A Runtime-Reconfigurable Hardware Encoder for Spiking Neural Networks

Sk Hasibul Alam
Min H. Kao Department of Electrical Engineering & Computer Science
The University of Tennessee
Knoxville, Tennessee, USA
hasib@utk.edu

Adam Foshie
Min H. Kao Department of Electrical Engineering & Computer Science
The University of Tennessee
Knoxville, Tennessee, USA
afoshie@vols.utk.edu

Garrett Rose
Min H. Kao Department of Electrical Engineering & Computer Science
The University of Tennessee
Knoxville, Tennessee, USA
garose@utk.edu

ABSTRACT

In order for raw sensory data to be processed by spiking neural networks (SNNs) an intermediary spike encoder must translate that data into a spike-train. Since there is no one-size-fits-all encoding method suitable for every neuromorphic application, the necessary encoding scheme differs from one implementation to the next. A similar circumstance exists concerning the encoding interval, or frame, that a spike-train is produced for. Although research exists on individual encoding schemes with a rationale for excluding other methods for a particular application, no neuromorphic implementation has addressed a dedicated hardware encoder that is compatible with multiple encoding methods. In this study, we introduce an encoder module which supports three major encoding schemes. The encoding method, as well as the encoding frame duration, can be easily tweaked at runtime. Both FPGA and VLSI implementations have been created for this encoder that are highly scalable and fast, with the latter running with a clock frequency of up to 530 MHz in a 65-nm process. The small area and power footprint of this design makes it attractive for any hardware-based neuroprocessor without needing any external software, or hardware, based process for data encoding.

CCS CONCEPTS

• **Hardware → Application specific integrated circuits**; **Datapath optimization**.

KEYWORDS

spike-train, rate encoding, temporal encoding, spiking neural network

ACM Reference Format:
Sk Hasibul Alam, Adam Foshie, and Garrett Rose. 2023. A Runtime-Reconfigurable Hardware Encoder for Spiking Neural Networks. In *Proceedings of the Great Lakes Symposium on VLSI 2023 (GLSVLSI '23), June 5–7, 2023, Knoxville, TN, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3583781.3590284

GLSVLSI '23, June 5–7, 2023, Knoxville, TN, USA

ACM ISBN 979-8-4007-0125-2/23/06…$15.00
https://doi.org/10.1145/3583781.3590284

1 INTRODUCTION

Brain-inspired computation in the form of spiking neural networks (SNNs) is a growing field of research largely thanks to its promising potential in terms of power efficiency and usefulness in a variety of real-time applications [13]. The way in which SNNs process data differs from traditional neural networks in that the data takes the form of a spike. These spikes have no value and instead rely on their spatio-temporal relation to one another to convey the information to be processed.

Despite the emergence of a number of hardware-based neuroprocessors in the last decade, encoding external data into spikes has remained a primarily software based endeavor [15]. This means that the neuroprocessor still relies on some conventional processing to enable it to process data from peripheral sensors. Even in the instances where custom hardware has been created for spike encoding, these implementations are often limited to a single spike encoding scheme and lack flexibility in terms of their runtime reconfigurability [3, 11, 16, 17].

In this work, we demonstrated an encoder that supports three major encoding techniques for value up to 6 bits, and a 4-bit destination address for the spike-train being generated. The module also supports three different sizes for the encoding frame – 16, 32, and 64 time-steps. Both the encoding method and frame duration can be reconfigured at runtime. For the VLSI implementation, a special 65-nm research PDK run at SUNY has been utilized. Using this node, our silicon implementation of the encoder can run at a maximum clock frequency of 530 MHz. The clever use of look-up table has made our hardware design highly scalable and fast, occupying minuscule area footprint.

2 BACKGROUND

When controlled by a spiking neural network, application performance is substantially impacted by the scheme chosen for encoding data into spikes [14]. By themselves, traditional spikes have no variable magnitude to convey information. In this study, three types of spike encoding schemes are used to convert numerical data into a spike train —

- Rate Encoding: The *frequency* of spikes in a spike-train is directly proportional to the value being encoded.
- Temporal Encoding: The *timing* of a spike in a spike-train is directly proportional to the value being encoded.
- Multi-Spikes Encoding: The *number* of spikes in a spike-train is equal to the value being encoded.

An ideal timing diagram that illustrates the differences between each of these encoding schemes is shown in Figure 1. An encoding

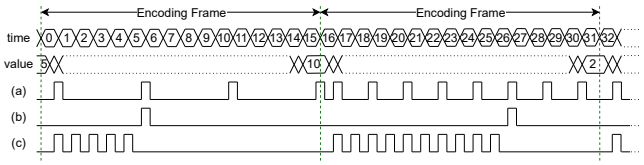


Figure 1: Ideal timing diagram for (a) rate, (b) temporal, and (c) multi-spikes encoding.

frame, as shown in Figure 1, is defined as a fixed number of time steps for which a value is encoded into spikes.

Of these three spike encoding schemes, the two that are most prominent in recent literature are rate encoding and temporal encoding. The temporal method for encoding spikes has been employed by Bohte et al., who utilized population coding to encode input temporally [2]. Temporal encoding has been employed by Mostafa for the classification of an MNIST dataset containing images of handwritten digits [9]. Rate encoding has been used by Liu et al. [8], Querlioz et al. [12], Bagheri et al. [1], for recognition of images with digits. A toolchain for Intel's Loihi has been presented by Lin et al., which includes a Python-based API, a compiler and a runtime library [7]. An SNN classifier has been made by Liang et al. using temporal encoding, where preprocessed pixel intensity is converted to spike time [6].

There are also some other works for hardware-based encoder, but most of them are limited to address-event representation (AER) based vision sensors [10]. Rate encoding has been employed by Culurciello et al. in a CMOS imager [3]. On the other hand, temporal encoding scheme has been utilized by Qi et al. [11], Shoushun and Bermak [16], in fabricated CMOS imager. Intel's Loihi 2 introduced hardware acceleration for encoding that supports graded spike events [5], whereas Loihi handles spike input using its x86 embedded processors and supports binary spike events [4]. A dedicated encoder module has been designed by Yi et al. using 0.18-µm process [17]. But that capacitance-based module only supports temporal encoding of analog data, where each analog neuron consumed a minimum power of 0.26 µW.

Although all previous works have emphasized on a single encoding method of their choice, none of them has demonstrated the capability or scalability to support others, let alone alter the method or frame size at runtime.

3 METHODOLOGY

The encoder in this study has three major parts — input handler, spike generator, and neuron selector. A block diagram of the encoder is shown in Figure 2. These are described in details in this section.

3.1 Input Handler

This part manages all binary inputs using registers and counters. The input space can be divided into eight segments:

3.1.1 `sc_in`. This is an one-time scan chain bitstream that is initially pushed into a system the encoder is part of.

3.1.2 `data_in`. This is a 10-bit input. Its least significant six bits contain the value which will be translated into a spike-train using

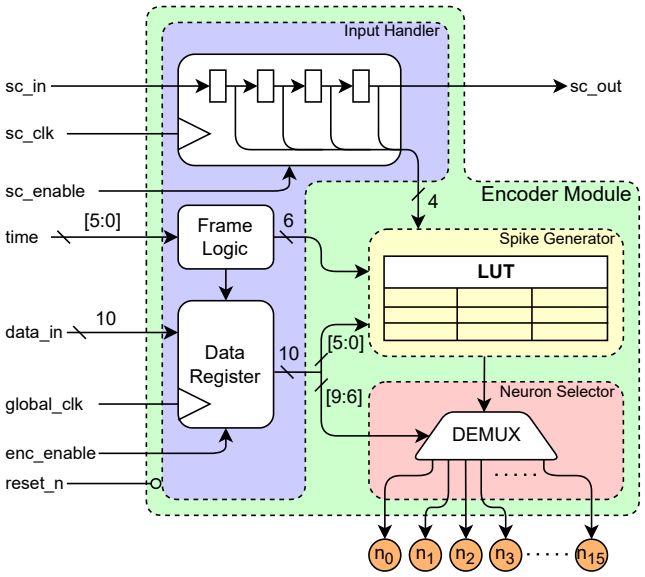


Figure 2: Block diagram of the Hardware Encoder.

the spike generator. The most significant four bits contain the destination address of the output spike-train.

3.1.3 `sc_clk`. This is one of the two asynchronous clock signals utilized in the encoder module. It is associated with the scan chain which determines how fast the `sc_in` moves through the system.

3.1.4 `global_clk`. This is the fastest clock signal which controls every register except those used for the scan chain.

3.1.5 `time`. This is a 16-bit signal that counts from 0 to $2^{16} - 1$. Depending on the scan chain, every k-number of time-steps make an encoding frame, where $k \in \{16, 32, 64\}$. With the help of `global_clk`, the `data_in` is sampled only at the beginning of each frame, e.g., if k is 16, the sampling will be done at `time` = 0, 16, 32, 48, and so on.

3.1.6 `sc_enable`. This is one of the two mutually dependent enable signals. It determines when to stop the scan chain. This pin is initially asserted high to move the scan chain through the system. As soon as it is asserted low, the stream of `sc_in` stops.

3.1.7 `enc_enable`. This determines when to start or stop the encoding process. It is initially asserted low as long as the `sc_in` is moving. It can be asserted high as soon as that `sc_enable` goes low.

3.1.8 `reset_n`. This is a synchronous, active-low signal that resets all registered signals inside the encoder module to their respective initial values.

3.2 Spike Generator

This part generates the spike-train using a 64 × 24-byte look-up table (LUT). The LUT takes three different set of inputs to build the train. As soon as the LUT obtains new inputs, the appropriate voltage, high or low, is available.

- Value: It comes from part of the data register. This is the value that is to be translated into spike-train. The range of this value is strictly equal to the duration of encoding frame.

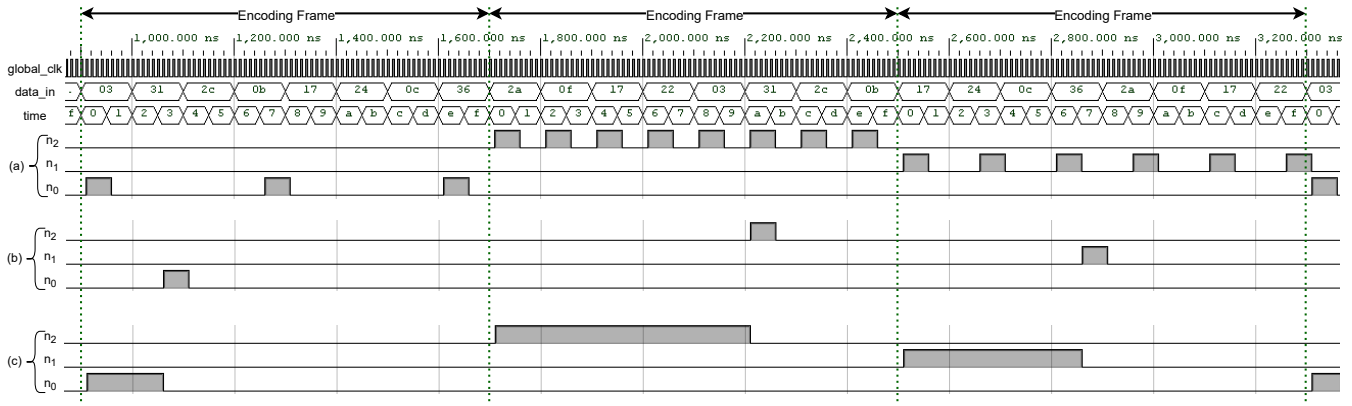


Figure 3: Timing diagram from Xilinx Vivado behavioral simulation demonstrating (a) rate, (b) temporal, and (c) multi-spikes encoding with the proposed reconfigurable spike encoder at a frame duration of 16 time-steps. The lower and upper nibble of `data_in` represents to-be-encoded value and destination address, respectively. Not all signals are included in this figure.

- Scheme: It comes from the shift register of scan chain that selects one of the three encoding schemes to be used.
- Timing: It comes from the frame logic block. This decides whether the current time-step is suitable for a spike or not.

3.3 Neuron Selector

It is a 4-to-16 demultiplexer. The spike-train from the spike generator goes to one of the sixteen different neurons, or clusters of neurons, using the most significant four bits from the `data_in`.

4 FPGA IMPLEMENTATION

The encoder has been successfully implemented on four different FPGA boards — Basys 3, PYNQ-Z1, Nexys A7, Arty A7. A timing diagram for the duration of three encoding frames is shown in Figure 3.

4.1 Timing and Power Analysis

For each board, this analysis has been done using the onboard oscillator as `global_clk`, and a multi-region clock capable (MRCC) I/O pin as `sc_clk`, at 100 MHz and 50 MHz, respectively. All other input and output ports have utilized similar switches and pins across different boards.

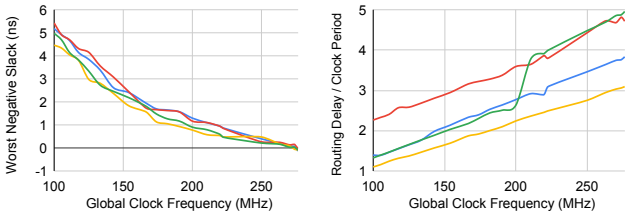
4.1.1 Setup and Hold Slack. The setup time and hold time constraint for each of the four boards have been met, where all four boards produced worst setup slack duration of at least 44.6% of `global_clk` period.

4.1.2 On-Chip Power. Basys 3 showed the lowest power consumption of 74 mW, while the highest one was 110 mW from PYNQ-Z1.

4.1.3 Routing Delay. It represents the delay after the design has been synthesized, implemented and routed in a device hardware. All four devices showed this delay between 11.14 and 13.47 ns.

4.2 Frequency Analysis

This analysis has been done to measure the maximum frequency at which the board can run until the slack timing fails. For each of the four boards, MRCC I/O pins have been used for `global_clk`, `sc_in`, and `sc_out`. Other ports used similar switches and pins.



(a) Slack vs Frequency. **(b) Routing Delay (as a fraction of clock period) vs Frequency.**

■: Basys 3 ■: PYNQ-Z1 ■: Nexys A7 ■: Arty A7

Figure 4: Frequency analysis on different FPGA boards using external clocks.

4.2.1 Maximum Frequency. Each board has successfully implemented the design up to 273 MHz until the setup slack became negative. The slack-vs-frequency graph is depicted in Figure 4a.

4.2.2 Routing Delay. For each board except Arty A7, the routing delay has remained almost the same. This translates to the delay as a fraction of `global_clk` period increasing linearly as the frequency increases. This delay is plotted against frequency in Figure 4b.

5 VLSI IMPLEMENTATION

The design has been synthesized using Synopsys Design Compiler, and the gate-level netlist has been assessed exhaustively using a testbench. The placement and routing has been done using Cadence Innovus for the 65-nm process run at SUNY, with a minimum target of 85% core density. The place-and-route netlist has manifested the same behavior against the testbench used for gate-level netlist.

Analysis has been done to determine the maximum frequency at which the slack becomes negative. Both `global_clk` and `sc_clk` have been swept from 100 MHz to 540 MHz with 10 MHz interval.

5.0.1 Worst Negative Slack. As the frequency was increased, the setup slack gradually decreased, but the hold slack had no significant change. The setup mode failed at 540 MHz.

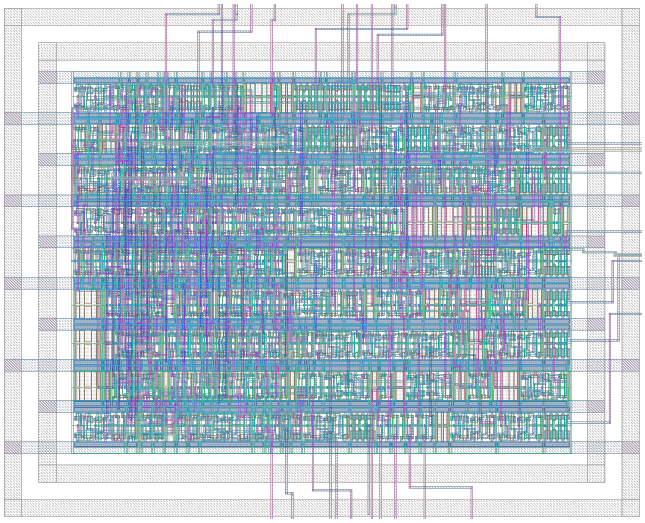


Figure 5: Physical layout of the encoder module (at frequency of 500 MHz) implemented in a 65-nm VLSI process occupying only 2134.67 μm^2.

5.0.2 Power. The total power of the design increased linearly with the frequency, consuming 0.54 mW at 100 MHz up to 2.98 mW at 540 MHz. The switching power contributed to 93.18(±0.24)% of the total power.

5.0.3 Wire Length. The process had five metal layers, and the distribution of wires did not exhibit any direct relation with the frequency. The worst hold slack changed with the change in wire distribution, as expected.

5.0.4 Core Area, Density and Capacitance. The core area increased by 5.44% after 370 MHz, but always stayed between 2024.61 and 2154.09 μm^2. The core density decreased slightly after 190 MHz, but remained at 87.35(±1.42)% overall. The total capacitance held at 35.15(±0.08) pF, also showing no correlation with the change in frequency.

6 CONCLUSIONS & FUTURE WORK

In this work we have proposed the first hardware-based spike encoder with support for multiple spike encoding schemes while also being reconfigurable at runtime. This design supports an encoding frame of up to 64 time-steps, a 6-bit runtime configurable value to be encoded, and a 4-bit runtime configurable address for one of 16 destination neurons. It can be comfortably incorporated into a neuroprocessor to alleviate the overhead that would normally come from performing spike encoding on an external hardware accelerator or software-enabled microcontroller.

Future improvements planned for this encoder include expanding the maximum size of the encoding frame and the resolution of values that can be encoded. More encoding schemes are also planned to be implemented beyond the three shown in this work. Additionally, the encoder presented here has a silicon layout that was fully generated with placement and routing algorithms, so additional performance enhancements could be gained by creating a custom layout which minimizes both the necessary silicon area and propagation delay. Finally, modifying the implementation of the the look-up table from combinational logic to NAND-based ROM could limit latency to less than one `global_clk` cycle.

ACKNOWLEDGMENTS

This material is based in part on research sponsored by Air Force Research Laboratory under agreement FA8750-21-1-1018. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory or the U.S. Government.

REFERENCES

[1] Alireza Bagheri, Osvaldo Simeone, et al. 2018. Training Probabilistic Spiking Neural Networks with First- To-Spike Decoding. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2986–2990. https://doi.org/10.1109/ICASSP.2018.8462410

[2] Sander M. Bohte, Joost N. Kok, et al. 2002. Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing* 48, 1 (2002), 17–37. https://doi.org/10.1016/S0925-2312(01)00658-0

[3] E. Culurciello, R. Etienne-Cummings, et al. 2003. A biomorphic digital image sensor. *IEEE Journal of Solid-State Circuits* 38, 2 (2003), 281–294. https://doi.org/10.1109/JSSC.2002.807412

[4] Mike Davies, Narayan Srinivasa, et al. 2018. Loihi: A Neuromorphic Manycore Processor with On-Chip Learning. *IEEE Micro* 38, 1 (2018), 82–99. https://doi.org/10.1109/MM.2018.112130359

[5] Intel. 2021. *Taking Neuromorphic Computing to the Next Level with Loihi 2.* Retrieved April 5, 2023from https://download.intel.com/newsroom/2021/new-technologies/neuromorphic-computing-loihi-2-brief.pdf

[6] Zhengzhong Liang, David Schwartz, et al. 2018. The impact of encoding–decoding schemes and weight normalization in spiking neural networks. *Neural Networks* 108 (2018), 365–378. https://doi.org/10.1016/j.neunet.2018.08.024

[7] Chit-Kwan Lin, Andreas Wild, et al. 2018. Programming Spiking Neural Networks on Intel's Loihi. *Computer* 51, 3 (2018), 52–61. https://doi.org/10.1109/MC.2018.157113521

[8] Chenchen Liu, Bonan Yan, et al. 2015. A Spiking Neuromorphic Design with Resistive Crossbar. In *Proceedings of the 52nd Annual Design Automation Conference* (San Francisco, California) *(DAC '15)*. Association for Computing Machinery, New York, NY, USA, Article 14, 6 pages. https://doi.org/10.1145/2744769.2744783

[9] Hesham Mostafa. 2018. Supervised Learning Based on Temporal Coding in Spiking Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems* 29, 7 (2018), 3227–3235. https://doi.org/10.1109/TNNLS.2017.2726060

[10] Christoph Posch, Teresa Serrano-Gotarredona, et al. 2014. Retinomorphic Event-Based Vision Sensors: Bioinspired Cameras With Spiking Output. *Proc. IEEE* 102, 10 (Oct. 2014), 1470–1484. https://doi.org/10.1109/jproc.2014.2346153

[11] Xin Qi, Xiaochuan Guo, et al. 2004. A time-to-first spike CMOS imager. In *2004 IEEE International Symposium on Circuits and Systems (ISCAS)*, Vol. 4. IV–824. https://doi.org/10.1109/ISCAS.2004.1329131

[12] Damien Querlioz, Olivier Bichler, et al. 2011. Simulation of a memristor-based spiking neural network immune to device variations. In *The 2011 International Joint Conference on Neural Networks*. 1775–1781. https://doi.org/10.1109/IJCNN.2011.6033439

[13] Garrett S Rose, Mst Shamim Ara Shawkat, et al. 2021. A system design perspective on neuromorphic computer processors. *Neuromorphic Computing and Engineering* 1, 2 (Nov. 2021), 022001. https://doi.org/10.1088/2634-4386/ac24f5

[14] Catherine Schuman, Charles Rizzo, et al. 2022. Evaluating Encoding and Decoding Approaches for Spiking Neuromorphic Systems. In *Proceedings of the International Conference on Neuromorphic Systems 2022*. ACM. https://doi.org/10.1145/3546790.3546792

[15] Catherine D. Schuman, James S. Plank, et al. 2019. Non-Traditional Input Encoding Schemes for Spiking Neuromorphic Systems. In *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE. https://doi.org/10.1109/ijcnn.2019.8852139

[16] Chen Shoushun and Amine Bermak. 2007. Arbitrated Time-to-First Spike CMOS Image Sensor With On-Chip Histogram Equalization. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 15, 3 (2007), 346–357. https://doi.org/10.1109/TVLSI.2007.893624

[17] Yang Yi, Yongbo Liao, et al. 2016. FPGA based spike-time dependent encoder and reservoir design in neuromorphic computing processors. *Microprocessors and Microsystems* 46 (2016), 175–183. https://doi.org/10.1016/j.micpro.2016.03.009