flow are illustrated in Fig. 4.

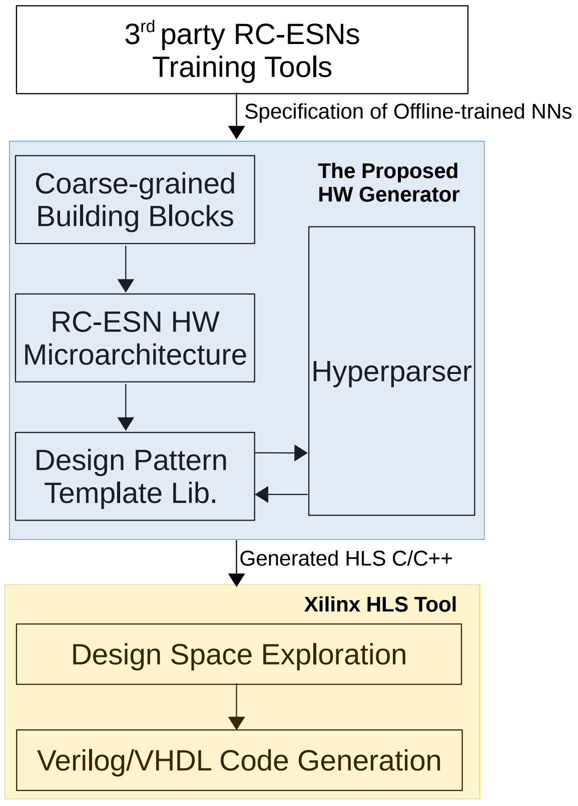
3rd party RC-ESNs Training Tools
Specification of Offline-trained NNs
Coarse-grained Building Blocks
The Proposed HW Generator
RC-ESN HW Microarchitecture
Hyperparser
Design Pattern Template Lib.
Generated HLS C/C++
Xilinx HLS Tool
Design Space Exploration
Verilog/VHDL Code Generation

Fig. 4. The Proposed Hardware Generator

V. Experimental Results

In this section, the proposed RC-ESNs hardware generator is demonstrated by the following case studies in the Plan4Act project [15]. The aim of Plan4Act is to provide new emerging technologies from spiking neural activity to proactive BMI control in the embedded systems, to show its effectiveness. Assuming that the hardware implementation exploits 32-bit floating point as precision to have similar prediction performance to the software one. The various sparsities are associated with the weight matrix of input-to-reservoir and reservoir-to-reservoir layer. For the continuous improvement of network performance, the software team retrains the RC-ESNs by utilizing new data sets obtained and collected from field tests at different times. It results in the run changes in both weight values and network topology.

As reported in Table I, there are 3 different offline-trained network topologies generated during the process of design iterations. Case 1 has 200 input neurons with sparse matrix 69.775%, 100 reservoir neurons with sparse matrix 80.23% and 10 output neurons. The variations of Case 2 and Case 3 fall in the spare matrices and number of reservoir neurons which sparsity is 70% in the input layer, 200 neurons in reservoir layer with sparsity 79% in Case 2, and 72% sparsity , 300 neurons in reservoir layer with 82% sparsity in Case 3, respectively. They all are generated by the proposed hardware generator within less than 90 sec. In the table, results of HLS synthesis are illustrated by two figures separated by backslash which represent the non-pipelined (left side) and pipelined version (right side) individually. For example, non-pipelined version consumes 296,662 cycles and pipelined version consumes 118,462 cycles with the same clock speed 9.55 ns. The pipelined version is obtained via compiler directives from design-space exploration in HLS. As it can be seen, the more reservoir neurons, the more significant BRAM consumption. From the perspective of hardware execution time in terms of latency for these two versions, Case 1 has 2.5X improvement, Case 2 gets 2.64X enhancement and Case 3 obtains 2.69X speed up in pipelined design. Furthermore, this work can support the arbitrary number of neurons with various sparsity and weight matrix configurations if the user targets FPGA with high area capacity.

The capability of the hardware generator is summarized and qualitatively compared to prior RC-ESNs accelerators as shown in Table II. Previous implementations are all based on RTL design in which the microarchitecture is fixed at design time. By leveraging the HLS methodology together with the design of hardware generator, it considerably mitigates the problem to support the scalable change of network topology. Moreover, according to the user's specific requirements, it also supports the design-space exploration in which the PPA is adjustable such as the pipelined method presented in the previous paragraph.

Overall, the RC-ESNs hardware generator provides noticeably benefit over previous implementations, which do not have scalability. Hence, the proposed work realizes rapid turn-around time and illustrates the promise of the solution in hardware design.

VI. Conclusions and Outlook

This paper has demonstrated a preliminary framework for the design automation of RC-ESNs hardware accelerator. The usages of the hardware generator are straightforward and effortless to not only hardware designers but also algorithm developers. The strength of the work hinges critically on the configurability and reusability of the design pattern template of the HLS C/C++ having the scalability. Therefore, it facilitates the whole reiterative procedure of update offline-trained network topology into hardware design in a few minutes.

Currently, the preliminary framework only supports RC-ESNs, but we plan to extend it to support other types of NNs like LSMs, TDRs, and feedforward NNs. In addition, we intend to provide a user interface, which is currently in development. Finally, it will be expanded to support the fixed-point format in accommodating the automatic determination

TABLE I
QUANTITATIVE ANALYSIS OF IMPLEMENTATION RESULTS OF DIFFERENT NETWORK TOPOLOGIES

	Case 1	**Case 2**	**Case 3**
Size of Input Layer, Sparsity	200, 69.775%	200, 70%	200, 72%
Size of Reservoir Layer, Sparsity	100, 80.23%	200, 79%	300, 82%
Size of Output Layer	10	10	10
Estimated Clock Speed (ns)	9.55/9.55	9.55/9.55	9.55/9.55
Latency (cycles)	296,662/118,462	1,152,962/436,562	2,569,262/954,662
BRAM Usage	57/57	147/147	215/215
DSP48E Usage	19/19	19/19	19/19
FF Usage	5,348/5,438	5,397/5,487	5,446/5,336
LUT Usage	7,346/7,376	7,355/7,345	7,371/7,361
IP Generation Time	<90 sec	<90 sec	<90 sec

TABLE II
QUALITATIVE COMPARISON WITH RELATED WORKS

	[8]	[10]	[9]	[11]	This work
Design Entry	RTL	RTL	RTL	RTL	HLS
Number of Reservoir Neurons	7	50	16/32/64	300	Arbitrary
Programmable Weight Values	YES	YES	YES	YES	YES
Scalable Network Topology	NO	NO	NO	NO	YES
Design Space Exploration	NO	NO	NO	NO	YES

of fixed-point word length for further automation of PPA optimization.

ACKNOWLEDGMENT

This research is supported by Horizon 2020 Framework Programme (FETPROACT-01-2016FET Proactive: emerging themes and communities) under grant agreement no. 732266 (Plan4Act).

REFERENCES

[1] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Computer Science Review*, vol. 3, no. 3, pp. 127–149, 2009.

[2] N. Soures, C. Merkel, D. Kudithipudi, C. Thiem, and N. McDonald, "Reservoir computing in embedded systems: Three variants of the reservoir algorithm." *IEEE Consumer Electronics Magazine*, vol. 6, no. 3, pp. 67–73, 2017.

[3] S. Che, J. Li, J. W. Sheaffer, K. Skadron, and J. Lach, "Accelerating compute-intensive applications with gpus and fpgas," in *Application Specific Processors, 2008. SASP 2008. Symposium on*. IEEE, 2008, pp. 101–107.

[4] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-level synthesis for fpgas: From prototyping to deployment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 4, pp. 473–491, 2011.

[5] R. Nane, V.-M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi *et al.*, "A survey and evaluation of fpga high-level synthesis tools," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 10, pp. 1591–1604, 2016.

[6] S. Dasgupta, F. Wörgötter, and P. Manoonpong, "Information dynamics based self-adaptive reservoir for delay temporal memory tasks," *Evolving Systems*, vol. 4, no. 4, pp. 235–249, 2013.

[7] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank, "A survey of neuromorphic computing and neural networks in hardware," *CoRR*, vol. abs/1705.06963, 2017.

[8] M. L. Alomar, V. Canals, N. Perez-Mora, V. Martínez-Moll, and J. L. Rosselló, "Fpga-based stochastic echo state networks for time-series forecasting," *Computational intelligence and neuroscience*, vol. 2016, p. 15, 2016.

[9] Y. Yi, Y. Liao, B. Wang, X. Fu, F. Shen, H. Hou, and L. Liu, "Fpga based spike-time dependent encoder and reservoir design in neuromorphic computing processors," *Microprocessors and Microsystems*, vol. 46, pp. 175–183, 2016.

[10] S. Buchanan and L. Liu, "Design and fpga implementation of a folded architecture echo state network processor with online training," 2016.

[11] M. Alomar, E. S. Skibinsky-Gitlin, C. F. Frasser, V. Canals, E. Isern, M. Roca, and J. L. Rosselló, "Efficient parallel implementation of reservoir computing systems," *Neural Computing and Applications*, pp. 1–15, 2017.

[12] E. Gamma, *Design patterns: elements of reusable object-oriented software*. Pearson Education India, 1995.

[13] A. DeHon, J. Adams, M. DeLorimier, N. Kapre, Y. Matsuda, H. Naeimi, M. Vanier, and M. Wrighton, "Design patterns for reconfigurable computing," in *12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2004, pp. 13–23.

[14] Xilinx, *Vivado Design Suite User Guide: High-Level Synthesis*, Feb,2017. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_2/ug902-vivado-high-level-synthesis.pdf

[15] *EU Plan4Act project*. [Online]. Available: http://plan4act-project.eu/index.php/about/