專題研究期末報告

撰寫人： B12901187 章宏瑞
指導老師：闕志達 教授
主題： 1. Network Quantization & PE Array Design
2. 5G MIMO Demodulation

ABSTRACT — This report is mainly divided into two main themes: A multiplier-less CNN inference accelerator and High-performance algorithm of MIMO communication system. Both topics hope to simplify the original high-complexity method without losing too much accuracy. In the first part of this report, I trained a well-known CNN models, MNIST, using PyTorch with 32-bit floating-point arithmetic initially. Subsequently, I applied the FloatSD method developed by our lab. This method not only significantly reduces the data transmission required for neural networks but also simplifies the convolution operations from multiplication and addition (MAC) to just addition. The CNNs for MNIST using SD4 weights achieved results comparable to their FP32-trained versions. Finally, I designed the hardware for the MAC circuit, processing element (PE) row, and processing element (PE) array to accelerate the convolution operations within the neural network. In second part of this report, I will introduce some algorithms decoding MIMO communication system. Next, use MATLAB to show the accuracy of these algorithms and analyze the complexity of algorithm, respectively. Then find the highest performance algorithm of decoding MIMO communication system. Last but not least, use the above algorithm to implement the hardware design of MIMO communication system which improve decoding speed without losing accuracy.

1. INTRODUCTION OF NETWORK QUANTIZATION AND PE DESIGN

Convolutional neural networks (CNNs) have achieved remarkable success in various fields. However, a significant drawback of CNNs is the extensive arithmetic operations required for their implementation. Consequently, many studies have already explored low-complexity CNN processing techniques. In this report, I introduce the FloatSD method, which reduces the bit-width of network weights from 32 bits to 4 bits. This approach decreases calculation complexity during CNN training. Despite the simplification of network weights and some training values, the FloatSD method maintains CNN accuracy that is comparable to FP32-trained CNNs.

Additionally, I use the above method to design a convolution acceleration circuit for SD4-weighted convolution layers. This accelerator called PE that

performs convolution using a 3x3 SD4 kernel with nine inputs in a 3x3 arrangement. Moreover, I created an array of PEs which provide four input and four output channels simultaneously. Using Design Vision, I assessed the timing, area and power in the PE array.

The rest of the first topic of this report is organized as follows. Section 2 shows the results of network quantization. Section 3 presents multiplier-less SD4 convolution circuit and section 4 shows the PE array design. In section 5 summarizes the result of what I have done in this topic.

2. NETWORK QUANTIZATION

In this section, I explain the methods and procedures employed to accomplish our goals and carry out network quantization on the MNIST model and CIFAR-10 model.

(1) MNIST

MNIST is a simple model with a network containing two convolutional layers and two linear layers, which is listed in Fig.1.

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5, 1)
        self.conv2 = nn.Conv2d(20, 50, 5, 1)
        self.fc1 = nn.Linear(4*4*50, 500)
        self.fc2 = nn.Linear(500, 10)

    def forward(self, x):
        x = self.conv1(x) #(Batch, 1, 24, 24)
        x = F.relu(x)
        x = F.max_pool2d(x, 2, 2)  #(Batch, 20, 12, 12)

        x = self.conv2(x)  #(Batch, 50, 8, 8)
        x = F.relu(x)
        x = F.max_pool2d(x, 2, 2)  #(Batch, 50, 4, 4,)

        x = x.view(-1, 4*4*50)
        x = self.fc1(x)
        x = F.relu(x)

        x = self.fc2(x)
        return x
```

Fig.1 Network structure for MNIST

The training results achieved 99.11% accuracy using our network. Training curves showed in Fig.2 and Fig.3 respectively.

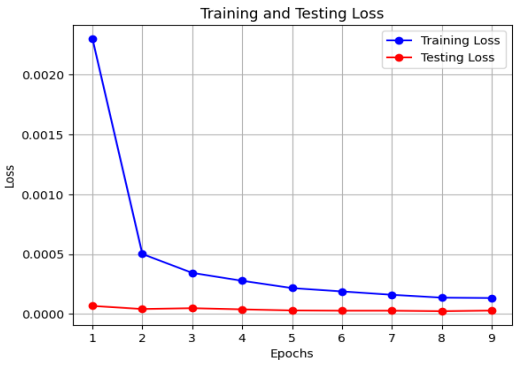

Fig.2 Training loss curve for MNIST using FP32

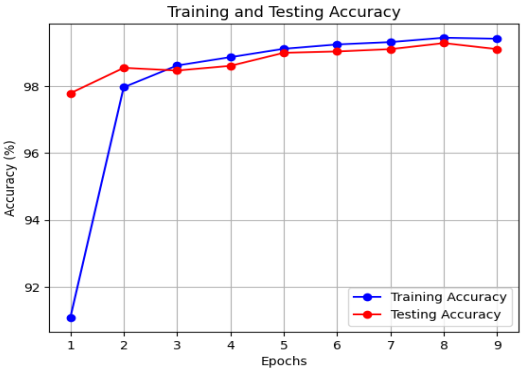

Fig.3 Training accuracy curve for MNIST using FP32

Then we use the autoSD developed in our laboratory to train the MNIST model and find that the accuracy is as high as 99.27%. Therefore, we can verify that when the number of access parameters becomes smaller, it will not affect the accuracy of training this model. In this way The benefit brought to us is the improvement of the efficiency of the timing and area. Training curves showed in Fig.4

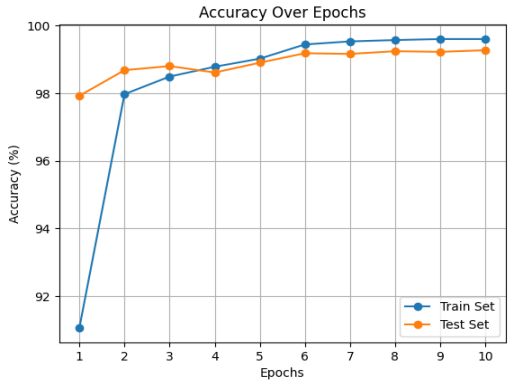

Fig.4 Training accuracy curve for MNIST using autoSD

(2) CIFAR10

CIFAR10 is a more complex data set and more parameter values than the previous MNIST model. Therefore, the accuracy of this model will be relatively lower than MNIST in the training process. I first set the parameters to FP32 and find that the accuracy is 92.65% over 100 epochs. Training curves accuracy can be seen in Fig.5.

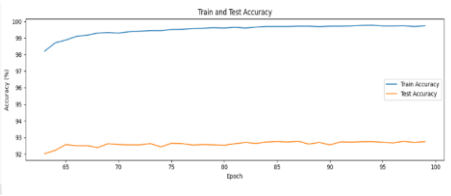

Fig.5 Train accuracy curve for CIFAR10

After completing the training of FP32, we trained the model again using the method of simplifying the number of parameter digits. We can find that although the accuracy has dropped by about 3% (92.65% to 89.72%), the overall performance is better than the high complexity of FP32. There is much less access space, so the time and area will be relatively reduced. Therefore, the performance of quantization method will be better than FP32.

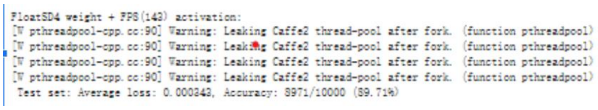

Fig.6 Train accuracy for CIFAR10 using FloatSD4 + FP8

3. HARDWARE DESIGN OF MAC

In this part, I will focus on explaining the design of SD4 3x3 MAC and how to use hardware to present the FloatSD4 representation we constructed in the previous topic.

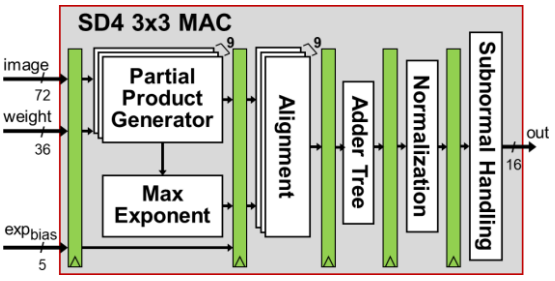

Fig.7 The block diagram of the five-stage SD4 3x3 MAC unit.

SD4 3x3 MAC takes in 9 FP8 pixels and 9 SD4 weights in each cycle.

The following figures show the block diagram and the hardware implement of the five stage of SD4 3x3 MAC.

(1) Partial Product Generator

The mantissa part is computed by cascading the product sign and the “leading 1” inherent in the FP8 number with the mantissa field. The exponent of the product is computed by summing the exponent fields of two inputs.

```
// image
// | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
// | S |     Exponent      | Mant. |
wire         image_sign = image[7];
wire [5-1:0] image_exp  = image[6:2];
wire [2-1:0] image_mant = image[1:0];

// weight
// | 3 | 2 | 1 | 0 |
// | S |  Exponent |
wire         weight_sign = weight[3];
wire [3-1:0] weight_exp  = weight[2:0];

// Add zero floag in case the image or weight is zero
wire zero_img_flag = image[6:0] == 0;
wire zero_wgt_flag = weight_exp == 3'b111;
wire zero_flag     = zero_img_flag || zero_wgt_flag;
```

Fig.8 Verilog implementation of partial product generator

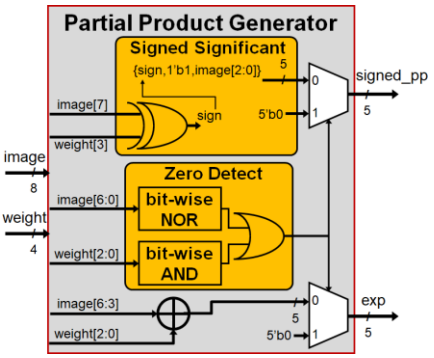

Fig.9 Block diagram of the partial product generator

(2) Max exponent & Alignment

The max exponent compares the nine products' exponent fields, finds the maximum value and computes the information needed for the nine partial products to properly align themselves forsummation in a later stage.

The shifted partial product is transformed into a two's complement representation based on its sign, resulting in a 16-bit output signal.

```
// align_pp
//                  exp_diff = 11     ----->          | 2 | 1 | 0 |
//                                                    | ld .      |
// | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
// | S |    .                                        G   R   S     |
assign align_pp = (pp_sign) ? ~{1'b0, shifted_unsign_pp} + 1'b1 : {1'b0, shifted_unsign_pp};
```

Fig.10 Verilog implementation of alignment

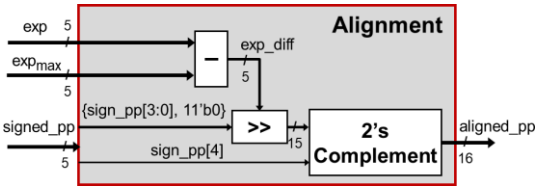

Fig.11 Block diagram of the alignment

(3) Adder Tree & Normalization

The adder tree circuit sums up nine aligned partial products from the previous block.

The normalization adjusts the sign_sum to norm_sum. First, it converts a signed integer to the corresponding unsigned number and find the leading one's position. Then we use a look-up-table shifter to shift the value according to the leading one's position. Last but not least, it needs to change the exponent field to the leading one's position.

```
// o_psum
//                  exp_diff = 11     ----->          | 2 | 1 | 0 |
//                                                    | ld .      |
// | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
// | S |                                             G   R   S     |
```

Fig.12 Verilog implementation of adder tree

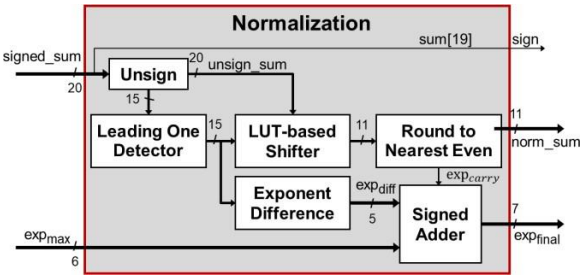

Fig.13 Block diagram of the normalization

(4) Subnormal handling

The last stage of SD4 3x3 MAC is subnormal handling, it adjusts the exponent term to become non-negative.

If signed_exp is non-positive, the normalized mantissa needs to be adjusted so that the exponent term is the lowest possible → 0.

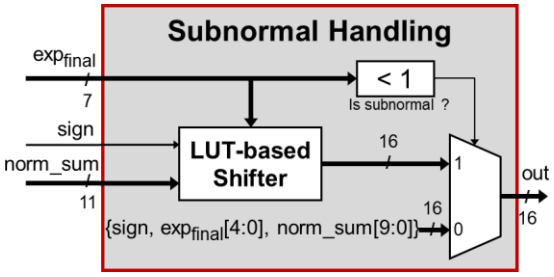

Fig.14 Block diagram of the subnormal handling

(5) Simulation Result

We input the picture into our MAC circuit and compare it with the output answer (stored in hexadecimal). We will find that it is consistent with the original.

802d	0056	0038	0056	0004	8063
8044	0027	001f	0076	0017	8023
0033	0055	006c	0050	8037	801c
0022	0041	003a	0012	8060	8016
001d	0039	00ad	8048	8079	801f
807e	0010	0052	000c	801d	001c
807d	0033	8033	0017	0037	0018
8077	8007	8083	005f	0035	0020
0064	8026	8077	0001	0021	8025
013a	8095	000c	8002	8025	8025
010f	8081	0021	8017	8039	802c
00a8	80a1	8016	8010	804d	0023
8032	8031	800f	8004	8034	0022
8029	0035	803d	0009	8026	002d
802f	007f	802c	0017	8008	8038
80ac	0083	8039	805a	0007	8034
80a1	801f	8030	8091	8031	803c
8097	803e	800f	80d5	8033	801b
802c	800	8048	8069	805f	806a
003f	80ac	803f	8048	803d	8061
0039	8097	8019	800e	8061	801d

Fig.15 Simulation result of MAC circuit

(A total of 126 kernels were counted)

4. PE DESIGN

(1) Processing element

The following figure shows the processing element (PE) for SD4 convolution with 3x3 kernels. As the kernel window slides through the image, the FP8 pixels for convolution are stored in the image buffer and fed to the SD4 MAC unit.

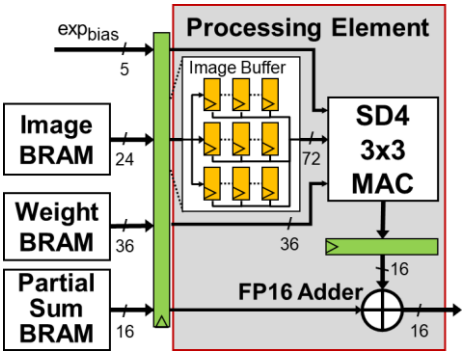

Fig.16 Block diagram of the processing element

(2) Processing element row

One PE row can calculate four independent input channels simultaneously. Each input channel feeds three image pixels in each cycle to a PE.

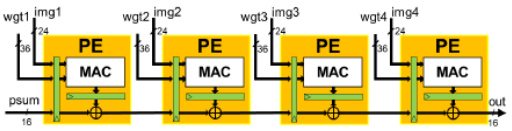

Fig.17 Architecture of the PE row

Now, we can connect the four MACs with wires and show the area of row. The following figure shows the wiring and calculation process.

```
assign conv_final = ff4_r + o_conv_4_r;
assign o_valid_final = row_valid_3_r;

always@(*) begin
    ff1_w = p_sum;
    ff2_w = ff1_r + o_conv_1_r;
    ff3_w = ff2_r + o_conv_2_r;
    ff4_w = ff3_r + o_conv_3_r;

    o_conv_1_w = o_conv_1;
    o_conv_2_w = o_conv_2;
    o_conv_3_w = o_conv_3;
    o_conv_4_w = o_conv_4;

    row_valid_1_w = ff1_valid_r;
    row_valid_2_w = row_valid_1_r;
    row_valid_3_w = row_valid_2_r;

end
```

Fig.18 Verilog implementation of row

```
Number of ports:                        10411
Number of nets:                         24304
Number of cells:                        12062
Number of combinational cells:           9795
Number of sequential cells:              1928
Number of macros/black boxes:               0
Number of buf/inv:                       1891
Number of references:                      13

Combinational area:                122363.867949
Buf/Inv area:                        9546.177467
Noncombinational area:              62311.552090
Macro/Black Box area:                   0.000000
Net Interconnect area:            1545981.280365

Total cell area:                   184675.420039
Total area:                       1730656.700404
```

Fig.19 Total area of row

(3) Processing element array

A PE array consists of four PE rows. Each PE row computes a distinct output channel, and all four rows in an array share the same set of four input channels. Note that the timing, area, and power are roughly 4 times larger than the PE row.

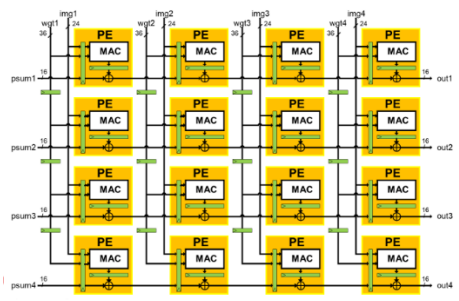

Fig.20 Architecture of the PE array

```
output                      o_valid_final_1,
output                      o_valid_final_2,
output                      o_valid_final_3,
output                      o_valid_final_4,
output signed   [15:0]      conv_final_1,
output signed   [15:0]      conv_final_2,
output signed   [15:0]      conv_final_3,
output signed   [15:0]      conv_final_4
```

Fig.21 The ouput wires of PE array

```
Number of ports:                        43450
Number of nets:                         98005
Number of cells:                        46953
Number of combinational cells:          37895
Number of sequential cells:              7712
Number of macros/black boxes:               0
Number of buf/inv:                       7289
Number of references:                       4

Combinational area:                481847.725993
Buf/Inv area:                       35020.756289
Noncombinational area:             249393.882145
Macro/Black Box area:                   0.000000
Net Interconnect area:            6084524.627228

Total cell area:                   731241.608138
Total area:                       6815766.235366
```

Fig.22 Total area of array

```
clock clk (rise edge)                                  10.00      10.00
clock network delay (ideal)                             0.50      10.50
clock uncertainty                                      -0.10      10.40
second/stg3/aligned_pp2_r_reg_14_/CK (DFFRX1)           0.00      10.40 r
library setup time                                     -0.22      10.18
data required time                                                10.18
-----------------------------------------------------------------------
data required time                                                10.18
data arrival time                                                -10.17
-----------------------------------------------------------------------
slack (MET)                                                       0.00
```

Fig.23 Total time of both row and array

Row : Area x time ≈ 1883045

Array : Area x time ≈ 7444033

We can calculate area multiplied by timing for row and array respectively, that is, we will find that the difference between the two is about 4 times. This is because we obtain the result of PE array after parallel processing of 4 PE rows.

5. CONCLUSION OF NETWORK QUANTIZATION AND PE DESIGN

During the half of this semester, I first learned to use software to implement the results of quantified parameters in our

laboratory, and compared them with the original FP32 representation. From the accuracy difference between the two, I can find that the performance after quantification will be much better than the original access. The decimal method is better, mainly because the speed is accelerated and the area is much smaller, and the accuracy will not decrease too much. It can be seen from the shallow learning MNIST model that the accuracy is almost perfect to 1, and the accuracy of the more complex model CIFAR10 is only 3% lower than the original FP32 representation. Therefore, we can be sure that if the parameters are quantized, it will be better than the original ones.

In the latter half of the semester, I designed a PE array based on the specifications in the SD4 paper. First, I connected 1 MAC into a PE that can read 9 pixels (a pixel of the image has 32 bits), then connected 4 PEs into PE rows so that four kernels can be counted at the same time, and finally connected 4 PE rows form PE arrays. After parallel operations, 16 kernels can be processed at one time, and we have also greatly improved the efficiency of calculating kernels in each image.

6. INTRODUCTION OF 5G MIMO DEMODULATION

A maximum likelihood decoder for a MIMO receiver functions by comparing the received signal vector to all possible noiseless received signals that could result from all potential transmitted signals. This method, under specific conditions, provides optimal performance by maximizing the likelihood of correct data detection. However, the complexity of such a decoder increases exponentially with the number of transmit antennas, making it impractical for large antenna arrays and high-order modulation schemes.

To solve the complexity of the above algorithm, the Fixed Sphere Decoder might be a choice of the solutions. It performs a search over only a fixed number of possible transmitted signals, generated by a small subset of all possible signals located around the received signal vector.

Now if there is a system with four antennas transmitting, we can use this method to filter out some values at each entry that are smaller than other options. The method I chose is each entry will select 1, 2, 4, and 4 sets of values, so compared to the maximum likelihood method that requires 256 calculations, we only calculate it 16+16+8=40 times.

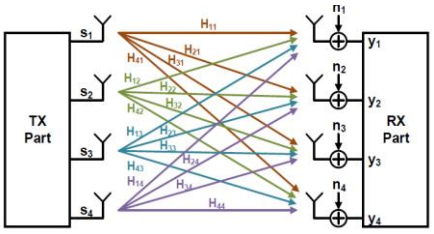

Fig.24 four antennas of MIMO system

s_1~s_4: one of $\left(\frac{1}{\sqrt{2}}+\frac{1}{\sqrt{2}}\mathbf{j}\right), \left(-\frac{1}{\sqrt{2}}+\frac{1}{\sqrt{2}}\mathbf{j}\right), \left(\frac{1}{\sqrt{2}}-\frac{1}{\sqrt{2}}\mathbf{j}\right)$, and $\left(-\frac{1}{\sqrt{2}}-\frac{1}{\sqrt{2}}\mathbf{j}\right)$

At $\sum_{i=1}^{4}\left|\left[\widehat{y_i}-\sum_{j=i}^{4}R_{ij}s_j\right]\right|^2$ part:

- 4th entry: $\widehat{y_4}-R_{44}s_4 = a+bj \rightarrow a^2+b^2$
- 3rd entry: $\widehat{y_3}-R_{33}s_3-R_{34}s_4 = c+dj \rightarrow c^2+d^2$
- 2nd entry: $\widehat{y_2}-R_{22}s_2-R_{23}s_3-R_{24}s_4 = e+fj \rightarrow e^2+f^2$
- 1st entry: $\widehat{y_1}-R_{11}s_1-R_{12}s_2-R_{13}s_3-R_{14}s_4 = g+hj \rightarrow g^2+h^2$

$\sum_{i=1}^{4}\left|\left[\widehat{y_i}-\sum_{j=i}^{4}R_{ij}s_j\right]\right|^2 = a^2+b^2+c^2+d^2+e^2+f^2+g^2+h^2$

Fig.25 Formula of Euclidean distance for each entry

7. MATLAB SIMULATION

Next, I will share several algorithms used in decoding communication networks, and I will also write the error rate obtained by each algorithm after actually running 1,000 tests using MATLAB and comparing it with the correct value in the following table.

	Soft decoder	Hard decoder	Zero forcing	Fixed sphere decoder
error rate(error/1000)	0.063	0.215	0.525	0.085
count	256	256	256	40
How to count	First calculate all data LLR then decide 0 or 1	First decide 0 or 1 of all data then calculate LLR	calculate inverse matrix(ignore noise)	calculate partial data using sphere decoder then decide some min at each entry respectively
complexity	4	2	1	3
accuracy	1	3	4	2
performance	2	3	4	1

Fig.26 The characteristic of some algorithms

Among them, we can first observe that the soft decoder calculates LLR for each possibility and then compares the minimum value, so the accuracy will be the highest.

Next is the method we finally chose — fixed sphere decoder. It first projects the data on a two-dimensional plane, uses four points to be evenly distributed on the sphere and calculates the distance. We will filter out how many points to retain based on different entries. minimum value, and then calculate the next entry.

Finally, because the error rate of zero forcing and hard decoder is too high, we can quickly eliminate it.

8. HARDWARE DESIGN OF FIXED SPHERE DECODER

After deciding on the algorithm, we start implementing it on the hardware.

(1) Calculator (algorithm implement)

A. First we can store all 4 values at the 4th entry, no sorting is needed at this entry.

```
always @(*) begin
    for(i=0; i<4; i=i+1) begin
        _4th_entry_dis_w[i] = _4th_entry_dis_r[i];
    end
    if((state_r == CAL_4TH_ENTRY && _4th_counter > 2) || (state_r == BUF_4TH)) begin
        _4th_entry_dis_w[_4th_counter_pipe] = general_entry_diff_square;
    end
    else if(state_r == IDLE) begin
        for(i=0; i<4; i=i+1) begin
            _4th_entry_dis_w[i] = max_value;
        end
    end
end
```

Fig.27 Implementation of 4th entry

B. Next, since we need to find the 4 smallest values among the 16 possible values at the 3rd entry. We use four registers to store labels.

```
if(_3rd_ped <= _3rd_entry_dis_r[0]) begin
    _3rd_entry_dis_w[3] = _3rd_entry_dis_r[2];
    _3rd_entry_label_w[3] = _3rd_entry_label_r[2];
    _3rd_entry_dis_w[2] = _3rd_entry_dis_r[1];
    _3rd_entry_label_w[2] = _3rd_entry_label_r[1];
    _3rd_entry_dis_w[1] = _3rd_entry_dis_r[0];
    _3rd_entry_label_w[1] = _3rd_entry_label_r[0];
    _3rd_entry_dis_w[0] = _3rd_ped;
    _3rd_entry_label_w[0] = _3rd_entry_label;
end
else if(_3rd_ped < _3rd_entry_dis_r[1]) begin
    _3rd_entry_dis_w[3] = _3rd_entry_dis_r[2];
    _3rd_entry_label_w[3] = _3rd_entry_label_r[2];
    _3rd_entry_dis_w[2] = _3rd_entry_dis_r[1];
    _3rd_entry_label_w[2] = _3rd_entry_label_r[1];
    _3rd_entry_dis_w[1] = _3rd_ped;
    _3rd_entry_label_w[1] = _3rd_entry_label;
end
else if(_3rd_ped < _3rd_entry_dis_r[2]) begin
    _3rd_entry_dis_w[3] = _3rd_entry_dis_r[2];
    _3rd_entry_label_w[3] = _3rd_entry_label_r[2];
    _3rd_entry_dis_w[2] = _3rd_ped;
    _3rd_entry_label_w[2] = _3rd_entry_label;
end
else if(_3rd_ped < _3rd_entry_dis_r[3]) begin
    _3rd_entry_dis_w[3] = _3rd_ped;
    _3rd_entry_label_w[3] = _3rd_entry_label;
end
```

Fig.28 Implementation of 3rd entry

C. Third, since we need to find the 2 smallest values among the 16 possible values at the 2nd entry. We use two registers to store labels.

```
if(_2nd_ped <= _2nd_dis_r[0]) begin
    _2nd_dis_w[1] = _2nd_dis_r[0];
    _2nd_label_w[1] = _2nd_label_r[0];
    _2nd_dis_w[0] = _2nd_ped;
    _2nd_label_w[0] = _2nd_label;
end
else if(_2nd_ped < _2nd_dis_r[1]) begin
    _2nd_dis_w[1] = _2nd_ped;
    _2nd_label_w[1] = _2nd_label;
end
```

Fig.29 Implementation of 2nd entry

D. Fourth, since we need to find the smallest values among the 8 possible values at the 1st entry. We use one register to store the final label.

```
if(_1st_ped < _1st_dis_r) begin
    _1st_dis_w = _1st_ped;
    _1st_label_w = _1st_label;
end
```

Fig.30 Implementation of 1st entry

E. In the end, we only need to calculate the distance for each entry to complete the calculation block.

(2) Finite State Machine

we begin from an IDLE state. Then, we initiate our calculation process by finding the specified minimum values for 4th entry to 1st entry. After completing the computation for the 1st entry, we transition to the OUT state.

```
case (state_r)
    IDLE : begin
        state_w = i_trig ? CAL_4TH_ENTRY : IDLE;
    end
    CAL_4TH_ENTRY : begin
        state_w = _4th_out_valid ? CAL_3RD_ENTRY : CAL_4TH_ENTRY;
    end
    CAL_3RD_ENTRY : begin
        state_w = _3rd_out_valid ? CAL_2ND_ENTRY : CAL_3RD_ENTRY;
    end
    CAL_2ND_ENTRY: begin
        state_w = _2nd_out_valid ? CAL_1ST_ENTRY : CAL_2ND_ENTRY;
    end
    CAL_1ST_ENTRY: begin
        state_w = _1st_out_valid ? OUTPUT : CAL_1ST_ENTRY;
    end
    OUTPUT : begin
        state_w = IDLE;
    end
endcase
```

Fig.31 Implementation of FSM

(3) Output

After we enter the output state, we only need to output the final label (1st label) we got (It will eventually be compared with the original input value that hasn't entered the communication system).

```
always @(*) begin
    store_flag = (state_r == OUTPUT);
    hb_to_store = _1st_label_r;
```

Fig.32 Implementation of Output

9. THE PERFORMANCE OF FIXED SPHERE DECODER

We use six difference testing packets to test the performance of this algorithm.

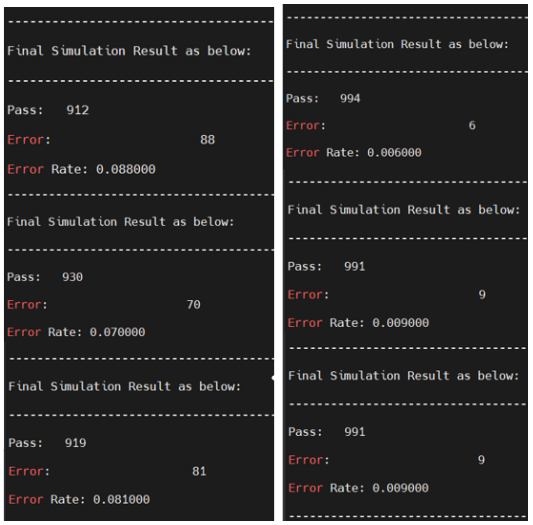

Fig.33 Error rate for each testing packet (Left : SNR=10dB, Right : SNR=15dB)

We can see that when the SNR (the ratio of signal to noise) is 15dB, the error rate is less than 1%. This algorithm not only reduces a lot of calculations, but also doesn't lose its original high accuracy.

10. CONCLUSION OF 5G MIMO DEMODULATION

On this topic, I first looked at the most typical decoder, which has a fairly high accuracy. However, once the number of antennas in the system increases, the high complexity will cause the speed to slow down. If we use the algorithm we mentioned above, not only can the speed be increased by at least 6 times, but the accuracy performance is also very good, so In the MIMO system, a specific number of smaller values should be appropriately selected so that points farther away can be excluded with high probability.

As for the future outlook, I hope to extend the number of antennas from 4 to n and find out how to express several smaller values for individual entries (perhaps there is no rule in this expression). Another question that can be extended is whether there is an algorithm with less calculation but better accuracy. Therefore, these two directions are possible motivations for future research.

REFERENCE

[1] Po-Chen Lin, Mu-Kai Sun , Chuking Kung, and Tzi-Dar Chiueh , *Fellow, IEEE "*FloatSD: A New Weight Representation and Associated Update Method for Efficient Convolutional Neural Network Training*"*

[2] Ming-Hang Hsieh, Yu-Tung Liu, and Tzi-Dar Chiueh, "A Multiplier-Less Convolutional Neural Network Inference Accelerator for Intelligent Edge Devices" in *IEEE Journal on Emerging and Selected Topics in Circuits and Systems,* 2021, p.739-750

[3] Q. Qi, C. Chakrabarti "Parallel High Throughput Soft output Sphere Decoder" in School of Electrical, Computer and Energy Engineering Arizona State University

[4] A. Shirly Edward , S. Malarvizhi "Architectural implementation of modified K-best algorithm for detection in MIMO systems"

[5] Yu Hsuan Tsai , Powerpoint:"5G MIMO Demodulation" in Graduate Institute of Electronics Engineering, National Taiwan University