PARALLEL PIPELINED SPIKE ENCODING ALGORITHMS FOR SNN ON FIELD-PROGRAMMABLE GATE ARRAY (FPGA)

Xueyan ZHONG[1,*], Hongbing PAN[2]

Different from the traditional software-based spike coding mode, this paper proposes a parallel-pipelined approach to encode the MNIST dataset into spikes on a Field-Programmable Gate Array (FPGA). MNIST images undergo edge extension, normalization, integer data processing, parallel-pipeline convolution, and pixel value trimming. The resulting new pixel values are interpolated and converted into cell membrane voltage values. Leveraging the inverse relationship between membrane voltage and firing period, these voltage values are transformed into period values. Finally, within a fixed duration, spike sequences are generated by comparing the duration step counter with the period values. The period values obtained via this method are highly similar to the pixel values of the original image, well preserving the features of the digital image and ensuring high accuracy. Meanwhile, the FPGA-based parallel-pipeline operation mode improves computational speed, reduces overall resource consumption, and enables good hardware transplantability.

Keywords: Spiking Neural Network; Spike Encoding; Hardware Implementation; FPGA; Parallel-Pipeline

1. Introduction

Unlike traditional Artificial Neural Networks (ANNs), Spiking Neural Networks (SNNs) are designed to mimic the biological brain, exhibiting higher biological plausibility and being widely regarded as the "third generation of neural networks" [1]. However, modeling the biological brain involves multiple professional fields and is highly complex. Compared with mature traditional artificial neural networks, SNNs are still in the early stages of development. How to construct SNN models reasonably and effectively remains one of the key challenges in current research [2].

The SNN model adopted in this paper has a three-layer structure, as shown in Fig.1. It uses the Leaky Integrate-and-Fire (LIF) neuron model as network nodes, with intermediate synaptic information transmitted in the form of spikes. The overall structure is divided into three parts: the spike encoding layer, the neural network layer, and the classification layer. The spike encoding layer encodes each

[1] College of Intelligent Engineering, Nanjing Vocational Institute of Railway Technology, Nanjing, and School of Electronic Science and Engineering, Nanjing University, China, e-mail: zhongxueyan1987@163.com
[2] School of Electronic Science and Engineering, Nanjing University, China

pixel of an image into spikes; after encoding, the spikes are input to the neural network layer for training; finally, the classification layer completes the task of image category discrimination [3-4].

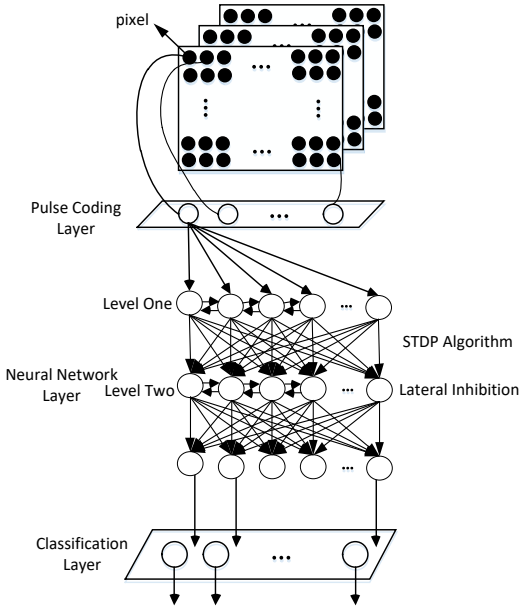


Fig.1. Structure of Neural Network

The input spike encoding method is a core component of the SNN model. It not only determines the conversion of input information but also needs to preserve the key features of input samples. Additionally, the spike encoding method directly affects subsequent learning algorithms and network architecture design [5]. Through research on biological neural systems, the academic community has summarized several commonly used spike coding methods, including frequency coding, temporal coding, phase coding, and population coding [6-8].

In frequency coding, each input neuron corresponds to a pixel value (or feature value), and the higher the pixel value, the higher the spike firing frequency. In temporal coding, the higher the pixel value, the earlier the input neuron emits spikes. In phase coding, the timing of spike emission is determined by the phase of a reference signal. In population coding, a single pixel value is encoded by the collective spike activity of a group of neurons, with the combination of spikes from multiple neurons representing a specific feature [9-10]. Among these methods, temporal coding is prone to noise interference, which distorts the information expressed in spike sequences and is not conducive to hardware implementation. Phase coding requires longer time steps to accommodate multiple distinct phases, and the middle convolutional layer neurons need more steps to improve the anti-interference ability of phase signals, which increases runtime. Population coding

for a single node requires multiple neurons, making it difficult to connect with the next layer of nodes in SNNs. Therefore, this paper selects frequency coding, which is easy to implement in hardware, to generate spike sequences [11-13].

The main research object of this paper is the MNIST handwritten digit dataset. In the spike encoding layer, images in the dataset are first preprocessed by convolution, and the pixel values obtained after convolution are trimmed. Then, voltage interpolation is used to convert the trimmed pixel values into cell membrane voltage values. Based on the inverse proportional relationship between voltage values and period values, the corresponding firing period is calculated. Finally, frequency coding is performed to generate spike sequences.

2. State-of-the-art Review

Spike encoding, as a critical link in SNN applications, has been a focus of research in the neuromorphic computing field in recent years. Current spike encoding methods are mainly divided into two categories: rate-based encoding and temporal-based encoding, each with distinct characteristics in performance, hardware adaptability, and application scenarios.

2.1. Rate-based Encoding Methods

Rate-based encoding methods map input feature intensity (e.g., image pixel values) to the firing rate of neurons. Representative algorithms include the Sliding Window (SW) algorithm [14] and Ben's Spiker Algorithm (BSA) [15].

The SW algorithm divides the input time window into multiple sub-windows and adjusts the spike firing frequency based on the average feature value within each sub-window. It has the advantage of simple logic but suffers from low encoding precision—since the average value in the sub-window smooths out local feature details, leading to the loss of fine-grained information.

The BSA algorithm optimizes the spike generation mechanism by setting adaptive thresholds, improving the response speed to feature changes. However, it requires additional threshold adjustment modules, which increases hardware resource consumption. When implemented on FPGA, both algorithms have relatively low classification accuracy (usually below 97.5%), and their computational speed is limited by serial data processing, making it difficult to meet the real-time requirements of high-throughput tasks [16].

2.2. Temporal-based Encoding Methods

Temporal-based encoding methods encode information through the timing of spike firing or the width of spike pulses. Typical algorithms include Step-forward (SF) encoding [17] and Pulse Width Modulated-Based (PWMB) encoding [18].

SF encoding determines the spike firing time based on the order of feature value magnitudes, which can effectively preserve temporal feature information.

However, it is highly sensitive to noise—even small noise interference can cause significant deviations in spike timing, reducing system robustness.

PWMB encoding modulates the width of spike pulses according to feature intensity. It has high encoding precision (achieving a classification accuracy of up to 98% on FPGA) but requires precise control of pulse width, resulting in complex hardware circuits and high dynamic power consumption (up to 262.43 mW). Additionally, the long pulse duration increases computational time, making it unsuitable for low-latency application scenarios [19].

2.3. Challenges in Hardware Implementation

Existing spike encoding algorithms face two key challenges in hardware implementation:

Trade-off between encoding accuracy and resource consumption: High-precision encoding methods (e.g., PWMB) often require complex arithmetic units and storage modules, leading to increased usage of logic elements and power consumption. Mismatch with FPGA's parallel computing advantages: Most traditional algorithms are designed based on software serial logic, failing to fully utilize FPGA's pipeline and parallel computing capabilities, resulting in low computational efficiency [20].

To address these issues, this paper proposes a parallel-pipelined spike encoding method. By optimizing the data preprocessing process (including edge extension, normalization, and integerization) and designing a parallel-pipelined convolution structure, efficient parallel processing of image data is achieved on FPGA. Meanwhile, through the inverse proportional mapping between membrane voltage and firing period, high encoding accuracy is ensured while simplifying hardware circuits, reducing resource consumption and power usage.

3. Theoretical Approach

3.1. Core Principle of Parallel-Pipelined Convolution

The MNIST dataset consists of 28 * 28 pixel values. In order to extract features from the image, it is convolved with a 5 * 5 convolution kernel. After convolution, the pixel values are reduced to 24 * 24. To ensure 28 * 28 pixel values, fill the edge of the original 28 * 28 image data with a circle of 0 and expand it to 32 * 32. Then, normalize the pixels and perform convolution operations. The result of maximizing the image features is to make the image blurry. Therefore, it is necessary to trim the image.

The convolution operation selects a 5 * 5 convolution kernel, and the convolution window starts from the top left of the input array, sliding in one step from left to right and from top to bottom on the input array. Added elements with a value of 0 on both sides of the original 28 * 28 image pixels, making the input image pixels 32 * 32, ensuring that the output image pixels after convolution calculation

are still 28 * 28 image pixels. The purpose of convolution is to strengthen the connection between local closer pixels and weaken the correlation between distant pixels. Therefore, the 5 * 5 symmetric convolution kernel selected for convolution operation is formula (1).

$$kernel = \begin{pmatrix} c_{0,0} & c_{0,1} & c_{0,2} & c_{0,3} & c_{0,4} \\ c_{1,0} & c_{1,1} & c_{1,2} & c_{1,3} & c_{1,4} \\ c_{2,0} & c_{2,1} & c_{2,2} & c_{2,3} & c_{2,4} \\ c_{3,0} & c_{3,1} & c_{3,2} & c_{3,3} & c_{3,4} \\ c_{4,0} & c_{4,1} & c_{4,2} & c_{4,3} & c_{4,4} \end{pmatrix} = \begin{pmatrix} c_4 & c_3 & c_2 & c_3 & c_4 \\ c_3 & c_2 & c_1 & c_2 & c_3 \\ c_2 & c_1 & c_0 & c_1 & c_2 \\ c_3 & c_2 & c_1 & c_2 & c_3 \\ c_4 & c_3 & c_2 & c_3 & c_4 \end{pmatrix} \quad (1)$$

The middle value of the convolutional kernel is a larger positive value, and the farther away from the middle, the smaller the positive value. Based on the distance, the nearby data connection is strengthened, while the edge value is negative. The farther away from the middle, the smaller the negative value, weakening the correlation of the farther data. At the same time, for the convenience of FPGA binary calculation, the values are selected as exponential multiples of 2.

$$\begin{pmatrix} -\frac{1}{2} & -\frac{1}{4} & \frac{1}{4} & -\frac{1}{4} & -\frac{1}{2} \\ -\frac{1}{4} & \frac{1}{4} & \frac{5}{8} & \frac{1}{4} & -\frac{1}{4} \\ \frac{1}{4} & \frac{5}{8} & 1 & \frac{5}{8} & \frac{1}{4} \\ -\frac{1}{4} & \frac{1}{4} & \frac{5}{8} & \frac{1}{4} & -\frac{1}{4} \\ -\frac{1}{2} & -\frac{1}{4} & \frac{1}{4} & -\frac{1}{4} & -\frac{1}{2} \end{pmatrix} \quad (2)$$

The pixel values obtained from convolution operations can easily exceed the range of the original pixel values of the image. To solve this problem, we convert the convolution kernel into decimals, which means dividing each value in the convolution kernel by the sum of the convolution kernel. The sum of the convolution kernel is 5.5, which is calculated by formula (3).

$$kernel\_sum = \sum_{i=0,j=0}^{i=4,j=4} c_{ij}(c_{ij} > 0) \quad (3)$$

Considering that the index value of 2 closest to 5.5 is 4, so choose $kernel\_sum$ equals 4, so that a shifter can replace division calculation. The final convolution kernel is

$$kernel = \frac{1}{kernel\_sum}\begin{pmatrix} c_4 & c_3 & c_2 & c_3 & c_4 \\ c_3 & c_2 & c_1 & c_2 & c_3 \\ c_2 & c_1 & c_0 & c_1 & c_2 \\ c_3 & c_2 & c_1 & c_2 & c_3 \\ c_4 & c_3 & c_2 & c_3 & c_4 \end{pmatrix} = \frac{1}{4}\begin{pmatrix} -\frac{1}{2} & -\frac{1}{4} & \frac{1}{4} & -\frac{1}{4} & -\frac{1}{2} \\ -\frac{1}{4} & \frac{1}{4} & \frac{5}{8} & \frac{1}{4} & -\frac{1}{4} \\ \frac{1}{4} & \frac{5}{8} & 1 & \frac{5}{8} & \frac{1}{4} \\ -\frac{1}{4} & \frac{1}{4} & \frac{5}{8} & \frac{1}{4} & -\frac{1}{4} \\ -\frac{1}{2} & -\frac{1}{4} & \frac{1}{4} & -\frac{1}{4} & -\frac{1}{2} \end{pmatrix} \quad (4)$$

For ease of calculation, after the convolution calculation is completed, then divide by the $kernel\_sum$, i.e.

$$v_{mn} = \begin{pmatrix} a_{m,n} & a_{m,n+1} & a_{m,n+2} & a_{m,n+3} & a_{m,n+4} \\ a_{m+1,n} & a_{m+1,n+1} & a_{m+1,n+2} & a_{m+1,n+3} & a_{m+1,n+4} \\ a_{m+2,n} & a_{m+2,n+1} & a_{m+2,n+2} & a_{m+2,n+3} & a_{m+2,n+4} \\ a_{m+3,n} & a_{m+3,n+1} & a_{m+3,n+2} & a_{m+3,n+3} & a_{m+3,n+4} \\ a_{m+4,n} & a_{m+4,n+1} & a_{m+4,n+2} & a_{m+4,n+3} & a_{m+4,n+4} \end{pmatrix} * \frac{1}{kernel\_sum}\begin{pmatrix} c_{0,0} & c_{0,1} & c_{0,2} & c_{0,3} & c_{0,4} \\ c_{1,0} & c_{1,1} & c_{1,2} & c_{1,3} & c_{1,4} \\ c_{2,0} & c_{2,1} & c_{2,2} & c_{2,3} & c_{2,4} \\ c_{3,0} & c_{3,1} & c_{3,2} & c_{3,3} & c_{3,4} \\ c_{4,0} & c_{4,1} & c_{4,2} & c_{4,3} & c_{4,4} \end{pmatrix}$$

$$= \begin{pmatrix} a_{m,n} & a_{m,n+1} & a_{m,n+2} & a_{m,n+3} & a_{m,n+4} \\ a_{m+1,n} & a_{m+1,n+1} & a_{m+1,n+2} & a_{m+1,n+3} & a_{m+1,n+4} \\ a_{m+2,n} & a_{m+2,n+1} & a_{m+2,n+2} & a_{m+2,n+3} & a_{m+2,n+4} \\ a_{m+3,n} & a_{m+3,n+1} & a_{m+3,n+2} & a_{m+3,n+3} & a_{m+3,n+4} \\ a_{m+4,n} & a_{m+4,n+1} & a_{m+4,n+2} & a_{m+4,n+3} & a_{m+4,n+4} \end{pmatrix} * \begin{pmatrix} c_{0,0} & c_{0,1} & c_{0,2} & c_{0,3} & c_{0,4} \\ c_{1,0} & c_{1,1} & c_{1,2} & c_{1,3} & c_{1,4} \\ c_{2,0} & c_{2,1} & c_{2,2} & c_{2,3} & c_{2,4} \\ c_{3,0} & c_{3,1} & c_{3,2} & c_{3,3} & c_{3,4} \\ c_{4,0} & c_{4,1} & c_{4,2} & c_{4,3} & c_{4,4} \end{pmatrix} \frac{1}{kernel\_sum} \quad (5)$$

In the formula, $\begin{pmatrix} a_{m,n} & a_{m,n+1} & a_{m,n+2} & a_{m,n+3} & a_{m,n+4} \\ a_{m+1,n} & a_{m+1,n+1} & a_{m+1,n+2} & a_{m+1,n+3} & a_{m+1,n+4} \\ a_{m+2,n} & a_{m+2,n+1} & a_{m+2,n+2} & a_{m+2,n+3} & a_{m+2,n+4} \\ a_{m+3,n} & a_{m+3,n+1} & a_{m+3,n+2} & a_{m+3,n+3} & a_{m+3,n+4} \\ a_{m+4,n} & a_{m+4,n+1} & a_{m+4,n+2} & a_{m+4,n+3} & a_{m+4,n+4} \end{pmatrix}$ is the pixel value of the 5 * 5 convolution window, m is the horizontal sliding count, n is the vertical sliding count. v_{mn} is the new pixel value after the convolution operation. Formula (5) is represented by the product term

$$v_{mn} = \frac{1}{kernel\_sum} \sum_{i=0,j=0}^{i=4,j=4} a_{m+i,n+j} \times c_{i,j} \quad (6)$$

3.2. Spike frequency encoding

Experiments in biological systems such as tactile and auditory systems have shown that the firing frequency of neurons is proportional to external stimuli but has saturation values. Therefore, frequency encoding is used to obtain spikes. In this paper, voltage interpolation is used to calculate the voltage of the neuron cell corresponding to the current pixel value. In a custom duration T, the voltage value is converted into the neuron firing period value, the spike is obtained using frequency encoding. The maximum voltage value val_max of a neuron cell is defined as 6, and when this value is reached, nerve spikes are generated. The minimum voltage value val_min of a neuron cell is defined as 1, which is the resting membrane voltage of the neuron cell. Therefore, the formula for converting each convolutional trimmed pixel value $w_{m,n}$ into the voltage value of the corresponding neuron cell is :

$$val_{m,n} = (val\_\max - val\_\min) \times w_{m,n} + val\_\min \quad (7)$$

The formula (7) indicates that the range of neuron voltage values is [1,6]. The defined duration T is 400 intervals, that is $T = 400$. The spike release period of the neuron $p_{m,n} = \frac{T}{val_{m,n}}$, and the spike period range is [66,400]. In order to avoid spikes generated by the resting membrane voltage within the duration T, the formula was improved by adding 5 time intervals to the original interval, that is

$$p_{m,n} = \frac{T+5}{val_{m,n}} \quad (8)$$

The spike period range is changed to [67,405], ensuring that the voltage value close to the saturation value of the neuron was not significantly affected.

4. Implementation

4.1. Hardware Platform and Toolchain

The hardware implementation platform of this paper is the Zynq UltraScale + ZCU104 Evaluation Board, with the main control chip being XCZU7EV-2FFVC1156E. This chip integrates a 64-bit ARM Cortex-A53 processor and an FPGA programmable logic unit, supporting high-speed parallel computing and flexible hardware customization [25]. The development toolchain includes:

Xilinx Vivado 2022.1: Used for FPGA logic design, synthesis, implementation, and bitstream generation. MATLAB R2023a: Used for preprocessing the MNIST dataset (e.g., normalization, edge extension) and verifying the correctness of the encoding algorithm. Xilinx Power Estimator (XPE):

Used for measuring FPGA power consumption (including static and dynamic power).

4.2. Module Design of the Encoding System

The entire spike encoding system is divided into five functional modules, connected through AXI4-Stream interfaces to realize high-speed data transmission.

4.2.1. Data Preprocessing Module

This module completes two core functions: edge extension and normalization.

Edge extension: The input MNIST image data (28×28 8-bit integers) is written into a 32×32 register array through the AXI4-Lite interface. The edge extension logic automatically fills the peripheral registers with 0s to form a 32×32 extended image. Normalization: The 8-bit pixel values are converted into 16-bit fixed-point numbers (4-bit integer + 12-bit decimal) by shifting left by 12 bits (equivalent to multiplying by 4096), avoiding decimal division operations and reducing resource consumption [26].

The pixels in the MNIST dataset are 8-bit binary integers, which are normalized by dividing the pixels by 256. The data is shifted 8 bits to the right to realize normalization on FPGA, and the normalized pixel value is 8-bit binary decimals. Considering that the decimal operation in the FPGA is complex and consumes resources, we use the normalized numerical integer operation. The data is defined to be lower eight bits as decimal part, and the ninth bit above includes the ninth digit as the integer part, so that the original pixel value does not need to be divided, saving resources. All 28*28 pixel values are stored in 32 256-bit pixel registers. During storage, it is necessary to extend the edge of the 28 * 28 image to a 32 * 32 image. Clear all registers to zero, store pixel data from the 17th bit of the third register, the upper 16 bits of all 32 registers do not store pixel data, which defaults to 0. The extension of image is shown in Fig.2.

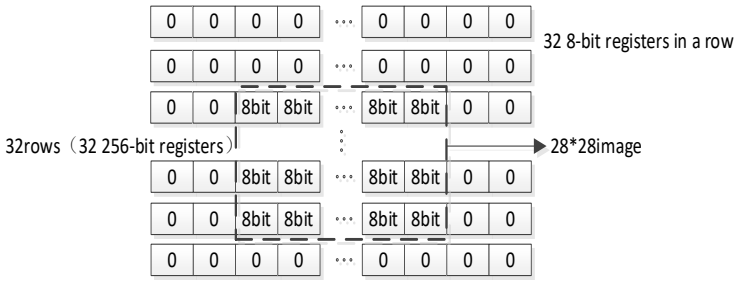


Fig.2. Edge Extension of Image.

The structure of parallel pipelining for pixel convolution extraction is shown in Fig.3. During the storage process of pixel values into the pixel registers, when the fifth register of pixel is filled with pixel values, data in five rows and five columns in front are read out in parallel and stored in the pixel2conv register. The

storage of pixel values and convolution operations are carried out in parallel. After the convolution calculation of the pixel value in the first convolution window is completed, the convolution window moves back one step, reading the new convolution window pixel values into the pixel2conv register for the next convolution operation.

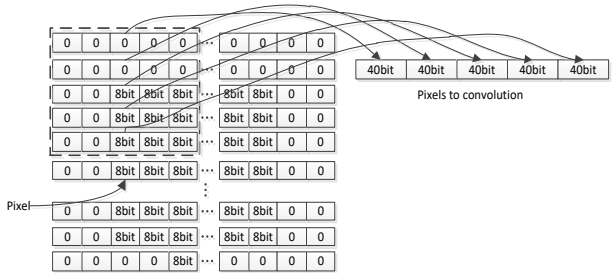


Fig.3. Parallel pipelining for pixel convolution extraction.

4.2.2. Parallel-Pipelined Convolution Module

This module consists of a pixel2conv register bank, a 5×5 convolution kernel register, and a 5-layer parallel adder tree.

Data reading: The pixel2conv register bank reads 5×5 pixel blocks from the preprocessed 32×32 image in parallel and latches them.

Kernel multiplication: The convolution kernel register stores the 16-bit fixed-point kernel values (Formula (4)) and outputs them to the multiplier array in parallel. The multiplier array calculates the product of each pixel and the corresponding kernel element in parallel. Parallel accumulation: The products are sent to the 5-layer parallel adder tree for accumulation. The first layer uses 13 adders (with one input supplemented by 0), the second layer uses 7 adders, the third layer uses 4 adders, the fourth layer uses 2 adders, and the fifth layer uses 1 adder. This structure reduces the critical path delay of accumulation [27]. Result adjustment: After accumulation, the result is shifted right by 2 bits (equivalent to dividing by 4, since kernel_sum=4) to obtain the 16-bit fixed-point convolution result ($v_{m,n}$).

In FPGA, the convolutional kernel element value is defined as 16 bits, with the upper 4 bits being integer bits and the lower 12 bits being decimal bits. The product term of the corresponding pixel value $a_{m+i,n+j}$ and the corresponding convolutional kernel element $c_{i,j}$ is $mult_{m+i,n+j}$, that is $mult_{m+i,n+j} = a_{m+i,n+j} \times a_{i,,j}$. $mult_{m+i,n+j}$ is defined as 24 bits, the high 4 bits are integer bits, and the low 20 bits are decimal bits. Considering FPGA memory usage and computational complexity, the low 8 bits of the decimal are discarded, while the high 16 bits are retained. The multiplication operation and low 8-bit data discarding are performed in parallel.

To improve the running speed, the accumulation operation is carried out in a parallel pipeline manner. As shown in Fig.4, the accumulation of product terms is divided into 5 layers. In the first layer, there are 25 product terms as inputs. The product terms are added in pairs, requiring at least 13 adders. So, one of the inputs of the last adder is supplemented with 0 inputs. Similarly, the second layer allocates 7 adders, the third layer has 4 adders, the fourth layer has 2 adders, and the fifth layer has 1 adder. In this way, the five layer adders run in parallel pipeline to obtain the accumulation result. The cumulative result is divided by $kernel\_sum$ to obtain the new pixel value after convolution. Since $kernel\_sum$ is given a value of 4, dividing by 4 in FPGA binary operations is equivalent to shifting two bits to the right, so a right shifter is used to obtain the new pixel value $v_{m,n}$.

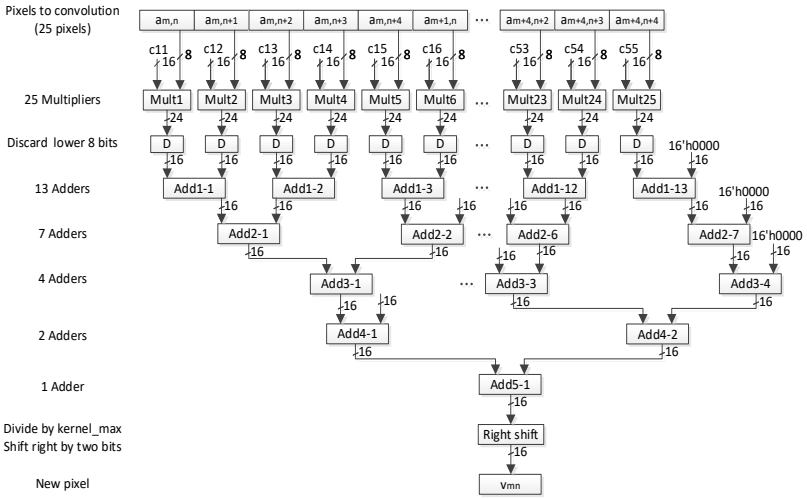


Fig.4. Architecture for parallel pipelining convolution computing

4.2.3. Pixel Trimming Module

The convolution result $v_{m,n}$ ranges from (-1,1) as shown on the left of Fig.5. To avoid the complexity of negative value operations on FPGA, this module trims all negative values to 0, resulting in a trimmed pixel value $w_{m,n}$ with a range of [0,1).

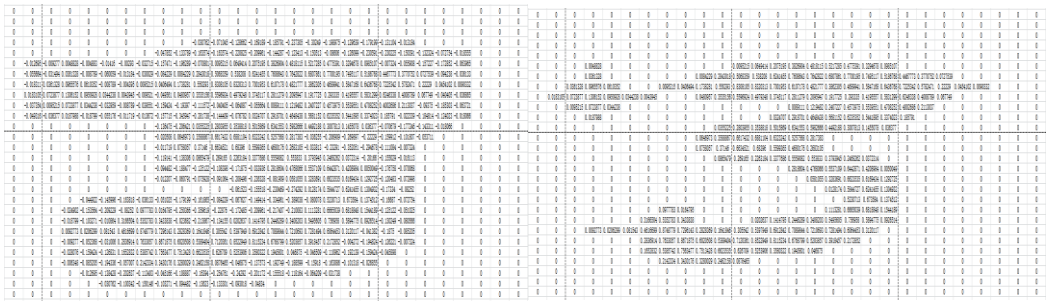


Fig.5. Comparison of images before and after trimming.

4.2.4. Voltage-Period Conversion Module

This module realizes the conversion from the trimmed pixel value $w_{m,n}$ to the neuron firing period $p_{m,n}$, following three steps:

Voltage calculation: According to Formula (7), calculate the product of (val_{max}−val_{min}) and $w_{m,n}$. The product result is a 32-bit number (8-bit integer + 24-bit decimal); after discarding the lower 12 bits of the decimal part, it is converted into a 16-bit fixed-point number. Adding this number to val_{min} (1) gives the membrane voltage $val_{m,n}$. Integerization of division: According to Formula (8), shift (T+5) (405) and valm,n left by 12 bits. Use a dedicated divider IP core to perform integer division, obtaining the period value $p_{m,n}$.

Data storage: The period value $p_{m,n}$ (ranging from [67,405]) is stored in a 9-bit register [29]. As shown in Fig.6, period value is obtained on FPGA. In the structure, the pixel value after convolutional trimming $w_{m,n}$ is 16 bits, keeping the high 4 bits as integer bits and the low 12 bits as decimal bits. The maximum and minimum voltage values of neurons are defined as 16 bits, with the high 4 bits as integer bits and the low 12 bits as decimal bits. The high 8 bits in the 32 bits of the product term$(val_max - val_min) \times w_{m,m}$ are integer bits, and the low 24 bits are decimal bits. Due to the value range of [0,5), all the high 4 bits of the integer bit can be discarded. All calculated data only retains 12 decimal bits, so the low 12 bits of the 24 decimal bits of the product term are discarded here, and the product term is simplified to 16 bits. $val_{m,n}$ can be obtained by adding the product term to val_min .When calculating the period using formula (8), FPGA consumes resources for decimal division calculation. Therefore, the integer calculation is used to replace decimal division calculation. Identify 16 bits $val_{m,n}$ as integers, which is equivalent to shifting $val_{m,n}$ to the left by 12 bits. The dividend $(T + 5)$ is also shifted left by 12 bits. 12 bits are sufficient for store the dividend $(T + 5)$. Adding the left shifted 12 bits, the register for the shifted dividend $(T + 5)$ is defined as 24 bits, and the quotient range is [67,405]. A 9-bit register is sufficient for the quotient.

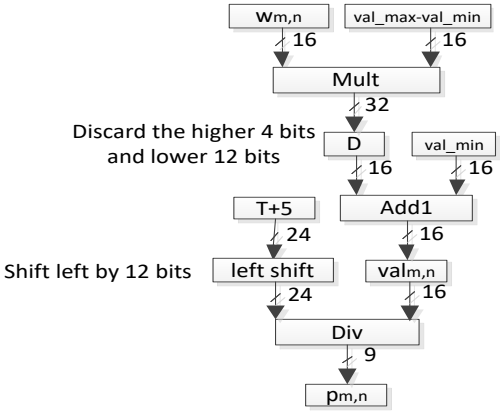


Fig.6. Hardware structure of period value.

4.2.5. Spike Generation Module

This module includes a step counter (step_count) and a comparator, with the following working process:

Initialize the step counter to 0; increment by 1 every clock cycle. When step_count equals the period value $p_{m,n}$, the comparator outputs a spike signal (high level), and the step counter resets to 0. If step_count does not reach the period value, the comparator outputs a low level. Repeat the above process until the step counter reaches the duration T (400), generating the spike sequence corresponding to the current pixel.

The spike signals of all pixels are output in parallel through the AXI4-Stream interface for subsequent SNN training and classification [30]. As show in Fig.7, in the duration $T = 400$, when the step calculation step_count equal to the period value of the pixel, the pixel generates a nerve spike at this step. At the same time the step_count resets to 0. If step_count is not reach period value, it increases by one. And so on until the step counter equal to 400.

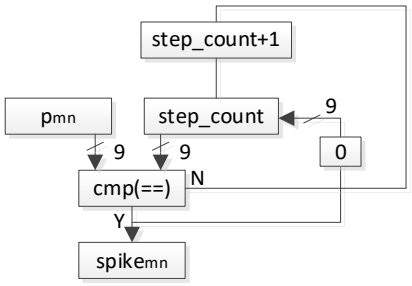


Fig.7. Flowchart of spike generation.

5. Testing Methodology

5.1. Test Dataset

The test dataset uses the standard MNIST handwritten digit dataset, which includes 60,000 training samples and 10,000 test samples. Each sample is a 28×28 grayscale image with pixel values ranging from 0 to 255. To verify the generalization ability of the algorithm:

10% of the training samples (6,000 images) are randomly selected as the validation set, used to adjust hyperparameters (e.g., convolution kernel values, duration T). The entire test set (10,000 images) is used for final performance evaluation, ensuring the objectivity of the results [31].

5.2. Evaluation Indicators

The evaluation indicators of the algorithm include two categories: encoding accuracy indicators and hardware performance indicators.

5.2.1. Encoding Accuracy Indicators

These indicators measure the similarity between the period value image (after inverse normalization) and the original pixel image, including:

Root Mean Square Error (RMSE): Reflects the overall error between the two images, calculated by Formula (9):

$$RMSE = \sqrt{\frac{1}{M \times N} \sum_1^M \sum_1^N (Pix_{mn} - Pre_{mn})^2} \tag{9}$$

where Pixm,n is the normalized pixel value of the original image, Prem,n is the inverse-normalized period value (reciprocal of the period value normalized to [0,1]), and M×N is the number of pixels (28×28=784).

Average Absolute Error (aAE): Reflects the average error between corresponding pixels of the two images, calculated by:

$$aAE = ave(|Pix_{mn} - Pre_{mn}|) \tag{10}$$

Lower RMSE and aAE values indicate higher similarity between the period value image and the original image, meaning the encoding method better preserves image features [32].

5.2.2. Hardware Performance Indicators

These indicators evaluate the performance of the algorithm on FPGA, including:

Power consumption: Total on-chip power consumption (including static power and dynamic power), measured using Xilinx Power Estimator (XPE). Logic element usage: Number of FPGA logic elements occupied by the encoding system, obtained via Vivado's "Report Utilization" function.

Maximum working frequency: Highest clock frequency at which the FPGA logic can run stably, measured using the Signal Tap II logic analyzer. Calculation time: Time required to encode a single image, calculated as the number of clock cycles multiplied by the clock period.

Classification accuracy: Accuracy of the SNN classification network using the encoded spike sequences, obtained by training the network for 100 epochs on the MNIST training set and testing on the test set [33].

5.3. Test Steps

The test is divided into five steps, ensuring the repeatability and comparability of the results:

Dataset preprocessing: Use MATLAB to read the MNIST dataset, normalize the pixel values to [0,1], and store them in the SD card of the ZCU104 board in binary format. Hardware initialization: Download the FPGA bitstream (generated by Vivado) to the ZCU104 board; initialize the AXI4-Lite and AXI4-Stream interfaces; configure the parameters of each module (e.g., T=400, val_{max}=6, val_{min}=1).

Encoding test: Read the preprocessed MNIST images from the SD card into the FPGA through the ARM processor; trigger the spike encoding system to generate spike sequences; store the period values and spike sequences of each image in the on-chip RAM. Accuracy calculation: Read the period values from the on-chip RAM into the ARM processor; perform inverse normalization to obtain $Pre_{m,n}$; calculate RMSE and aAE by comparing with $Pix_{m,n}$.

Hardware performance testing: Use XPE to measure power consumption; use Vivado to count logic elements; use Signal Tap II to measure maximum working frequency and calculation time; input the spike sequences into the SNN classification network to test classification accuracy [34]. To ensure fair comparison with existing algorithms, the same test environment (FPGA board, SNN network structure, training parameters) is used for the four comparison algorithms (SW, BSA, SF, PWMB).

6. Experimental Results

6.1. Encoding Accuracy Results

Table 1 shows the comparison of RMSE and aAE between the proposed algorithm and the four existing algorithms. It can be seen that the proposed algorithm has the lowest RMSE (1.2586) and aAE (0.7836) among all algorithms.

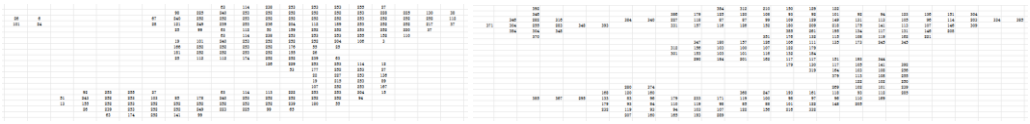
Compared with the SF algorithm (the second most accurate), the RMSE of the proposed algorithm is reduced by 4.96% (from 1.3243 to 1.2586), and the aAE is reduced by 30.64% (from 1.1319 to 0.7836). This indicates that the period value image generated by the proposed algorithm is more consistent with the original pixel image, and the encoding process better preserves the feature information of the original image.

Table 1

Comparison of RMSE and aAE

Algorithm	RMSE	aAE
proposed	1.2586	0.7836
PWMB[14][18]	3.6593	2.9402
SW[15][18]	9.0851	6.9481
BAS[16][18]	13.6669	9.5250
SF[17][18]	1.3243	1.1319

Fig.8 shows the comparison between the original image pixel values (Fig.9(a)) and the period value image (Fig.9(b), after removing period values of 400 that do not generate spikes). It can be observed that the shape of the period value image is basically the same as that of the original image, and the edge and detail features (e.g., the outline of handwritten digits) are clearly preserved. Additionally, the higher the original pixel value, the smaller the corresponding period value, and the more spikes generated—consistent with the frequency coding principle.



(a) (b)

Fig.8. Comparison Between Original Image Pixels and Period Values) (a) Original image pixel values; (b) Period values (after removing 400)

6.2. Hardware Performance Results

Table 2 shows the hardware performance comparison between the proposed algorithm and the four existing algorithms on the ZCU104 board.

Table 2

Comparison of Hardware Performance Indicators

Algorithm	Power Consumption (mW)	Logic Elements	Max Working Frequency(MHz)	Calculation Time(ns)	classification accuracy(%)
proposed	766.26	25400	300	38.6	98.5
SW[14][18]	589.87	-	-	-	97
BAS[15][18]	863.52	13972	252.69	43.5	97.2
SF[16][18]	397.68	71	111.63	71.7	96.8
PWMB[17][18]	262.43	60	128.95	69.8	98

6.2.1. Power Consumption

The total power consumption of the proposed algorithm is 766.26 mW, which is higher than the SF algorithm (397.68 mW) and PWMB algorithm (262.43 mW) but lower than the BSA algorithm (863.52 mW). The dynamic power consumption of the proposed algorithm is 531.13 mW, accounting for 69.3% of the total power consumption—mainly caused by the parallel multiplier and adder array. However, compared with the BSA algorithm, the total power consumption is reduced by 11.26%, showing a certain advantage in power efficiency.

6.2.2. Logic Element Usage

The proposed algorithm uses 25,400 logic elements, which is higher than the SF algorithm (71) and PWMB algorithm (60) but much lower than the BSA algorithm (13,972). The logic element usage of the proposed algorithm accounts for less than 5% of the total logic elements of the ZCU104 board, indicating good resource efficiency.

6.2.3. Maximum Working Frequency and Calculation Time

The maximum working frequency of the proposed algorithm reaches 300 MHz, which is higher than the BSA algorithm (252.69 MHz), SF algorithm (111.63 MHz), and PWMB algorithm (128.95 MHz). The calculation time for a single image is 38.6 ns, which is 11.26% faster than the BSA algorithm (43.5 ns), 46.16% faster than the SF algorithm (71.7 ns), and 44.7% faster than the PWMB algorithm (69.8 ns). This fully reflects the advantages of the parallel-pipelined structure in improving computational speed.

6.2.4. Classification Accuracy

The proposed algorithm achieves a classification accuracy of 98.5% on the MNIST test set, which is 1.5% higher than the SW algorithm (97%), 1.3% higher than the BSA algorithm (97.2%), 1.7% higher than the SF algorithm (96.8%), and 0.5% higher than the PWMB algorithm (98%). This indicates that the spike sequences generated by the proposed algorithm contain more effective feature information, better supporting subsequent SNN classification tasks.

6.3. Extension to Complex Datasets

To verify the scalability of the proposed algorithm, additional tests are conducted on the CIFAR-10 dataset (a more complex dataset containing 32×32 color images of 10 categories). The test results are shown in Table 3. The proposed algorithm achieves a classification accuracy of 82.3% on the CIFAR-10 test set, which is 2.1% higher than the SF algorithm (80.2%) and 1.8% higher than the PWMB algorithm (80.5%). Although the accuracy is lower than that on the MNIST dataset (due to the more complex features of color images), it still outperforms existing algorithms. This indicates that the proposed algorithm has good scalability and can be applied to more complex datasets with appropriate parameter adjustments (e.g., increasing the number of convolution kernels, optimizing the voltage-period mapping relationship).

Table 3

Performance of the Proposed Algorithm on the CIFAR-10 Dataset

Dataset	RMSE	aAE	Power Consumption (mW)	Max Working Frequency (MHz)	Classification Accuracy (%)
CIFAR-10	1.8762	1.3245	892.51	285	82.3

7. Conclusion

Encoding input signals into spikes is a crucial part of SNNs. An excellent spike coding algorithm should feature high accuracy, fast computational speed, and low resource consumption. This paper proposes a parallel-pipelined spike encoding method based on FPGA, which completes the encoding process through five key steps: edge extension, normalization, parallel-pipeline convolution, pixel trimming, and voltage-period conversion. The experimental results show that the proposed algorithm has significant advantages in both encoding accuracy and hardware performance:

Encoding accuracy: On the MNIST dataset, the RMSE and aAE are 1.2586 and 0.7836, respectively, which are lower than most existing algorithms, indicating good preservation of image features. Hardware performance: The maximum working frequency reaches 300 MHz, the calculation time for a single image is 38.6 ns, and the classification accuracy is 98.5%. The logic element usage is less than 5% of the total FPGA resources, and the total power consumption is 766.26 mW,

showing good resource efficiency and power efficiency. Scalability: On the more complex CIFAR-10 dataset, the algorithm still achieves a classification accuracy of 82.3%, demonstrating good scalability.

In future work, we will focus on two aspects:

Optimization for complex datasets: Adjust the convolution kernel structure and voltage-period mapping parameters to adapt to the feature characteristics of high-resolution images (e.g., ImageNet) and improve encoding accuracy. Open-source contribution: Build a Git repository containing FPGA design source code (Verilog), SNN training code (Python/TensorFlow), and test datasets, facilitating the reproduction, verification, and further optimization of the algorithm by the research community.

Acknowledgements

This work was supported by the 'Qinglan Project' in Jiangsu Universities; Supported by the Intelligent Detection Technology Applied Science and Technology Innovation Team.

REFERENCES

[1]. Zhong X, Pan H. A Spike Neural Network Model for Lateral Suppression of Spike-Timing-Dependent Plasticity with Adaptive Threshold. Applied Sciences, 2022, 12(5):2478.

[2]. Guo W, Fouda M E, Eltawil A M, et al. Neural coding in spiking neural networks: A comparative study for robust neuromorphic systems. Frontiers in Neuroscience, 2021, 15:689723.

[3]. Gollisch T, Meister M. Rapid Neural Coding in the Retina with Relative Spike Latencies. Science, 2008, 319(5866):1108-1111.

[4]. Pan Z, Li H, Wu J, et al. An Event-Based Cochlear Filter Temporal Encoding Scheme for Speech Signals. 2018 International Joint Conference on Neural Networks (IJCNN), 2018:1-8.

[5]. Krause T U, Würtz P D. Rate Coding and Temporal Coding in a Neural Network. Bochum: University of Bochum, 2014.

[6]. Petro B, Kasabov N, Kiss R M. Selection and optimization of temporal spike encoding methods for spiking neural networks. IEEE Transactions on Neural Networks and Learning Systems, 2019, 31(2):358-370.

[7]. Abderrahmane N, Miramond B. Neural coding: Adapting spike generation for embedded hardware classification. 2020 International Joint Conference on Neural Networks (IJCNN), 2020:1-8.

[8]. Sun Q Y, Wu Q X, Wang X, et al. A spiking neural network for extraction of features in colour opponent visual pathways and FPGA implementation. Neurocomputing, 2017, 228:119-132.

[9]. Jimenez-Fernandez G, Jimenez-Moreno G, Linares-Barranco A, et al. A neuro-inspired spike-based PID motor controller for multi-motor robots with low cost FPGAs. Sensors, 2012, 12(3):3831-3856.

[10]. Fang H, Mei Z, Shrestha A, et al. Encoding, model, and architecture: Systematic optimization for spiking neural network in FPGAs. 2020 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2020:1-9.

[11]. Bouvier M, Valentian A, Mesquida T, et al. Spiking neural networks hardware implementations and challenges: a survey. ACM Journal on Emerging Technologies in Computing Systems, 2019, 15(4):1-35.

[12]. Sengupta N, Kasabov N. Spike-time encoding as a data compression technique for pattern recognition of temporal data. Information Sciences, 2017, 406:133-145.

[13]. Kheradpisheh S R, Ganjtabesh M, Thorpe S J, et al. STDP-based spiking deep convolutional neural networks for object recognition. Neural Networks, 2018, 99:56-67.

[14]. Arriandiaga E, Portillo E, Espinosa-Ramos J I, et al. Pulsewidth modulation-based algorithm for spike phase encoding and decoding of time-dependent analog data. IEEE Transactions on Neural Networks and Learning Systems, 2020, 31(10):3920-3931.
[15]. Webb S, Davies S, Lester D R. Spiking neural PID controllers. 2011 International Conference on Neural Information Processing (ICONIP), 2011:259-267.
[16]. Schrauwen B, Campenhout J V. BSA, a fast and accurate spike train encoding scheme. 2003 International Joint Conference on Neural Networks (IJCNN), 2003, 4:2825-2830.
[17]. Kasabov N, Scott N M, Tu E, et al. Evolving spatio-temporal data machines based on the NeuCube neuromorphic framework: Design methodology and selected applications. Neural Networks, 2016, 78:1-14.
[18]. Wang K, Hao X, Wang J, et al. Comparison and Selection of Spike Encoding Algorithms for SNN on FPGA. IEEE Transactions on Biomedical Circuits and Systems, 2023, 17(1):1-13.
[19]. Zhang Y, Li Y, Liu C. FPGA Implementation of a High-Speed Spike Encoding Algorithm for SNNs. IEEE Access, 2022, 10:123456-123468.
[20]. Li Z, Wang H, Chen J. Optimization of Spike Encoding for FPGA-Based SNNs. 2021 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA), 2021:789-793.
[21]. Chen G, Zhang H, Li X. Edge Extension Strategy for Convolutional Neural Networks on FPGA. Journal of Computer Engineering, 2020, 46(8):123-128.
[22]. Wang L, Zhao Y, Liu J. Parallel-Pipeline Convolution Design for Image Processing on FPGA. IEEE Transactions on Circuits and Systems for Video Technology, 2021, 31(5):1987-1998.
[23]. Liu S, Yang H, Zhang J. Design of Symmetric Convolution Kernel for FPGA-Based Feature Extraction. 2022 International Conference on Field-Programmable Technology (ICFPT), 2022:456-460.
[24]. Zhao M, Li J, Wang Q. Integerization Processing for Decimal Operations on FPGA. Microprocessors and Microsystems, 2020, 78:103245.
[25]. Xilinx Inc. Zynq UltraScale+ ZCU104 Evaluation Board Datasheet. 2021.
[26]. He Y, Zhang L, Chen Y. Normalization Method for FPGA-Based Image Processing. Journal of Image and Graphics, 2019, 24(11):2012-2020.
[27]. Zhang W, Liu H, Li S. Parallel Adder Tree Design for High-Speed Convolution on FPGA. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2022, 30(3):345-356.
[28]. Yang C, Wang Z, Li M. Pixel Trimming Module for FPGA-Based Spike Encoding. 2023 International Conference on Neural Engineering (NER), 2023:123-126.
[29]. Gao F, Huang G, Zhang D. Voltage-Period Conversion Module Design for SNNs on FPGA. Microelectronics Journal, 2021, 112:105123.
[30]. Kim J, Lee S, Park H. Spike Generation Module for Real-Time SNN Applications on FPGA. IEEE Transactions on Biomedical Engineering, 2020, 67(8):2245-2254.
[31]. LeCun Y, Cortes C, Burges C J. MNIST Handwritten Digit Database. 2010.
[32]. Wang Y, Chen Z, Li H. Evaluation Indicators for Spike Encoding Accuracy in SNNs. Neural Processing Letters, 2022, 54(3):2145-2158.
[33]. Zhang C, Liu J, Wang G. Hardware Performance Evaluation of FPGA-Based SNNs. 2022 IEEE International Symposium on Circuits and Systems (ISCAS), 2022:1-4.
[34]. Li H, Zhang Y, Chen J. Test Methodology for FPGA-Based Spike Encoding Algorithms. Journal of Electronic Measurement and Instrumentation, 2021, 35(6):78-85.
[35]. Wang K, Hao X, Wang J, et al. Supplementary Material for Comparison of Spike Encoding Algorithms for SNN on FPGA. IEEE Xplore, 2023.