SpikeGAN: An Energy-Efficient Spiking Generative Adversarial Network Design

Yuga Hanyu, Atharv Sharma and Aruki Komatsuzaki
School of Computer Science and Engineering, University of Aizu, Fukushima, Japan
Email: {m5282026, m5292015, s1310244}@u-aizu.ac.jp

***Abstract*—The use of generative AI in our society has been increasing rapidly. At the same time, generative tasks, such as image generation, are computationally expensive, leading to major carbon emissions due to powering the machines. However, there is a possibility to reduce carbon emissions from generative AI by adapting Spiking Neural Networks (SNNs), thanks to their event-driven nature and energy efficiency. In this study, we developed a framework that converts a Generative Adversarial Neural (GAN) Network consisting of Artificial Neural Networks (ANNs) into an SNN, which can be deployed on an FPGA and implemented in ASIC 45nm CMOS technology. The output images from software and RTL simulations show great correspondence (3.02e-05 to 1.00e-04 MSE values).**

***Index Terms*—FPGA, Image Generation, Spiking Neural Networks, Artificial Neural Networks, Generative Adversarial Networks**

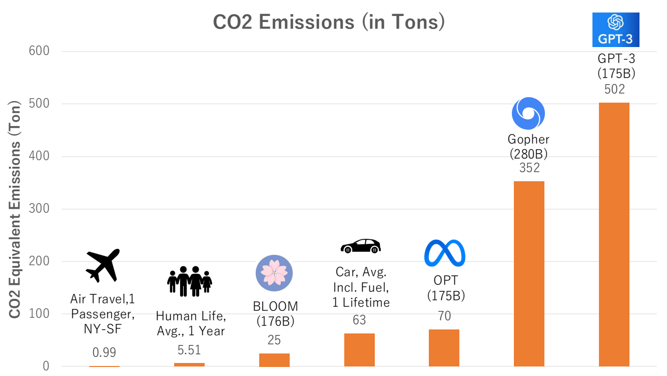


Fig. 1. CO2 emissions comparison between AI systems and others (Source: Luccioni et al., 2022; Strubell et al., 2019 — Chart: 2023 AI Index Report)

I. Introduction

In recent years, generative AIs such as ChatGPT have been incorporated into our society. One of the useful tasks they can perform is image or video generation, in which the Generative Adversarial Networks (GANs) [1] have shown to be versatile and powerful. While generative AI has made notable improvements and is becoming increasingly popular among general consumers, the computation of AI requires a massive amount of electricity, causing associated carbon emissions. The estimation of total carbon emissions from AI accounts for 2.1% to 3.9% of the global carbon emissions [2].

On the other hand, unlike conventional Artificial Neural Networks (ANNs), Spiking Neural Networks (SNNs) offer a low-energy solution to AI systems due to the sparsity of spikes and multiplication-less operation. Additionally, SNNs are noise-resilient and suitable for chip deployment, as various low-power techniques can be applied with minimal performance loss, such as quantization [3], pruning [4], approximation [5], and compression [6]. In addition to the low-power techniques applied to the model itself, hardware-level energy optimizations such as power gating [7], clock gating [8], [9], and dynamic voltage frequency scaling [10].

Training of SNNs can be performed in different ways, including (1) training an equivalent ANN and converting it to an SNN [11], (2) training directly on spikes using backpropagation [12], and (3) biologically plausible learning methods such as Spike Timing Dependent Plasticity (STDP) [13].

Adapting SNNs to generative AI systems can solve the problems of the huge carbon emissions, especially for computationally intensive image generation. However, there are several challenges that need to be addressed:

- *Learning Complexity*: For SNNs, a complex learning task, such as generating contents is still a challenge and may limit the learning efficiency.
- *Huge Model Size*: For an ANN for content generation, the output quality depends on scaling and often has a huge number of parameters. Incorporating a large SNN often results in limited energy efficiency.
- *Output Equivalency*: To tackle the learning complexity with SNNs, training an ANN and converting to the equivalent SNN is an option. However, the converted SNN can not perfectly match the original ANN, prompting the need for managing the difference.

To address these challenges, we propose a framework that (1) trains a reasonably sized GAN model with consideration for the later conversion and (2) converts the generator ANN into the equivalent SNN.

The structure of the report is as follows. Section II explains the background theories, section III shows the details and results of the hardware implementation of our proposed design. Finally, section IV concludes the paper.

II. Design

In this section, the theory of GAN training, Spiking Neural Networks, Conversion from ANN to SNN, and the exportation of the SNN generator are detailed.

A. GAN Training

The principle of GANs is the adversarial training of two neural networks: (1) a generator that tries to generate fake data, and (2) a detector that tries to detect the fake data from the generator (decipher fake from the real data). During training, the objectives for both networks are summarized as:

$$\min_G \max_D V(D,G) = \frac{1}{m}\sum_{n=1}^{m}\Big[\log D(X_n) + \log(1 - D(G(Z_n)))\Big] \tag{1}$$

Where D and G are the discriminator and generator networks, m is the total number of input noise (Z) to the generator and real data (X). Here, the generator tries to minimize the $V(D, G)$ (bypass the discriminator), while the discriminator tries to maximize (return 1 for the real data, 0 otherwise).

For each training step, parameters for the discriminator are updated by ascending its stochastic gradient.

$$\nabla_{\theta_D}\frac{1}{m}\sum_{n=1}^{m}\Big[\log D(X_n) + \log(1 - D(G(Z_n)))\Big] \tag{2}$$

∇_{θ_d} is the set of parameters for the discriminator. Here, the loss function for the discriminator is unchanged from Eq. 1. For the generator, the parameters are updated by descending its stochastic gradient:

$$\nabla_{\theta_G}\frac{1}{m}\sum_{n=1}^{m}\log(1 - D(G(Z_n))) \tag{3}$$

∇_{θ_G} is the set of parameters for the generator. It only involves the second term for Eq. 1.

In principle, the training of GAN can be applied to any gradient-based learning.

B. Spiking Neural Networks

SNNs process information through a series of discrete spikes (spiketrain), unlike other types of Neural Networks that use continuous signals. Fig. 2 shows an example architecture of a 3 feedforward layer SNN. First, the scalar values in the input data such as images, are converted to the corresponding spiketrains as:

$$S_{ij} = \begin{cases} 1 & \text{if } x_i > RNG_{ij} \times \alpha \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

S_{ij} denotes the jth spike of the ith spiketrain, x_i is the ith scalar input, RNG is a random number, and α is the scaling factor. Here, the ratio of spike (1) in the ith spiketrain corresponds to the normalized value of the ith input.

The excitatory layer and output layer consist of Leaky Integrate-and-Fire (LIF) neurons. Membrane potential of a LIF neuron is updated as:

$$\tau_m \frac{dV_m(t)}{dt} = -(V_m(t) - V_{rest} + R_m I(t)) \tag{5}$$

Here, the parameters are:

- $V_m(t)$: Membrane potential at time t
- V_{rest}: Resting membrane potential

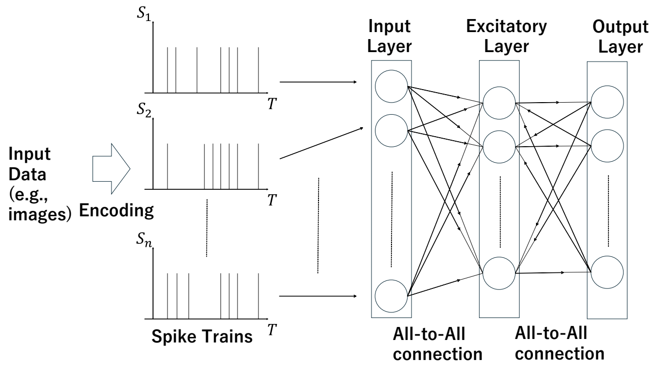


Fig. 2. Example architecture of a 3 feedforward layer SNN

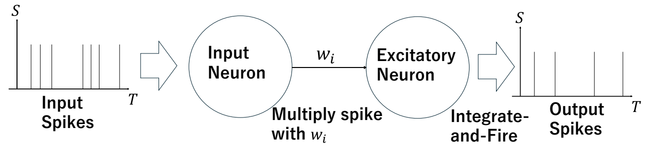


Fig. 3. Spike processing of a Leaky-Integrate-and-Fire neuron

- R_m: Membrane resistance
- $I(t)$: Input synaptic current
- V_{th}: Firing threshold
- V_{reset}: Reset potential

Initially, membrane potential is reset to V_{reset}. Then, V_m increases according to $R_m I(t)$. Once V_m goes over the threshold V_{th}, the neuron outputs a spike and V_m is set to V_{rest}. Fig. 3 shows the spike processing of a LIF Neuron. The input spike is multiplied with the weight w assigned to the neuron, then the value is accumulated in the membrane potential of the neuron. The neuron operates according to Eq. 5.

C. Conversion from ANN to SNN

After training a GAN model that consists of ANNs, the generator network is converted to an equivalent SNN. The principle of conversion is to mimic the behavior of the ReLU function in the ANN with the LIF neuron. The conversion starts with weight normalization:

$$\lambda_l = max\{a_i^l\} \tag{6}$$

$$w' = w\lambda_{l-1}/\lambda_l \tag{7}$$

λ_l is the maximum activation value of layer l. Each weight in w is normalized based on the maximum activation of the current and the previous layer. For the network architecture, the encoding mechanism and IF (or LIF) layer are added. Fig. 4 shows the conceptual conversion. After the conversion, spikes are fed to the network instead of a scalar value, as well as an IF layer replacing the ReLU function.

D. Hardware Architecture

Fig. 5 shows the architecture of the SNN generator on hardware, and the surrounding modules, such as memory for the weights. Input spikes are loaded through the Spike Rom

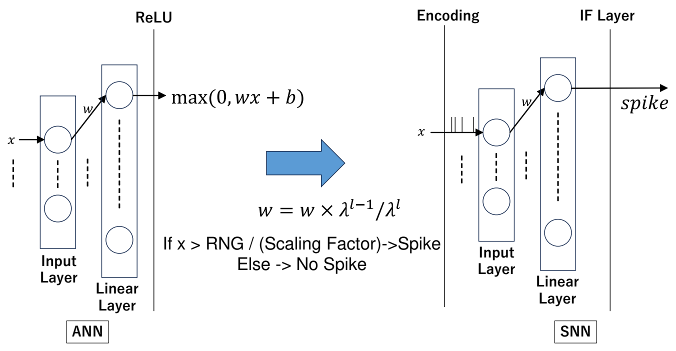


Fig. 4. Conversion from ANN to SNN

module, multiplied by the stored weights, and the output spikes come from each layer. In each layer, the LIF neurons access the stored weights through XBAR.

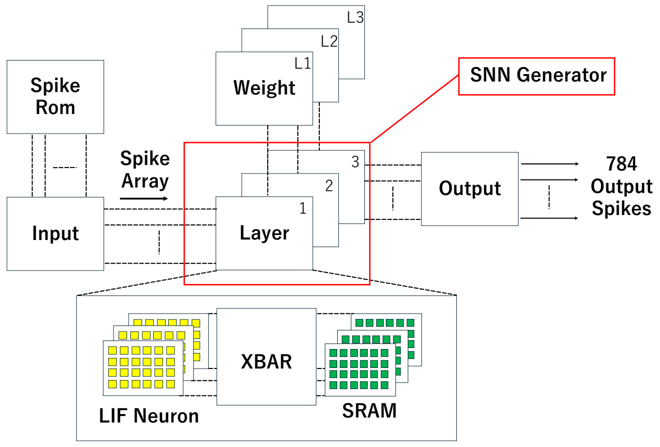


Fig. 5. Hardware architecture of the GAN model

III. Implementation Result

In this section, first, we export the SNN generator to HDL and compare the outputs between software and HDL simulations. Then the SNN generator is exported to both the FPGA and ASIC.

A. Export SNN From Software to HDL

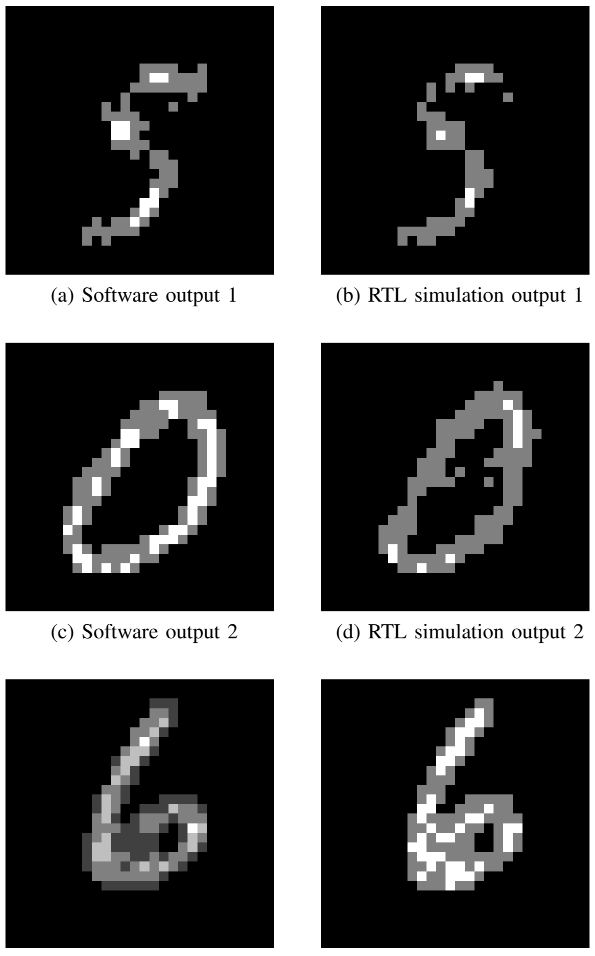
Once the converted SNN generator is obtained and runs successfully on the software, the model parameters, including the trained weights for each layer, are exported for RTL simulation. The model architecture of the original ANN generator is summarized in Table. I. The model architecture remains consistent for the ANN and SNN.

Fig. 6 compares the output images obtained from software and RTL simulation. The Mean Squared Error (MSE) values for (a) and (b), (c) and (d), and (e) and (f) are 3.0196e-05, 1.00e-04, and 8.95e-05, respectively.

Fig. 7 compares the spike timings of some neurons in software and RTL simulation. The pink wave in the waveform is the system start signal. The spike pattern in software and

TABLE I
Model Architecture and Training Parameters

Parameter	Value
Training Data	MNIST Handwritten Digits
Training Numbers	600 Images per Batch $\times$ 15 Epochs
Input Size	100
Layer 1	1200 Neurons
Layer 2	1200 Neurons
Layer 3	784 Neurons



(a) Software output 1 (b) RTL simulation output 1
(c) Software output 2 (d) RTL simulation output 2
(e) Software output 3 (f) RTL simulation output 3

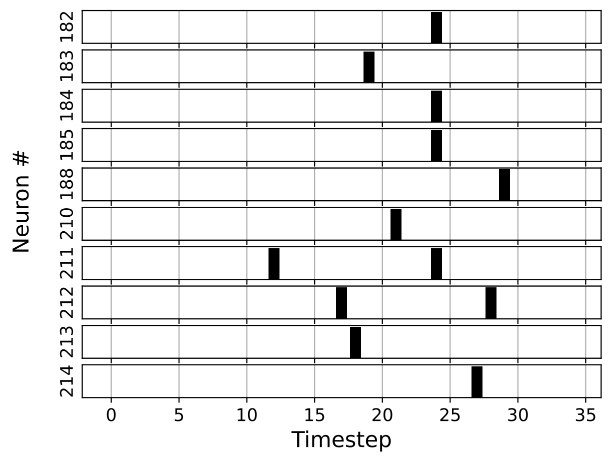
Fig. 6. Processing results for test images

RTL simulations are observed to be similar as illustrated in Fig. 7. It can be seen that the total count of output spikes for software (a) and RTL (b) matches, except for neuron #188.

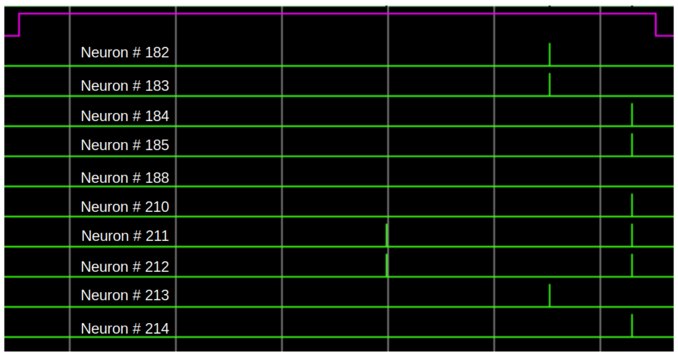
B. FPGA Implementation

The FPGA implementation was conducted using Vivado 2019.1 and Artix-100T FPGA evaluation board was used. Due to the memory constraint, the number of neurons in Layers 1 and 2 were changed from 1200 to 64 neurons.

Fig. III shows the resource utilization. The proposed design occupies 2,952 flip-flops (23.28% utilization) and 46,920 LUTs (74.01% utilization), on the targeted FPGA Chip.



(a) Timing snapshot of software



(b) Timing snapshot of RTL simulation with ModelSim

Fig. 7. Processing results for test images in both software and RTL simulation.

C. ASIC Implemenation

In order to calculate the area consumption of GAN model Design Compiler Shell is used. Design Compiler Shell is command-line interface of Synopsys Design Compiler used to perform logic synthesis, where RTL descriptions are translated into optimized gate-level netlists based on timing, area, and power constraints.

A clock period of 2 ns is specified, and synthesis is performed using the CMOS PDK45 technology node.

Table. II shows the area consumed by combinational logic, sequential logic, and memory, resulting to a total of around 222,972 μm^2 by combinational and sequential logic. For the total number of bits required by memory:

$$\text{Layer } 1 = 100 \times 64 = 6{,}400 \text{ bits} \tag{8}$$

$$\text{Layer } 2 = 64 \times 64 = 4{,}096 \text{ bits} \tag{9}$$

$$\text{Layer } 3 = 64 \times 784 = 50{,}176 \text{ bits} \tag{10}$$

Combining the bits of all these 3 layers, we get 485,376 bits in total for memory.

IV. Conclusion

In this report, an ANN-based GAN model was trained, and the generator network was converted to an SNN model. The SNN model was simulated both on software and HDL (ASIC + FPGA), and the correspondence in output images and spike patterns was observed. Synthesis was also successfully performed for ASIC and FPGA both. For future work, we are planning to scale-up the model, and perform prototyping on FPGA.

TABLE II
Area summary of ASIC synthesis

Category	**Area**
Combinational logic	151,198.92 μm^2
Sequential logic	71,773.18 μm^2
Memories	485,376 bits
Total	222,972.10 μm^2

TABLE III
Resources Utilization of FPGA implementation

Resource	**Utilization**	**Available**	**Utilization** (%)
LUT	46920	63400	74.01
LUTRAM	7296	19000	38.40
FF	29523	126800	23.28

References

[1] V. L. T. De Souza, B. A. D. Marques, H. C. Batagelo, and J. P. Gois, "A review on generative adversarial networks for image generation," *Computers & Graphics*, vol. 114, pp. 13–25, 2023.

[2] K. Kirkpatrick, "The carbon footprint of artificial intelligence," *Commun. ACM*, vol. 66, no. 8, p. 17–19, Jul. 2023. [Online]. Available: https://doi.org/10.1145/3603746

[3] M. Nagel *et al.*, "A white paper on neural network quantization," *arXiv preprint arXiv:2106.08295*, 2021.

[4] D. Blalock *et al.*, "What is the state of neural network pruning?" *Proceedings of machine learning and systems*, vol. 2, pp. 129–146, 2020.

[5] R. Kobayashi *et al.*, "An efficient hardware implementation of spiking neural network using approximate Izhikevich neuron," in *2024 9th International Conference on Integrated Circuits, Design, and Verification (ICDV)*. IEEE, 2024, pp. 13–18.

[6] S. Han *et al.*, "Deep compression: Compressing deep neural networks with pruning, trained quantization, and huffman coding," *International Conference on Learning Representations (ICLR)*, 2015, arXiv:1510.00149.

[7] N.-D. Nguyen *et al.*, "Power-Aware Neuromorphic Architecture With Partial Voltage Scaling 3-D Stacking Synaptic Memory," vol. 31, no. 12, pp. 2016–2029, 2023.

[8] Q. Wu *et al.*, "Clock-Gating and Its Application to Low Power Design of Sequential Circuits," vol. 47, no. 3, pp. 415–420, 2000.

[9] N.-D. Nguyen *et al.*, "An in-situ dynamic quantization with 3D stacking synaptic memory for power-aware neuromorphic architecture," *IEEE Access*, vol. 11, pp. 82 377–82 389, 2023.

[10] T. Kolpe *et al.*, "Enabling Improved Power Management in Multicore Processors through Clustered DVFS," in *2011 Design, Automation & Test in Europe*. IEEE, 2011, pp. 1–6.

[11] B. Rueckauer *et al.*, "Conversion of continuous-valued deep networks to efficient event-driven networks for image classification," *Frontiers in neuroscience*, vol. 11, p. 682, 2017.

[12] M. Dampfhoffer *et al.*, "Backpropagation-based learning techniques for deep spiking neural networks: A survey," *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[13] M. R. Azghadi *et al.*, "Spike-based synaptic plasticity in silicon: design, implementation, application, and challenges," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 717–737, 2014.