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: {m5291018, m5292015, s1310244}@u-aizu.ac.jp

I. Overview

Figure 1 shows the overall design flow. A compact ANN-based GAN model is trained on the MNIST dataset using 100-dimensional noise input and a three-layer generator consisting of 1200, 1200, and 784 neurons. Training was performed with 600 images per batch for 15 epochs. After training, only the generator network is converted to a Spiking Neural Network (SNN) to enable spike-based hardware execution.

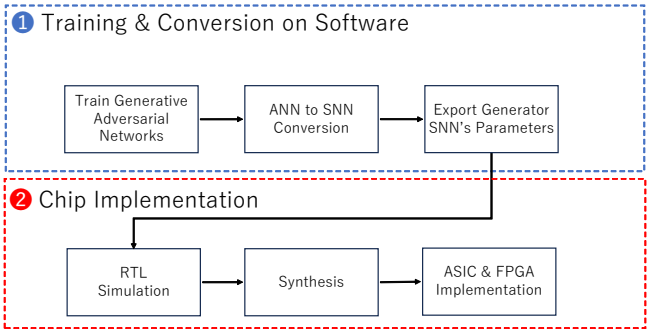


Fig. 1. Overall design flow of GAN training and chip implementation

Figure 2 shows the GAN training and conversion from ANN generator and SNN generator. The ANN-to-SNN conversion preserves the original network structure while replacing ReLU activations with Integrate-and-Fire (IF) neurons. Weight normalization is applied using maximum activation per layer to maintain functional equivalence. Scalar inputs are encoded into spike trains, and neuron firing rates approximate ANN activations over multiple timesteps. This approach allows the SNN to reproduce the behavior of the trained ANN while enabling hardware-friendly spike processing.

II. Hardware Architecture

Figure 3 shows the architecture of chip implementation, and Figure 4 shows the computing core that stores the learned weights for each layer. Here, we did not utilize the STDP part, a learning mechanism for SNNs, since the learning was conducted by the source ANN model. The design consists of data readings for spike and weight for each layer, crossbar-based weight memory, arrays of IF neurons, and global control logic. Input spikes are multiplied by stored weights and accumulated into membrane potentials. When the membrane potential exceeds the threshold, an output spike is generated, and the potential resets. Layer-by-layer spike propagation

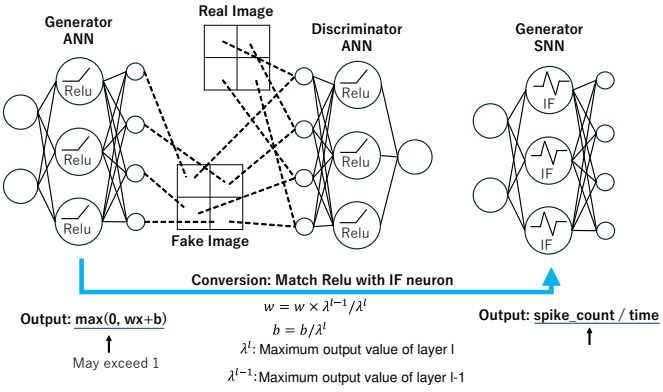


Fig. 2. GAN training and conversion from ANN to SNN

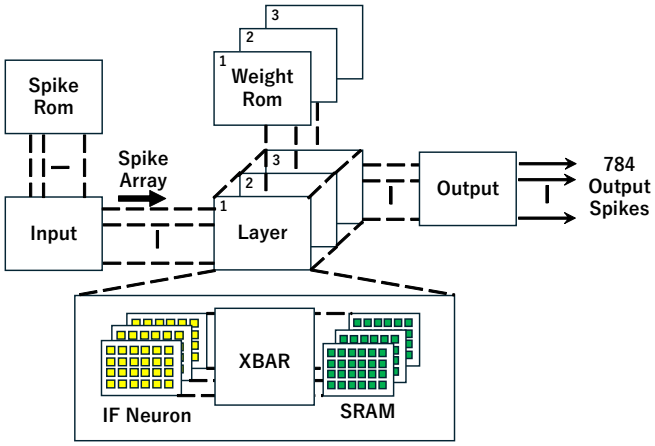


Fig. 3. Architecture of chip implementation

produces the final output image. The same RTL design is used for both FPGA implementation and ASIC synthesis.

III. Implementation Results

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. 5 compares the output images obtained from software and RTL simulation. The Mean Squared Error (MSE) value for (a) and (b) is 3.02e-05.

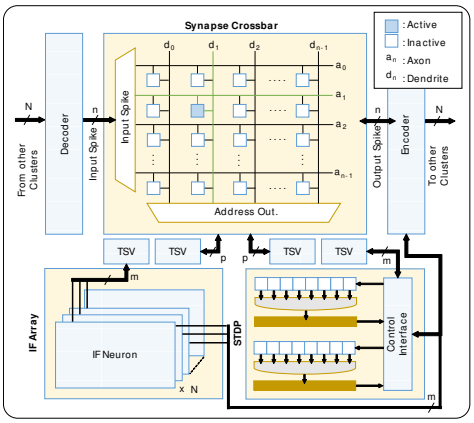
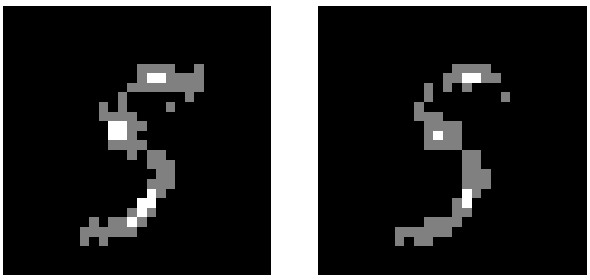


Fig. 4. Computing Core

TABLE I
MODEL ARCHITECTURE AND TRAINING PARAMETERS

Parameter	Value
Training Data	MNIST Handwritten Digits
Training Numbers	600 Images per Batch × 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

Fig. 5. Output image comparison between software and RTL simulation

A. 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. II 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.

B. 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
RESOURCES UTILIZATION OF FPGA IMPLEMENTATION

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

TABLE III
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 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:

$$\text{Layer } 1 = 100 \times 64 = 6{,}400 \text{ bits} \quad (1)$$

$$\text{Layer } 2 = 64 \times 64 = 4{,}096 \text{ bits} \quad (2)$$

$$\text{Layer } 3 = 64 \times 784 = 50{,}176 \text{ bits} \quad (3)$$

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

IV. CONCLUSION

This work presents SpikeGAN, a hardware-oriented framework that converts an ANN-based GAN generator into an SNN suitable for spike-based implementation. The proposed approach maintains output equivalence between software and RTL, achieves successful FPGA deployment, and demonstrates ASIC synthesis feasibility in 45 nm technology. The results indicate the potential of SNN-based generative models for energy-efficient AI hardware systems.