An Event-Driven Recurrent Spiking Neural Network Architecture for Efficient Inference on FPGA

DOI:
10.1145/3546790.3546802

An Event-driven Recurrent Spiking Neural Network Architecture for Efficient Inference on FPGA

Anand Sankaran
a.sankaran@student.utwente.nl
UT-EEMCS & IMEC
Eindhoven, The Netherlands

Paul Detterer
p.detterer@imec.nl
IMEC
Eindhoven, The Netherlands

Kalpana Kannan
kalpana.kannan@imec.nl
IMEC
Eindhoven, The Netherlands

Nikolaos Alachiotis
n.alachiotis@utwente.nl
UT-EEMCS
Twente, The Netherlands

Federico Corradi
f.corradi@tue.nl
TU/e & IMEC
Eindhoven, The Netherlands

ABSTRACT

Spiking Neural Network (SNN) architectures are promising candidates for executing machine intelligence at the edge while meeting strict energy and cost reduction constraints in several application areas. To this end, we propose a new digital architecture compatible with Recurrent Spiking Neural Networks (RSNNs) trained using the PyTorch framework and Back-Propagation-Through-Time (BPTT) for optimizing the weights and the neuron's parameters. Our architecture offers high software-to-hardware fidelity, providing high accuracy and a low number of spikes, and it targets efficient and low-cost implementations in Field Programmable Gate Arrays (FPGAs). We introduce a new time-discretization technique that uses request-acknowledge cycles between layers to allow the layer's time execution to depend only upon the number of spikes. As a result, we achieve between 1.7x and 30x lower resource utilization and between 11x and 61x fewer spikes per inference than previous SNN implementations in FPGAs that rely on on-chip memory to store spike-time information and weight values. We demonstrate our approach using two benchmarks: MNIST digit recognition and a realistic radar and image sensory fusion for cropland classifications. Our results demonstrate that we can exploit the trade-off between accuracy, latency, and resource utilization at design time. Moreover, the use of low-cost FPGA platforms enables the deployment of several applications by satisfying the strict constraints of edge machine learning devices.

CCS CONCEPTS

• **Computer systems organization → Neural networks**.

KEYWORDS

spiking neural networks, FPGA, embedded hardware

ACM Reference Format:
Anand Sankaran, Paul Detterer, Kalpana Kannan, Nikolaos Alachiotis, and Federico Corradi. 2022. An Event-driven Recurrent Spiking Neural Network Architecture for Efficient Inference on FPGA. In *International Conference on Neuromorphic Systems (ICONS 2022), July 27–29, 2022, Knoxville, TN, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3546790.3546802

ICONS 2022, July 27–29, 2022, Knoxville, TN, USA
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9789-6/22/07.
https://doi.org/10.1145/3546790.3546802

1 INTRODUCTION

Spiking neural networks (SNNs) are efficient computational models that enable neural-inspired processing at the edge. SNNs are inspired by principles of biological systems as potentially more powerful processing models, as they replicate the neurons' efficiency and ability to deal with temporal streams of information in novel non-von-Neumann computer architectures. These architectures span from fully custom application-specific integrated circuits (ASICs) in mixed-signal design [1] to fully digital processors [2–4]. While ASICs offer high performance and power efficiency, they are fixed at fabrication. Thus, they do not provide a similar level of programmability as Field-Programmable Gate Arrays (FPGAs). Recent theoretical advances showed that SNNs perform comparably to classical artificial neural networks (ANNs) by exploiting new supervising training strategies [5, 6].

However, deploying SNN models in resource-constrained devices requires that the hardware executes these models with high fidelity with respect to their mathematical abstraction while meeting the tight constraints of edge devices. Furthermore, the hardware should be sufficiently flexible to accommodate possible model changes.

Previous work has proposed to realize programmable spiking neural network architectures in FPGA, implementing large-scale bio-realistic models for neuroscientific and neurocomputational modeling. Some works implement conductance-based neuron models [7, 8], while others implement the leaky-integrate-and-fire (LIF) neuron model for both neuroscientific simulations [9] and fast and efficient inference [10–13]. A limitation of the previous architectures is the use of mean rate models, which require computing the average of the firing activities of neurons over several spikes to obtain an accurate estimate of the represented real value.

In this work, we propose an event-driven architecture compatible with models where the spike activity is limited during training, offering low-computational requirements and fast inference [6].

We present the first digital RSNN architecture compatible with a software-supervised training strategy that exploits surrogate gradient and Back-Propagation-Through-Time (BPTT) for inference in FPGA hardware. Furthermore, our digital architecture achieves

100% accuracy for the timing of every single spike against the software model while allowing us to exploit the trade-off between execution speed and resource utilization.

Thus, the main contributions of this work are the following:
i) We present a fully digital, event-driven architecture with discrete time-steps that achieves 100% accuracy between software simulations and hardware models.
ii) We propose a simplified leaky-integrate-and-fire neuron model that is compatible with discrete-time synchronization strategies and does not require additional memory for storing spike-time information.
iii) We introduce a synchronization scheme that uses an external request signal among neurons and layer operations. As a result, the execution speed solely depends on the number of events in the network without requiring the storage of time information in memory.
iv) We efficiently utilize the hardware resources while exploiting the trade-offs among accuracy, latency, memory, and energy for two use cases (MNIST benchmark and radar-images sensory fusion for cropland classification).

2 BACKGROUND

SNNs are composed of spiking neurons that mimic biological neural networks more closely than traditional analog neural networks (ANN). An essential concept in SNN models is the use of time. Spiking neurons store the weighted history of the synaptic inputs into their membrane potential and only communicate with a pulse when the membrane potential passes a threshold value. Furthermore, ANNs exploit point neuron models as rectified linear units (ReLU) and sigmoid neurons, whereas spiking neurons communicate employing binary pulses (i.e., action potentials or spikes). Several neuron models have been proposed, the Izhikevich [14], the Leaky-Integrate-and-Fire-Neuron (LIF)[15], the Adaptive Leaky-Integrate-and-Fire-Neuron (ALIF) [16], the FitzHugh-Nagumo [17], and more biologically detailed models like the Axon-Hillock model [18]. Nevertheless, to obtain a compact model better suited for digital hardware, it is often required to simplify the neuron model and include fewer biological details. The LIF model is commonly employed as a good compromise between realistic behaviors and the simplicity of the implementation, both in hardware and software models.

Several implementations of SNNs on FPGA have been proposed recently. The optimization of various architectures usually targets specific performance gains (latency, energy, power, throughput, or a combination of those) and, in some cases, application-dependent. Different neuron models are used, and some architectures focus on bio-realistic implementations. Some notable ones are listed in Table 1. In contrast to other designs, our architecture focuses on executing RSNNs with single spike time precision for obtaining high software-to-hardware fidelity, thus allowing efficient and accurate inference as proposed by Yin et al. [6].

3 GENERAL SYSTEM DESIGN

3.1 Event-driven Architecture

Our architecture targets the implementation of neural networks organized in layers of spiking neurons. Figure 1 depicts a block diagram of the proposed architecture, showing a three layers implementation. The architecture supports fully connected layers with and without full recurrent connectivity. When a neuron emits spikes, the spike is delivered to all the neurons in the next layer and every neuron within the same layer. Each layer contains an input queue that stores the incoming spikes from the previous layer and the recurrent activity. Also, each layer requires a weight memory for storing the synaptic weight values.

The SNN architecture is described at the register-transfer level (RTL). It is highly parameterized and allows the creation of different design points using a configuration file. At the architectural level, it is possible to select the number of neurons per layer, the number of layers, the type of connectivity between layers and within a layer (i.e., fully connected and recurrently connected), and the depth of the queues. Furthermore, at the neuron level, it is possible to define the bit-width of the neuron's accumulator (membrane voltage), the synaptic weight resolution (bit-width), and the refractory period. Since FPGAs can be reprogrammed to efficiently implement a custom computer architecture, we decided to leverage the sparsity of network connections during synthesis and not implement synaptic connections with zero weights. By defining the weights as constant values in the RTL description, the FPGA synthesis and subsequent place-and-route optimizations leverage the sparsity of connections as weights of zero value will not consume any resources, and the hardware resources that store weights with the same numerical value will be shared among several neurons. If the application requires weights that can be changed without FPGA reprogramming, we also provide efficient memory mapping techniques in section 4.2. The mapping technique includes a set of parameters that make it easy to use on-chip memories, such as Block RAM (BRAM) and Ultra RAMs.

Figure 1 also shows the internal organization of the RSNN submodules. The layer module contains the following submodules: the request controller, the input spike queue, the weight controller, the weight memory, the neuron wrapper, and the output spike buffer. In the request controller, a finite state machine controls the operations within a layer module and its interface at the input and output sides. The input spike queue acts as a buffer to store the incoming spikes. It contains separate queues for forward spikes (i.e., incoming from the previous layer) and recurrent spikes (from the same layer). The weight controller fetches the right weights from the weight memory by decoding the memory address depending on the source address of the input spikes. The neuron cluster contains an array of simple LIF neurons, described in section 4.1. Finally, the output spike buffer communicates the output spikes to the input spike queue of the next layer. The network communicates by employing spikes both at the input and output interfaces. The spikes are represented through the address of the spiking origin. The behavior of the output layer can be monitored in two ways; each spike is streamed through a buffer. Additionally, it is possible to monitor the membrane potential of each neuron within the output layer.

3.2 Event-driven Time Discretization

Importantly, and in contrast to previous SNN digital architectures in FPGA [10, 13], our architecture does not require a timer module since our simplified neuron model does not rely on time-dependent

Name	Neuron Model	Driven	Network Topology	Algorithm
Bluehive [22]	Izhikevich	Time-driven	Feed-forward and Recurrent	Mean-rate
FDF [23]	Conductance	Time-driven	Feed-forward	Mean-rate
n-Minitaur [24]	LIF	Event-driven	Feed-forward	Mean-rate
Pani [9]	Izhikevich	Time-driven	Recurrent	Mean-rate
Tsinghua [12]	LIF	Hybrid (time and event)	Feed-forward	Mean-rate
Gyro [13]	LIF	Event-driven	Feed-forward and Recurrent	Mean-rate
Irmak et al. [25]	LIF and ReLU	Hybrid (time and event)	CNN, MLP and SNN	ANN and Mean-rate
This work	Simplified LIF	Event-Driven	Feed Forward and Recurrent	Single spike control (BPTT)

Table 1: Spiking neural networks on FPGA.

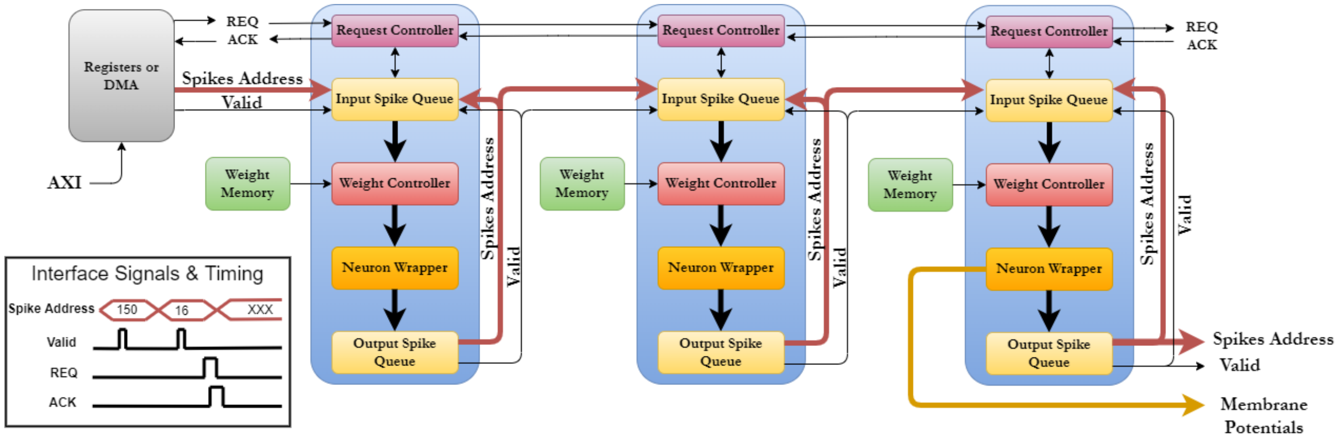


Figure 1: Block diagram of an instance of a recurrent spiking neural network of three layers.

operations. Instead, the membrane potential update and the voltage decay operations are executed on-demand only upon the arrival of input spikes. In addition, each evaluation of the neuron membrane potential against the threshold is solely based on the request signal propagated locally between layers, which provides synchronization. This mechanism serves to synchronize the evaluation of the membrane potentials against the threshold for all the neurons in a layer, only at designated times (i.e., upon request). It ensures that the neuron's evaluation happens only after receiving the full input sequence, enabling time discretization. This mechanism is crucial for high fidelity between software and hardware, as each spike will virtually reach the neuron's membrane potential within the same time window. It is important to note that the synchronization signal does not need to be triggered at fixed intervals. The request controller uses a four-phase handshake mechanism, common in neuromorphic architectures.

4 DESIGN OF FPGA SYSTEM

4.1 Neuron Model

The neuron model is a low-complexity digital version of the LIF model [15]. The membrane potential is updated upon the arrival of a spike. However, the evaluation of the membrane potential against the threshold value is carried out upon the arrival of a request signal (REQ in Fig. 1 and in Fig. 2). The request signal is shared among all neurons in a single layer. The membrane potential is increased or decreased with a step of size equal to the synaptic weight value, i.e., W^{ij}. Since the emission of spikes happens only upon the request signal, the update algorithms do not require storing the time information, as time is divided into discrete steps (i.e., a sequence of request signals). Furthermore, since the request signal is shared between all neurons in the same layer, the neurons in a layer emit spikes simultaneously. This allows to quickly estimate the maximum spike burst size, equal to the number of neurons in a layer, with the buffer queues sizes for the worst-case scenario. Additionally, the membrane potential is not allowed to go negative. It will remain to zero upon the arrival of excessive inhibitory inputs. The neuron only performs computation (i.e., the integration of synaptic input) upon the arrival of spikes, and it implements the leakage operation only upon the arrival of the request signal. The decay of membrane voltage is implemented using a shift register, which calculates a programmable, but fixed decay before evaluation against the threshold value. No dynamic calculation is needed since this is a fixed decay and not a time-dependent value. Decay of the neuron voltage can be enabled or disabled at design time. Decay is not calculated for every input spike but is calculated once per request cycle. The neuron model also uses a fixed refractory, calculated using digital counters based on the number of request cycles. Significantly, sparse spike activity results in faster execution.

The execution time depends on the number of spikes in the layer's queue.

Until the request signal is received, the neuron only performs membrane voltage updates on incoming spikes. The time period between two requests is a discrete chunk of time, and all the input spikes inside that period are accumulated until a request is received. Upon receiving the request, the membrane voltage is compared against the threshold value. An output spike is produced depending on the result. Figure 3 shows the neuron block diagram, and Figure 2 shows the behavior of the neuron. The membrane potential can overshoot the threshold. However, when a REQ signal reaches the neuron, a spike is emitted, and the membrane potential is reset to the resting value (in fig. 2 $Vres = 0$).

4.2 Weight Memory Mapping

Weight memory mapping is a key feature of the proposed architecture since weight storage and on-chip memory access directly affect the latency of the system and memory bandwidth. Every weight-memory module stores a weight matrix representing interlayer and recurrent connections.

Figure 4 (left) shows an example 4×2 network, i.e., 4 pre-synaptic and 2 post-synaptic neurons, and two alternative weight-mapping patterns. The postsynaptic neurons have both forward and recurrent connections. Every column of weights in the weight mapping patterns shown in Figure 4(right) is stored in a dedicated memory module, thereby allowing to read an entire row in a single clock cycle. For the first mapping pattern, when a neuron in the visible layer spikes, all neurons in the hidden layer are updated, and the corresponding weights are fetched from the same memory row in one clock cycle. When a neuron in the hidden layer spikes, the weights for the two postsynaptic neurons are read in a single clock cycle. This results in a throughput of two neuron updates every 6 clock cycles for both forward and recurrent spikes, or two neuron updates every 4 clock cycles for a feed-forward network since only one weight is read per clock cycle. For the second mapping pattern, when a neuron in the visible layer spikes, the two corresponding weights are read from one row in one clock cycle. When a neuron in the hidden layer spikes, the weights for the two neurons are read in a single clock cycle, and the resulting throughput is two neuron updates per 3 clock cycles for both forward and recurrent spikes, or two neuron updates per 2 clock cycles for a feed-forward network since two weights are read per clock cycle.

We provide a generic parameter-based description of the weight-mapping scheme below. For storing the weight matrices between V visible neurons and H hidden neurons, the memory is visualized as a three-dimensional block with x, Y, and Z coordinates as depicted in Figure 4 (right). Here, X represents the bit width of a single weight, Y represents the number of weights in a single memory row (total memory width), and Z represents the number of distinct memory modules. The memory width is described in terms of the number of weights.

When visualized in a three-dimensional manner, as shown in Figure 4 (right), the highlighted block with dimensions X_1, Y_1, X_1, contains the set of H weights corresponding to a single visible neuron V. The volume of this block is the number of hidden neurons H, $X_1 \times Y_1 \times Z_1 = H$. This block is repeated X_2 times in the X dimension, Y_2 times in the Y dimension, and Z_2 times in the Z dimension for the entire forward weight matrices between the 2 layers. The number of block instances is equal to the number of visible neurons since there is a single block for each forward spiking neuron. To update a set of hidden neurons upon spiking of a forward neuron, X_1 by Y_1 weights are read from Z_1 memories.

The recurrent connections are stored with an offset in the Y dimension. A block for a single recurrent spiking neuron is structurally the same as for a forward spiking neuron (represented by X_1, Y_1, Z_1) and is repeated X_2 times in the X dimension, Y_2_r times in the Y dimension, and X_2_r times in the X dimension. The values Y_2_r and X_2_r are derived from the previous parameters, where Y_2_r is the minimum of Y_1 and $Z_1 \times X_1$, while Z_2_r is the maximum of Y_1 and Z_1. Increasing the Z parameter requires more distinct memories and allows more weights to be read in parallel in a single clock cycle, effectively increasing the memory bandwidth. To update a set of hidden neurons upon spiking of a hidden neuron X_1 by Y_1 weights are read from Z_1 memories. The parameters X_n and Z_n indicate how many weights are read in a clock cycle, thereby allowing $X_n \times Z_n$ neurons to be updated per clock cycle ($X_n \times Z_n$ is the neuron cluster size C).

4.3 Finite State Machine / Request Controller

The finite state machine controls the internal functioning of the layer and communication to the next layer by scheduling and triggering the sub-block operations. It manages the interlayer communication based on its internal state, state of the queues, network input signals, or output signals of other layers. The state machine contains five states: *idle*, *process*, *communicate*, *acknowledge*, and *finish*. Each state and its function, along with control signals, are as follows. In the *idle* state, the layer is ready to receive input spikes from the network input or the previous layer. The incoming spikes are stored in the spike queue. Upon request from the network input or previous layer, the finite state machine switches from idle to process state. In the *process* state, the neuron spikes are processed, and the membrane voltage is accumulated for each of the input spikes. The spikes in the queue received before the request signal are processed until both the recurrent queue and forward spike queue are empty. The spikes received after receiving a request signal are not processed until the next request is received and are buffered in the queue until then. Once the input spike queues are emptied and neuron voltages updated, the finite state machine switches to *communicate* state, and the threshold evaluation of the LIF neurons is triggered. The neuron membrane voltage is evaluated against the threshold voltage and generates an output spike depending on whether it is above the threshold voltage or not. If a neuron spikes, the output spike is stored in the output spike queue. The finite state machine switches into the acknowledge state after storing all resulting spikes in the output spike queue. In the *acknowledge* state, the layer sends the output spikes out. This conveys that data processing is complete to the input side, thereby completing the request-acknowledge 4-phase handshake. The output spikes are sent as a burst of spikes to the output or next layer and to the recurrent spike queue within the same layer. In the *finish* state, the request signal for the next layer is set as high, which triggers the next layer into the same state transitions through which it processes

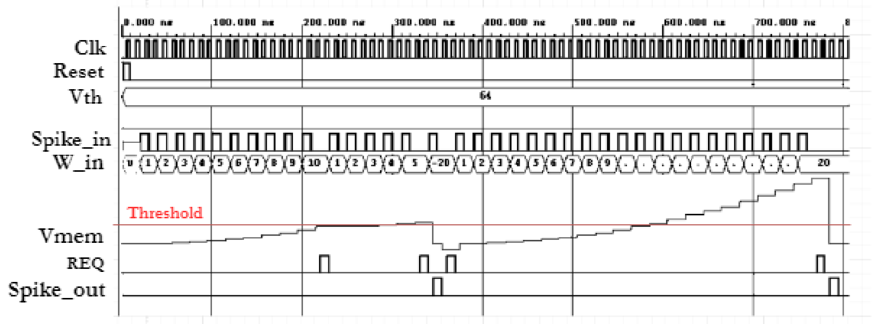


Figure 2: Neuron behavioral simulation. The neuron evaluates the membrane potential against the threshold only upon the arrival of the request signal.

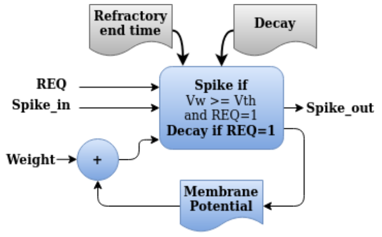
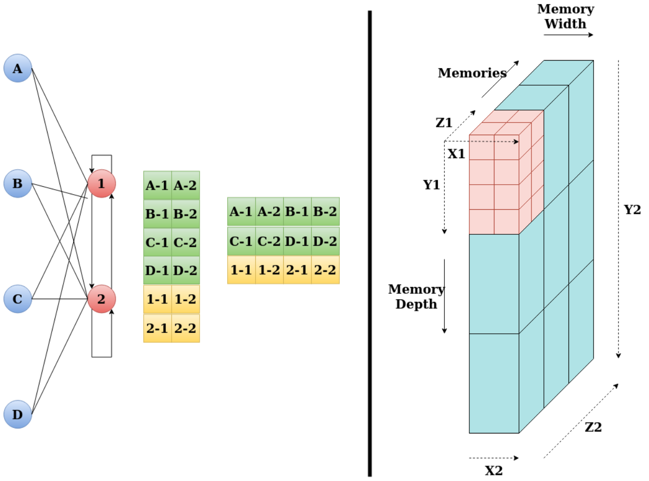


Figure 3: Simplified neuron's block diagram.



A-1	A-2
B-1	B-2
C-1	C-2
D-1	D-2
1-1	1-2
2-1	2-2

A-1	A-2	B-1	B-2
C-1	C-2	D-1	D-2
1-1	1-2	2-1	2-2

Figure 4: Left: A sample network of two layers and two weight mapping patterns. Right: 3-dimensional visualization of memory

the input spikes received from the current layer. Upon receiving an acknowledge signal from the next layer or output, the finite state machine switches back to the idle state to reset its control signals. This completes the request-acknowledge handshakes on both the layer's input and output sides.

4.4 Hierarchy/Structure

The spike queue receives two streams of spikes, one for forward spikes from the previous layer and one for recurrent spikes from the same layer. Each stream is separately stored using a First In, First Out (FIFO) buffer. The input queues are emptied based on triggers from the state machine. The recurrent spikes are processed first, followed by the spikes in the forward spike queue. The state machine schedules the reading of the spike. The input of the recurrent spike queue is an internal signal. The output from the neuron wrapper is sent to both the output of the layer and to the recurrent queue. The maximum length of the queues is also parameterized and, therefore, it is user-controllable.

The weight controller fetches weights from memory depending on the spike source address. The spike source address and spike direction (recurrent or forward) are latched from the spike queue in the initial state. The memory address for the required weights is derived from the source address by the address decoder. The state machine latches the spike source address and the direction to the address decoder, and it generates the memory addresses for forwarding and recurrent weights. The weight control module fetches the weights as vectors and then provides the weights to the neurons for spike processing.

The neuron wrapper consists of the simplified LIF spiking neurons and a parallel to serial converter to buffer and serialize the output spikes. Neurons operations are executed in parallel. The output spikes from a layer are serialized into a single stream of spike source addresses. An N-bit parallel-in serial-out shift register serializes the output, where N is the total number of neurons in the layer. It takes in the output spikes from N neurons as an N bit vector. For every neuron that spiked, an output spike is generated as the address of the neuron that spiked. The output spikes are sent out one per clock cycle. The maximum number of spikes that can originate from a layer is the total number of neurons for every request signal since each neuron is evaluated against the threshold only once per request. This is advantageous as it eliminates the need to multiplex the output spikes between neuron clusters.

The input spikes of a layer are a combination of a valid binary signal and a source address of the input neuron. Source address conveys which input neuron spiked, and the valid signal signals at the receiver the readiness of the address. Additionally, each layer has a request signal that is used to generate discrete time steps (the acknowledge-request cycles). The activity of the output layer can be observed in three ways. Firstly, as a stream of sequential output spikes encoded with the source addresses of the active neurons, secondly, as a parallel bus with the bit positions representing the address (i.e., a compressed spike map). Thirdly, it is possible to output the membrane potential of neurons at each request acknowledge cycle. Since the threshold evaluation of every neuron happens in parallel, the output spikes are sent out at the same instant and can be observed as a burst of output spikes.

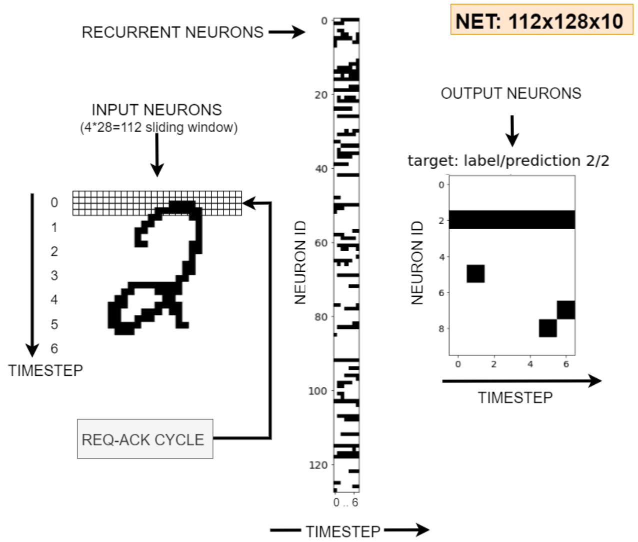


Figure 5: Network activity for the SRNNs classifying a stream of spikes (black pixels) representing an MNIST digit. A whole digit is provided in seven timesteps (i.e., req-ack cycles, from 0 to 6). The output neuron corresponding to the input class is the most active, showing correct inference.

5 EXPERIMENTAL RESULTS

5.1 Recurrent Spiking Neural Networks for Classification

5.1.1 MNIST handwritten digit dataset. To demonstrate the application of the digital SNN architecture, we trained a spiking neural network to classify handwritten digits. We used the PyTorch framework and implemented a detailed model of our simplified digital LIF neuron model. We performed quantization and scaling to make the training compatible with the integer precision of our RSNN hardware[1]. Additionally, we trained the RSNN using surrogate gradient and BPTT to achieve high accuracy and high activation sparsity as in [6]. The input to the RSNN is a binary stream of spikes that matches handwritten digits, as shown in Figure 5. Several networks with a different number of layers and input sizes have been trained. We used three input dimensions, a single row of input per request-acknowledge cycle (28 input at each timestep, see Figure 5), with four rows of input spikes for each request-acknowledge cycle ($28 \times 4 = 112$ inputs neurons), and with 14 rows of input spikes for each request-acknowledge cycle ($28 \times 14 = 392$ neurons). The accuracy and the effect of the quantization can be seen in Figure 6. We used quantization-aware training to four bits, and less than 4-bit weights significantly deteriorate accuracy.

Figure 5 shows an RSNN of three layers of neurons. The input layer contains 112 neurons, representing four-row of the binary MNIST input digits. The second layer contains 128 neurons which receive 112 inputs and are fully recurrently connected. The first layer is fully connected to the second layer, which contains 10 output neurons whose activity reflects the output classification. In one request-acknowledge cycle, the inputs from 112 neurons represent four rows of the handwritten digit. Figure 5 shows the spiking of the network. The most active neuron in the output layer represents the digit classification. Increasing the number of parallel pixels presented at the input makes it possible to process the full picture in fewer time steps. This comes at the cost of more hardware memory for the input layer. Nevertheless, the time steps required to perform a classification are reduced. In addition, a larger input size also reduces the number of spikes required to perform a full inference, as neurons are only allowed to spike only once per time step. Finally, in Table 2 we show an exploration of input size (28,112,392) versus accuracy and resources.

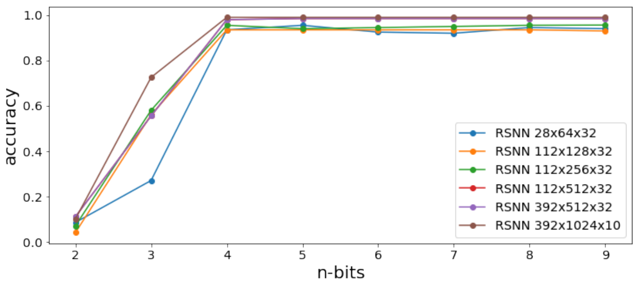


Figure 6: MNIST accuracy in function of weight-bit resolution and network size.

5.1.2 Sensory Fusion of Radar Data and Images for Remote Sensing. We used a publicly available dataset from a real remote sensing application that requires the classification of fused optical and radar data [20]. The dataset contains many features acquired from an optical camera on the RapidEye satellite and radar-based information collected by the Uninhabited Aerial Vehicle Synthetic Aperture Radar (UAVSAR) system over the agricultural region near Winnipeg, Manitoba, Canada in 2012. The dataset contains 2x49 radar features and 2x38 polarimetric radar features. The dataset contains seven crop types: corn, peas, canola, soybeans, oats, wheat, and broadleaf. Class distribution in the cropland dataset is highly unbalanced and it is as follows: corn 12,02%, peas 1,10%, canola 23,22%, soybeans 22,73%, oats 14,46%, wheat 26,12%, and broadleaf 0,35%. [21] (see Fig. 7). The dataset contains 174 features per pixel. However, after a feature correlation analysis as proposed in [21], we filtered 72 features as they show high correlations (> 0.95) with other features. The resulting number of optical-radar features is 102 per pixel. In addition, the dataset has been split into 80% training and 20% testing, maintaining the same class distribution as in the dataset. We used 260667 pixels for training and 65167 pixels for testing. As for the MNIST use case, we trained an RSNN with 102 inputs (i.e., one inference for each pixel) and 600 hidden neurons with 8 output neurons, one for each output class. The input pixel value is fed with a 6-bit intensity representation to the first layer of 600 hidden spiking before providing the first request (REQ) signal. This allows encoding input values at a higher resolution than binary by repeating the input spikes multiple times before proving a REQ signal.

The network configuration is 17-600-8 with a clock frequency of 250 MHz on the Zynq UltraScale+ MPSoC ZCU104 evaluation board

[1] code will be made available in Github

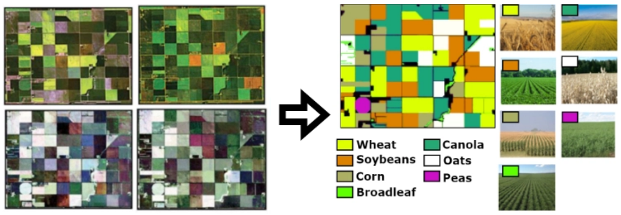


Figure 7: Dataset of radar and image data used for pixel-wise classification [20].

for the cropland classification test case. A network of 600-8 neurons was implemented for this classification. It uses 17 inputs in six-time steps to provide a full image for classification. The parameters x_1, y_1, z_1 for weight memory mapping are 1-20-30 for the first layer. The throughput for each layer is calculated and gives a total of 8.14 GSOPS. Since we trained the network with sparse recurrent connectivity, we exploited weight placement during synthesis for both the MNIST and cropland applications. This resulted in the use of LUTs and BRAMs due to the optimization procedure. The resource utilization for a fully connected layer is quadratically proportional to the number of neurons in a layer. However, this can be limited if training results in sparse connectivity. We achieved weight sparsity by simply removing some of the connections during training. The 17-600-8 network required 54,516 LUTs, 21,756 FFs and 312 BRAMs. We achieved an accuracy of 95% with a 4-bits synaptic weight resolution.

5.2 Resource utilization and accuracy

We measure resource utilization in FPGA in terms of the Lookup Tables (LUTs), Flip-Flops (FF), and BRAMs used. Table 2 shows resource utilization for the implementation of several networks of different sizes in the case of MNIST classification. The table shows how the accuracy varies according to the size of the network. The resource utilization here does not include the external registers, resets, or the AXI interface logic.

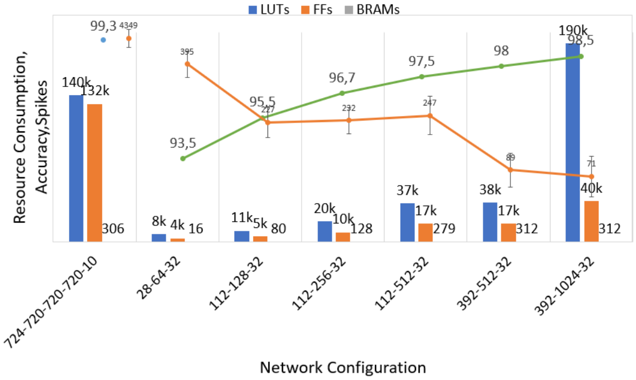


Figure 8: Resource utilization for different network sizes. The accuracy increases for larger networks.

The resource utilization is directly proportional to the number of neurons in a network, which can be observed in Figure 8. More neurons require more configurable logic for implementation, and the synaptic memory takes up most resources. All networks were implemented with 4-bit-weight resolution. Figure 6 shows a plot of accuracy against different weight bit resolutions for the same network size. A relatively low bit resolution of 4 bits is not a limiting factor in achieving high accuracy. But a sufficient number of neurons in the hidden layers are significant for reducing the error rate for low weight bit resolution.

5.3 Throughput

We define throughput as the number of synaptic operations per second (SOPS). When a neuron generates an output spike to N neurons, it causes N synaptic operations. The peak throughput depends upon the weight mapping parameters and the clock frequency. The weight mapping parameters define the memory size and layer size. Upon receiving a forward input spike, the layer does $X_1 \times Z_1$ synaptic operations per clock cycle. The layer requires Y_1 clock cycles plus one clock cycle overhead to update all neurons in the layer. The peak throughput in synaptic operations per time unit for a layer is calculated for every neuron in the layer, which is $X_1 \times Y_1 \times Z_1$. The total throughput is the peak throughput for every individual layer added. The peak throughput of the RSNN therefore is:

$$\sum_{l=2}^{L} \frac{X_1 \times Y_1 \times Z_1}{t_{clk} \times (Y_1 + 1)}, \tag{1}$$

where l indicates the hidden layers, L is the total number of hidden layers, and X_1, Y_1, Z_1 are the parameters of the respective layer.

Throughput is maximized by minimizing Y_1. However, to maximize the utilization of the memory in terms of memory depth, it is favorable to maximize $Y_1 \times Y_2$ to the maximum available memory depth. So a trade-off can be made by minimizing Y_1 to increase the throughput and increasing Y_2 proportionately to maximize the utilization of memory in terms of depth. Throughput values calculated for multiple networks of different sizes can be observed along with the respective weight mapping parameters for 2 hidden layers in Table 3.

6 CONCLUSIONS

We present a novel digital architecture for implementing RSNNs with perfect software-to-hardware fidelity. Our results demonstrate accurate inference and low spike activity. The architecture is fully digital, and it is compatible with FPGA hardware. It offers adjustable weight resolution, layers count, number of neurons, programmable recurrent connectivity, and other parameters. Furthermore, we have proposed a memory layout for on-chip memories (BRAM and UltraRAM) that helps achieve desired peak throughput performances at design time. We demonstrated two inference applications, the handwritten digit (MNIST) and the sensory fusion task of images and radar data for cropland classification. The presented architecture enables network size, memory resources (weight resolution), throughput, and accuracy trade-offs. In the MNIST benchmark, we have set up several network configurations with different input sizes. We demonstrated that we can exploit the recurrent connections to store information over previous time steps while still accurately solving the task (see Table 2). However, with input layers

Network	# Neu	# LUT	# FF	# BRAM	Accuracy	nbit	Spikes/Inf.	Resource Savings [x]
784-720-720-720-10 [13]	2954	140,206	131,977	306	99.3%	6	4349 ± 352	1(LUTs), 1(FFs), 1(BRAMs)
28-64-10	124	7,559	3,604	16	93.5%	4	395 ± 38	18x(LUTs), 36x(FFs), 36x(BRAMs)
112-128-10	272	10,783	5,284	80	95.5%	4	227 ± 44	18x(LUTs), 13x(FFs), 25x(BRAMs)
112-256-10	400	19.537	9.345	128	96.7%	4	232 ± 38	10x(LUTs), 7x(FFs), 2.3x(BRAMs)
112-512-10	656	37,010	17,447	279	97.5%	4	247 ± 55	3.7x(LUTs), 4x(FFs), 1.1x(BRAMs)
392-512-10	936	37,706	17,492	312	98.0%	4	89 ± 48	3.7x(LUTs), 7.5x(FFs), 0.98x(BRAMs)
392-1024-10	1439	189,561	39,170	312	98.5%	4	71 ± 58	0.74x(LUTs), 3.3x(FFs), 0.98x(BRAMs)

Table 2: Resource utilization, accuracy, average spike per inference, and resource savings for MNIST handwritten classification networks. Resource savings has been normalized to Gyro [13].

Network	Parameters (X_1, Y_1, Z_1)	Throughput (GSOPS)
28-64-32	1-16-4,1-8-4	1.83
112-128-32	1-32-4,1-8-4	1.85
112-256-32	1-64-4,1-8-4	1.87
112-512-32	1-64-8,1-8-4	2.85
392-512-32	1-64-8,1-8-4	2.85
392-1024-32	1-64-16,1-8-4	4.82
17-600-8	1-20-30,1-1-8	8.14

Table 3: Peak throughput for different network sizes and weight mapping parameters.

of low neuron counts, it is required to provide more input data chunks in sequence for presenting the whole input digit, trading off accuracy and latency with FPGA resources. When fewer neurons are at the input layer, the task becomes more challenging because the network needs to remember the previous timesteps to make the right decision via recurrent activity and neuronal dynamics, resulting in more latency and spikes but requiring fewer FPGA resources.

ACKNOWLEDGMENTS

This project received funding from the ECSEL Joint Undertaking (JU) under grant agreement No. 826610. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Spain, Austria, Belgium, Czech Republic, France, Italy, Latvia, and Netherlands.

REFERENCES

[1] Moradi S, Qiao N, Stefanini F and Indiveri G. "A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs)." IEEE transactions on biomedical circuits and systems, 2017, 12(1), pp.106-122.

[2] Davies M, Srinivasa N, Lin TH, Chinya G, Cao Y, Choday SH, Dimou G, Joshi P, Imam N, Jain S and Liao Y. "Loihi: A neuromorphic manycore processor with on-chip learning." IEEE Micro. 2018 Jan 16;38(1):82-99.

[3] Akopyan F, Sawada J, Cassidy A, Alvarez-Icaza R, Arthur J, Merolla P, Imam N, Nakamura Y, Datta P, Nam GJ, Taba B. "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip." IEEE transactions on computer-aided design of integrated circuits and systems. 2015 Aug 28;34(10):1537-57.

[4] Mayr C, Höppner S, Furber S. "SpiNNaker 2: A 10 Million Core Processor System for Brain Simulation and Machine Learning." In Communicating Process Architectures 2017 & 2018 2019 (pp. 277-280). IOS Press.

[5] Neftci EO, Mostafa H, Zenke F. "Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks." IEEE Signal Processing Magazine. 2019 Nov 5;36(6):51-63.

[6] Yin B, Corradi F and Bohté SM. "Accurate and efficient time-domain classification with adaptive spiking recurrent neural networks." Nat Mach Intell 3, 905–913 (2021). https://doi.org/10.1038/s42256-021-00397-w

[7] Smaragdos G, Isaza S, van Eijk MF, Sourdis I, Strydis C. "FPGA-based biophysically-meaningful modeling of olivocerebellar neurons." In Proceedings of the 2014 ACM/SIGDA international symposium on Field-programmable gate arrays 2014 Feb 26 (pp. 89-98).

[8] Wang RM, Thakur CS, Van Schaik A. "An FPGA-based massively parallel neuromorphic cortex simulator." Frontiers in neuroscience. 2018 Apr 10;12:213.

[9] Pani D, Meloni P, Tuveri G, Palumbo F, Massobrio P, Raffo L. "An FPGA platform for real-time simulation of spiking neuronal networks." Frontiers in neuroscience. 2017 Feb 28;11:90.

[10] Neil D, Liu SC. "Minitaur, an event-driven FPGA-based spiking network accelerator." IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 2014 Jan 9;22(12):2621-8.

[11] Mostafa H, Pedroni BU, Sheik S, Cauwenberghs G. "Fast classification using sparsely active spiking networks." In IEEE International Symposium on Circuits and Systems (ISCAS) 2017 May 28 (pp. 1-4).

[12] Han J, Li Z, Zheng W, Zhang Y. "Hardware implementation of spiking neural networks on FPGA." Tsinghua Science and Technology. 2020 Jan 13;25(4):479-86.

[13] Corradi F, Adriaans G, and Stuijk S. "Gyro: A Digital Spiking Neural Network Architecture for Multi-Sensory Data Analytics." In Proceedings of the 2021 Drone Systems Engineering and Rapid Simulation and Performance Evaluation: Methods and Tools Proceedings, pp. 9-15. 2021.

[14] Izhikevich EM. "Simple model of spiking neurons." IEEE Transactions on neural networks. 2003 Nov;14(6):1569-72.

[15] Abbott LF. "Lapicque's introduction of the integrate-and-fire model neuron (1907)." Brain research bulletin. 1999 Nov 1;50(5-6):303-4.

[16] Bellec G,Salaj D, Subramoney A, Legenstein R, and Maas W. "Long short-term memory and learning-to-learn in networks of spiking neurons." In Advances in neural information processing systems, v. 31. 2018.

[17] Izhikevich EM, FitzHugh R. "Fitzhugh-nagumo model." Scholarpedia. 2006 Sep 23;1(9):1349.

[18] Danneville F, Loyez C, Carpentier K, Sourikopoulos I, Mercier E, Cappy A. "A Sub-35 pW Axon-Hillock artificial neuron circuit." Solid-State Electronics. 2019 Mar 1;153:88-92.

[19] Stuijt J, Sifalakis M, Yousefzadeh A, Corradi F. "μBrain: An event-driven and fully synthesizable architecture for spiking neural networks." Frontiers in neuroscience. 2021 May 19;15:538.

[20] Dataset available at UCI Machine Learning Repository. Query: "Crop mapping using fused optical-radar data set Data Set." https://archive.ics.uci.edu/ml/datasets/Crop+mapping+using+fused+optical-radar+data+set

[21] Khosravi I, Alavipanah SK. "A random forest-based framework for crop mapping using temporal, spectral, textural and polarimetric observations." International Journal of Remote Sensing. 2019 Sep 17;40(18):7221-51.

[22] Moore SW, Fox PJ, Marsh SJ, Markettos AT, Mujumdar "A. Bluehive-a field-programable custom computing machine for extreme-scale real-time neural network simulation." In IEEE 20th International Symposium on Field-Programmable Custom Computing Machines 2012 Apr 29 (pp. 133-140).

[23] Wang R, Hamilton TJ, Tapson J, van Schaik A. "An FPGA design framework for large-scale spiking neural networks." In IEEE International Symposium on Circuits and Systems (ISCAS) 2014 Jun 1 (pp. 457-460).

[24] Kiselev I, Neil D, Liu SC. "Event-driven deep neural network hardware system for sensor fusion." In IEEE International Symposium on Circuits and Systems (ISCAS) 2016 May 22 (pp. 2495-2498).

[25] Irmak H, Corradi F, Detterer P, Alachiotis N, Ziener D. "A Dynamic Reconfigurable Architecture for Hybrid Spiking and Convolutional FPGA-Based Neural Network Designs." Journal of Low Power Electronics and Applications. 2021 Sep;11(3):32.