Circuits and Systems
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ens.ewi.tudelft.nl/

CAS-2021-11

M.Sc. Thesis

Hardware Spiking Neural Network based Sbox AES

Hanyu Ma

Abstract

Hardware cryptographic algorithm implementation is easy to attack by side-channel attacks. The power-based side-channel attacks are powerful among several side-channel attacks. This attack methods use the relationship between the leakage model and power traces to reveal the secret key. Some existing countermeasures like mask and hide can protect the algorithms from attacking. However, they can not break the relationship between power traces and the leakage model. Based on the property of the neural network, the linear relationship can be easily broken. Furthermore, the spiking neural network is more hardware-friendly than a conventional neural network. The design replaces the sbox in AES with a pipeline spiking neural network-based sbox and implements it in hardware. The help of the FPGA attack platform demonstrates that the proposed design can resist DPA, CPA, Template Attacks, and Deep Learning-based attacks.

Hardware Spiking Neural Network based Sbox AES

Thesis

submitted in partial fulfillment of the
requirements for the degree of

Master of Science

in

Microelectronic

by

Hanyu Ma
born in Wu Han, China

This work was performed in:

Circuits and Systems Group
Department of Microelectronics & Computer Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Delft University of Technology

Delft University of Technology
Department of
Microelectronics & Computer Engineering

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled **"Hardware Spiking Neural Network based Sbox AES"** by **Hanyu Ma** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: My Graduation Date

Chairman: _____
dr.ir. T.G.R.M van Leuken

Advisor: _____
dr.ir. David Aledo Ortega

Committee Members: _____
dr.ir. Mottaqiallah Taouil

_____

Abstract

Hardware cryptographic algorithm implementation is easy to attack by side-channel attacks. The power-based side-channel attacks are powerful among several side-channel attacks. This attack methods use the relationship between the leakage model and power traces to reveal the secret key. Some existing countermeasures like mask and hide can protect the algorithms from attacking. However, they can not break the relationship between power traces and the leakage model. Based on the property of the neural network, the linear relationship can be easily broken. Furthermore, the spiking neural network is more hardware-friendly than a conventional neural network. The design replaces the sbox in AES with a pipeline spiking neural network-based sbox and implements it in hardware. The help of the FPGA attack platform demonstrates that the proposed design can resist DPA, CPA, Template Attacks, and Deep Learning-based attacks.

Acknowledgments

I would like to thank my advisor dr.ir. T.G.R.M van Leuken for his assistance during the writing of this thesis, and David who also gives many advice during project. And the Abdullah from CE group provides the experiment platform for me to do the side channel attack. Without them, this would not have been possible.

Hanyu Ma
Delft, The Netherlands
My Graduation Date

Contents

List of Figures

List of Tables

Nomenclature

Physics Symbol

υ	Voltage
I	Current
C	Capacitance
R	Resistance
t	Time

Mathematical Symbol

$\sum_a^b$	mathematical sum
$\oplus$	Xor function
$\frac{\mathrm{d}x}{\mathrm{d}y}$	Derivation
$GF(2^n)$	The finite field of two elements
$[]_{i,j}$	Entry (i, j) of matrix
μ	The mean of amount of numbers
$g_{i,j}$	The pearson's correlation function

Other Symbol

G	The array of pearson's correlation coefficient array
S	The hypothetical Sbox result array
H	The hypothetical leakage model result array
T	The collected power traces array
P	The plaintext array
$f(x)$	The probability density distribution function
c_i, j	The covariance function

1 Introduction

1.1 Motivation

The development of computers, the Internet, and communications make the world becomes a highly digital information world in today's technology. The increasing reliance on technology is also a double-edged sword, as they give a chance for the attacker to get the data by various attack methods. It is becoming more and more essential to secure every aspect of information and data. Encryption is a way of scrambling data so that only authorized parties can understand the information preventing people from economic, reputation, and privacy loss. However, encryption can not guarantee safety due to thousands of attackers in the real world. This leads to the demand for countermeasures and resistive attack methods.

There are already many encryption algorithms in the encryption field. And this research focuses on the advanced encryption standard(AES)[18], also known by its original name Rijndael. It is symmetric key encryption that is implemented in software and hardware throughout the world to encrypt sensitive data.

While some attacks can be protected by software-based security, they cannot defend from all attacks. For example, a device is used for data transmission, but if the attacker gets access to the physical device, then no amount of software routine can provide protection. Thus, this is where hardware security comes in and why it is becoming increasingly popular with SoCs, microcontrollers, and microprocessors.

However, even the system equipped with hardware security is also vulnerable to side-channel attacks(SCA)[32]. SCA takes advantage of patterns in the information exhaust that hardware constantly gives off: power[32], time[17], radiation[35], noise[4], etc. The attacks can recover the encryption key without any expensive tools. Furthermore, attackers can perform SCA even when they can not access the device. Thus, an effective countermeasure against SCA is becoming a hot topic recently.

Comparing with other SCA, power-based SCA has the advantage of less sensitivity to noise, easy implementation, and focusing on the point of attack. Power-based SCA has been proved the most powerful and attractive to many attackers. In different abstraction levels in devices, different countermeasures against power-based SCA have been implemented. And those of them are proven to be different degrees of success with the cost of the area, power, speed, etc.

Therefore, this thesis focuses on the countermeasures against power-based SCA in AES. In AES, the attackers can easily get the Hamming Weight or Distance model from substitute phases(S-box). The other phases will give out less information even uses good models. So the S-box is the weakest point because of the correlation between power consumption and the leaky model. In order to break out the correlation between those two, [54] already replaced the sbox with conventional neural network sbox in

software. We introduce the spiking neural network into AES for hardware. Exploration of hardware security while using artificial intelligence methods such as spiking neuron networks to improve the resistibility of hardware AES and power efficiency.

1.2 Goal

The neural network based sbox can protect the AES from side channel attack has been demonstrated by [54]. However, not all types of neural network can achieve the task. In this thesis, we want to demonstrate the spiking neural network can protect the AES in hardware aspect and find its advantages.

1.3 Methodology

1. Find a suitable spiking neural network simulator to train the network properly.
2. Building a hardware single spike neuron hardware spiking neuron network to substitute the S-box inside AES. The input information going through the S-box would be encoded by clock-driven coding to decrease the spikes amount.
3. The SNN architecture consists of several neuron cores which use AER communication protocol to improve performance.
4. In order to do the side-channel attack. To synthesize the digital implementation and use the chipwhisperer simulation tool to get the power waveforms, and perform DPA and CPA side-channel attacks on the proposed SNN with trained weight.

1.4 Contribution

The contribution of this thesis work are following:

- Encoding the dataset input and then obtain the corresponding values from the simulator. Furthermore, design the digital hardware machine learning spiking neural network RTL model for S-box.
- Instead of using the Spice model to get the power traces, I choose to implement the model on FPGA. The power traces collected from the FPGA can be easily analyzed using Differential Power Analysis(DPA) and Correlation Power Analysis.

1.5 Thesis Outline

- Chapter 2 introduces the background of side-channel attacks and the details of the implementation methods.
- Chapter 3 introduces the overview of spiking neural networks in spike representation, neural model, communication methods, and weight memory mapping.

- Chapter 4 presents the details of the implementation of the proposed spiking neural networks in simulator.
- Chapter 5 presents the details of the hardware design of the proposed spiking neural network.
- Chapter 6 presents the implementation of the test model and an evaluation of the results.
- Chapter 7 presents the conclusion from the above implemetation and the future works needs to be done.

Background of Advanced Encryption Standard and Power based Side-channel Attacks

2

In this chapter, the Advanced Encryption Standard(AES) details are described to understand the most popular encryption algorithm. Furthermore, the power-based side-channel attacks, which are very powerful to attack the AES, are introduced to prepare for the experiment. Finally, the state-of-art countermeasures to power-based side-channel attacks are listed for comparison with the thesis's countermeasure.

2.1 Advanced Encryption Standard

AES (Advanced Encryption Standard) is a symmetric key block cipher based on substitution–permutation network. It comprises a series of linked operations, some involving replacing inputs with specific outputs (substitutions) and others shuffling bits around (permutations). The data packet length must be 128 bits, and the key length used is 128, 192, or 256 bits. The number of rounds is different with the key length, 10 rounds for 128 bits keys, 12 rounds for 192 bits keys, and 14 rounds for 256 bits keys. The three AES algorithms with different key lengths are called "AES-128", "AES-192", and "AES-256" respectively. The target in this thesis is "AES-128".

The AES encryption transformation function does not have the Feistel structure. Instead, the round transformation comprises three distinct invertible uniform transformations. The linear mixing layer, the non-linear layer, the key addition layer. With exception of the last round, each round transformation involves four operations: **SubBytes**, **ShiftRows**, **MixColumns**, and **AddRoundKey**. The decryption process is the corresponding reverse operation. Since each operation is reversible, the plaintext can be recovered by decrypting in the reverse order. The key for each round of encryption and decryption is obtained by the initial **KeyExpansion**. The workflow is shown in Figure 2.1 [18].

Interestingly, the AES treats bytes level, representing the finite field GF(2^8) rather than the bits level. AES ranges the 128 bits data into 4-byte words, which can form the intermediate cipher result, also called State. Each column has 32 bits. The cipher key is a rectangular matrix with four rows. Furthermore, after round transformation, the array will create a same-type array for the next round.

2.1.1 Round transformation

In order to execute an available attack, it is necessary to understand the details of the AES algorithms. The central part inside the AES is the round transformation

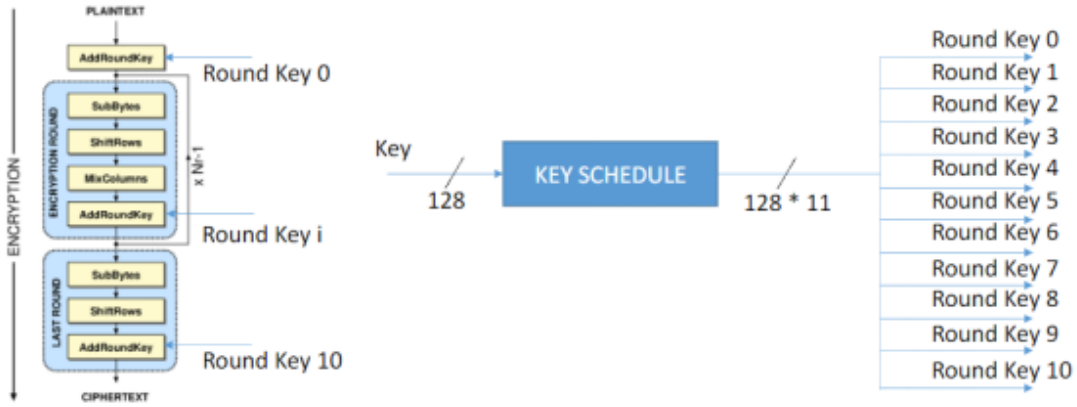

Figure 2.1: The AES workflow and the key expansion [52].

which contains four different operations. As mentioned above, every round has four operations, but the last round does not have the MixColumns step.

2.1.1.1 SubBytes

The SubByte transformation is the unique layer that is the only non-linear byte substitution, operating on each state byte independently. The function is invertible and constructed by the following two functions [8].

- First, Implementing multiplicative inverse to state in GF(2^8) using an irreducible polynomial ($x^8 + x^4 + x^3 + x + 1$).
- An affine transformation will then apply on the inverse state over GF(2).

Considered the actual memory and computation complexity, storing the result as a look-up table(LUT) in the memory like Figure 2.2 can decrease the computation complexity but bring the memory storage problem. Figure 2.3 can better explain the subbytes process.

As mentioned before, S-box has good non-linear properties and is resistant to linear and differential cryptanalysis. However, the attacker can still create a leakage model using the power traces from the device to crack the key. Instead of using the LUT, the multiplication with two can be implemented with a shift and conditional exor. In a straightforward implementation, the execution time of this operation will depend on the input value. This may allow an attacker to mount a timing attack. Insert a NOP-operations can make the execution time equal. Nevertheless, introduce the weakness of the power analysis attack. So the use of a LUT effectively counters these types of attacks.

2.1.1.2 ShiftRows

ShiftRows is easier to understand and more direct. The idea of shift rows is to shift the position of states to make a more diffusion array. Figure 2.4 helps to provide a visual operation.

	0	1	2	3	4	5	6	7	8	9	0a	0b	0c	0d	0e	0f
0	63	7c	77	7b	f2	6b	6f	c5	30	1	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	4	c7	23	c3	18	96	5	9a	7	12	80	e2	eb	27	b2	75
40	9	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	0	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	2	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	6	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	8
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	3	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figure 2.2: The Sbox look up table contains the 256 cases conversion [55].

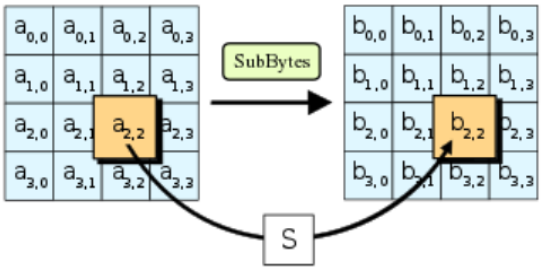

Figure 2.3: In the SubBytes step, each bytes in the array is substituted with the corresponding value in the sbox LUT [55].

Bytes in the state matrix will shift following a standard. The first row will stay still. The second row will shift to the left by one, in the third by two, and in the fourth by three, illustrated in Figure 2.4.

2.1.1.3 MixColumns

In short, MixColumns is a linear transformation of the array. It contains matrix multiplication and bitwise XOR operation. The addition and multiplication will perform over GF(2^8) like Equation 1.1. The multiplication will be performed on the result matrix of ShiftRows. It must be noted that all the columns of the matrix will perform this function. Moreover, the result matrix will have the same size as the original one. Figure 2.5 shows the MixColumns process.

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \tag{1.1}$$

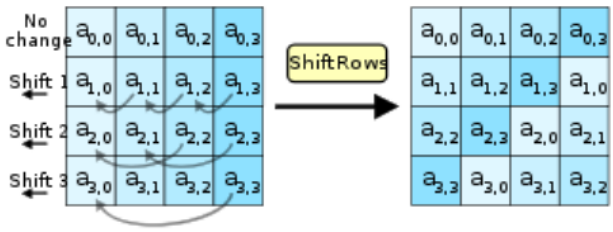

Figure 2.4: In the ShiftRows step, bytes in each row of the array are shifted cyclically to the left [55].

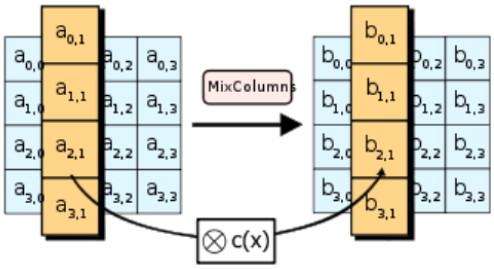

Figure 2.5: In the MixColumns step, each column of the array is multiplied with a fixed polynomial [55].

2.1.1.4 KeyExpansion and AddRoundKey

The AES will perform Addroundkey at the beginning. In other words, the plaintext will do bitwise xor with the original key. In the round transformation, Addroundkey will apply a key expansion process to generate the round keys.

The AES key expansion algorithm inputs a four-word (16-byte) key and produces a linear array of 44 words (176 bytes). That key is sufficient to provide a four-word round key for the initial AddRoundKey stage. The expansion key algorithm is resistant to known cipher analytic attacks. The key will be divided into four 32bits words. Each added word w[i] depends on the immediately preceding words. The word whose position is a multiple of 4, a g function is used. The other words will perform XOR operations. Figure 2.6 describes the generation of the key and the g function. The key expansion can be done beforehand to enhance the operations, and Rijndael can be specified in terms of this expanded key.

Moreover, AES performance in software is low unless performance-optimized implementations are used like T-tables (precomputed tables and require memory). The large key size makes it hard for an attacker to use a brute force attack. However, those cryptographic devices still leak some side-channel information that the attacker can obtain to decipher the secret.

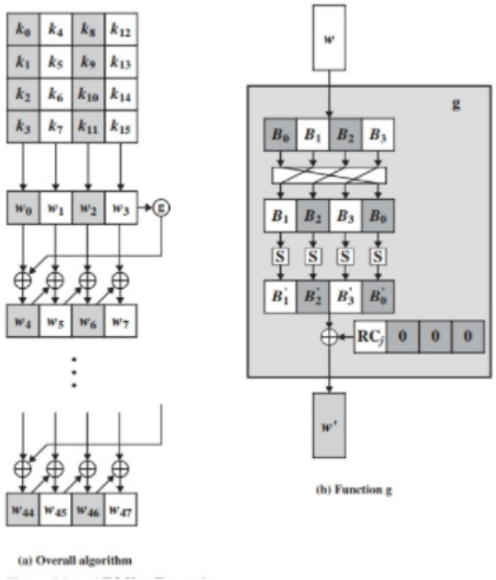

Figure 2.6: The KeyExpansion involves the XOR operations and g function [18].

2.2 Power Analysis Attacks

As has been mentioned above, AES is the standard algorithm to protect the device from attack. In recent years, several attack methods have stood out. Their primary goal is to reveal the key inside the device. The implementation measures are various. Based on the different categories standard, the attack measures can have several different categories. The most straightforward criterion is the Passive and Active. As the name suggests, the Passive way reveals the secret key by measuring the physical properties of the cryptographic devices. The device usually leaks execution time information, power consumption information, and memory cache information when the device works. When the attacker performs an Active attack, they typically put some special input data and manipulate the environment to make it unusual. The abnormal information can reveal the secret key. This thesis aim at power analysis attack. Due to the easy implementation methods, the power attack analysis is prevalent and powerful to those standard security algorithms. It brings a threat to many cryptographic devices. From the reliability view and security side, it is essential to give more attention to the power consumption at the hardware register level.

The instant power consumption of a device depends on two factors which are data-dependency and operation-dependency [37]. The digital circuit consists of many CMOS inverters, which consist of transistors. The power consumption of an inverter consists of two parts which are the static power consumption P_{stat} and dynamic power consumption P_{dyn}.P_{stat} is typically very low. However, with the growth of the modern process technologies, P_{stat} increases significantly. P_{dyn} is still the dominant part of the power consumption. The P_{dyn} occurs when the output signal switches (e.g. 0 - 1, 1 - 0). The output capacitance in a circuit usually needs to be charged or discharged

Power Analysis Attack List	
Non-profiled Attacks	Profiled Attacks
Simple Power Analysis (SPA) Differential Power Analysis (DPA) Correlation Power Analysis (CPA) Higher Order Differential Power Analysis (HODPA)	Template Attack Deep Learning based Template Attack

Table 2.1: Power Analysis Attacks List.

according to the change of input value when the switch occurs. It has a great impact on power. The short-circuit current caused by the instant short of PMOS and NMOS also provides large power consumption. The power analysis attacks track the power during execution. After that, they perform mathematical/statistical analysis with the traces to reveal the key. The attacker usually uses the special leakage model, which correlates with the secret key bit under operation to get the key.

In addition to the passive and active classification method, the profiled and Non-profiled criteria can also divide power analysis attacks into two parts. The profiled attacks are considered the most potent because the attacker needs to have a clone of the device and input particular plain text and secret keys to create a template. In that particular case, the attacker can reveal the secret key with the template. In comparison, the non-profiled attacks need the attackers to know the cryptographic algorithm and not the devices. The substantive difference between profiled attacks and non-profiled attacks is that the profiled attacks can have access to the device. However, the attackers can not always perform profiled attacks in reality because they can not always have control of those cryptographic devices. On the other side, the non-profiled attacks can still threaten those devices even the attacker can not get the device. It is needless to say that a good understanding of those various attack methods can help security engineers to design a resistant device. Thus, an explanation of those attack methods and summary has an outstanding contribution to this research. Table 2.1 shows the popular Power Analysis Attack methods and each one will be explained in the following subsections.

2.2.1 Non-Profiled Attacks

For the non-profiled attacks, it is weaker than profiled attack from the assumption. Because attackers can not access the device and only have plaintext and ciphertext information, they need to obtain many power traces and use statistical analysis to reveal the key. As the Table 2.1 shows, non-profiled attacks has SPA, DPA, CPA, and HODPA. With the countermeasure development, traditional attacks methods like DPA and CPA can not succeed in recovering the key. So HODPA comes out, which is the method to deal with some protected devices.

2.2.1.1 Simple Power Analysis

When attackers perform SPA, the voltage drop and up showed in the power traces leak the information to the attackers. However, only using SPA can not reveal the key. It can help to locate the interesting point in the traces. From Figure 2.7, the attackers can infer that the device starts to do nine AES encryption round at 0.3ms. Furthermore, at 4.1ms, the last round is performed. Because the final round does without Mixcolumn processes, the 4ms traces becomes different from previous power traces. However, it is impossible to reveal the AES key only from one power trace.

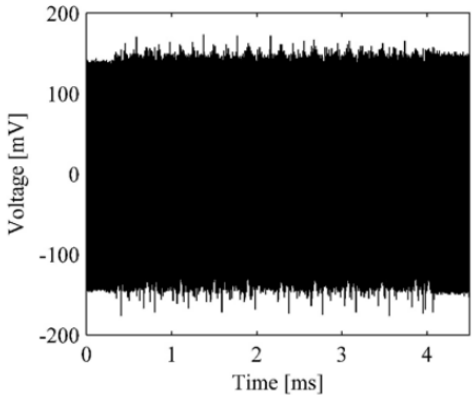

Figure 2.7: Power consumption of the AES encryption[37].

However, the SPA can still play a role in some implementations. For example, Figure 2.8 shows the unprotected RSA implementation. When multiply occurs, the power trace leak '1' information outside; meanwhile, the square operation leaks '0' information. Once the attackers get the whole power trace of RSA implementation, they can get the final key.

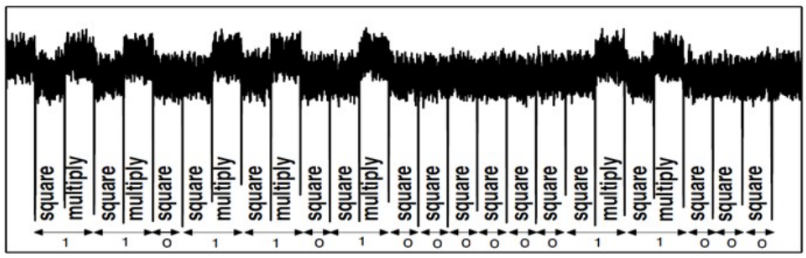

Figure 2.8: square-and-multiply RSA implementation[37].

The attacks which only use one or few power traces to recover the key are treated as SPA [37]. Thus, the manual effort compared to other methods is larger. Moreover, the lousy signal-to-noise ratio makes SPA challenging to use.

2.2.1.2 Classical 1-bit Differential Power Analysis

The power traces in Figure 2.9 contain the information of instruction sequence and have also been affected by the manipulated data values [32]. However, the measurement error and noise sometimes conceal the data power variations. So some attacks aimed at revealing the data power information come out. For example, DPA involves dealing with many power traces, and hypothesis key statistical analysis can recover the secret key. The DPA attacks have seven steps. At first, the attackers need to collect many power traces with varying inputs. And then select sensitive intermediate values like sbox result in AES. As sbox is commonly used and the attackers can know sbox. Then, for each possible value of K, compute the hypothesis result like $S(p \oplus k)$. They are choosing one bit in the hypothesis result like MSB or LSB. According to the value of that bit, partition the power traces into two groups '1' '0' , compute the mean and get the difference of those two means. Because AES 8 bits key has 256 cases, the attackers can get 256 differential power traces graphs. One of them has the peak power traces is the correct key for that bit. The essential idea behind it is that if a wrong key is implemented, the intermediate data is uncorrelated to the actual power traces by the device. Moreover, in theory, the mean difference would approach zero, causing the difference in power traces graphs to flat. As showed in the Difference plots for the key guesses, the attacker can determine the 119 is one of the key. Thus, this model can use 1-bit to recover one byte key. This processes needs to be repeated several times to get the full key. To be more specific about DPA, following list explains more details about DPA.

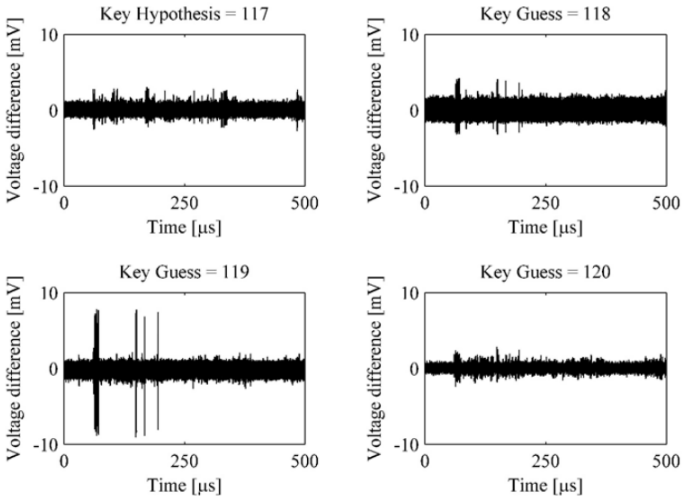

Figure 2.9: Difference plots for the key guesses[37].

1. **Select point of interest** In AES, the first operation in each round is SubByte. As mentioned in the AES part, the first inputs of the first SubByte are plaintext and original key. The SubByte operation is a byte level and non-linear function. This makes it suitable for attacker to perform DPA.

2. **Measure power traces** The attacker needs to collect the random plaintexts power traces with a constant key. The amount of traces may vary from different algorithms. In Equation 2.1 and Equation 2.2, the P is the recorded plaintexts, and the T is the corresponding power traces. Moreover, the n represents the measurement point, and the d represents the number of random plaintexts.

$$P = \begin{bmatrix} p_0 & ... & p_d \end{bmatrix} \tag{2.1}$$

$$T = \begin{bmatrix} t_{0,0} & ... & t_{0,n} \\ . & ... & . \\ . & ... & . \\ t_{d,0} & ... & t_{d,n} \end{bmatrix} \tag{2.2}$$

3. **Compute hypothetical S-box intermediate value** After collecting the power traces, the attackers can choose one byte in the plaintext and all the possible keys (e.g., AES key 256 cases) to perform SubByte $S(p \oplus k)$. The attackers will perform these steps many times and choose different bytes until all the 128 bits key are recovered. Equation 2.3 shows the hypothesis intermediate results.

$$S = \begin{bmatrix} S_{0,0} & ... & S_{0,255} \\ . & ... & . \\ . & ... & . \\ S_{d,0} & ... & S_{d,255} \end{bmatrix} \tag{2.3}$$

4. **Partition traces into two groups** In classical 1-bit DPA, the T power traces array can be divided into two groups based on one bit in the S intermediate value array. For example, the MSB bit of the first column in the S array(key 00 case). If MSB bit is 1, add this trace into group A in Equation 2.4. If MSB is 0, add this trace into group B in Equation 2.5.

$$A = \begin{bmatrix} t_{0,0} & ... & t_{0,n} \\ t_{2,0} & ... & t_{2,n} \\ t_{3,0} & ... & t_{3,n} \\ . & ... & . \\ t_{d,0} & ... & t_{d,n} \end{bmatrix} \tag{2.4}$$

$$B = \begin{bmatrix} t_{1,0} & ... & t_{1,n} \\ t_{4,0} & ... & t_{4,n} \\ t_{7,0} & ... & t_{7,n} \\ . & ... & . \\ t_{d-1,0} & ... & t_{d-1,n} \end{bmatrix} \tag{2.5}$$

5. **Compute average and two groups means difference** In this step, each group needs to average the power trace to get more robust power traces. And then do the subtraction of A and B to get the correlation of hypothetical key and real key. If the key is correct, a large peak will show in the different power traces. While

if the key is incorrect, the power traces will become flat, as the graph shows. Equation 2.6 shows the collection of each maximum value in each subkeys.

$$D = \begin{bmatrix} d_0 & d_1 & ... & d_{255} \end{bmatrix} \tag{2.6}$$

6. **Identify the correct key** After comparing all the 256 different difference power traces in D, The attacker can determine which hypothesis key is the correct key byte.

7. **Repeat the processes** In order to get the full byte key, the attacker need to repeat step 3 - 6 several times. However, the power traces is still the same, which can save time for the attacker.

This attack method tells the truth that the cryptographic device depends on intermediate value is still vulnerable to the attack. If the attackers find a suitable point of interest, they can perform the corresponding DPA on this device. This method still has its limitation, the 1-bit guess is time consuming. So a different more powerful method will be introduced in the next section.

2.2.1.3 Correlation Power analysis

As mentioned above, the power consumption is compromised of two part P_{stat} and P_{dyn}. Comparing to P_{stat},P_{dyn} takes over the dominant part. The data inside the device moves from 1 to 0 (or vice versa), the capacitance needs to be (dis)charged. Some leakage models can represent this kind of switch in the data, like the Hamming Distance and the Hamming Weight. The correlation power analysis (CPA) is the method to use those hypothetical power consumption and the measured power traces to do statistical analysis. The correlation coefficient can determine the linear relationships between data. So, the attacker can use the correlation coefficient in CPA statistical analysis to reveal the secret key [12]. In order to get full understanding of the CPA, the two generic leakage models details needs to be explained.

Hamming-Distance Model : The HD model is very suitable for data transmission and register simulation. The power consumption of the data bus is proportional to the HD model. The registers are triggered by the clock signal so that they only change in each clock cycle. However, there are many glitches inside combinational cells in combinational cells, which makes it difficult to simulate. Because the HD describes the transition of data, the calculation methods count how many transitions happen. For example, an 8-bit data "0100_0011" transfers to "0000_0000", three transitions are causing HD to 3. And the HD model assumes all the transmission contribute to the power consumption equally. In order to use HD model, the attacker usually needs to know the previous or following values in the algorithms. So they needs to know a little about the algorithm.

Hamming-Weight Model : The HW model counts the number of logic ones in the n-bit bus. It is usually used when the attacker has no idea about the algorithms. Because the HW model only describes the current state of the data and ignores the processed before and after data, it can not simulate the device's transition. However,

CMOS device's power consumption largely depends on data transitions. So this model is not very well to CMOS devices. But in some special cases, this kind model can be used. Furthermore, a easy way to calculate the HD is $HD = HW(v_0 \oplus v_1)$.

The following is the steps of a correlation coefficient based power analysis.

1. **Select point of interest** It is same that the AES sbox is still the interest point of CPA.

2. **Measure power traces** Like the DPA step 2, the CPA records the random plaintext data as a P array in Equation 2.7. For each plaintext during encryption, the same key is used to simulate actual conditions. And the collected power traces is the array T like Equation 2.8.

$$P = \begin{bmatrix} p_0 & ... & p_d \end{bmatrix} \tag{2.7}$$

$$T = \begin{bmatrix} t_{0,0} & ... & t_{0,n} \\ . & ... & . \\ . & ... & . \\ t_{d,0} & ... & t_{d,n} \end{bmatrix} \tag{2.8}$$

3. **Compute hypothetical sbox result** In AES, the small subkeys have 256 choices. To guess the correct result of the key, all 256 keys need to do SubByte with each plaintext to get a hypothetical sbox array S like Equation 2.9.

$$S = \begin{bmatrix} S_{0,0} & ... & S_{0,255} \\ . & ... & . \\ . & ... & . \\ S_{d,0} & ... & S_{d,255} \end{bmatrix} \tag{2.9}$$

4. **Compute hypothetical H by leakage model** The CPA step 2 and step 3 are the same as the DPA step 2 and step3. Unlike DPA step 4, dividing the power traces into two groups, the CPA uses the leakage model HD or HW to calculate hypothetical traces to simulate the corresponding power consumption. Equation 2.10 shows the hypothetical traces array.

$$H = \begin{bmatrix} h_{0,0} & ... & h_{0,255} \\ . & ... & . \\ . & ... & . \\ h_{d,0} & ... & h_{d,255} \end{bmatrix} \tag{2.10}$$

5. **Compute the Pearson's Correlation** One way to evaluate the correlation between the hypothetical power traces and measured power traces can help identify the correct key. Equation 2.11 named Pearson's Correlation is a helpful tool to find the relationship. Furthermore, i is the column from the H array, and j is the column from the T array. All the g values can build a G array like Equation 2.12, which is used to choose the best guess key. For each subkey i, find the highest

value of this row. Then, comparing the 256 maximum value, the location of this column is the best key. It correlated more with the measured traces than any other guess. Note that the value is the absolute value. The linear relationship is the most interesting point. The time information can omit either.

$$g_{i,j} = \frac{\sum_0^d (h_{k,i} - \mu_{h,i})(t_{k,j} - \mu_{t,j})}{\sqrt{\sum_0^d (h_{k,i} - \mu_{h,i})^2 \sum_0^d (t_{k,j} - \mu_{t,j})^2}} \tag{2.11}$$

$$G = \begin{bmatrix} g_{0,0} & \cdots & g_{255,n} \\ \cdot & \cdots & \cdot \\ \cdot & \cdots & \cdot \\ g_{0,0} & \cdots & g_{255,n} \end{bmatrix} \tag{2.12}$$

6. **Put together the best subkey together** And after above steps, the attackers only get one byte key. So in order to get the full key, 16 times processes needs to be performed.

Figure 2.10 shows a 7-bit subkey correlation coefficient graph, we can infer from the graph that 42 is the most related key.

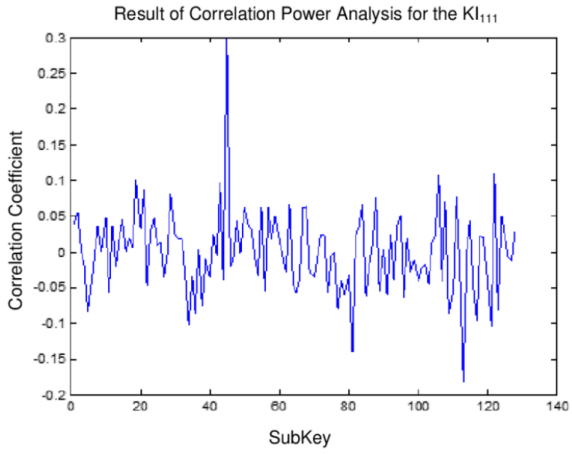

Figure 2.10: Correlation result for 7-bit subkey [38].

2.2.1.4 Higher order Differential Power Analysis

With the development of attack methods, there are also countermeasures occur to prevent the device from attacking. **Masking** and **Hiding** is the main trend countermeasures. They will be introduced in later. However, like higher-order DPA and template-based DPA can succeed attacking **Masking**. The DPA and CPA which is discussed above only use one intermediate value. So these power analysis attacks are treated as *first-order* power analysis attacks. If several intermediate value is used , the this is called *high-order* power analysis attacks.

The mask conteract attacks by mask the plaintext or the key at the begining. However, even the mask is used, the operation in the algorithms will keep the mask information. If the attacker uses more intermediate value, they can recover the masked key. In practical, *second-order* attack is sufficient to exploit the leakage . And the two intermediate value can be two value masked by the same mask or a masked value and corresponding mask [40].

2.2.2 Profiled Attacks

In this attack method, the attackers control the device and can build a good leakage model by using this device. Then this model, also called template, can be used to exploit the actual power traces secret information. The profiled attacks have two processes, which are profiling and extraction. The profiling step is offline and aims at capturing the behavior of the model. In the profiling step, the model may need thousands of power traces. However, the benefit is that this model can extract secret keys in the same device by a few power traces. The attacker can use the same template to attack the different user's keys in the same device. And based on the way of manipulating data, there are many different methods. For example, the template power analysis and deep learning based power analysis is now the advanced side channel attack.

2.2.2.1 Template Attacks

Before to describe the details of template attacks, it is necessary to understand what is a multivariate distribution and how to use it in the extraction processes. Take noise distribution as an example, the noise signal always exists to influence the measured result. It is like $x_{result} = x_{actual} + noise$. A Gaussian distribution can describe this random variable value. Figure 2.11 describe this more straightforward. F(x) is the probability density function and indicates the final result. The attacker can use this model in the extraction to identify the correct key. If the probability density is very small, the guessed key may be wrong. However, this only works for one measurement, the multivariate distribution can help to model several random values.

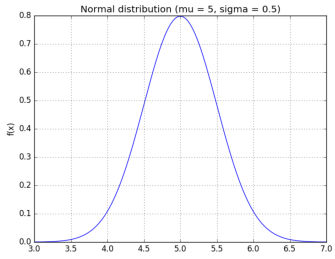

Figure 2.11: The Gaussian Distribtuion indicates the possibility density distribution [50].

For example, if there are two random values, X and Y. Calculating each Gaussian distribution are not helpful because they are independent. They do not have any correlation. So it is necessary to model them together (X, Y). Equation 2.13 is the

three random values model. Equation 2.14 shows the mean array of this model.

$$\sum = \begin{bmatrix} Var(x) & Cov(x,y) & Cov(x,z) \\ Cov(y,x) & Var(y) & Cov(y,z) \\ Cov(z,x) & Cov(z,y) & Var(z) \end{bmatrix} \tag{2.13}$$

$$\mu = \begin{bmatrix} \mu_x \\ \mu_y \\ \mu_z \end{bmatrix} \tag{2.14}$$

However, the probability density distribution will look like more complicated than just one variable value. $f(x) = \frac{1}{\sqrt{(2\pi)^k |\sum|}} e^{\frac{-(x-\mu)^T \sum^{-1} (x-\mu)}{2}}$. After the attacker gets the probability density distribution, they can put k point of one power trace into this model. If the f(x) is high, then this might be the correct key (vise versa).

1. Creating the template

   (a) **Select target operation** It is the same as first-order attacks, the higher-order attack also need to find a attack operation to implement attack. In AES, the sbox is usually protected by the mask. So the related sbox operation XOR is the interested point.

   (b) **Measure power traces** Because the attackers need to build a good distribution to model the power traces, the tens of thoudsands of power traces are needed. The measured power traces is array T like Equation 2.15.

$$T = \begin{bmatrix} t_{0,0} & ... & t_{0,n} \\ . & ... & . \\ . & ... & . \\ t_{d,0} & ... & t_{d,n} \end{bmatrix} \tag{2.15}$$

   (c) **Compute hamming weight and divide into 9 model and calculate the mean of those model** If the attacker builds a distribution model for every key, it will be considerable work. Another clever method to reduce the workload is to model the strong relation between a leakage model and target operation. The relationship between the Hamming weight and the Sbox can help the attacker reduce the model to only nine models. The H array like Equation 2.16 is the hamming weight value data. And then, divide the power traces into nine groups and calculate the according mean of each group as Equation 2.17.

$$H = \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ ... \\ h_d \end{bmatrix} \tag{2.16}$$

$$M = \begin{bmatrix} m_0 \\ m_1 \\ m_2 \\ ... \\ m_8 \end{bmatrix} \tag{2.17}$$

(d) **Find point of interest** In this case, the attackers do not need to create a model for every key or point of power traces. Because the key does not affect the whole power traces, and the high sample rate records some more points. In other words, the sample rate is faster than the operation rate, and some extra points can be omitted. One of the simplest method to find the point of interest(POI) is the sum of differences method like Equation 2.18.

$$D_i = \sum_{j=0}^{i} (M_i - M_j) \tag{2.18}$$

Based on Equation 2.18, a sum of differences graph can be got like Figure 2.12. As the graph shows, the attackers need to pick the highest value as the POI

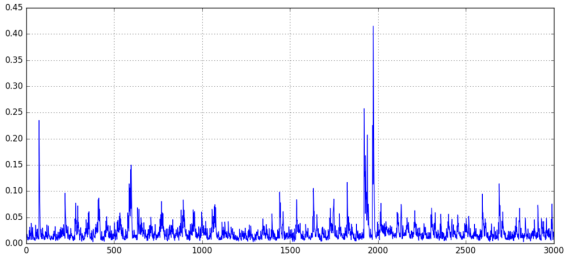

Figure 2.12: Points of Interest [50].

and then ignore the near N numbers because of the extra sample point reason. Then repeat the process again to get the enough points of interest.

(e) **Create the template** After the attacker already find the points of interest, then they can find a mean and covariance matrix for every key (256 in AES). Take key 01 as an example, there are T traces of key 01 and $t_{j,i}$ means the number j trace at POI i. Then the mean of POI can be calculated like $\mu_i = \frac{1}{d}\sum_{j=0}^{d} t_{j,POI}$. Equation 2.19 shows the array of those means.

$$\mu_i = \begin{bmatrix} \mu_0 \\ \mu_1 \\ \mu_2 \\ ... \\ \mu_i \end{bmatrix} \tag{2.19}$$

Based on the mean, calculating the variance can use $\upsilon_i = \frac{1}{d}\sum_{j=0}^{d}(t_{j,POI} - \mu_i)^2$. Then the covariance can be calculated like $c_{i,j} = \frac{1}{d}\sum_{k=0}^{d}(t_{k,i} - \mu_i)(t_{k,j} - \mu_j)$. At last, for every group, the attacker can get an array containing variance and

covariance to indicate the possibility distribution of the input power trace. Equation 2.20 shows the multiple random model coefficient array.

$$\sum = \begin{bmatrix} v_1 & c_{1,2} & c_{1,3} & \dots \\ c_{2,1} & v_2 & c_{2,3} & \dots \\ . & . & . & \\ . & . & . & \end{bmatrix} \tag{2.20}$$

1. Using the template
   (a) **Identifies the target operation** Like the other methods, the Sbox is the target operation.
   (b) **Measure power traces** Comparing to the other attack methods, this method may need lesser measured power traces.
   (c) **Creating POI array** After getting the power traces, the attackers only needs the point of interest power traces points. D traces of n points of interest power traces array can be collected like Equation 2.21.

$$A = \begin{bmatrix} a_{0,0} & \dots & a_{0,n} \\ . & \dots & . \\ . & \dots & . \\ a_{d,0} & \dots & a_{d,n} \end{bmatrix} \tag{2.21}$$

   (d) **Correlate the array with template** Then using the $f(x)$ function for every row in array A, the attackers can obtain the possibility distribution of the guessed key.
   (e) **Combining the results** One way to evaluate the $f(x)$ is to multiply them together. However, after multiplying those values, the result may be too large or small to fit. So an alternative way is to do a *log* calculation and then compare the different results to determine which key is the most relative.

2.2.3 Deep Learning-based Side Channel Attack

As mentioned above, the template attack usually needs pre-processing and finding the point of interest to reduce the noise influence, alignment, and dimension reduction [29]. So, the pre-processing procedure is essential and based on human manipulation. Moreover, some machine-learning-based side-channel attack also needs the pre-processing and POI. However, according to some studies, the Deep learning-based side-channel attack(DLSCA) may not need to do the pre-processing and POI steps, enhancing the possibilities of success.

The creating template steps are different from the classical template attack. After collecting the power traces, the attacker already knows the intermediate value after the operation, like the sbox result or hamming weight. So, the attackers can use the power traces and the intermediate value as input data and the label of a neural network to create an inference model. Furthermore, the property of Convolution Neural Network(CNN) is that it is good at image classification. Although the noise and

measurement methods impact data dramatically, the CNN can still have the ability to identify the correct result by advanced input data augmentation technique. Figure 2.13 shows the training process with CNN.

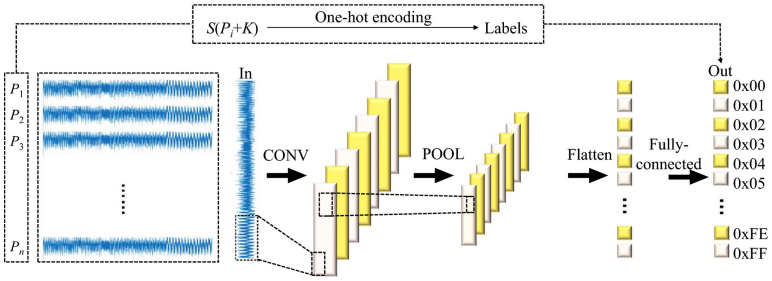

Figure 2.13: CNN procedure for intermediate value [29].

After getting the trained CNN model, the attacker can use it for unknown power traces to get the relative value as array A. Array A indicates the possible result of the power traces. Then, like steps in the CPA, the 256 hypothetical subkey leakage model array can be built. The attacker needs the hypothetical array B and the template array A to identify which result has the highest value. After that, the attacker can decide which key is the correct result.

All in all, the DLSCA can avoid the pre-processing and POI steps. The steps are more straightforward than the classical template attack. However, the main factor is the trained CNN model. If the input power traces is too random and have little to train, the attack can not guarantee success. So, the amount and the regularity of the input power traces are essential for this attack method.

2.3 Countermeasures

The attack methods above are all interested in attacking the S-box in the AES algorithms. Because the attackers use the S-box operation can get the intermediate value for DPA, CPA, and Template attack. The intermediate value can recover the relationship between the actual power traces and the correct key. So there are many countermeasures to change the power consumption in the device. At least to reduce the dependency of those power relationships. The recent countermeasures can be divided into two major classes, hide and mask. Hide aims to hide the power relationship by adding random power or consuming equal power every clock. However, hide sill uses the same intermediate value as the unprotected algorithms use. The mask mainly replaces the intermediate value and still gets the correct result. In that way, the attackers can not use their already known intermediate value to attack this protected algorithm.
Furthermore, the mask can be used either at the cell level or algorithms level. It is more popular than hiding in the scientific field and is widely discussed. Both of them are independent of the relationship between the power traces and the intermediate data. The big difference between hiding and mask is whether use the same intermediate data as the unprotected device. The following section will introduce more details of the hide and the mask countermeasures to understand resisting attacks methods better.

Table 2.2 shows the two mainstream countermeasures.

Side-Channel Attack Countermeasure List	
Hide	Mask

Table 2.2: Power Analysis Attacks Countermeasures List.

2.3.1 Hide

The purpose of hiding is to counteract the attack method by cutting the relationship between power consumption and data operations. The designer can design the device to consume the same power every clock or random power consumption. Although those two ways are too ideal for implementation, there are still approaches to get the desired result. One method to random or equal the consumption is to change the operations power characteristic directly. Thus, those proposals belong to the amplitude dimension approaches. The other one is called the time dimension approach, which changes the operation time during one execution. So if the operations of the algorithms are randomized during the execution, the power consumption also will become random. The following subsections will describe some hiding countermeasures from the time dimension and amplitude dimension.

2.3.1.1 Time dimension

The power trace needs to be pre-processed from the CPA and some machine-learning-based power analyses, and the POI is essential for the attacker to perform a successful attack. Moreover, the point of interest highly depends on the operations. If the algorithm's operation can run at a different time during each execution, the power traces will become more random [37]. Furthermore, the designer usually uses two methods inserting dummy operations and shuffling to change the composition of the power traces. The below is the details of those two approaches.

1. **Inserting dummy operations**
   The motivation of inserting dummy operations is to change the operation run time. The designer can implement this either on algorithms side or on hardware side. For example, some dummy operations is inside the C code to resist attacks. And the number of hummy operations differs from each other. The more random the number of dummy operations between algorithms operations, the more random the power traces. While in the hardware side, the designer can add random delay of FIFOs in pipeline architectures [34]. The random delay FIFOs can make the data stay random when the algorithms run. Thus, the execution time will change. Moreover, the throughput will decrease as the inserting dummy operations consumes more time.

2. **Shuffling**
   As some operations' sequence can be changed and the algorithms can still work correctly. In other words, the AES 16 SBOX substitution usually works in a fixed

order. If the order sequence can be shuffled every time the AES is executed, the random number of sequences is difficult to attack. Furthermore, the shuffle has less impact on throughput, like inserting dummy operations. However, the shuffle can only be used in some limited positions because some sequences in the algorithms can not be upset. So, the shuffling and inserting dummy operations are usually combined to protect the algorithms.

2.3.1.2 Amplitude dimension

In the time dimension, two methods to randomize the power consumption are described. In the amplitude dimension, the power characteristic can directly be modified to randomize the power and even balance the power in every operation. The noise and signal power has a significant influence on the real power. Furthermore, to increase the noise, the noise power will dominate the whole power. On another side, to decrease the operation power to a degree, the operation power will become equal and make the power trace flat.

1. **Increasing the noise**
   The straightforward way to increase the noise in the device is to add the noise generator module. The generator can then add extra noise into the device to distort the power. In the hardware aspect, the more independent operations run, the more noise generated. So a wider datapath hardware architecture is more difficult to attack. Moreover, in the [45], the storage elements are distributed, and the algorithms are changed based on the logistical datapath, which can help the device resist attack.

2. **Decreasing the signal power**
   The power analysis can reveal the slight differences in the power traces. So making all the operations consume equal power can enhance the security of the device. Moreover, this is not trivial. One straightforward way to make power equal is to use a filter. The filter can be used on the device to make the power constant. Because all the power is based on the CMOS cell on the device, a dedicated cell can be designed. If every cell can consume constant power, then the whole power trace will become flat.

2.3.2 Mask

Even if the device has a data dependency power consumption, the mask method can break the relationship because the mask does not use an intermediate value like the unprotected device. Moreover, the mask can be applied to the algorithms level and cell level. The basic idea of the mask is to use an attacker's unknown value and the original intermediate to create a new value to use in the algorithms. For example,$v_m = v \oplus m$ use m to do xor operation with original sbox input value. And sometimes the operation can be changed according to the purpose. The mask is usually used to the plaintext and the secret key. Thus, the encryption output is the masked result too. So, the algorithms need to convert the masked data into the original one. Nevertheless, the

attack can not perform this step, and the power traces they got is not like the normal one. Furthermore, the intermediate value needs to be masked all the time. In other words, the masked intermediate value may become the original one by accident. So, it is crucial to pick the mask and mask time carefully. And several different numbers of masks will decrease the performance.

As the power analysis mentioned, the power analysis can work because of the relationship between the intermediate data and the power consumption. So the mask methods try to protect the device by a mask the intermediate value, which directly destroys the relationship. Furthermore, because the masked intermediate value's power distribution does not depend on the original intermediate value, this can only resist first-order DPA. In order to resist higher-order DPA, the secret sharing scheme needs to be used. It means several masks are used on the intermediate value. It has been proved that n masks can resist n-th order DPA [13]. However, more masks are used leads to more memory resources and more computation time. Hence, to resist the higher-order DPA, the hiding and mask methods are combined.

Background of Spiking Neural Network

3

In order to understand the meaning of the spiking neural network(SNN), the background needs to be described. This chapter introduces the traditional artificial neural network and the characteristics of spiking neural networks. The different encoding methods have a significant influence on SNN. Three primary encoding methods are introduced. Furthermore, some hardware implementations of spiking neural networks are also described to guide the thesis.

3.1 Artificial Neural Network

Artificial Neural Network (ANN) like Figure 3.1 is used to simulate human brain intelligence. The idea was come out firstly in 1943 by Warren McCulloch and Walter Pitts. This model contains the inputs ,calculation and outputs. Those three different parts can imitate the information transmission in the brain. The input is like dendrite. The calculation function is like the cell itself. The output is like a synapse that transmits the spike to the next dendrite. At that time, the neuron can not learn by themselves. The weights are fixed numbers. Until 1949, Hebb introduced the learning rate, which lay the foundation for learning algorithms. In 1958, a simple two-layer neural network, called Perceptron, led research in neural networks. However, Minsky pointed out that the perceptron can only solve simple linear classification tasks. So until 1986, the neural network was in the winter period. Rumelhart and Hinton introduced the backpropagation (BP) algorithms which helped the neural network return to the research field. Recent research demonstrated that the ANN could be used in auto driving, computer vision, and voice recognition. Nowadays, the neural network has more layers and more neurons than before. So the calculation becomes more and more complex. A new neural network that works more like a real human brain called a spiking neural network, can solve this problem.

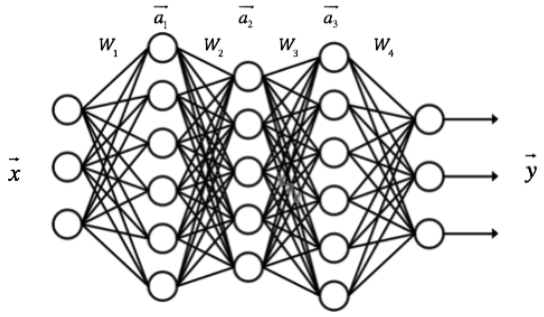

Figure 3.1: Artificial Neural Network Architecture[11].

3.2 Spiking Neural Network

The ANN has already lasted for fifty years. However, in the biological neuron model, the actual information transmission can be imitated by the voltage in the circuit. Some scientists introduced a new generation neural network, spiking neural network. It can diminish the distance between deep learning and neuroscience. The SNN uses the spikes to imitate the biological-inspired manner by discrete function. This section introduces some details of the spiking neural network. It can help the reader get a compact brief intuition of the spiking neural network [48].

3.2.1 Neuron Model

The neurons in the cell throughout the nervous system transmit information. As shown in Figure 3.2, the neuron consists of three major parts. It transmits information by electricity. Before the neuron sends the information to the next motor neuron, it needs to receive a chemical message from the former neuron on its dendrites around the cell. The cell body then converts this chemical message into an electrical message through the neuron, and the electrical message is called the action potential. This neuron impulse then travels through the high-speed channel axon, which is covered with a myelin sheath. At the axon terminal, the electrical one is collected and converted to the chemical message. The axon terminal can then convert the chemical message to the next neuron. The connecting point between different neurons is called the synapse. The analog neural network imitates the function in the neuron. However, they use the fixed number rather than the electrical pulse to transmit information and do not consider the time information. Inspired by the activities of neurons, some neuron models to imitate the actual activity of neurons are invented, like Hodgkin–Huxley model, the integrate-and-fire model, and the Izhikevich model. Many of them have already been demonstrated can be used in the implementation.

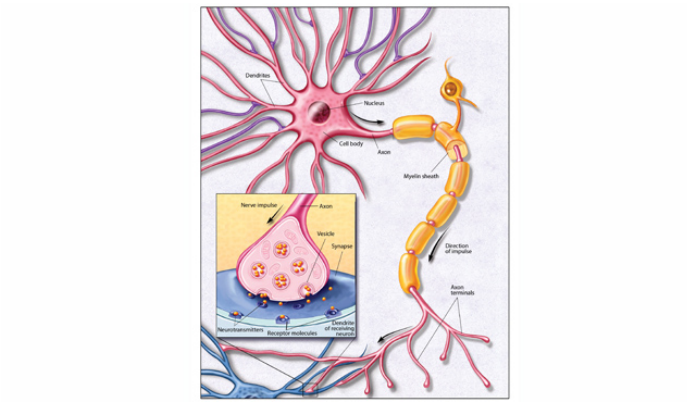

Figure 3.2: The Neuron in the Brain[51].

3.2.1.1 Hodgkin-Huxley Model

The Hodgkin-Huxley (H&H) model was firstly invented in 1952 [26]. It is a conductance-based model which describes the ionic mechanisms underlying the initiation and propagation of action potential in squid giant axon [26]. It is a mathematical function that consists of many non-linear differential functions. The basic theory can be explained by Figure 3.3. It treats every component as electrical elements. It has three different processes, voltage-gated ion channels, leak channels ,and pumps and exchangers. However, this model is hard to be implemented in the silicon compared to the other model. Its complexity constrains its usage range.

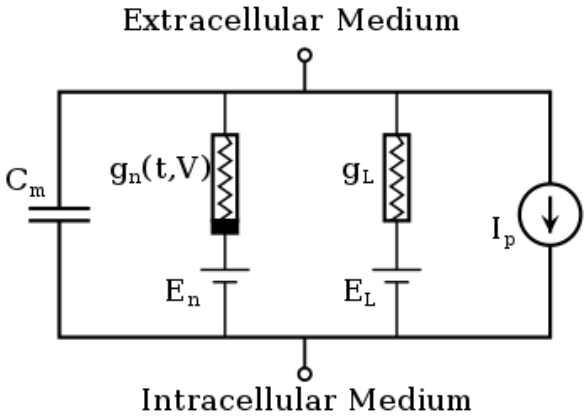

Figure 3.3: The Hodgkin-Huxley model[56].

3.2.1.2 Izhikevich Model

The Izhikevich model combines the biological plausibility of Hodgskin-Huxley dynamics and the computational efficiency of integrate-and-fire neurons [28]. It can be used in large scale neuron systems. However it still has the limitation that the spike shape is always the same. There are four parameters that determine the spiking and bursting behaviour of the neurons. Various parameters can result in various firing patterns. Equation 3.1 and Equation 3.2 explain the input voltage change. With the change of a,b,c,d four different parameters, the neuron can have several different types.

$$v = 0.04v^2 + 5v + 140 - v + I \tag{3.1}$$

$$u = a(bv - u) \tag{3.2}$$

With the condition Equation 3.3:

$$if\, v \geq 30mv, v \leftarrow c, u \leftarrow u + d \tag{3.3}$$

3.2.1.3 Integrate-and-Fire Model

The integrate-and-fire (I&F) model is the widely used model in the spiking neural network. Because it is very simple compared to the other models, it was first born

in 1907 by Lapicque. Moreover, the mechanism can be described by Equation 3.4. When the input stimulus comes, the membrane potential increases over time. After the membrane potential exceeds the threshold, a new spike is generated, and the membrane potential is set to reset value.

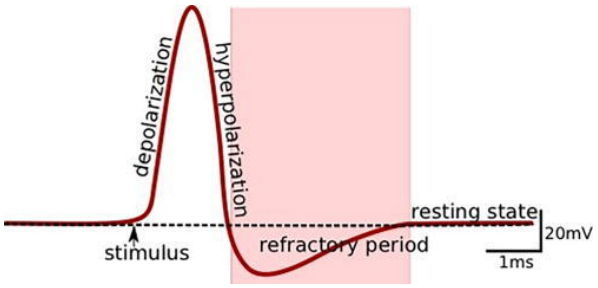

Figure 3.4: The action potential the fire period, refractory period , and the reset state [46].

However, this model still has its limitation. It can not describe the refractory period in the real neuron activity. The membrane potential will increase to infinite. The leaky integrate-and-fire(LIF) model can be used to solve this problem. Figure 3.4 shows the voltage change inside LIF model and Equation 3.4 explains its mechanism.

$$I(t) - \frac{V_m(t)}{R_m} = C_m \frac{dV_m(t)}{dt} \tag{3.4}$$

3.2.2 Spike Encoding

In the spiking neural network, the information can be encoded by many different ways. Different encoding methods give different impact on the neural network. There are three major popular encoding methods, rate coding , temporal coding ,and population coding. The following sections give a detail description of three coding methods. And the final result judgement criterion is determined by the frequency rate or the arrive time of spikes, which is different from the criterion in the ANN. Figure 3.5 shows the firing coding general SNN structure.

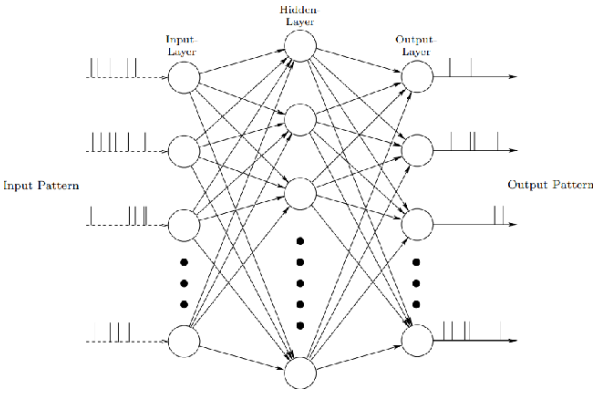

Figure 3.5: General Structure of SNN[27].

3.2.2.1 Rate Coding

Rate coding is a widely used coding method in the spiking neural network. The principle of the rate coding is that the neurons corresponding to inputs with the highest intensities fire more frequently. In other words, the scheme treats the input pixel as a firing rate and converts it into Poisson spike train with firing rate [24]. The firing rates can be used to describe the properties of sensory or cortical neurons. There are two calculation methods for calculating firing rates. Firstly, the spike-count rate, also called temporal average, can be obtained by counting the spikes number during the trial time and divided by the trial time. The time window period needs to be set corresponding to the input stimulus. The advantage of spike-count rate is that it can be obtained by only one trial but at the cost of losing all temporal resolution in neuron response during this trial [57]. The second scheme is the time-dependent firing rate that averages the number of spikes during a short interval, divided by the interval. The same input sequence needs to be repeated several times. The time-dependent firing rate is an effective way to evaluate neuron activity. However, it can not be used to imitate the neuron in the brain because the neuron in the brain can not wait for the repeated input sequence. Nevertheless, the neuron in the experiment can use this method to get a more precise firing rate.

3.2.2.2 Temporal Coding

The temporal coding like time-to-first coding is that the neuron fire first with the highest intensity. In other words, the larger the input pixel is, the more information is, the earlier it fires a spike.

3.2.2.3 Population Coding

Population coding scheme is one way to describe method by the connection activity between neurons. For example, the spike times of several input neurons are used to represent the input data. Figure 3.6 shows the three different encoding method.

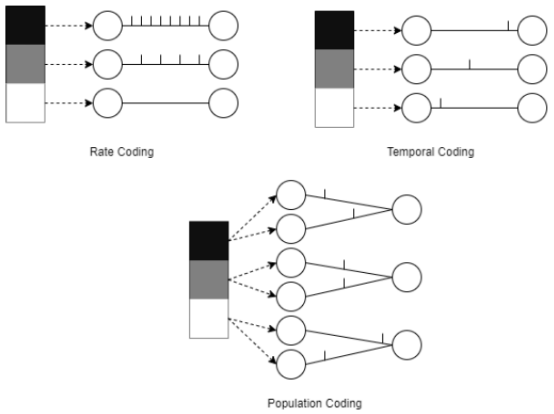

Figure 3.6: Three Different Encoding Schemes [43].

3.2.3 SNN Learning Method

One of the most used learning algorithms is spike-time-dependency plasticity(STDP). The STDP changes the weight in proportion to the pre-synaptic firing rate and the temporal rate of the changes of the activity on the post-synaptic side [5].

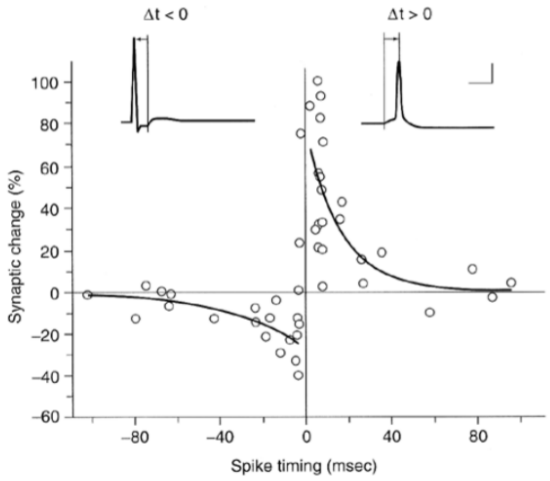

Figure 3.7: Biological observation of STDP weight change [5].

In Figure 3.7, the horizontal axis represents the time differences between neurons. Moreover, the vertical axis represents the change of weight. It can be concluded from Figure 3.7 that the positive changes occur when the pre-synaptic spike before the post-synaptic spike, and the closer the time difference is, the larger the change is. The negative changes occur on the opposite when the pre-synaptic spike after the post-synaptic spike. The changes diminish to zero when the time differences become larger. And the temporal vicinity of the post-synaptic spike changes the weight significantly.

3.2.4 SNN Hardware Implementation

Greatly boosts deep learning spiking neural networks both in the scientific and engineering fields [49]. According to the implementation methods, the SNN hardware platforms can be categorized as multi-processor [30], FPGA [41], or analog/mixed-signal [7]. However, most of the platforms using analog circuits are mostly based on the fact that complex mechanisms can be implemented with transistors in the subthreshold regime compared to digital circuits. Analog implementation can faithfully create large-scale neuromorphic circuits and complex bio-plausible tasks. However, digital implementation can also realize bio-plausible mechanisms. Many different bio-plausible models are also created to fit hardware implementation. Machine learning may choose simple IF and LIF models to achieve high accuracy and power-efficient hardware implementation. The advantage of digital implementation of spiking neural networks is that a complex multiplier operation can be reduced to addition and compared to MAC operators in ANN. Thus, the digital SNN can only use adders to replace MAC, reducing operations complexity and improving power efficiency.

Several groups already create some different spiking neural network hardware. For example, the Neurogrid [6] designed and built by Stanford University is a neuromorphic system for simulating large-scale neural models in real time [6]. The SpiNNaker, created by the University of Manchester, consists of several ARM processors. The ARM architecture provides the ability of parallel computing with custom digital multi-core chips. Up to 16,000 neurons and 8 million synapses can be simulated in a single chip [1]. Moreover, the TrueNorth created by IBM is a processor that contains 5.4 billion transistors. The purpose of the processor is pattern recognition. The creative method behind TrueNorth is neuromorphic computing. Furthermore, because the asynchronous clock principle can speed up the data transmission and consume less power, the Dynamic Neuromorphic Asynchronous Processor(DYNAP) [47] combines inhomogeneous sub-threshold analog circuits, and fast programmable digital circuits were created. The designer can implement the spike-based neural processing architectures on this chip. It solves the von Neumann bottleneck problem. The real-time multiplexed communication of spiking events can also be harnessed. The intel also created a processor named Loihi with many cores by the intel's 14-nm process. It has a unique programmable microcode learning engine for on-chip training [42]. And the adaptive self-training event-driven SNN can be implemented effectively on the chip. The European human brain project also created a chip named BrainScaleS in the 180 nm process. And it can not use pre-programmed code but evolves with the real electronic circuits.

Selecting an implementation method is a design choice that needs to follow a design rule as our design replaces the S-box in AES, which is an embedded design and is not a large-scale neuromorphic design. Our task is like a classification task in machine learning. SNN, compared to ANN, is more hardware-friendly and thus attracts researchers and engineers. Hardware AES is usually implemented on portable devices. Thus SNN is suitable for hardware AES implementation. When in the application-specific domain, ASIC or FPGA implementation of SNN is proved to improve hardware efficiency ann get more than 97% accuracy for MNIST dataset classification in just hundreds of nanojoules. Table 3.1 shows the state-of-the-art SNN hardware digital implementation.

Items	Park ISSCC'19[44].	Chen JSSC'18[14].	Yin'17[59].	Chuang ACM'20[16].
Types	Digital	Digital	Digital	Digital
Technology	65nm(simulated)	10nm	28nm(simulated)	90nm
Inference Energy(uJ/inf.)	0.24	1.70	0.286	0.59
MNIST Accuracy(%)	97.83	97.9	96.86	98.01
Chip size(mmˆ2)	1.87	1.72	1.65	0.2

Table 3.1: State of the art digital SNN implementation.

Different application has employed power/accuracy tradeoff technics. Although ANN hardware implementation has a similar power consumption, it costs almost 3x larger area compared to other SNN hardware accelerators. And in some implementations employ energy/accuracy trade-off technics, which would improve power efficiency in losing accuracy. It demonstrates that a digital implementation of SNN can be used in real-life applications.No matter what kind of design methodology is adopted. People

need to describe the complex electronic system in the SOC era to select a suitable system architecture, divide the software, and divide the software-hardware, and simulate algorithms. To measure the security of the hardware, performing attacks on the back end is too late. To better understand the design, establishing a chip model is an effective way to make chip architecture decisions early.

Spiking Neural Network Simulator Implementation

4

In this section, the proposed spiking neural network needs to be simulated by the simulator. However, many different simulators can simulate the spiking neural network. It is essential to find a simulator that can get the hardware friendly values. So in the following sections, the characteristics of the snn simulators are described. And the details of the implemented snn are described.

4.1 Spiking Neural Network Software Implementation

Machine Learning(ML) is the best alternative method to replace conventional human coding for AI. Significantly, the deep learning methods can do classification tasks, image recognition, and speech recognition. So a helpful simulator system for deep learning is needed. Furthermore, in some unique aspects like the finance field, much data needs to be processed fast and find the best point to invest in. By the way, the IT engineers need to achieve a reasonable inference as soon as possible. The learning time for a new module can not be very long. Otherwise, it can not capture the great point. Nowadays, based on the effort of many researchers, many advanced systems play an essential role in the industry field and research field. Many systems choose to use python to implement the module.

Choosing a python module has many advantages. First, a great library ecosystem can help designers save build time for a neural network. A library contains the pre-written class and functions to be chosen directly. By doing this, the designer does not need to code the function by themselves. Second, Python is an easy code language to get familiar with. The low entry barrier can help the designer to pick up the module more quickly. Furthermore, the Python is very flexible for the designer to use. The designer can choose a comfortable style to code the neural network. However, the execution speed of Python compared to C/C++ is slow because everything in Python is an object which costs more memory access. In C/C++, the pointer directly points to the first data of an array. In Python, the pointer points to the buffer of other pointers, which makes for inefficient memory access [36]. However, in this thesis, this is not a big issue because of the small neural network structure.

The main trends library for the artificial neural network is Pytorch and Tensorflow. Google designed Tensorflow, and the Pytorch has similar functionality. Compared to the Numpy library, Pytorch and Tensorflow have the advantage of building the GPU module. The Numpy can only be used on the CPU. Getting the benefit from the modern GPU computation power, the neural network can be trained faster. Furthermore, Tensorflow and Pytorch have their strengths. The PyTorch is more researcher-friendly than Tensorflow, because the extended new classes and functions are easy to add to the library. Moreover, the advanced research paper has many Pytorch based modules.

In the computational graph aspects, the Pytorch can define graph on the go, which is helpful for the variable-length inputs in RNN [36]. One more powerful computational device named TPU can also be used for neural network training. However, the TPU is only supported by Tensorflow. Furthermore, many companies prefer Tensorflow to the Pytorch for production purposes.

As for SNN, some laboratories also create a simulator to simulate complex human brain activities. For instance, Genesis, NEURON, Brian, Brian2, NEST, and Bindsnet are commonly used. Some simulators are based on the Pytorch library like Bindsnet. Some are created from the bottom. They all can train the spiking neural network and gets a reasonable inference. However, many of the libraries are built by one more language. As has been mentioned above, some languages have a faster execution time at the lower level structure. And in some high levels, Python or Matlab are used to build the structure faster. Thus, when a tailored structure needs to be implemented, the designer needs to change the structure to two different languages, both high level and low level. This is not flexible for the designer. So the Bindsnet based on the Pytorch has all the Pytorch advantages. Moreover, the Bindsnet can be connected to the unique hardware and get a fast inference.

4.2 Bindsnet implementation

It is owing to the torch.Tensor object in the Pytorch, the Bindsnet, can be implemented on the CPU and GPU. And the torchvision.datasets have been added into the library for evaluating the ability to spike neural networks for image recognition tasks. The Bindsnet can be used for the unsupervised learning task and supervised-learning task. And the big difference between the spiking neural node and the artificial neural node is that the spiking node integrates the input during the time. The SNN considers time as the third dimension, and it has the advantage of dealing with time-related tasks.

The neuron's characteristic is mainly dependent on their weights. In other words, the synapses connected to the neuron have a contribution to the neuron activity. And the weight needs to be trained based on the input spikes and output labels. So, a suitable learning mechanism is important for a spiking neural network. The Bindsnet library provides the bindsnet.learning module to update the weight. It contains popular learning methods, including Hebbian learning, a variant of spike-timing-dependent plasticity (STDP), and less well-known methods such as reward-modulated STDP (MSTDP) [25].

Different from the ANN, the neuron of SNN has some different types that can construct the NN. So the **bindsnet.network.node** can provide several neural nodes to let the designer choose. Moreover, the **bindsnet.network.topology** can give the connection types in the networks. Different connection types have different performances in various situations. Furthermore, the **t** variance has a significant impact on the performance. If the variance is considerable, the simulation time will become very long, but better accuracy, and vice versa. The designers determine this trade-off depending on their design.

4.2.1 ANN

The initial idea to replace the sbox with a neural network is from Snet [54]. In that thesis, they use a multiple target neural network in software to make it resistant to side-channel attacks. And they collected the traces on the MCU. The output of that ANN has eight different neuron nodes. Each node can represent one bit of the result of an eight bits sbox. The mechanism is that each bit has different integer results like +456, -134. Then, based on the integer sign, it generates 1 when the sign is positive (vice versa). A multiple-target neural network can save many memory resources in the hardware. Furthermore, they also optimize the neural network by removing the bias, quantizing the weight, and constraining the weight range. On the other hand, the output result of sbox has 256 different. The neural network can also be constructed with 256 output targets. In this thesis, two different versions of ANN are built to get a deep understanding of the S-net project. Those two neural networks are built with Pytorch library. Table 4.1 shows the comparison of two types ANN.

Artificial Neural Network		
types	structure	accuracy
Multiple targets	8-80-8	100%
Single target	8-96-256	100%

Table 4.1: Sbox Artificial Neural Network.

Thanks to the flexibility of the Pytorch library, the structure of the neural network can be changed many times until the best structure. The network is a class in the Pytorch containing characteristics of the layer and the activation functions. The chosen activation function is based on the tasks. In this thesis, the Relu function is chosen. The best advantage of Relu functions is that it does not activate all the neurons simultaneously. Furthermore, the dataset and data loader library let the designer build their own data library for their specific task, which is very helpful for different scientific researches. After building the specific training datasets, the neural network can train by themselves with the learning algorithms. The Pytorch provides the back-propagation methods. With the help of the loss function and the optimizer, the weights and bias in the neural network will be updated automatically. The difference between the multiple targets and the single target is the loss function, also called criterion. In the multiple targets, the *BCEWithLogitsloss()* function can help to get the multiple loss together and train the neural network. The single targets neural network uses the normal loss functions to do the training process. Finally, both of them can reach 100% accuracy, which can replace the sbox with artificial neural network totally. The weights are float64 now.

In order to make them more suitable for hardware implementation, quantization like fixed-point methods can help convert the weights to fewer bits with a bit of accuracy loss. In the Pytorch library, the *quantization* library can help convert the weights from float64 to integer or float32. Three different quantization methods can choose whether to quantize weight and activation. The straightforward way is dynamic quantization which only quantizes the weight with activation read/write float. The static quantization, also called post-training quantization, is the second one to quantize both the

weights and activation function after training. The quantization aware training can quantize the weight and activation functions during training. Then after getting the integer weight result, the hardware can be implemented with fewer resources.

4.2.2 ANN to SNN conversion

The ANN is so popular that some researchers want to convert ANN directly to SNN. The bridge to convert the ANN to SNN is essential. Fortunately, there is a strong correlation between the Relu functions and the firing rate in the SNN. By using Relu function and removing bias, the problem of negative representation of values can be solved. The Relu function is an estimater of IF neuron's firing rate. And the output of the Relu function is proportional to the spikes number of IF neurons. Based on this, the ANN to SNN conversion can be possible. Furthermore, the weight normalization can decrease the weight that the Relu function does not overestimate. So based on the effort of [20], the previous ANNs can be converted to SNN. But, the Bindsnet SNN is fire rate based SNN. The SNN's accuracy is determined by the highest fire rate. Thus, the multiple targets ANN can not be converted to the firing rated SNN. Because the output neurons fire the spikes can not indicate it has the highest fire rate. So, the single target ANN can be converted to the SNN with the help of Bindsnet.

As ANN is built on Pytorch, so a pytorch based SNN library to convert the ANN is more straightforward. Then, the Bindsnet library contains a conversion library which can convert the Pytorch based ANN directly. As has been mentioned above, the weight quantization method can reduce the resources usage in the hardware. However, the quantized neural network based on quantization library in Pytoch can not be directly converted to SNN in Bindsnet. The weights still need to be quantized. The Fxp library [2] is a python library for fractional fixed-point arithmetic and binary multiplication with Numpy. The weight stored in the neural network can be extracted as numpy array and converted to fixed point type by the Fxp library. And the bits of weights are determined by the designer. In order to choose the smallest bits, various fractional bits needs to be test. Figure 4.1 shows the different fractional bits effects on the ANN accuracy.

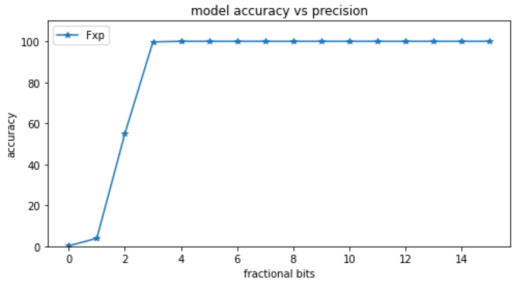

Figure 4.1: Different Fractional Bits Accuracy.

Then with the help of *ann_to_snn* functions, the ANN can be converted to the SNN. Besides, the Bindsnet provides the monitor in the library to monitor the spikes and network structure. Figure 4.2 describes the structure of SNN.

```
===============
NETWORK SUMMARY
===============
batch size:1
 ----------------------------------------
 Layer: 'Input' (trainable)
8 neurons (8,)
 ----------------------------------------
 Layer: '1' (trainable)
96 neurons [96]
 ----------------------------------------
 Layer: '4' (trainable)
256 neurons [256]
 ==========================
Total neurons: 360 (360 trainable)
Total synapses weights: 0 (0 trainable)

{('0', '1'): Connection(
  (source): Input()
  (target): IFNodes()
), ('3', '4'): Connection(
  (source): IFNodes()
  (target): IFNodes()
)}
```

Figure 4.2: SNN structure.

The quantized weight used in the SNN is the same as the quantized weight used in the ANN. However, the accuracy in those two neural networks is different because of the different mechanisms of those two neural networks. The accuracy of those two neural networks with different fractional bits are in Figure 4.3. It shows that the ANN reaches 100% at 3 bits and the SNN reaches 100% at 7 bits. Then 7bits weight is the best weight value in this Bindsnet SNN and can be used in the hardware SNN module.

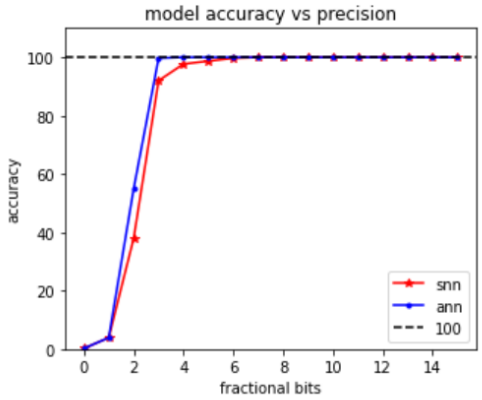

Figure 4.3: ANN and SNN Accuracy with different fractional bits.

However, the Bindsnet SNN is still a fire-rate-based neural network. The output neuron will generate several spikes during the time window. So, in the hardware, fewer spikes can reduce the power consumption, which is the target of spiking neural networks. In the [31], it introduces a one-spike neural network. The neurons in this network only fire one time to reduce the power consumption. The sources code is in [31], it uses a new supervised learning method which uses temporal coding rank-order-coding. Different from the fire rate coding, the firing order in the rank-order coding can carry the information. Thus, the first fire neuron in the output neurons determines the

class. This model is well suited for hardware implementation than the fire-rated one. In the following section, a new Pytorch-based SNN can be used to generate a one-spike neural network which is well suited for the hardware implementation.

4.3 SpikingJelly implementation

The SpikingJelly [21] like Bindsnet is also built based on the Pytorch. It is beneficial for the engineer who is familiar with Pytorch. Different from the bindsnet, it uses the surrogate gradient learning method to train the neural network. The SNN is like the RNN. The neuron's input is the voltage increment, and the output spikes are determined by comparing current voltage with the threshold. And then, the voltage is reset to zero or the reset value. The RNN can use differentiable gating functions like tanh function to train the neural network. However, the SNN activation function is not differentiable. Furthermore, SNN can not be trained by gradient descent and back-propagation. So, a differentiable function similar to the activation function can replace the function so that the SNN can use the back-propagation learning method. In SpikingJelly, the recommended surrogate gradient function is a sigmoid function similar to the firing spike mechanism. And it can be used to replace the gradient of the spiking function. So different neuron models like IF, LIF, and EIF models can be implemented based on the surrogate gradient function. The neuron model has basic variance like threshold voltage, step time, and reset voltage. The designer can change those values corresponding to their purpose.

After solving the learning problem, the spiking neuron can be embedded into the neural network as an activation function in the artificial neural network. Then the ANN will be converted to SNN. The training method in the SNN is to let the network run several step times and collect the number of output spikes during the step times. Moreover, the target neuron usually has the maximum number of spikes. The loss function in the SNN is also firing frequency. So the multiple target model can not be implemented either. Based on their research, the MSE optimizer has the best performance for training. The SNN is a memory-state neural network. So the whole network needs to be reset before each forward process. The SpikingJelly also provides the spikes and voltage monitor to monitor the network. Figure 4.4 can describe how the SNN works.

4.3.1 Sbox spiking neural network

The sbox spiking neural network is very similar to the one in the Bindsnet. However, the SpikingJelly one uses an 8-24-256 structure. After the experiment, the time window is chosen 7, which has the smallest multiple firing output. In other words, the target label output neuron can fire seven spikes during time steps. However, there is another neuron may also fire two or one spike during time steps. The correct result can be chosen by comparing those fire rates. Nevertheless, if the model is used in the hardware, a fire rate comparator needs to be designed, which will add extra effort. So the fewer multiple neurons fire, the better the neural network is. After weight quantization, only six results have two output neurons fire, which is not trivial. However in the bindsnet,

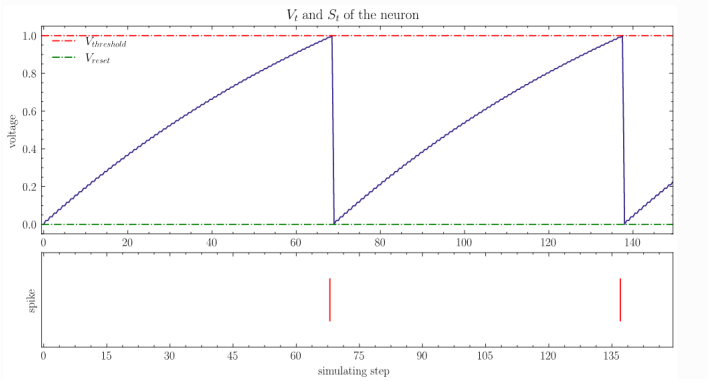

Figure 4.4: SNN Output Neuron Spikes and Voltage.

almost three or four output neurons will fire spikes. The output spikes can be explained more straightforward with Figure 4.5. Figure 4.5 indicates that the result neuron fires spike at every time step and other neuron do not fire spikes, which is suitable for hardware design. And the spiking neural network can reach 99% accuracy. The all zero inputs is a special case for spiking neural network. Because no spikes can transfer information between layers.

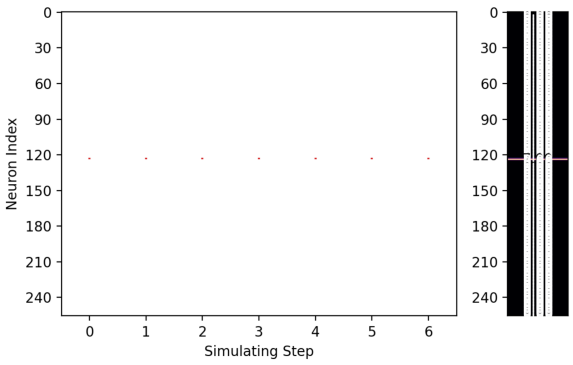

Figure 4.5: Sbox SNN Output Neuron Spikes.

After getting the trained SNN, the post weight quantization needs to be applied like Bindsnet SNN. The neural network can reach 100% after weight quantization except the special case. Furthermore, the all float64 bits can be converted to 8bits integer. The SNN can still keep 100% accuracy but have some multiple output neuron spikes. However, only 6 results have this situation which is not a trivial.

4.4 Comparison

There are many powerful spiking neural network simulators. However, the hardware SNN can maximize the advantage of SNN, a suitable simulator for hardware needs to be selected. Two different models based on two different libraries can handle the sbox task. Table 4.2 shows the characteristics of two SNN simulator. The SpikngJelly based SNN compared to bindsnet based SNN has several advantages. For instance, the SNN

structure is smaller, and the weights are 8 bits integers. Furthermore, the SpikingJelly based SNN can also transfer one spike in the neural network like S4NN [31]. Although two models use the same loss function, fire rate, the SpikingJelly one simultaneously has fewer multiple neuron fires. Those advantages can help reduce hardware resources, memory storage, and power consumption. So the inference hardware design will be based on the SpikingJelly model.

Spiking Neural Network		
Simulator	Bindsnet based	SpikingJelly based
Structure	8-96-256	8-24-256
Accuracy	100%	100%
Loss function	fire rate	fire rate
Spikes type	multiple spikes	one spike
Weight format	8bits integer + 7bits fractional bits	8bits integer

Table 4.2: Sbox Spiking Neural Network.

Spiking Neural Network Hardware Implementation

5

In this chapter, the proposed spiking neural network is implemented in the hardware. Each part in the RTL models is described. Moreover, the simulation results are posted to verify the functionality of each part.

5.1 IF Neuron model

To imitate the biological behavior in neural networks, an integrate-and-fire neuron model can be used to describe the synaptic inputs and the injected current that it absorbs. In other words, to achieve IF neuron and optimize the resources in hardware, some designs and adaptions need to be made.

5.1.1 Model adaptation

IF neuron model can be simply looked at as three main processes: integration of inputs, pulse input, and the triggers action potentials in terms of the threshold.

Initially, the binary number system is usually used in digital circuits to represent all types of information inside the computers. There are several types of number representation techniques, like fixed-point representation and float point representation. In the training phase, the spiking neural network model usually will use 16 bit or 32-bit float point weights and bias to train the network. However, the disadvantages of using float point numbers in hardware are that more storage and computation resources will be needed. The fixed point representation can solve this problem and still keep an acceptable accuracy. This representation has a fixed number of bits for the integer part and fractional part. The designer can decide the fixed number of bits. As in this thesis, transforming a 32bit float point to 8bit fixed-point number can help to save 4x resources.

Secondly, in order to build a phenomenological model of neuronal dynamics, the critical voltage for spike initiation by a formal threshold v. If the voltage $u(t)$ reaches v, the neuron fires a spike to the next synapses. This moment defines the firing time t [22]. Although the elementary laws from the theory of electricity can describe the mechanism, the continuous voltage calculation can not be achieved in a digital system. Because a digital circuit works with digital signals, where all values are discrete. Therefore action potentials are reduced to 'events' that happen at a precise moment in time [22]. A neuron cell will receive spikes from the front end, which is like a good insulator. Moreover, the short $I(t)$ carrying weight is injected into the neuron, it will charge the neuron. The neuron will work like a capacitor C with Equation 5.1. To implement the input addition in the digital circuit, just using adder and register can achieve the addition when the input spike comes in and keep the value until comparing with the

threshold.

$$I(t) = C\frac{\mathrm{d}V(t)}{\mathrm{d}t} \tag{5.1}$$

On the other hand, after the front neuron has already transferred all the addresses of the spikes. A neuron fire enables signal will let the membrane compare with the threshold.

5.1.2 Architecture

As explained above, a flexible modular has been proposed, including all the functions and control logic in the neuron.

Figure 5.1 is the digital block of IF neuron. This neuron model has six input ports, including the clock reset signal, three control signals, and one data signal. The input weight data is chosen from the ROM by the address of input spikes. The done_en signal indicates the end of the spikes address transmission, making the continuous-time into discrete-time. Once the done_en signal comes into the neuron, the neuron will determine whether this neuron can fire spike and reset the membrane to zero. Moreover, the Fired and Next signals can help to control the add processes in the neuron. The Next signal shows that the new address is different from the previous address. The Fired signal can make the neuron only fire one clock time and save the fire power. Only when Next is true and Fired is false, the add processes will work to add new weight into membrane potential.

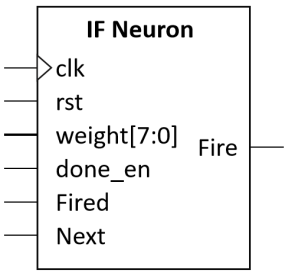

Figure 5.1: Digital block of a neuron.

Go deep into next level of the neuron, shown in Figure 5.2. The neuron consists of sequential and combinational logic. The reset signal will reset all the sequential signals to zero. There are two registers in this neuron. One will keep the add process data membrane, and the other will determine the fire signal. Two combinational blocks for the internal add signal and compare block with the threshold.

To be more specific, the whole working process will be explained below. Firstly, it will receive a reset signal to set all the signals to the original state. The weight data will change corresponding to the input address of the AER communication bus in the high-level Neuron Core automatically. When a new address comes in, the Next signal will let the neuron do add process. After all the new address weight has been added to the membrane potential, the done_en signal will let the neuron compare to determine

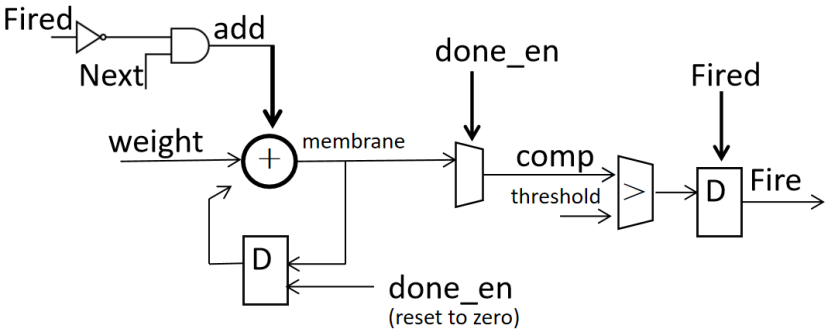

Figure 5.2: Digital implementation of the neuron equations.

whether to fire a spike and reset the membrane. If a new spike is fired, the Fired signal will let the fire signal be zero. Then a simple neuron period will end and start again when a new AER address comes in.

5.1.3 Simulation

Figure 5.3 shows the simulation waveform of the neuron model in the ModelSim. All the signals worked properly as the above workflow. The weight data will change every clock keeping the next signal activated. Several clock cycles later, the done_en signal is activated, and therefore in the next clock cycle, the neuron fires the spike. The Fired is stimulated by the fire spike and stops fire spike firing. Finally, the neuron finishes generating impulsed and resets every internal signal to zero.

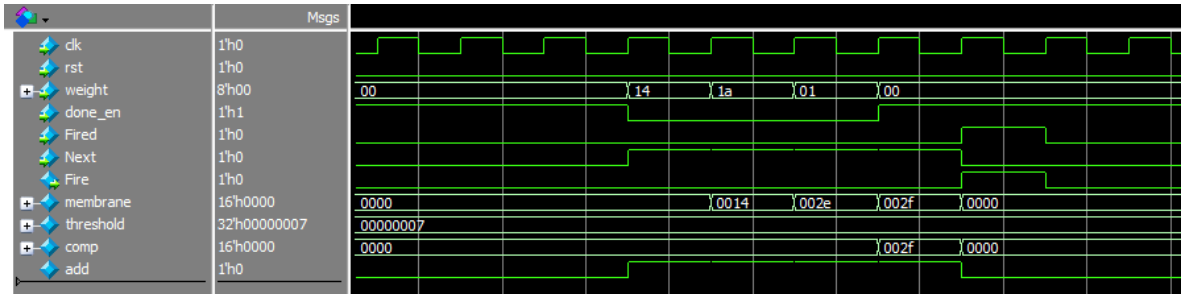

Figure 5.3: Neuron simulation.

5.2 AER protocol

The neuron core can read the address information from the Address-Event Representation(AER) bus. The AER protocol only uses one digital bus line to broadcast the complete address information serially. In order to achieve the asynchronous AER protocol, the encoder generates several corresponding binary addresses to the receiver chip, where a decoder decodes the binary address to spike pulse again. The four-phase handshake protocol can achieve the communication by an *acknowledge* signal and a *ready*

signal [9]. The AER design adds some changes to the [39] to make it more suitable for this thesis.

However, the AER is designed by an encoder, which is different from the asynchronous AER protocol to keep all designs in an asynchronous process. The synchronous AER can also read the spikes from neurons to send the neural address to the next layer through the AER bus.

Moreover, the encoder consists of a priority encoder, which can compress multiple binary inputs into a decimal outputs address. And the encoder can solve the address transmitting out-of-order problem. It transmits addresses one by one in every clock cycle.

5.2.1 Design and Architecture

Figure 5.4 shows the AER block and the input and output ports. The clk and rst port is the normal port for all the block. The *start full empty fifo_out_zero* and *cal_finished* ports are used to control the *control* block to control the work flow of the AER. The *din* and *aer* ports are the data communication ports for input and output respectively. The *load* and *fifo_out_en* ports are control inputs, which receive the control block output signals.

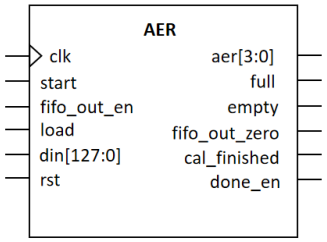

Figure 5.4: Digital block of AER.

The *start* port indicates the AER starts to work and reset the inside FIFO to zero. Then the control block sends the load signal, making the FIFO store the input spike vector. Then the FIFO generates a full signal to stimulate the control state from the load state to the calculation state. In the thesis case, the substitute(sbox) deals with 128 bits vectors consisting of 16 8-bit vectors. The 8-bit vector, for example, "0000_0101". A "1" means the input neuron has a spike pulse. There are two spikes in this vector. Then the AER can write 2 to the AER address bus. At the next clock cycle, the AER address bus would transmit 0. In order to keep order and avoid collapsing, the AER address data can be written to the bus every clock cycle. After finishing transmitting the AER address of one 8-bits vector, the cal_finished port generates one pulse to change the state of the control block. Then the control block would be a fire state which can give a fire enable signal to the neuron layer to fire the spike to the next layer. Then, the control block would generate one fifo_out_en pulse to let the AER deal with the next 8-bit vector. Finally, after all the 16 8-bit vectors have been processed, the FIFO produces an empty signal to stop the air and make the control state idle until

the next start signal.

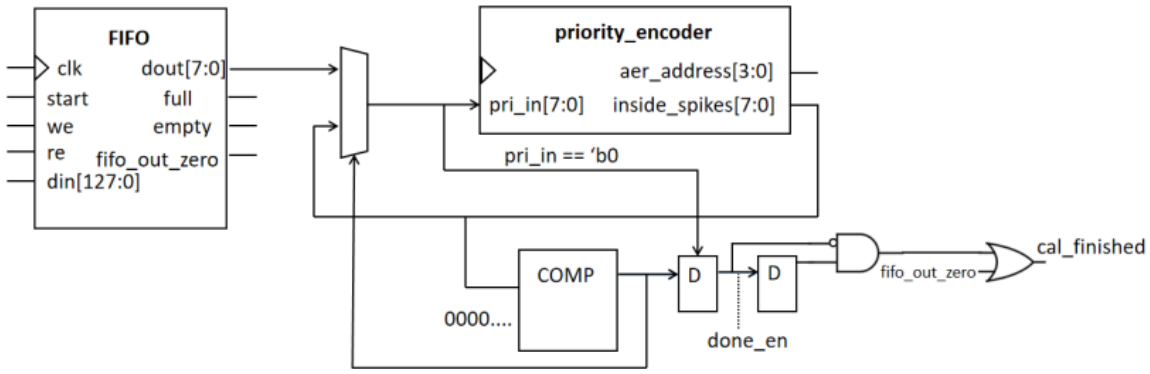

Figure 5.5: AER Schematic.

Clearly, the RTL level of the AER is shown in Figure 5.5. The area consists of three major parts, asynchronous FIFO, a priority_encoder to transfer spike information to address information, an inside control logic part. The user can work properly with the inside control logic and the external control signal.

Firstly, the FIFO loads all 16 8-bit vectors. Using the "0000_0101" as an example, if the inside_spikes and pri_in are zero, the FIFO gets the fifo_out_en signal from the control block, the priority_encoder can get a new 8-bit vector. In the next clock cycle, the priority_encoder translates the 8-bit vector from left to right. When detecting a "1" the corresponding position address, "2" is immediately transmitted to the user address bus. Then the priority_encoder generates another inside_spikes signal "0000_0001", which determines the next step.

Then the inside_spikes compares with an all-zero 8-bit vector to determine whether the translation process is done. If not, the inside_spikes comes back to the priority_encoder as pri_in to continue the address transmitting process. As in the above case, the encoder still needs to transmit another "0" address information. By the way, the fault address output of the encoder is "8", which is out of the address range of the input vector. If the inside_spikes is all zero, the pri_in gets a new fifo_out and generates a done_en signal. Besides, the fifo_out is determined by the report of FIFO, where a fifo_out_en signal would get from the control block when the control state is in CAL state. A posedge detector would generate a pulse for the cal_finished to change the control state from the CAL state to the FIRE state. After the fire state returns to CAL state, the FIFO reads a new vector for the pri_in. The FIFO works flow is like Figure 5.6, which is different from the design in [39].

In order to avoid writing data out of order, the design in the previous has the restriction for load data. It loads one 8 bit vector in every cycle and processes the translating meanwhile. However, in this thesis design loads all the 16 8-bit once, which can solve the load data problem. And adding some logic pulse to makes the clock more compact.

In the thesis, there are two layers, AER protocol is applied to both layers. The

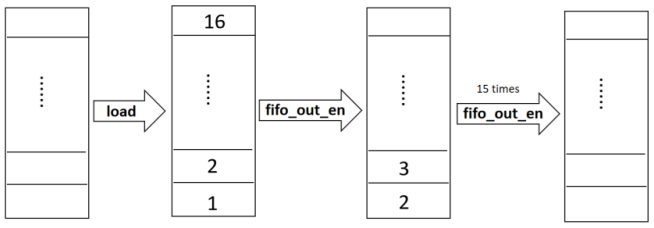

Figure 5.6: FIFO work flow.

second layer AER has a slight difference from the first layer. Because the first layer generates a 24 vector to the second layer, the priority encoder needs to be extended to 24 spikes detector. The address range extends to 0 23. Another difference is that the FIFO workflow would change. The FIFO loads all 128 bits of data in one clock cycle in the first layer and reads the data in the following clock cycles. Because the second layer needs more time to process the 24 bits vector, the following new 24 bits vector arrives at the second layer when the second layer deals with the old one. So, the second FIFO is capable of writing and reading the data separately. In other words, the FIFO can write data and read data simultaneously. Additionally, the empty signal indicates the end of the substitute process.

All "0000_0000" inputs have their unique transfer channel for the first AER and the second AER. In this thesis, the fifo_out_zero port is used to control the logic and transfer the corresponding zero information. If the system needs to load an all-zero vector into the FIFO, the control logic can use a load signal to help.

5.2.2 Simulations

The AER simulation in the Modelsim is shown in Figure 5.7 and Figure 5.8. In the waveform, it can display the detail of behaviors when the neural network starts to work.

As shown in the waveform Figure 5.7, the AES address default output is "8". In the next layer's weight room, address 8 stores zero, which has no impact on the calculation of membrane potential. When the start signal comes into the neural network, a load signal is activated to stimulate the ram to store the spikes_in. After the store is finished, the control logic sends a fifo_out_en signal to the address transmission process according to the above functionality. The AER address port starts to send '6' '4' '1' corresponding to the spikes in the 8-bit vector. It can be expressed in the pri_in and inside_spikes signal. After transmission is finished, the cal_finished signal and done_en signal are stimulated to stop the cal and go to the next FIRE state, shown in the neuron stimulation waveform. Until the next fifo_out_en True signal comes, the FIFO would not send a new 8-bit vector to the priority encoder.

In Figure 5.8 shows the second AER protocol behaviors. The default AER address output is 24. When a new spikes_in arrives at the second AER layer, the load signal is activated, and the ram starts to load the spikes into the ram. And then wait for the fifo_out_en, the air starts to send the addresses '21' '16' '11' ... '5' like the first layer are. Differently, the ram also loads the spikes_in when the load signal is '1'. The ram in the waveform shows appropriately.

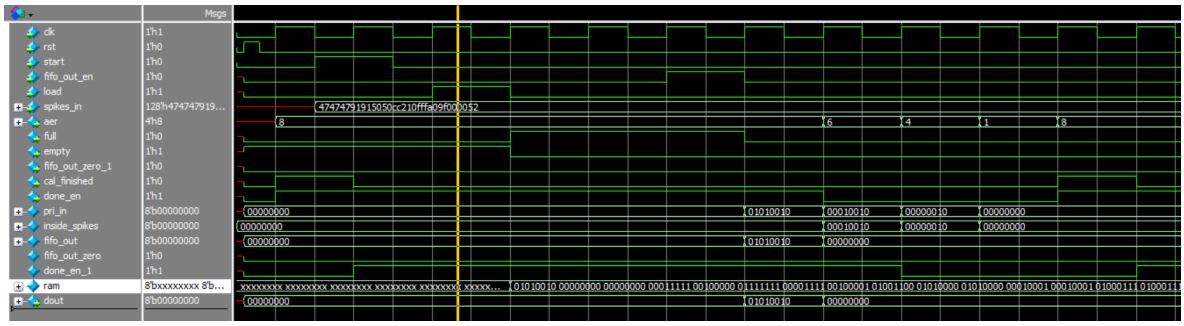

Figure 5.7: First layer AER simulation.

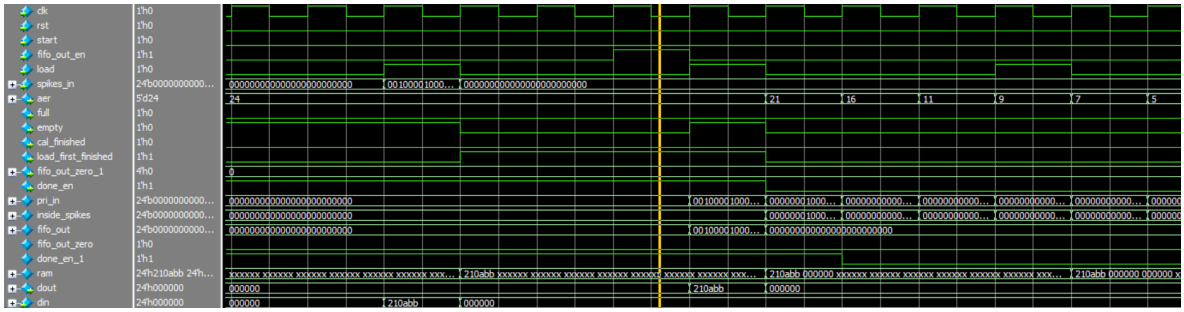

Figure 5.8: Second layer AER simulation.

5.3 Neuron Core

Directly transmitting the AER address information to the IF neuron brings to a mess of the design. The neuron core consists of several IF neurons, some control logic part and weight ROM can help to make the design more *tight* and *neat*. With the help of the neuron core, the memory of the weight data can be distributed in the whole architecture. The neuron core can work in parallel, reducing the computation and storage complexity of different layers of neural networks. Thus, using proper communication protocols like AER and router can enhance the throughput in the chip. A well-designed mapping strategy is also essential and can fully exploit the capability of the neural network[10].

Therefore, in this chapter, the details of the neuron design are described. The AER address using block can be used to detect whether the new address is not the same as before and allow the IF neuron to do the calculation process. After the IF neuron has finished the calculation process, the core can also stop the IF neuron from firing and send the spikes to the next layer. That is called FIRED control block.

5.3.1 Design and architecture

The main design idea comes from [33] and adding some modifications. The Neuron Core block is like Figure 5.9. The global clk and rst signal can drive the Neuron Core to work. The *done_en* comes from the AER protocol block, which can determine whether the IF neuron should fire spikes. The input_spike_addr can read the aer address data from the aer address communication bus. When the whole layer finishes the calculation,

the control logic sends the Next_step signal to make the Neuron Core fire spikes to the next layer. The Output_spikes bus is used to send the spikes to the next layer. The Done signal tells the control block, the firing process is finished, and the control state is switched to the IDLE state.

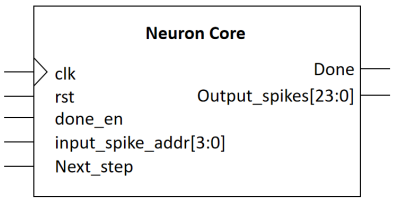

Figure 5.9: Digital block of Neuron Core.

Going deep into the schematic of the Neuron Core like Figure 5.10, the internal signal control logic can be explained more clearly. The default data of pri_addr is "1000" which is out of the input address range. Once the Neuron Core reads a new address, the pri_addr will compare with the new address and generate a Next signal to drive Neuron Array to calculate. Besides, the pri_addr is updated to the new address in this clock cycle. When a new address is read in the next clock cycle, the pri_addr has already been updated to the old one and can determine whether the new address is different from the old one. Meanwhile, the ROM weight array sends the corresponding weight data to the Neuron Array.

In the Neuron array, the working mechanism has been described in the IF neuron section. The combinational logic creates the Fired signal. Thus it can change as the same as Fire signal and give feedback to the Fire signal to stop the Fire signal. However, the out_Fired signal can last until the Next_step signal comes. When done_en and Fired are activated, the Done signal can tell the control logic at the end of the Neuron Core and send the Next_step signal. Once the Fired Control receives the Next_step signal from the control block, the Fired Control sends the 24 bits to vector Output_spikes to the next layer.

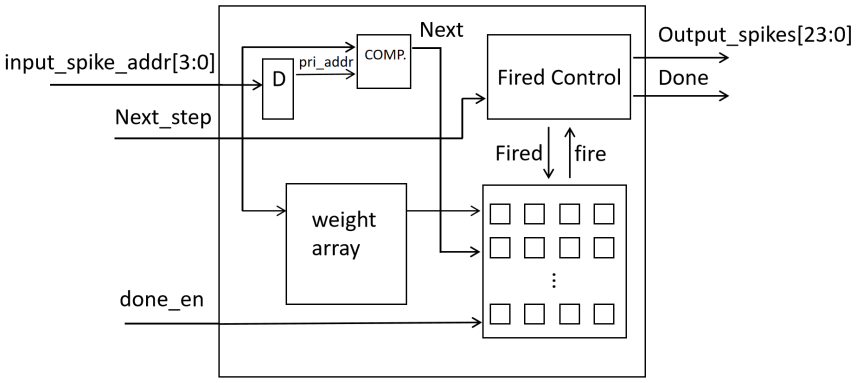

Figure 5.10: Digital block of Neuron Core Schematic.

5.3.2 Simulation

In Figure 5.11, the simulation waveform is also created by the Modelsim in order to show the functionality of neuron core. In the timeline simulation, there are all the signals mentioned above for a better explanation.

The weight ROM array stores all the weight data. There are 24 IF neurons in this neuron core as the new input_spike_addr is the key to starting the neuron core's workflow. Because of this, When a new address '6' is read by the bus, the Neuron Core starts to work. The next signal is activated until the calculation process is finished. Then, Fire, Fired, and out_Fired signal gets the 24 vector spikes information, waiting for the Next_step to send the spikes to the Output_spike. Meanwhile, the Output_spike only lasts for one clock cycle. The waveform exhibited the same workflow as the schematic's explaination.

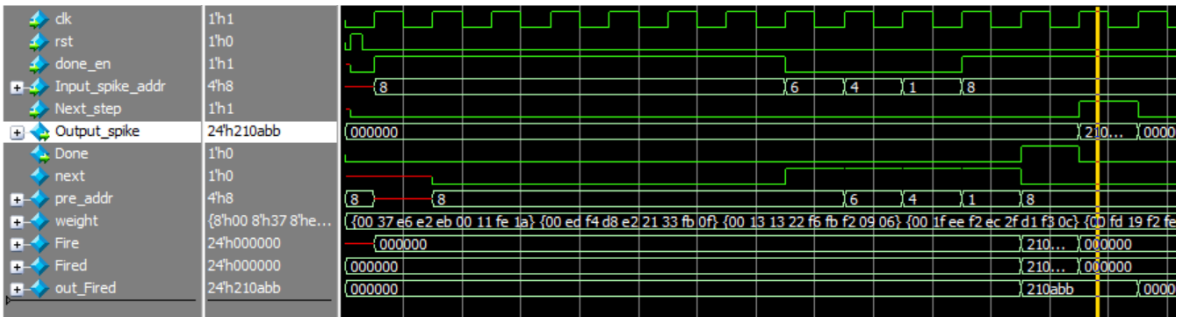

Figure 5.11: Neuron Core simulation.

5.4 SNN Emulation

As introduced in the introduction section, three main modules have been proposed to build a spiking neural network inference. The IF neuron module, the aer address protocol, and the Neuron core are the essential parts of this system. Besides, the control logic parts are also significant to connect the different parts to make them work properly.

In this section, different from previous sections to describe one of the major parts in the spiking neural network, this section intends to introduce the whole system how to build by the proposed digital modules in the above sections. In other words, how to connect those modules in order to get a neural network to perform the substitute task. Moreover, if the resource is available, thousands of neuron layers can be implemented in the spiking neural network following the construction standard.

5.4.1 Design and Architecture

First of all, a neural network consists of some neuron layers, consisting of several neurons core. The input 128 vector data works as 128 continuous spikes are received into the hidden layers. And the hidden layers perform the corresponding functionality to

transmit the spikes inside the neural network. Finally, the output obtains the substitute results and passes to the outside.

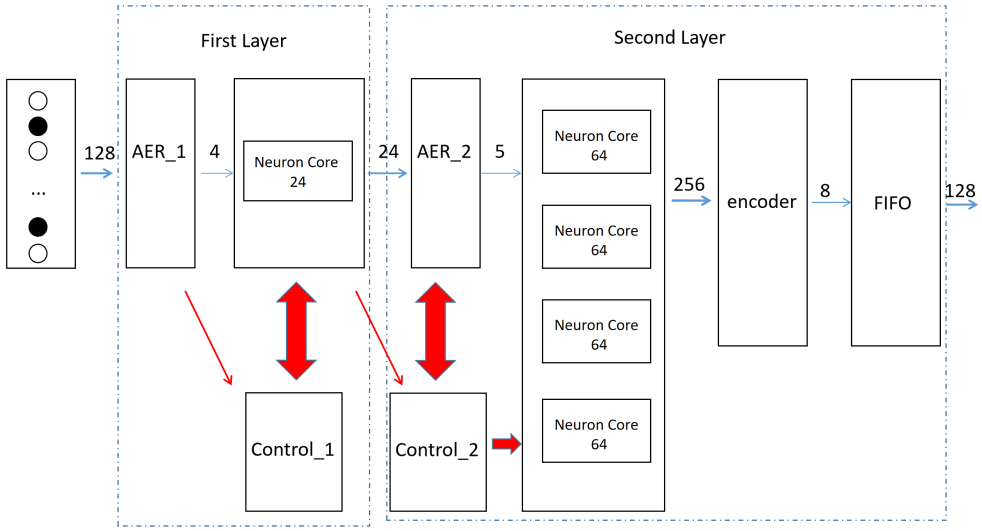

Figure 5.12: Digital block of SNN Schematic.

As shown in Figure 5.12, the aer_1 is responsible for building the communication bridge between different layers. The aer modules can store the input spikes and translate the position of the spikes to the address information. Then, transmitting the address information to the Neuron Core to obtain the appropriate weight data. Additionally, the done_en signal generated from aer module also can stop the neuron.

In this thesis, the two-layer consists of the two different control logic, the most important module to drive the whole system to work. The first layer consists of three major parts: aer_1, one neuron core consisting of 24 IF neurons, and a control_1 module. The first layer can send 24 vector stimuli to the next layer every time due to 24 IF neurons. Because the number of spikes in every input 8 bits vector is different, the 24 vector transmission time interval would be variable. It is the same in the second layer due to the different spikes in the second input 24 vector. The second layer consists of five different parts. The aer_2 can read address and write input 24 vectors into FIFO, which can help speed up the process. Because the output has 256 different results, the second layer consists of 256 neurons, which are divided into four neuron cores consisting of 64 IF neurons. When one classification task is completed, only one-second layer neuron fires one spike to the encoder. The encoder then translates this spike to the 8-bit vector as the substitute process acts. The following FIFO then absorbs this 8-bit vector into FIFO. When the FIFO gets 16 8-bit vectors, the second layer broadcasts the 128 bit to the outside, indicating one substitute process is finished.

Additionally, the control logic is built by the Moore finite state machine(FSM) whose output values are determined by the current state. In Figure 5.13 and Figure 5.14, two FSM workflows establish the functionality of the first layer and second layers' functionality. The first layer control state has four states: IDLE, LOAD, CAL, and FIRE. When the system starts to work, a start signal drives the IDLE to the LOAD state.

After the first FIFO stores all the 128-bit vectors, a full signal works as load_finished signal stimulates the LOAD state to the CAL state. In the CAL state, the control logic sends the fifo_out_en signal to read the 8-bit vector from the FIFO and do the calculation process. After the calculation processes are finished, the FIRE states fire the spikes to the next layer. If the FIFO is not empty and the layer is already fired, the control logic returns to the CAL state to calculate the next new 8-bit vector. When the FIFO is empty, the control logic receives the substitute finished signal and goes to the IDLE state to wait for the next start signal to restart the whole process.

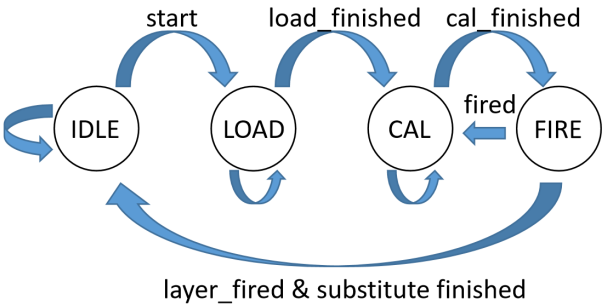

Figure 5.13: First Layer Control State.

The second layer control state is almost the same as the first layer control state. Because of the different mechanisms of the second FIFO, the second Moore FSM can work appropriately without the load state. The load_first_finished signal can work as the start signal in the first control logic to simulate the whole system. And the substitute finished signal is not the empty signal like the first control logic, while it is the full signal in the output FIFO.

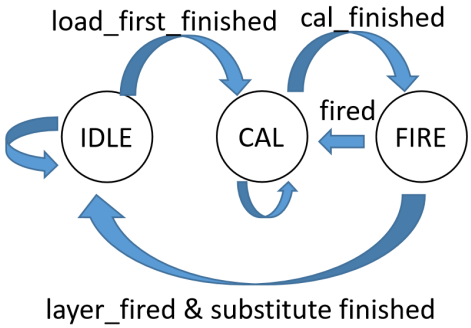

Figure 5.14: Second Layer Control State.

Those two control logic make the whole system work successfully and make the work state easily controlled. With the help of the control, all the spikes can only fire in one clock cycle.

5.4.2 Simulation

In this section, the simulation result comes from the Modelsim. As shown in Figure 5.15, after several clocks when the 128 bits was loaded in the module, the first layer finished the process and transmitted the 24-vector spikes to the second layer. In the meanwhile, the second layer started to do its calculation. However the second layer did not finishes its first process when the first layer already sent its second spikes. The fifo between the first layer and the second layer can store the second 24 vector spikes waiting for the second layer to do second process. In Figure 5.16 yellow line position, the second layer finished its process and sent the spike to the outside encoder.

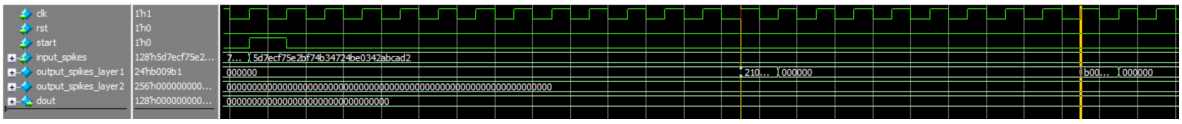

Figure 5.15: First Layer Spike Simulation.

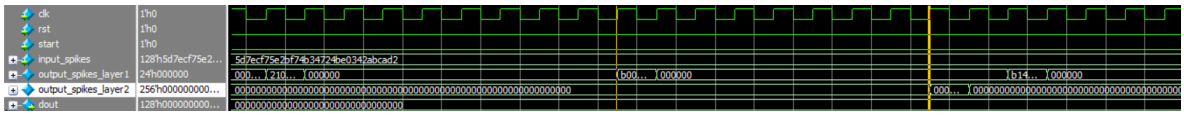

Figure 5.16: Second Layer Spike Simulation.

Figure 5.17 shows the final result after ten rounds in AES algorithm.

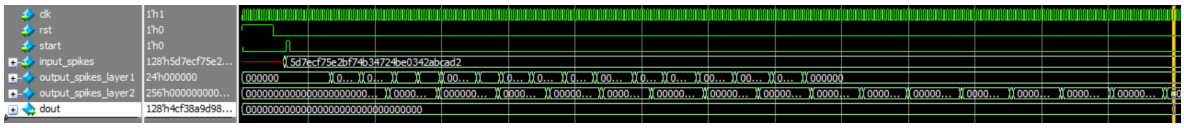

Figure 5.17: Substitute Result.

6 Experiment

This chapter describes the experimental platform, the power traces collection method, and the side-channel attack scripts, including DPA and CPA. Finally, a data analysis conclusion and demonstration are described.

6.1 Platform

In order to perform the power side-channel attack on the target design, it is necessary to use a method to collect the power traces of the design. However, it is time-consuming to wait for the actual product. And in the post-silicon stage, the design is hard to change. Thus, the designer needs to check whether the design is resistant against the SCA on the design stage. Some EDA tools can help analyze the design in the pre-silicon stage like [23]. It usually uses the RTL code to design the hardware first and then synthesize the RTL code to the netlist. Then the place and route process would put the synthesis netlist on the target silicon technology and connection resources. After that, the power analysis EDA tool can obtain the power traces for data analysis. Alternatively, using the Spice model to simulate the circuit characteristics [19]. Figure 6.1 shows the work flow of using Spice model to perform DPA.

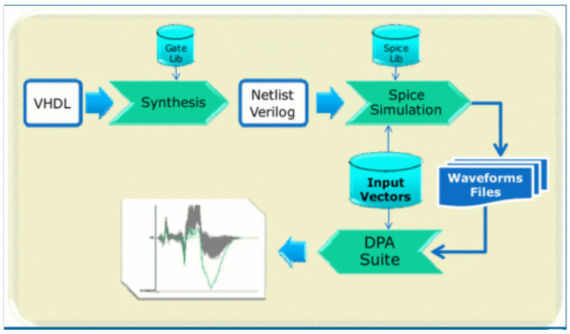

Figure 6.1: The DPA work flow of Spice model [19].

Except for the simulation process above, the FPGA can also provide the power-based side-channel attack in reality. The flexibility of FPGA has an excellent advantage for the designer to implement the digital design. Furthermore, the FPGA is a real-time application that it can simulate the actual power in reality. However, only using the FPGA is insufficient. Power measurement equipment needs to be used to collect the power trace. The chipwhisperer & CW305 board [15] is power analysis equipment on FPGA implementation of AES. The chipwhisperer-lite tool shown in Figure 6.2 integrates high-speed power measurement, programmer, and fault-injection. And the

designed algorithms are implemented on the CW305 FPGA board like Figure 6.3. This FPGA also has ARM Cortex-M1 Cortex-M3 to run.

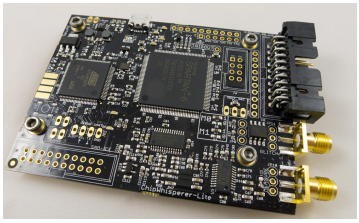

Figure 6.2: Chipwhisperer-Lite tool.

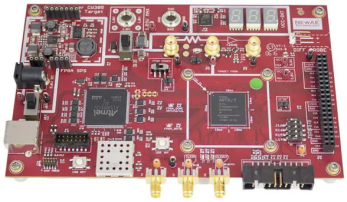

Figure 6.3: CW305 FPGA Board.

Like the Spice model-based SCA, Figure 6.4 shows the workflow of the FPGA-based SCA experiment. Firstly, building the RTL model and then the model needs to be run on the FPGA. Use the whisperer-lite board to collect the corresponding power traces and perform power analysis. If the model is resistant then it is a security hardware. If not, this model need to be improved.

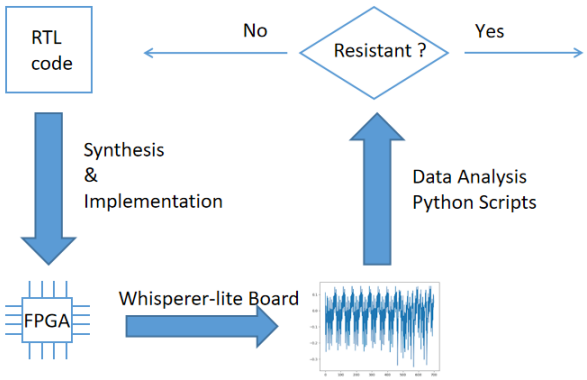

Figure 6.4: Experiment Flowchart.

6.2 SNN Sbox Power Analysis

After collecting the power traces of the SNN sbox based AES in the FPGA, the power-based side-channel attack needs to be performed on those power traces. As mentioned in chapter 2, several different power analysis methods can reveal the secret key. In this thesis, both the Non-profiled Attacks and the Profiled Attacks will be performed to test the ability of SNN Sbox. The four mainstream attack methods will be adopted. They are 1-bit DPA, CPA, Template Attack, and DLSCA. The power traces points are up to 8000. From Figure 6.5, there are ten power consumption blocks. They indicates the ten rounds in the AES. It indicates the AES runs correctly. And comparing the result shows that the AES works well after replacing the sbox with SNN based sbox. The total power traces is up to 10000.

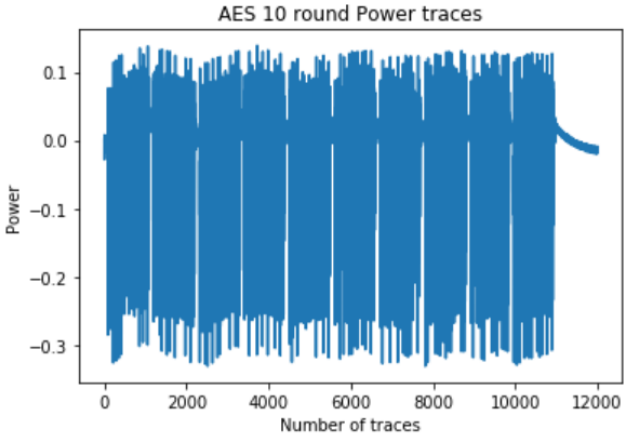

Figure 6.5: AES Power Traces.

6.2.1 Classical 1-bit DPA

First of all, the most straightforward attack method DPA, among those four ways, needs to be performed first. The data analysis script is based on [58]. However, its classification criteria is based on the hamming weight of the hypothesis sbox result. So, I changed it to the classical 1-bit DPA criteria. It divides the power traces into two groups based on 1 bit in the hypothesis sbox result. Because the correct key result will have a peak, and the uncorrelated key result has a almost flat line. So it is easy to determine the correct result by comparing the maximum value among 256 graphs. The hardware unprotected sbox using GF(2^8) calculation and affine transformation is set as the baseline to compare with the SNN sbox implementation. Unlike directly comparing the result of the guessing key with the correct key, partial guessing entropy is more helpful in analyzing the result. The partial guessing entropy, also called rank analysis, is a metric to rank the correct key. It can indicate how robust the implementation is. Figure 6.6 is the first byte rank analysis baseline result and snn sbox rank analysis.

The original sbox's rank becomes one at 2000 power traces. In other words, the correct key ranks first among the guess keys, and the original sbox is attacked successfully. However, the SNN sbox even uses 10000 power traces can not reveal the correct

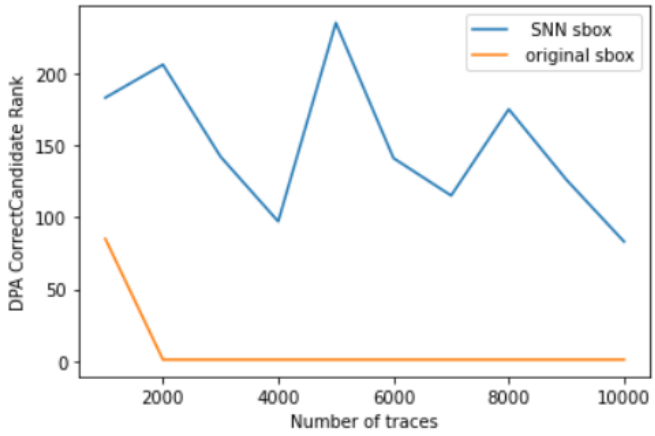

Figure 6.6: DPA Rank Analysis(original vs SNN).

key and the correct key is ranking outside 100. It demonstrates that the SNN sbox can protect implementation from DPA.

6.2.2 CPA

Furthermore, the CPA is slightly better than the DPA. Because CPA needs fewer power traces than DPA [3], thanks to the scripts in [53], the CPA analysis can be easily implemented. The CPA uses Pearson's Correlation to determine which subkey is the correct result. However, the SNN sbox can still resist the CPA. The Hamming distance model is used in CPA to model the power consumption. The unprotected original sbox CPA rank analysis and protected SNN Sbox CPA rank analysis show in Figure 6.7 and Figure 6.8.

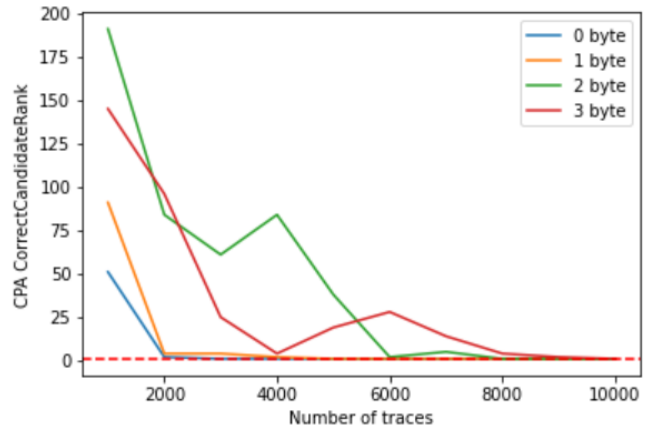

Figure 6.7: Unprotected Original Sbox CPA Rank Analysis.

From Figure 6.7 and Figure 6.8, the unprotected sbox's rank will decrease with the increase of the power traces. Moreover, some bytes' ranks decrease to one quickly. However, the ranks in the protected SNN sbox fluctuate randomly. Most of the ranks

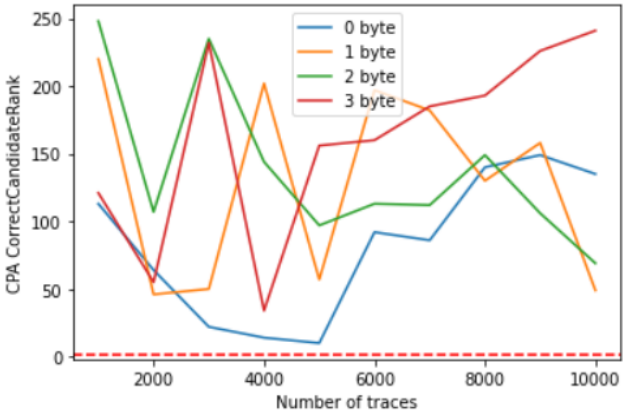

Figure 6.8: Protected SNN Sbox CPA Rank Analysis.

are out of range 50. And, even the number of power traces becomes larger, the rank becomes bigger. It demonstrates that the SNN sbox can avoid the implementation from CPA.

6.2.3 Template Attack

One of the profiled attacks, template attack, is more potent than the non-profiled attack. Before implementing the attack, the template of the target needs to be created. Moreover, the template is created by random key power traces. First of all, the 10000 power traces need to be sorted according to the hamming weight. And selecting the points of interest can reduce the data analysis load. Figure 6.9 shows the points of interest. However, it is different from the usual. There are many peaks in Figure 6.9. It may indicate that the template attack did not work. After selecting enough points of interest, the multivariate distributions at each point for each Hamming weight can be built. Then it is time to perform the attack. The final result is determined by the

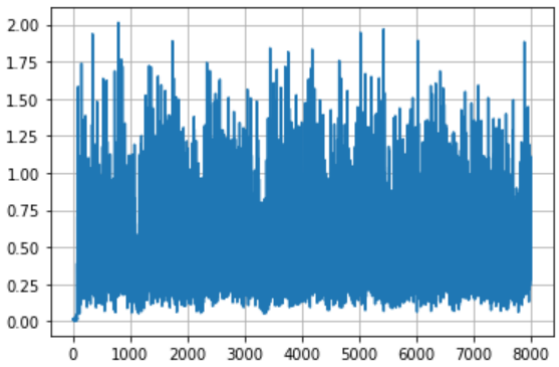

Figure 6.9: Template Attack Points of Interest.

maximum data in the result array. The resulting array is updated when the template evaluates new attack power traces. However, even 10000 power traces were calculated,

the final result is different from the correct result. Then rank analysis in Figure 6.10 and Figure 6.11 show the result more reasonably. It is the same as the CPA and DPA rank analysis. The template attack rank of unprotected sbox decreases with the number of power traces increases. However, the SNN sbox protected implementation ranks fluctuate, and most of them are out of rank 100. In other words, the template attack can not attack it successfully.

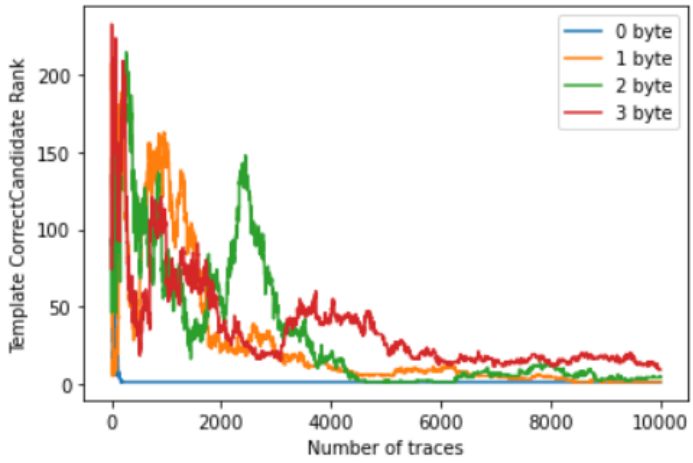

Figure 6.10: Unprotected Original Sbox Template Attack Rank Analysis.

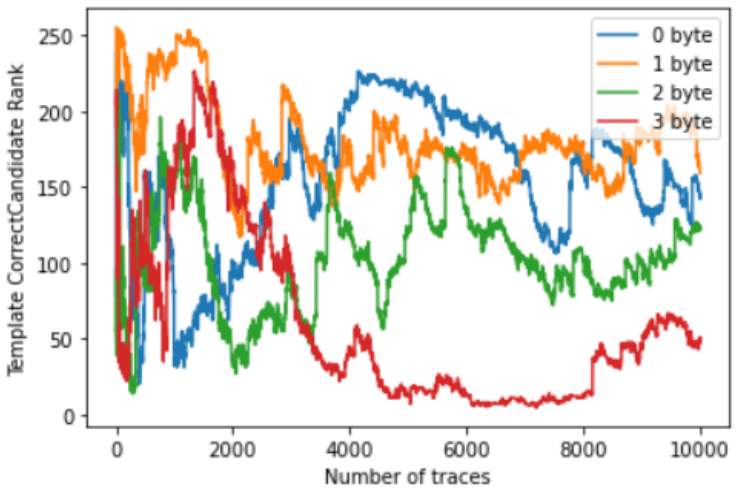

Figure 6.11: Protected SNN Sbox Template Attack Rank Analysis.

6.2.4 DLSCA

As in the template attack, the points of interest Figure 6.9 is too challenging to determine which points are good to select. So deep learning-based side-channel attacks can help to solve this problem. The DLSCA does not need pre-processing to select

points of interest. And it already demonstrates that the baseline can be attacked by the template attack. So it does not needs to perform it again with DLSCA. The SNN sbox needs to be tested by DLSCA. Figure 6.12 shows the rank analysis. It can be concluded that the SNN Sbox also resist the DLSCA and the rank is almost out of 100.

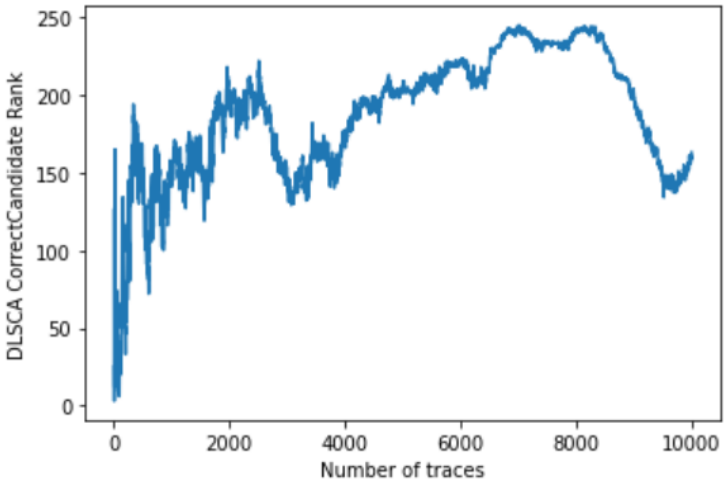

Figure 6.12: SNN Sbox DLSCA Rank Analysis.

6.3 Performance Analysis and Comparison

In the work of S-Net [54], they used standard ANN in the software to protect AES from attacking. Instead of comparing the software ANN and hardware SNN, the comparison of hardware S-Net and SNN Sbox is needed. The comparison is based on AES-128. Both of them can resist the above four implementations.The original unprotected AES requires 10 clock cycles for one encryption. The hardware S-Net requires 913 clock cycles for one AES-128 encryption. However, the SNN Sbox needs 2750 clock cycles for one AES-128 encryption. The SNN Sbox is three times slower than the S-Net. The AES-128 has ten rounds for one encryption. So each round in the S-Net needs approximately 91.3 clock cycles to do substitution for 8 bits input. And the SNN sbox requires approximately 275 clock cycles for sixteen 8 bits inputs. If the S-Net works in pipeline for 16 inputs, it requires $91.3 \times 16 \div 2 = 730.4$ clock cycles. So the SNN is quicker than S-Net when both of them are used in the pipeline mode. As for the resources aspect, the corresponding resources of those two different AES implementation are shown in Table 6.1 and Table 6.2. The S-Net hardware uses more memory and DSP resources than the SNN hardware because the S-Net uses 16 blocks to substitute in parallel and sacrifices the area. Furthermore, the SNN structure is larger than the S-Net in theory. So the SNN uses the pipeline principle to reduce the resources used and sacrifices the speed. As for the power consumption aspect, the total On-Chip Power of S-Net is 0.570W, and SNN is 0.488W. The SNN saves power.

Furthermore, the SNN can not be stimulated by the all-zero input. So the all zero

Resource	Estimation
LUT	24146
LUTRAM	16
FF	7749
IO	44
BUFG	2

Table 6.1: SNN based Sbox AES Hardware Resources.

Resource	Estimation
LUT	15583
FF	7475
BRAM	128
DSP	128
IO	42
BUFG	4

Table 6.2: S-Net based Sbox AES Hardware Resources.

input case is replaced by a LUT in the SNN sbox. Thus the implementation is not complete compared with S-Net. However, the input has 256 different cases, and the all-zero case is only a particular case. Figure 6.13 uses the Hamming Distance to describe the distribution of nine input classes. The number of all zero cases is so small that it can be ignored in the whole process. Due to the pipeline SNN, the leakage power can be hidden in the processes.

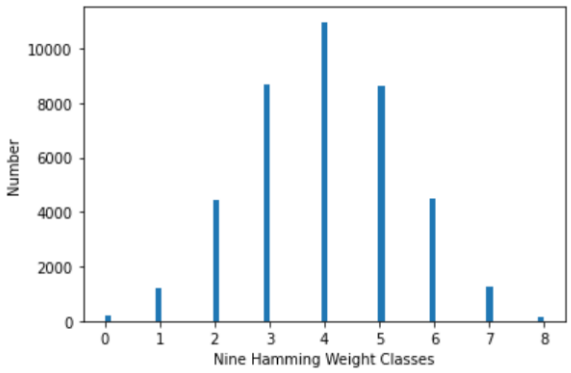

Figure 6.13: The Distribution of Nine Input Classes.

7 Conclusion and Future Work

This chapter restates the purpose of this thesis and gives a conclusion of this work. Furthermore, some future work can be implemented on this work.

7.1 Conclusion

The side-channel attack is a powerful attack method to reveal secret keys in AES. There are already two countermeasures Hide and Mask. However, those two attack methods can not disconnect the relationship between the power traces and the final result. Because of the non-linear characteristic of neural networks, the neural network can disconnect the relationship. In this thesis, a spiking neural network is selected rather than a normal neural network. The simplest neural model I&F model was chosen for the hardware implementation. Two different SNN simulators were applied to obtain better hardware-friendly parameters. However, the learning methods in the two simulators are different. In order to get low resource utilization, some optimizations were implemented in the SNN model. The bias was removed, and the weight's bits were quantized to integer rather than float. And the spiking neural network implemented in the hardware is only one spike neural network. In other words, the neurons in the model only fire once during each classification, which can save much power compared to the several spikes models.

In order to test the SNN sbox AES, four experiments were implemented. They all indicate that the designed SNN sbox aes can resist power-based SCA. The reason that the SNN can resist the SCA has two significant reasons. The first one is that the calculation process can break the relationship between power traces and the leakage model. The second one is that the 128 bits sbox substitute process can hide the information inside the power traces. The sbox is usually 8 bits and replaces 8 bits every time. Because the SNN has two layers, while the second layer does the calculation to indicate the 8 bits result, the first layer can calculate the next 8 bits data. And the next 8 bits of data are variable, leading to variable power traces. Owing to those two reasons, the SNN sbox AES is secure against these attacks.

Although the SNN in this thesis can not achieve multi-target detection like S-Net, the hardware structure of the SNN sbox is much simpler than the S-Net. It does not need the multiplier inside the FPGA so that the usage of DSP resources is zero. And the power consumption of SNN is smaller than the S-Net, which demonstrates the hardware friendly of SNN. And the two layers work simultaneously that can enhance the complexity of power consumption.

7.2 Future Work

Although this SNN sbox can resist these attack, it still needs to be improved.
First of all, this SNN is only single target SNN. The SNN can be converted to multi-targets SNN. Then the output layer only needs 8 outputs rather than 256 outputs neurons. In this way, the amount of memory to store weight can be reduced and also the number of connection can be reduced. After that, this multi-targets SNN can compare with ANN based sbox hardware implementation to demonstrate whether SNN based sbox implementation is better.

Secondly, because the all zero input can not stimulate the SNN to transmit spikes, in this thesis the SNN do not consider the all zero condition. However, the all zero inputs may leak power information to the attackers. So the designer can add some extra bits to represent the all zero situation or some other methods.

Furthermore, the SNN replacement method can be used in some other substitution algorithms like DES. Whether snn based sbox works well in the other algorithms needs to be investigated.

The communication AER protocol inside the SNN can only transmit one address every clock cycle. This is very time-consuming. So asynchronous protocol can be considered to use to increase the communication speed.

Bibliography

[1] URL: https://www.humanbrainproject.eu/en/silicon-brains/how-we-work/hardware/.

[2] Franco Alcaraz. *fxpmath*. 2020. URL: https://github.com/francof2a/fxpmath.

[3] Massimo Alioto, Massimo Poli, and Santina Rocchi. "Power analysis attacks to cryptographic circuits: a comparative analysis of DPA and CPA". In: *2008 international conference on microelectronics*. IEEE. 2008, pp. 333–336.

[4] Michael Backes et al. "Acoustic Side-Channel Attacks on Printers." In: *USENIX Security symposium*. Vol. 9. 2010, pp. 307–322.

[5] Yoshua Bengio et al. "STDP as presynaptic activity times rate of change of postsynaptic activity". In: *arXiv preprint arXiv:1509.05936* (2015).

[6] Ben Varkey Benjamin et al. "Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations". In: *Proceedings of the IEEE* 102.5 (2014), pp. 699–716. DOI: 10.1109/JPROC.2014.2313565.

[7] Ben Varkey Benjamin et al. "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations". In: *Proceedings of the IEEE* 102.5 (2014), pp. 699–716.

[8] Alex Biryukov and Dmitry Khovratovich. "Related-key cryptanalysis of the full AES-192 and AES-256". In: *International conference on the theory and application of cryptology and information security*. Springer. 2009, pp. 1–18.

[9] Kwabena A Boahen. "Point-to-point connectivity between neuromorphic chips using address events". In: *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing* 47.5 (2000), pp. 416–434.

[10] Maxence Bouvier et al. "Spiking neural networks hardware implementations and challenges: A survey". In: *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 15.2 (2019), pp. 1–35.

[11] Facundo Bre, Juan Gimenez, and Víctor Fachinotti. "Prediction of wind pressure coefficients on building surfaces using Artificial Neural Networks". In: *Energy and Buildings* 158 (Nov. 2017). DOI: 10.1016/j.enbuild.2017.11.045.

[12] Eric Brier, Christophe Clavier, and Francis Olivier. "Correlation power analysis with a leakage model". In: *International workshop on cryptographic hardware and embedded systems*. Springer. 2004, pp. 16–29.

[13] Suresh Chari et al. "Towards sound approaches to counteract power-analysis attacks". In: *Annual International Cryptology Conference*. Springer. 1999, pp. 398–412.

[14] Gregory K Chen et al. "A 4096-neuron 1M-synapse 3.8-pJ/SOP spiking neural network with on-chip STDP learning and sparse weights in 10-nm FinFET CMOS". In: *IEEE Journal of Solid-State Circuits* 54.4 (2018), pp. 992–1002.

[15] *CHIPWHISPERER*. URL: https://www.newae.com/chipwhisperer.

[16] Po-Yao Chuang et al. "A 90nm 103.14 tops/w binary-weight spiking neural network cmos asic for real-time object classification". In: *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE. 2020, pp. 1–6.

[17] Bart Coppens et al. “Practical mitigations for timing-based side-channel attacks on modern x86 processors”. In: *2009 30th IEEE symposium on security and privacy*. IEEE. 2009, pp. 45–60.
[18] Joan Daemen and Vincent Rijmen. “AES proposal: Rijndael”. In: (1999).
[19] Giorgio Di Natale, Marie-Lise Flottes, and Bruno Rouzeyre. “An integrated validation environment for differential power analysis”. In: *4th IEEE International Symposium on Electronic Design, Test and Applications (delta 2008)*. IEEE. 2008, pp. 527–532.
[20] Peter U Diehl et al. “Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing”. In: *2015 International joint conference on neural networks (IJCNN)*. ieee. 2015, pp. 1–8.
[21] Wei Fang et al. *SpikingJelly*. https://github.com/fangwei123456/spikingjelly. 2020.
[22] Wulfram Gerstner. *Neuronal Dynamics*. URL: https://neuronaldynamics.epfl.ch/index.html.
[23] Behnam Ghavami, Hossein Pedram, and Mehrdad Najibi. “An EDA tool for implementation of low power and secure crypto-chips”. In: *Computers & Electrical Engineering* 35.2 (2009), pp. 244–257.
[24] Wenzhe Guo et al. “Neural Coding in Spiking Neural Networks: A Comparative Study for Robust Neuromorphic Systems”. In: *Frontiers in Neuroscience* 15 (2021), p. 212. ISSN: 1662-453X. DOI: 10.3389/fnins.2021.638474. URL: https://www.frontiersin.org/article/10.3389/fnins.2021.638474.
[25] Hananel Hazan et al. “Bindsnet: A machine learning-oriented spiking neural networks library in python”. In: *Frontiers in neuroinformatics* 12 (2018), p. 89.
[26] AL Hodgkin and AF Huxley. “A quantitative description of membrane current and its application to conduction and excitation in nerve”. In: *Bulletin of mathematical biology* 52.1-2 (1990), pp. 25–71.
[27] Taras Iakymchuk et al. “Fast spiking neural network architecture for low-cost FPGA devices”. In: *7th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)* (2012), pp. 1–6.
[28] Eugene M Izhikevich. “Simple model of spiking neurons”. In: *IEEE Transactions on neural networks* 14.6 (2003), pp. 1569–1572.
[29] Sunghyun Jin et al. “Recent advances in deep learning-based side-channel analysis”. In: *ETRI Journal* 42.2 (2020), pp. 292–304.
[30] Muhammad Mukaram Khan et al. “SpiNNaker: mapping neural networks onto a massively-parallel chip multiprocessor”. In: *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*. Ieee. 2008, pp. 2849–2856.
[31] Saeed Reza Kheradpisheh and Timothée Masquelier. “Temporal backpropagation for spiking neural networks with one spike per neuron”. In: *International Journal of Neural Systems* 30.06 (2020), p. 2050027.
[32] Paul Kocher, Joshua Jaffe, and Benjamin Jun. “Differential power analysis”. In: *Annual international cryptology conference*. Springer. 1999, pp. 388–397.
[33] Mingxuan Liang, Jilin Zhang, and Hong Chen. “A 1.13 μJ/classification Spiking Neural Network Accelerator with a Single-spike Neuron Model and Sparse

Weights". In: *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2021, pp. 1–5.

[34] Kuan Jen Lin, Chih Ping Weng, and Tsai Kun Hou. "Enhance hardware security using FIFO in pipelines". In: *2011 7th International Conference on Information Assurance and Security (IAS)*. IEEE. 2011, pp. 344–349.

[35] Jake Longo et al. "SoC it to EM: electromagnetic side-channel attacks on a complex system-on-chip". In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2015, pp. 620–640.

[36] Andrew Luashchuk. *8 Reasons Why Python is Good for Artificial Intelligence and Machine Learning*. May 2019. URL: https://towardsdatascience.com/8-reasons-why-python-is-good-for-artificial-intelligence-and-machine-learning-4a23f6bed2e6.

[37] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks: Revealing the secrets of smart cards*. Vol. 31. Springer Science & Business Media, 2008.

[38] Massoud Masoumi and Sobhan Saie Moghadam. "A simulation-based correlation power analysis attack to FPGA implementation of KASUMI block cipher". In: *International Journal of Internet Technology and Secured Transactions* 7.2 (2017), pp. 175–191.

[39] Eduard-Guillem Merino Mallorquı. "Digital system for spiking neural network emulation". B.S. thesis. Universitat Politècnica de Catalunya, 2017.

[40] Thomas S Messerges. "Using second-order power analysis to attack DPA resistant software". In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2000, pp. 238–251.

[41] Hesham Mostafa et al. "Fast classification using sparsely active spiking networks". In: *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2017, pp. 1–4.

[42] *Neuromorphic Computing - Next Generation of AI*. URL: https://www.intel.com/content/www/us/en/research/neuromorphic-computing.html.

[43] Osaze Shears and Ahmadhossein Yazdani. "Spiking Neural Networks for Image Classification". en. In: (2020). DOI: 10.13140/RG.2.2.27001.80486. URL: http://rgdoi.net/10.13140/RG.2.2.27001.80486.

[44] Jeongwoo Park, Juyun Lee, and Dongsuk Jeon. "7.6 A 65nm 236.5 nJ/classification neuromorphic processor with 7.5% energy overhead on-chip learning using direct spike-only feedback". In: *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE. 2019, pp. 140–142.

[45] Francois Poucheret et al. "Spatial EM jamming: A countermeasure against EM Analysis?" In: *2010 18th IEEE/IFIP International Conference on VLSI and System-on-Chip*. IEEE. 2010, pp. 105–110.

[46] George H. Rutherford et al. "Analog implementation of a Hodgkin–Huxley model neuron". In: *American Journal of Physics* 88.11 (2020), pp. 918–923. DOI: 10.1119/10.0001072. eprint: https://doi.org/10.1119/10.0001072. URL: https://doi.org/10.1119/10.0001072.

[47] Dimitry G. Sayenko et al. "Facilitation of the soleus stretch reflex induced by electrical excitation of plantar cutaneous afferents located around the heel". In: *Neu-*

roscience Letters 415.3 (2007), pp. 294–298. ISSN: 0304-3940. DOI: https://doi.org/10.1016/j.neulet.2007.01.037. URL: https://www.sciencedirect.com/science/article/pii/S0304394007000778.

[48] Aboozar Taherkhani et al. "A review of learning in biologically plausible spiking neural networks". In: *Neural Networks* 122 (2020), pp. 253–272. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2019.09.036. URL: https://www.sciencedirect.com/science/article/pii/S0893608019303181.

[49] Amirhossein Tavanaei et al. "Deep learning in spiking neural networks". In: *Neural Networks* 111 (2019), pp. 47–63.

[50] *Template Attacks*. URL: https://wiki.newae.com/Template_Attacks.

[51] *The Neuron*. 2012. URL: https://www.brainfacts.org/brain-anatomy-and-function/anatomy/2012/the-neuron.

[52] user1449user1449 and mikeazomikeazo. *Aes addroundkey*. June 1961. URL: https://crypto.stackexchange.com/questions/8043/aes-addroundkey/8044#8044.

[53] *V4:Tutorial B6 Breaking AES (Manual CPA Attack)*. URL: https://wiki.newae.com/V4:Tutorial_B6_Breaking_AES_(Manual_CPA_Attack).

[54] Pradeep Venkatachalam. "S-Net, A Neural Network Based Countermeasure for AES". In: (2019).

[55] Wikipedia contributors. *Advanced Encryption Standard — Wikipedia, The Free Encyclopedia*. [Online; accessed 18-October-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Advanced_Encryption_Standard&oldid=1044936366.

[56] Wikipedia contributors. *Hodgkin–Huxley model — Wikipedia, The Free Encyclopedia*. [Online; accessed 15-October-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Hodgkin%E2%80%93Huxley_model&oldid=1049127797.

[57] Wikipedia contributors. *Neural coding — Wikipedia, The Free Encyclopedia*. [Online; accessed 16-October-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Neural_coding&oldid=1047918079.

[58] Yan1x0s. *Side Channel Attacks-Part 2 (DPA CPA applied on AES Attack)*. Apr. 2021. URL: https://yan1x0s.medium.com/side-channel-attacks-part-2-dpa-cpa-applied-on-aes-attack-66baa356f03f.

[59] Shihui Yin et al. "Algorithm and hardware design of discrete-time spiking neural networks based on back propagation with binary activations". In: *2017 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. IEEE. 2017, pp. 1–5.