Efficient Nonlinear Autoregressive Neural Network Architecture for Real-Time Biomedical Applications

Brooks Olney, Shakil Mahmud, and Robert Karam
University of South Florida
{brooksolney, shakilmahmud, rkaram}@usf.edu

***Abstract*—Medical devices, such as continuous glucose monitors (CGMs) and drug-delivery pumps, are often combined in closed-loop systems for treating chronic diseases. Generally, these systems consist of sensors and actuators whose operation is modulated based on sensed stimuli. Closed-loop systems may be susceptible to a number of different security and reliability issues which may result in incorrect operation which may endanger patients. Nonlinear autoregressive neural networks (NARNNs) may be used in such systems for error detection and correction due to their predictive capabilities; however, an efficient implementation is needed for use in wearables and biomedical implants. In this paper, we present an area- and energy-efficient, pipelined NARNN hardware architecture suitable for such constrained devices. The architecture was tested on FPGA to confirm functionality, then synthesized targeting the SAED 32nm EDK. This NARNN implementation requires an estimated area of $0.02\ mm^2$, $0.54\ \mu s$ and $0.76\ nJ$ per inference.**

I. Introduction

Biomedical devices with closed-loop or semi-closed loop control systems, such as the implantable pacemaker, artificial pancreas, or certain neurostimulators, can be used to manage chronic disease and improve patient care and comfort. However, as these devices are used in real-time environments with extremely high cost for failure, it is critical to ensure their security and reliability. One potential issue could stem from incorrect sampling of the target biosignal. Sensor drift, for example, due to environmental changes, moisture absorption, system aging, etc., or faults, whether naturally occurring or due to interference by malicious actors, can result in incorrect treatments, e.g. drug dosage or neurostimulation amplitude/duration, which can be life threatening. By utilizing a predictive model, the biosignals measured in real-time can be validated against *expected* behavior before they are processed by the control algorithm, adding another layer of security and reliability to the system.

As an example, we consider the artificial pancreas system (APS) [1]. The APS delivers a measured dose of insulin as needed by the patient, and is comprised of a small continuous glucose monitor (CGM), an insulin infusion pump, and a control algorithm. The sensor is implanted subcutaneously and continuously monitors the patient's blood glucose concentration. When the glucose level crosses a predefined threshold, the adaptive control algorithm adjusts, either by increasing, decreasing, or stopping the insulin delivery such that the target blood-glucose concentration is maintained. As the insulin pump is modulated in response to the glucose concentration as measured by the sensors, an error in measurement or fault during algorithmic processing could be harmful. Hence, high reliability is critical.

Step-ahead prediction is one possible technique to improve reliability in such systems. Using various signal processing and statistical techniques, an estimate for a likely range of expected next measurements could be used to confirm the true next measurement or identify a potential issue. Measurements far outside the expected range could be corrected, again using a variety of techniques. To this end, machine learning (ML) techniques are well-suited to this problem space, and have seen usage in biomedical applications such as early disease prediction, forecasting risk factors, and treatment [2], [3]. In particular, we focus on the use of nonlinear autoregressive neural networks (NARNNs) which have been used for time series prediction for biosignals [4].

Broadly, artificial neural networks (ANNs) can forecast time-series data by modeling complex data with nonlinear relationships using multiple hidden layers to improve prediction and classification accuracy. Implementation of ANNs in hardware can be very costly in terms of area and power requirements; simultaneously, bioimplantables have high efficiency and ultra-low power design requirements. In this paper, we present an efficient pipelined hardware architecture for step-ahead prediction of biosignals with a 5 neuron, 16 tap NARNN, using CGM data from the open D1NAMO CGM dataset [5] as a case study. The dataset consists of blood-glucose recordings from 9 human subjects with Type 1 diabetes and provides a sufficient baseline for evaluating the hardware implementation. The implementation is verified on FPGA and synthesized with Synopsys Design Compiler using the SAED 32nm library. The proposed architecture is estimated to occupy an area of $0.02\ mm^2$, requiring 107 clock cycles / $0.54\ \mu s$ and $0.76\ nJ$ per inference. To the best of our knowledge, this is the first hardware architecture of the NARNN targeting ultra-constrained devices such as bioimplantables.

II. Background

A. Hardware Implementation of ANNs

ANNs are generally considered computationally expensive, and as such require optimizations to both the model architecture and parameters for implementation in hardware. There are

many examples in the literature of ANN acceleration in various hardware platforms, e.g. field-programmable gate arrays (FPGAs) or application-specific integrated circuits (ASICs).

In [6], the authors presented three hardware accelerators for recurrent neural networks (RNNs) which are well suited for time-series data. Similar to our work, the authors experimented with different on-chip and off-chip memory access patterns and quantizing the weights to varying Q-format fixed-point (FP) numbers. In [7] Gao et al. improved upon this and presented a power efficient RNN implementation in a MiniZed FPGA. In their approach, they reduced computational complexity by quantizing the weights to 8-bit FP and by enabling data reuse between arithmetic units. With total on-chip power consumption at 1.48 W, their accelerator can achieve between 389 nJ and 79 nJ per inference, depending on operation mode. While lower than in [6], this approach may not be suitable for ultra-low power applications.

There has also been some work specifically towards predicting blood glucose over time. Zhu et al. [8] presented a deep learning based approach that utilizes long short-term memory (LSTM). They implemented their approach on an STM32 microcontroller and measured the power consumption at 8 μW with total execution time of 22 ms – translating to 176 nJ per inference. While the model performance and power requirements are good, the longer execution time translates to larger energy requirements per inference. This is partially due the size of the models they used and the operations taking place in floating point on a general-purpose processor. In practice, a more lightweight and pipelined implementation may be more desirable, especially in battery powered devices to increase the overall system lifetime.

B. NARNN Algorithm

The non-linear autoregressive network (NAR) uses *internal delay states* to encapsulate the trends from the last d timesteps to predict the next value in the time series – with the number of states defined by the delay timesteps d. The model uses d past values of the time series data Y to predict the next value in the time series Y_t using the following equation:

$$Y_t = f(Y_{t-1}, Y_{t-2}, ..., Y_{t-d}) \tag{1}$$

where f is a nonlinear function that can be approximated by a neural network. This is accomplished by training the neural network on a given time series, which will iteratively adjust the weight and bias terms to improve the accuracy to the targets. The architecture of the NARNN is defined by the number of neurons in the hidden layer N and the number of delay states d. Each neuron has a weight term for every delay state and a single bias term, with a nonlinear activation function ϕ – typically hyperbolic tangent (tanh). Once the model has been trained, the feedforward operation can be formalized as follows:

$$Y_t = \sum_{i=0}^{N} w_{oi} * \phi\Big(b_i + \sum_{j=0}^{d} w_{i,j} * D_j\Big) + b_{oi} \tag{2}$$

III. Methodology

A. NARNN Software Implementation

To generate and train the NARNN models, we used the *neural network time series* application in Matlab R2021a. We were able to configure N and d, select one or more data sources to train on, set different splits for training, validation and testing data, and train the model repeatedly. We found the most suitable architecture to be ($N = 5, d = 16$) for the D1NAMO dataset [5].

For hardware acceleration, we experimented with FP quantization using different representations of integer and exponent bits to find the lowest bit-width that could still accurately predict the time series data. To evaluate the tradeoffs of quantization, we built a testbed for the NARNN in Python which uses the Fxpmath library to quantize the NARNN parameters at different FP representations. This allowed us to quantize the weights at smaller precisions, compare the results against traditional floating point results and compare intermediate results with the hardware implementation. Using this approach, we found signed 10-bit FP with 1 integer bit and 8 exponent bits to be the best option for storage savings and model performance.

Aside from evaluating the effects of weight quantization, the testbed automates outputting model weights and subject input data into formats parsable by the hardware description language (HDL) code. Results in FPGA simulation are expected to be exactly the same, aside from some rounding errors, so this provides a good baseline for our hardware implementation.

B. Hardware Architecture

The architecture of the proposed hardware accelerator is shown at a high level in Figure 1(a). The accelerator features a pipelined processing architecture that is controlled by a finite state machine (FSM) and implements the neurons as self-contained processing elements (PEs).

The state diagram of the FSM is shown in Figure 1(b). The circuit waits for a new measurement, Y_t, then begins execution of the first layer. This is done by iteratively loading a column of weights and latching the results from the PEs. Once the first matrix multiplication is complete, the tanh activation function is approximated through a lookup table (LUT). The tanh values feedback into the PEs to be multiplied by the output layer weights, and the results are summed to derive the final output for that timestep. The internal delay states are updated every time a new update is received and an output calculated, and are maintained by using counters and storage registers.

The PE is designed to be self-contained and compact, with support for multiply-accumulate (MAC) operations in different FP representations. A diagram of the PE is shown in Figure 1(c). The PE has reset and enable signals and 3 dedicated operand inputs, one for the bias term and two for the multiplier and multiplicand. It has internal storage registers, one of size Q(m+m).(n+n) for the product of w and x, and another for accumulating the result with an extra bit for overflow. Aside from the activation function ϕ, the PE encapsulates all functionality of the NARNN neuron.

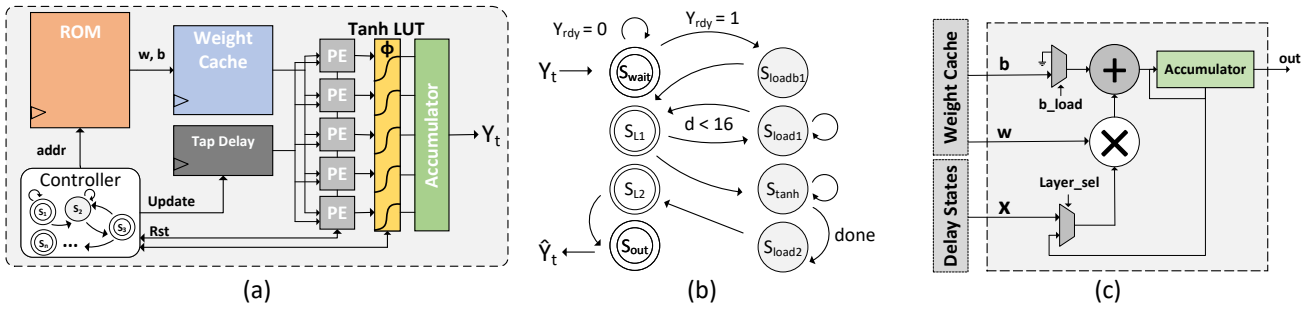

Fig. 1: (a) Overall hardware architecture consisting of FSM control logic, an array of 5 PEs, a single tanh LUT and output accumulator. (b) Controller FSM begins execution once data is received - controlling the updating of delay states, loading of weight columns, layer swapping and computing the final output. (c) Internal structure of PE consisting of three data inputs (w, x, b), a multiplier, adder and accumulator.

A single neuron can be used to implement the full functionality of any NARNN. However, as N increases, so to will the latency as more computations are required per data sample – we used a network with $(N = 5, d = 16)$. Since this network size is relatively small (91 total parameters), we implemented the NARNN with 5 PEs (same number of neurons). Loading the weights takes 5 clock cycles and the result is latched on the subsequent clock cycle. With 16 total delay states, the hidden layer takes a total of 96 clock cycles to compute the intermediate result, plus 5 to compute ϕ. Weights for the output layer only need to be loaded once since they fit entirely in the weight cache – thus it takes 5 cycles to load the weights and one cycle to compute the result. Once the system receives an input, it will produce a result after 107 clock cycles.

With a small number of parameters, the entire network could potentially be stored in registers for fast real-time inference, albeit at great hardware costs. In our case with the CGM data, measurements are only taken every 5 minutes, so the timing requirements are not so tight as to require a response on the order of nanoseconds. Moreover, the network weights encapsulate the behavior *for a single user*, hence there must be a mechanism for easily programming a user's weights into the device. This can be achieved by using an ultra-low power nonvolatile memory (NVM), such as that in [9]. Since the algorithm has no need for writing data to memory, we simply require high read endurance.

IV. Results

A. Model Performance

Performance was evaluated using the Python testbed described in Section III. While we experimented with different FP notations, we only show results comparing the performance of the software and hardware implementations. Results for the best and worst subjects in the D1NAMO dataset (in terms of achievable model performance) are shown in the plots in Figure 2. There are some regions around upper/lower peaks where the hardware simulation is slightly above or below the expected. A comprehensive listing of the results is provided in Table I. We found that the RMSE of the two implementations does not differ drastically (μ=0.044 $\pm$ 0.031).

TABLE I: Comparison of results from software and FPGA.

Subject	RMSE		
	Software	Hardware	Abs. Difference
S1	0.279	0.311	0.032
S2	0.306	0.380	0.074
S3	0.581	0.632	0.051
S4	0.375	0.322	0.053
S5	0.230	0.321	0.091
S6	0.702	0.698	0.004
S7	0.523	0.524	0.001
Average	0.428	0.455	0.044

The behavior between the software and hardware is expected to be slightly different, due to quantization error and inconsistencies in rounding between the systems. Because there are a relatively small number of parameters in the NARNN, it can be very sensitive to quantization error. It is important to select a FP notation that can accurately represent all of the NARNN parameters – we used FP-10/8, which has a range of [-2, 1.99609375]. Retraining was done in Matlab iteratively to derive networks with weights that could be accurately represented within that range. In general, these results demonstrate that the optimizations to the NARNN architecture (parameters and execution flow) have no drastic effect on model performance.

B. FPGA and Custom Hardware Results

The resource requirements of our platform targeting a the smallest available Spartan-7 FPGA (part no. 7s6cpga196) are shown in Table II. Registers are allocated for the parameter cache, delay state matrix, accumulator results and intermediate MAC results in each neuron. Most hardware overheads are in the controller logic between the memory and logic elements. This implementation is suitable for wearable or other external applications, with an estimated power consumption of 38 mW, inference time of 1.07 μs and 35.31 nJ per inference.

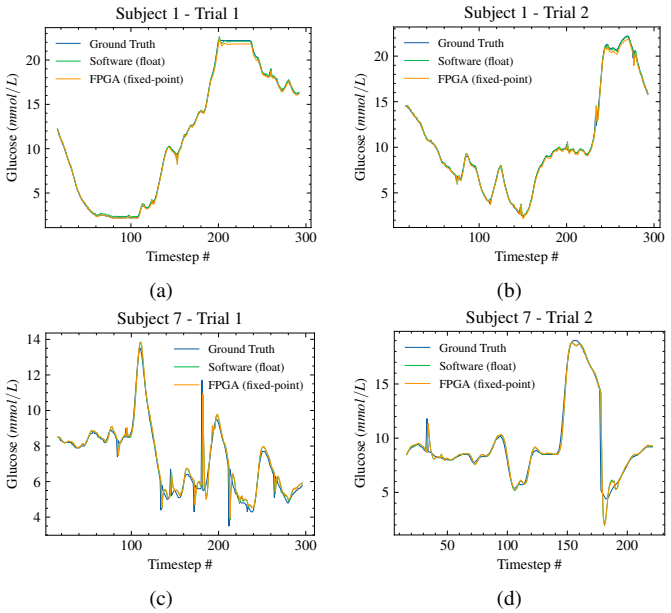

Fig. 2: Plots of predictions both in software and hardware simulations. (a-b) Subject 1, model with lowest RMSE. (c-d) Subject 6, model with highest RMSE.

For ultra-low-power applications like implanted devices, a custom implementation is needed. The proposed architecture was synthesized with Synopsys Design Compiler targeting the SAED 32nm EDK and high V_T gates, excluding the tanh LUT and the weights ROM, which were modeled as black boxes. Memory area and power consumption were estimated using CACTI 6 [10]. Memories were modeled as direct mapped scratch RAMs with no tag array and a blocksize of 2 bytes, far larger than necessary for the required storage. The total area, including synthesized logic and memory arrays, is about $0.02\ mm^2$, which is suitable for integration with an ultra-constrained bioimplantable device.

Low power design options such as array, wordline, and interconnect power gating were enabled for the memories. Despite this, the tool estimates significant leakage power of about $1\ mW$ for the memory arrays, which would not be suitable for an ultra-low power implant. In reality, higher density and lower leakage ROMs or other NVMs must be used, as this will help alleviate the leakage issue. Thus, the CACTI results provide a high upper bound on the area and power estimate. For the synthesized logic, Design Compiler reports static and dynamic power of 317.2 uW and 29.3 uW, respectively, with a cycle time of $5\ ns$. This high static leakage can also be addressed in a number of ways, namely, 1) reducing V_{DD}, trading off delay with power, and 2) coarse-grained power gating using sleep transistors. Option 2 has been shown to reduce leakage power by over 90% in some cases [11] and has been applied successfully in other biomedical circuit applications [12]. In this case, the device only needs to operate for about $200ns$ every 5 minutes (or when a new sample is acquired), for an effective duty cycle of $6.7x10^{-8}\%$, promising a dramatic reduction in static power. The use of emerging NVM technologies would also enable retention of state information during sleep, leading to lower overall power consumption and faster wake times.

TABLE II: Utilization on Spartan-7 (7s6cpga196) FPGA.

	LUT	FF	BRAM	DSP	Power	Latency
Available	3750	7500	5	10	-	-
Required	361	405	1	10	38 mW	1.07 μs

V. Conclusion

In this paper, we have presented an efficient pipelined NARNN architecture targeting bioimplantables. The network can perform error checking and correction to improve security and reliability of these devices in a real-time, closed-loop system. The 5 neuron, 16-tap NARNN was verified in FPGA and matches the software implementation to within 0.044 RMSE on average. Using the SAED 32nm EDK, the synthesized implementation occupies an area of just $0.02\ mm^2$ and requires around $0.71\ nJ$ per inference, making it suitable for use in ultra-constrained devices. Future work will focus on system-level evaluation and energy efficiency improvements.

References

[1] C. K. Boughton and R. Hovorka, "Advances in artificial pancreas systems," *Science translational medicine*, vol. 11, no. 484, 2019.

[2] M. Chen, Y. Hao, K. Hwang, L. Wang, and L. Wang, "Disease prediction by machine learning over big data from healthcare communities," *Ieee Access*, vol. 5, pp. 8869–8879, 2017.

[3] D. Dave, D. J. DeSalvo, B. Haridas, S. McKay, A. Shenoy, C. J. Koh, M. Lawley, and M. Erraguntla, "Feature-based machine learning model for real-time hypoglycemia prediction," *Journal of Diabetes Science and Technology*, vol. 15, no. 4, pp. 842–855, 2021.

[4] M. Asad, U. Qamar, and M. Abbas, "Blood Glucose Level Prediction of Diabetic Type 1 Patients Using Nonlinear Autoregressive Neural Networks," *Journal of Healthcare Engineering*, vol. 2021, 2021.

[5] F. Dubosson, J.-E. Ranvier, S. Bromuri, J.-P. Calbimonte, J. Ruiz, and M. Schumacher, "The open D1NAMO dataset: A multi-modal dataset for research on non-invasive type 1 diabetes management," *Informatics in Medicine Unlocked*, vol. 13, pp. 92–100, 2018.

[6] 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)*. IEEE, 2017, pp. 1–4.

[7] C. Gao, A. Rios-Navarro, X. Chen, T. Delbruck, and S.-C. Liu, "Edge-drnn: Enabling low-latency recurrent neural network edge inference," in *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2020, pp. 41–45.

[8] T. Zhu, L. Kuang, K. Li, J. Zeng, P. Herrero, and P. Georgiou, "Blood glucose prediction in type 1 diabetes using deep learning on the edge," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5.

[9] K. Jeon, J. Kim, J. J. Ryu, S.-J. Yoo, C. Song, M. K. Yang, D. S. Jeong, and G. H. Kim, "Self-rectifying resistive memory in passive crossbar arrays," *Nature Communications*, vol. 12, no. 1, p. 2968, Dec. 2021.

[10] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "CACTI 6.0: A tool to model large caches," *HP laboratories*, vol. 27, p. 28, 2009.

[11] J. W. Tschanz, S. G. Narendra, Y. Ye, B. A. Bloechel, S. Borkar, and V. De, "Dynamic sleep transistor and body bias for active leakage power control of microprocessors," *IEEE Journal of Solid-State Circuits*, vol. 38, no. 11, pp. 1838–1845, 2003.

[12] R. Karam, S. J. Majerus, D. J. Bourbeau, M. S. Damaser, and S. Bhunia, "Tunable and lightweight on-chip event detection for implantable bladder pressure monitoring devices," *IEEE transactions on biomedical circuits and systems*, vol. 11, no. 6, pp. 1303–1312, 2017.