Fault-Tolerant Systolic Array Based Accelerators for Deep Neural Network Execution

Introduction:

With the rapid development of AI, the demand for AI-accelerated chips is getting higher and higher.

In order to achieve faster and more efficient computing AI models, custom AI or SOC chips have become a trend.

The following figure shows the basic architecture of the AI accelerator, or Neural-network Processing Unit (NPU) which is mainly composed of Systolic Array (SA), whose advantage is that the matrix operation is fast, and because it does not need to frequently access the memory, so it is much more energy efficient than CPU and GPU.

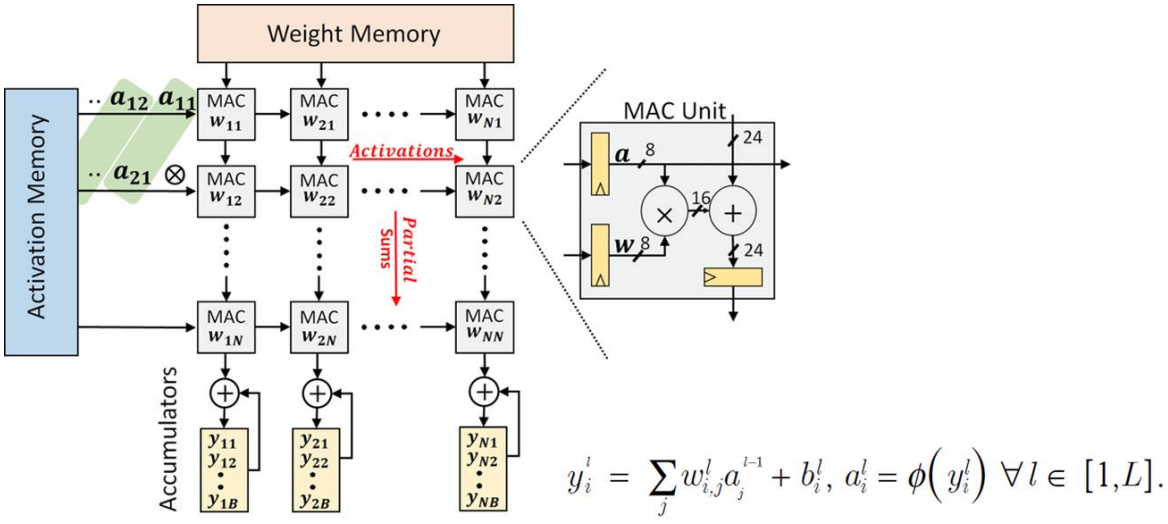


Fig. 1 Basic architecture of the AI accelerator [1]

However, because the characteristic of Systolic array is passed one by one, and the number of PEs is large, the operation error of a PE often becomes a fatal error of neural network. Fig. 2 show the classification accuracy drop due to stuck-at fault MACs. Therefore, we have adopted the following design to try to avoid operation errors caused by hardware errors, which is showed as Fig. 3.

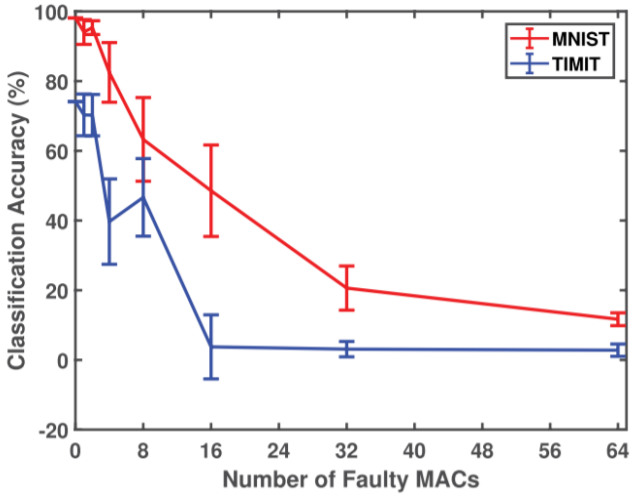


Fig. 2 Impact of classification accuracy drop due to stuck-at fault MACs

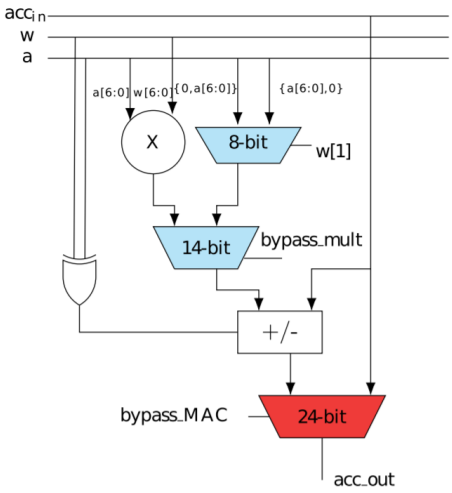
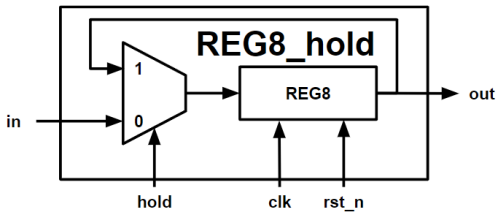


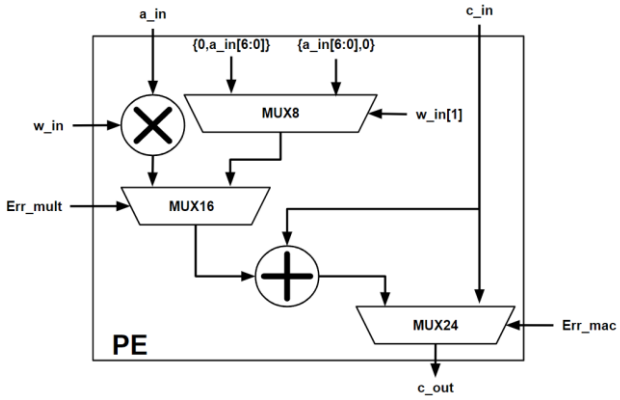
Fig. 3 Illustration of Fault-Tolerant MAC unit

Experimental and programming:



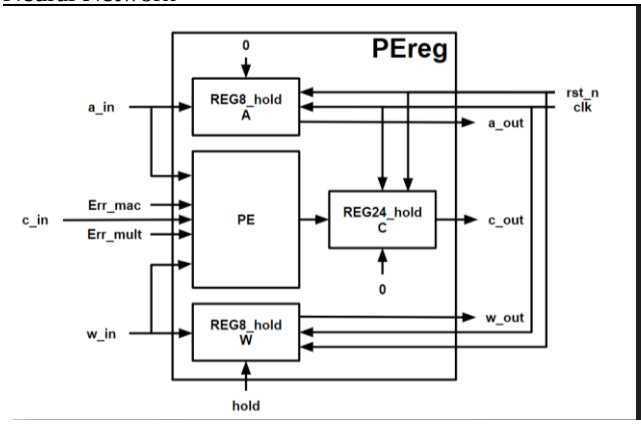

```verilog
module REG8_hold(clk,rst_n,in,out,s);
parameter N=8;
input clk,s,rst_n;
input signed [N-1:0] in;
output signed [N-1:0] out;
reg signed [N-1:0] in_wire;
reg signed [N-1:0] out;
always @ (*)
begin
    case(s)
    1'b0:in_wire <= in;
    1'b1:in_wire <= out;
    endcase
end
always @ (posedge clk or negedge rst_n)
begin
    if (!rst_n) out <= {N{1'b0}};
    else out <= in_wire;
end
endmodule
```

Fig. 4 REG diagram & Verilog code




```verilog
module PE(Err_mac, Err_mult, w_in, a_in, c_in, c_out);
input signed [7:0] w_in, a_in;
input signed [23:0] c_in;
input Err_mac , Err_mult;

output signed [23:0] c_out;

wire signed [15:0] product;
wire signed [7:0] shift;
wire signed [15:0] acc_in;
//reg signed [23:0] acc_in_24;
wire signed [23:0] acc_out;

//multiplier multi(product, w_in, a_in);
assign product = w_in * a_in;
MUX8 sft(w_in[1],{1'b0,a_in[6:0]},{a_in[6:0],1'b0},shift);
MUX16 bypass_mult(Err_mult,product,{8'h00,shift},acc_in);
MUX24 bypass_MAC(Err_mac,acc_out,c_in,c_out);
//wire [23:0] acc_in_24 = { 8{acc_in[15]} , acc_in[15:0] };

/*
always @ (*)
begin
    //if(acc_in[7]==1) acc_in_24 = {8'b1111_1111,acc_in};
    //else acc_in_24 = {8'b0000_0000,acc_in};

end
*/

assign acc_out = c_in + acc_in;
endmodule
```

Fig. 5 PE diagram & Verilog code




```v
module PEreg(Err_mac, Err_mult, w_in, a_in, c_in, w_out, a_out, c_out, clk, rst_n,
hold);

input signed [7:0] w_in, a_in;
input signed [23:0] c_in;
input Err_mac , Err_mult;
input clk;
input rst_n;
input hold;

output signed [7:0] w_out, a_out;
output signed [23:0] c_out;

wire signed [23:0] p_sum;

PE PE_no_reg(Err_mac, Err_mult, w_out, a_out, c_in, p_sum);
//PE(Err_mac, Err_mult, w_in, a_in, c_in, c_out);

REG8_hold regW(clk, rst_n, w_in, w_out, hold);
REG8_hold regA(clk, rst_n, a_in, a_out, 1'b0);
REG24_hold regC(clk, rst_n, p_sum, c_out, 1'b0);

endmodule
```

Fig. 6 PEreg diagram & Verilog code

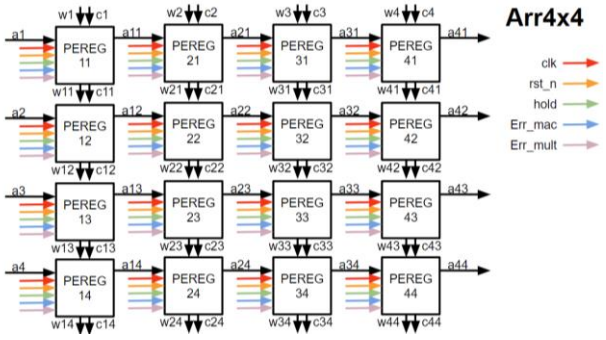


Fig. 7 Arr4x4 diagram

```v
module Arr4x4(Err_mac, Err_mult,
              w1_in, w2_in, w3_in, w4_in,
              a1_in, a2_in, a3_in, a4_in,
              w1_out, w2_out, w3_out, w4_out,
              a1_out, a2_out, a3_out, a4_out,
              c1_out, c2_out, c3_out, c4_out,
              clk, rst_n, hold);
//PEreg(Err_mac, Err_mult, w_in, a_in, c_in, w_out, a_out, c_out, clk, rst_n, hold);
input [7:0] w1_in, w2_in, w3_in, w4_in;
input [7:0] a1_in, a2_in, a3_in, a4_in;
input clk, rst_n, hold;
input [15:0] Err_mac, Err_mult;

output [7:0] w1_out, w2_out, w3_out, w4_out;
output [7:0] a1_out, a2_out, a3_out, a4_out;
output [23:0] c1_out, c2_out, c3_out, c4_out;



wire [7:0]  w11_out, w21_out, w31_out, w41_out,
            w12_out, w22_out, w32_out, w42_out,
            w13_out, w23_out, w33_out, w43_out,
            w14_out, w24_out, w34_out, w44_out;
wire [7:0]  a11_out, a21_out, a31_out, a41_out,
            a12_out, a22_out, a32_out, a42_out,
            a13_out, a23_out, a33_out, a43_out,
            a14_out, a24_out, a34_out, a44_out;
wire [23:0] c11_out, c21_out, c31_out, c41_out,
            c12_out, c22_out, c32_out, c42_out,
            c13_out, c23_out, c33_out, c43_out,
            c14_out, c24_out, c34_out, c44_out;

//PEreg(Err_mac, Err_mult, w_in, a_in, c_in, w_out, a_out, c_out, clk1, clk2);
PEreg PE11( Err_mac[0],  Err_mult[0], w1_in, a1_in,   24'sh00_0000, w11_out, a11_out, c11_out, clk, rst_n, hold);
PEreg PE21( Err_mac[1],  Err_mult[1], w2_in, a11_out, 24'sh00_0000, w21_out, a21_out, c21_out, clk, rst_n, hold);
PEreg PE31( Err_mac[2],  Err_mult[2], w3_in, a21_out, 24'sh00_0000, w31_out, a31_out, c31_out, clk, rst_n, hold);
PEreg PE41( Err_mac[3],  Err_mult[3], w4_in, a31_out, 24'sh00_0000, w41_out, a41_out, c41_out, clk, rst_n, hold);
```

```
39   PEreg PE12( Err_mac[4],  Err_mult[4], w11_out, a2_in,   c11_out, w12_out, a12_out, c12_out, clk, rst_n, hold);
40   PEreg PE22( Err_mac[5],  Err_mult[5], w21_out, a12_out, c21_out, w22_out, a22_out, c22_out, clk, rst_n, hold);
41   PEreg PE32( Err_mac[6],  Err_mult[6], w31_out, a22_out, c31_out, w32_out, a32_out, c32_out, clk, rst_n, hold);
42   PEreg PE42( Err_mac[7],  Err_mult[7], w41_out, a32_out, c41_out, w42_out, a42_out, c42_out, clk, rst_n, hold);
43
44   PEreg PE13( Err_mac[8],  Err_mult[8], w12_out, a3_in,   c12_out, w13_out, a13_out, c13_out, clk, rst_n, hold);
45   PEreg PE23( Err_mac[9],  Err_mult[9], w22_out, a13_out, c22_out, w23_out, a23_out, c23_out, clk, rst_n, hold);
46   PEreg PE33(Err_mac[10], Err_mult[10], w32_out, a23_out, c32_out, w33_out, a33_out, c33_out, clk, rst_n, hold);
47   PEreg PE43(Err_mac[11], Err_mult[11], w42_out, a33_out, c42_out, w43_out, a43_out, c43_out, clk, rst_n, hold);
48
49   PEreg PE14(Err_mac[12], Err_mult[12], w13_out, a4_in,   c13_out, w14_out, a14_out, c14_out, clk, rst_n, hold);
50   PEreg PE24(Err_mac[13], Err_mult[13], w23_out, a14_out, c23_out, w24_out, a24_out, c24_out, clk, rst_n, hold);
51   PEreg PE34(Err_mac[14], Err_mult[14], w33_out, a24_out, c33_out, w34_out, a34_out, c34_out, clk, rst_n, hold);
52   PEreg PE44(Err_mac[15], Err_mult[15], w43_out, a34_out, c43_out, w44_out, a44_out, c44_out, clk, rst_n, hold);
53   //PEreg(Err_mac, Err_mult, w_in, a_in, c_in, w_out, a_out, c_out, clk, rst_n, hold);
54
55   assign c1_out = c14_out;
56   assign c2_out = c24_out;
57   assign c3_out = c34_out;
58   assign c4_out = c44_out;
59
60   assign w1_out = w14_out;
61   assign w2_out = w24_out;
62   assign w3_out = w34_out;
63   assign w4_out = w44_out;
64
65   assign a1_out = a41_out;
66   assign a2_out = a42_out;
67   assign a3_out = a43_out;
68   assign a4_out = a44_out;
69
70   endmodule
```

Fig. 8 Arr4x4 Verilog code

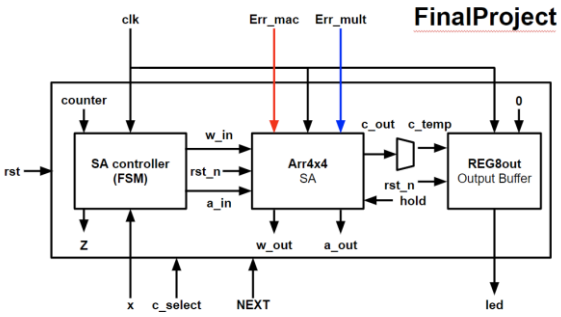


Fig. 9 FinalProject diagram

```
1   module FinalProject(clk,x,rst,led,Err_mac_in,Err_mult_in,c_select,NEXT);
2
3   input NEXT;
4   input clk;
5   input x, rst;
6   input Err_mac_in, Err_mult_in;
7   input [1:0] c_select;
8
9   output [7:0] led;
10
11  parameter S0 = 2'b00,S1 = 2'b01,S2=2'b10;
12
13  assign rst_n = !rst;
14
15  wire [1:0] z;
16  wire [3:0] counter;
17
18  reg [7:0] w1_in;
19  reg [7:0] w2_in;
20  reg [7:0] w3_in;
21  reg [7:0] w4_in;
22
23  reg [7:0] a1_in;
24  reg [7:0] a2_in;
25  reg [7:0] a3_in;
26  reg [7:0] a4_in;
27
28  reg [7:0] c_temp;
```

```
35  reg hold;
36  reg clk_s;
37  reg [28:0] clk_counter;
38  reg [15:0] Err_mac, Err_mult;
39  always @(*)
40  begin
41      if(Err_mac_in)
42      Err_mac = 16'b0000_0000_0001_0000;
43      else
44      Err_mac = 16'b0000_0000_0000_0000;
45
46      if(Err_mult_in)
47      Err_mult = 16'b0000_0010_0000_0000;
48      else
49      Err_mult = 16'b0000_0000_0000_0000;
50  end
51
52  FSM fsm(clk,x,z,rst,counter);
53  REG8out temp(clk_s, rst_n, c_temp, led, 1'b0);
54  Arr4x4 SA(Err_mac, Err_mult,
55                w1_in, w2_in, w3_in, w4_in,
56                a1_in, a2_in, a3_in, a4_in,
57                w1_out, w2_out, w3_out, w4_out,
58                a1_out, a2_out, a3_out, a4_out,
59                c1_out, c2_out, c3_out, c4_out,
60                clk, rst_n, hold);
61
```

```verilog
always @(*)
begin
    case(c_select)
        2'b00 :  begin c_temp = c1_out[7:0];end
        2'b01 :  begin c_temp = c2_out[7:0];end
        2'b10 :  begin c_temp = c3_out[7:0];end
        2'b11 :  begin c_temp = c4_out[7:0];end
        default :  c_temp = c1_out[7:0];
    endcase
end
```

```verilog
always @(*)
begin
        case(z)
            S0 : begin
                 clk_s = NEXT;
                 hold = 0;
                 end
            S1 : begin
                    hold = 0;
                    clk_s = clk;
                    case(counter)
                    4'h0 : begin w1_in=4; w2_in=3; w3_in=2; w4_in=1; end
                    4'h1 : begin w1_in=8; w2_in=7; w3_in=6; w4_in=5; end
                    4'h2 : begin w1_in=4; w2_in=3; w3_in=2; w4_in=1; end
                    4'h3 : begin w1_in=8; w2_in=7; w3_in=6; w4_in=5; end
                    endcase
                 end
            S2 : begin
                    hold = 1;
                    clk_s = clk;
                    case(counter)
                    4'h0 :begin a1_in=8; a2_in=0; a3_in=0; a4_in=0; end
                    4'h1 :begin a1_in=7; a2_in=8; a3_in=0; a4_in=0; end
                    4'h2 :begin a1_in=6; a2_in=7; a3_in=8; a4_in=0; end
                    4'h3 :begin a1_in=5; a2_in=6; a3_in=7; a4_in=8; end
                    4'h4 :begin a1_in=4; a2_in=5; a3_in=6; a4_in=7; end
                    4'h5 :begin a1_in=3; a2_in=4; a3_in=5; a4_in=6; end
                    4'h6 :begin a1_in=2; a2_in=3; a3_in=4; a4_in=5; end
                    4'h7 :begin a1_in=1; a2_in=2; a3_in=3; a4_in=4; end
                    4'h8 :begin a1_in=0; a2_in=1; a3_in=2; a4_in=3; end
                    4'h9 :begin a1_in=0; a2_in=0; a3_in=1; a4_in=2; end
                    4'ha :begin a1_in=0; a2_in=0; a3_in=0; a4_in=1; end
                    4'hb :begin a1_in=0; a2_in=0; a3_in=0; a4_in=0; end
                    default: begin a1_in=0; a2_in=0; a3_in=0; a4_in=0; end
                    endcase
                 end
        endcase
end
endmodule
```

Fig. 10 FinalProject Verilog code

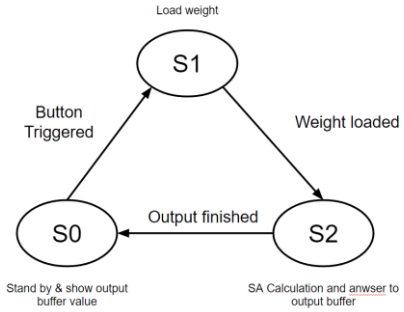


Fig. 11 FSM diagram

```verilog
module FSM(clk,x,z,rst,counter);

input clk;
input x, rst;

output [1:0] z;
output reg [3:0] counter;

reg [1:0] Z;
reg [2:0] counter1;
reg [3:0] counter2;
reg [1:0] next_state;
reg [1:0] current_state;
//state encoding
parameter S0 = 2'b00,S1 = 2'b01,S2=2'b10;
```

```verilog
//counter1
always @(posedge clk , posedge rst)
begin
    if(rst==1)
        counter1 <= 3'b000;
    else if(counter1 < 3'b011 && current_state==S1)
        counter1 <= counter1 +1;
    else
        counter1 <= 3'b000;
end
```

```
//counter2
always @(posedge clk , posedge rst)
begin
    if(rst==1)
        counter2 <= 4'b0000;
    else if(counter2 < 4'b1110 && current_state==S2)
        counter2 <= counter2 +1;
    else
        counter2 <= 4'b0000;
end

//CS
always @(posedge clk , posedge rst)
begin
    if(rst==1)
        begin
        current_state <= S0;


        end
    else
        current_state <= next_state;
end

//Z <= 2'b00;
//next_state <= S0;
//OL
assign z = Z;
always @(*)
begin
if(current_state == S1)
        counter <= counter1;
    else
        counter <= counter2;
end
```

```
//NL
always @(*)
begin
        case(current_state)
            S0 : begin
                Z = 2'b00;
                        if(x == 1)begin
                                next_state <= S1;

                                end
                        else begin
                                next_state <= S0;

                            end
                    end

            S1 : begin
                Z = 2'b01;
                        if(counter1 < 3'b011)begin
                                next_state <= S1;
                                end
                        else    begin
                                next_state <= S2;

                                end
                    end

            S2 : begin
                Z = 2'b10;
                        if(counter2 < 4'b1110)begin
                                next_state <= S2;

                                end
                        else    begin
                                next_state <= S0;

                                end
                    end
            default: next_state <= current_state;

        endcase
end
```

Fig. 12 FSM Verilog code

Results:

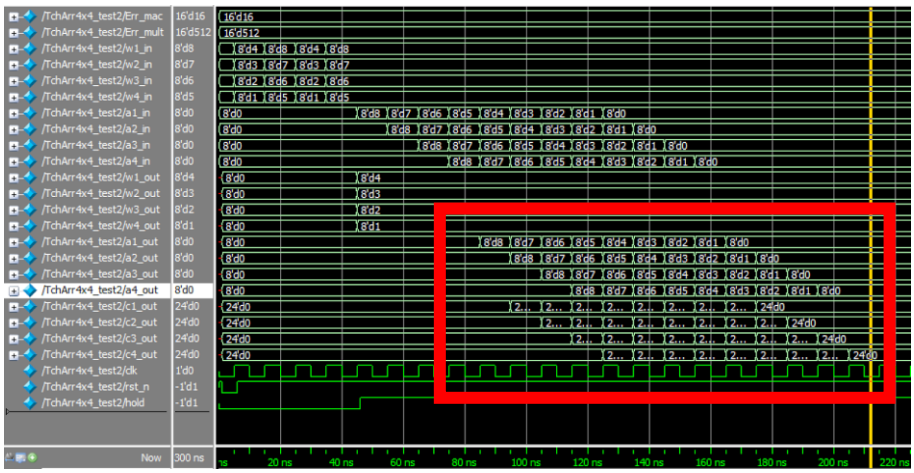


Fig. 13 simulation result

Reference:

[1] J. Zhang, Z. Ghodsi, S. Garg and K. Rangineni, "Enabling Timing Error Resilience for

Low-Power Systolic-Array Based Deep Learning Accelerators," in IEEE Design & Test, vol. 37, no. 2, pp. 93-102, April 2020, doi: 10.1109/MDAT.2019.2947271.

[2] J. J. Zhang, K. Basu, and S. Garg. Fault-tolerant systolic array based accelerators for deep neural network execution. IEEE Design Test, 2019.