

dchip

一維一階分解

程式碼

```
module TOP(clk, reset, in_data, d1_out, d2_out, d3_out, d4_out, ld1_out,
ld2_out, ld3_out, ld4_out, ld5_out, L_out, H_out);
    input clk;
    input reset;
    input [7:0] in_data;
    output [7:0] d1_out, d2_out, d3_out, d4_out;
    output [7:0] ld1_out, ld2_out, ld3_out, ld4_out, ld5_out;
    output [7:0] L_out;
    output [7:0] H_out;

    parameter ODD = 1'd0;
    parameter EVEN = 1'd1;
    reg [7:0] d_out [3:0];
    reg [7:0] ld_out [4:0];

    wire [7:0] sub1;
    wire [7:0] sub2;

    assign sub1 = in_data - d_out[0];
    assign sub2 = d_out[2] - d_out[0];
    assign H_out = d_out[3];
    assign L_out = ld_out[4];
    assign d1_out = d_out[0];
    assign d2_out = d_out[1];
    assign d3_out = d_out[2];
    assign d4_out = d_out[3];
    assign ld1_out = ld_out[0];
    assign ld2_out = ld_out[1];
    assign ld3_out = ld_out[2];
    assign ld4_out = ld_out[3];
    assign ld5_out = ld_out[4];
    reg state, nstate;

    always@(posedge clk, posedge reset)begin
        if(reset)begin
            state <= ODD;
        end
        else begin
            state <= nstate;
        end
    end
end
```

```

always@(*)begin
    case(state)
        ODD : nstate = EVEN;
        EVEN : nstate = ODD;
        default: nstate = ODD;
    endcase
end

always@(posedge clk, posedge reset)begin
    if(reset)begin
        d_out[0] <= 8'd0;
        d_out[1] <= 8'd0;
    end
    else begin
        case(state)
            ODD : d_out[0] <= in_data >> 1;
            EVEN: d_out[1] <= sub1;
        endcase
    end
end

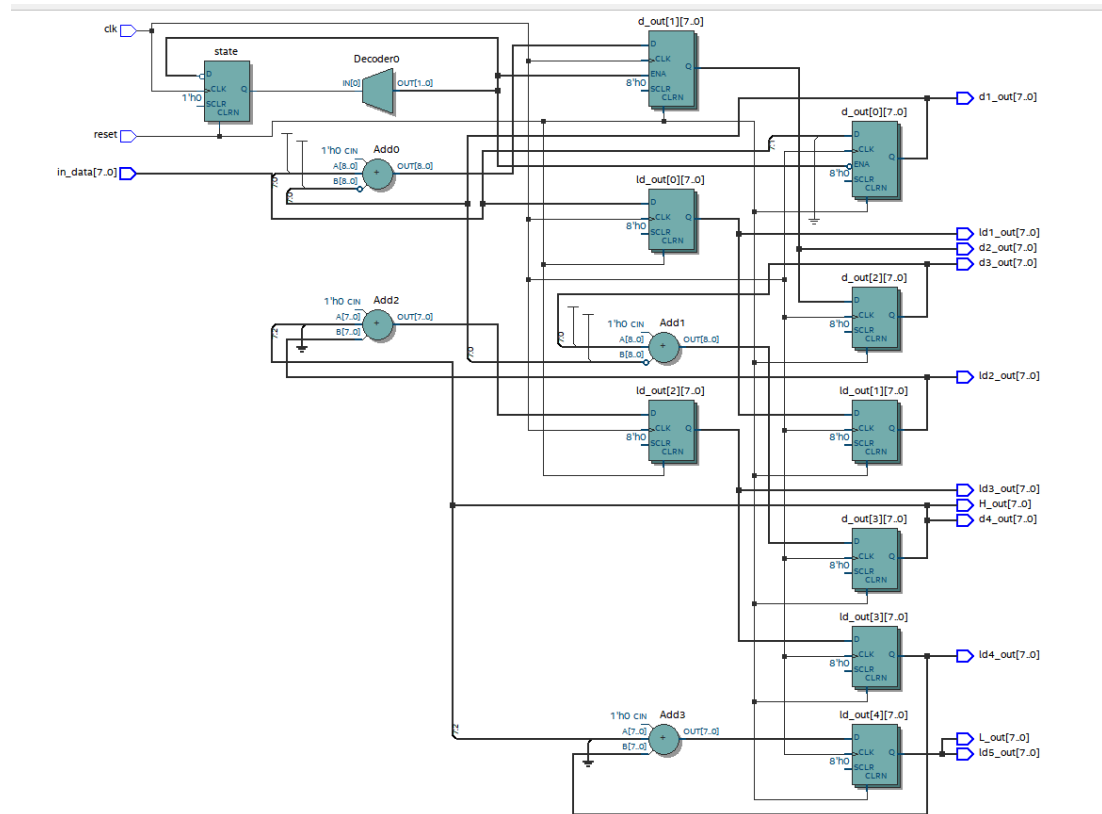
always@(posedge clk, posedge reset)begin
    if(reset)begin
        d_out[2] <= 8'd0;
        d_out[3] <= 8'd0;
    end
    else begin
        d_out[2] <= d_out[1];
        d_out[3] <= sub2;
    end
end

integer x;
always@(posedge clk, posedge reset)begin
    if(reset)begin
        for (x = 0; x < 5; x = x + 1) begin
            ld_out[x] <= 8'd0;
        end
    end
    else begin
        ld_out[0] <= in_data;
        ld_out[1] <= ld_out[0];
        ld_out[2] <= (d_out[3] >> 2) + ld_out[1];
        ld_out[3] <= ld_out[2];
        ld_out[4] <= (d_out[3] >> 2) + ld_out[3];
    end
end

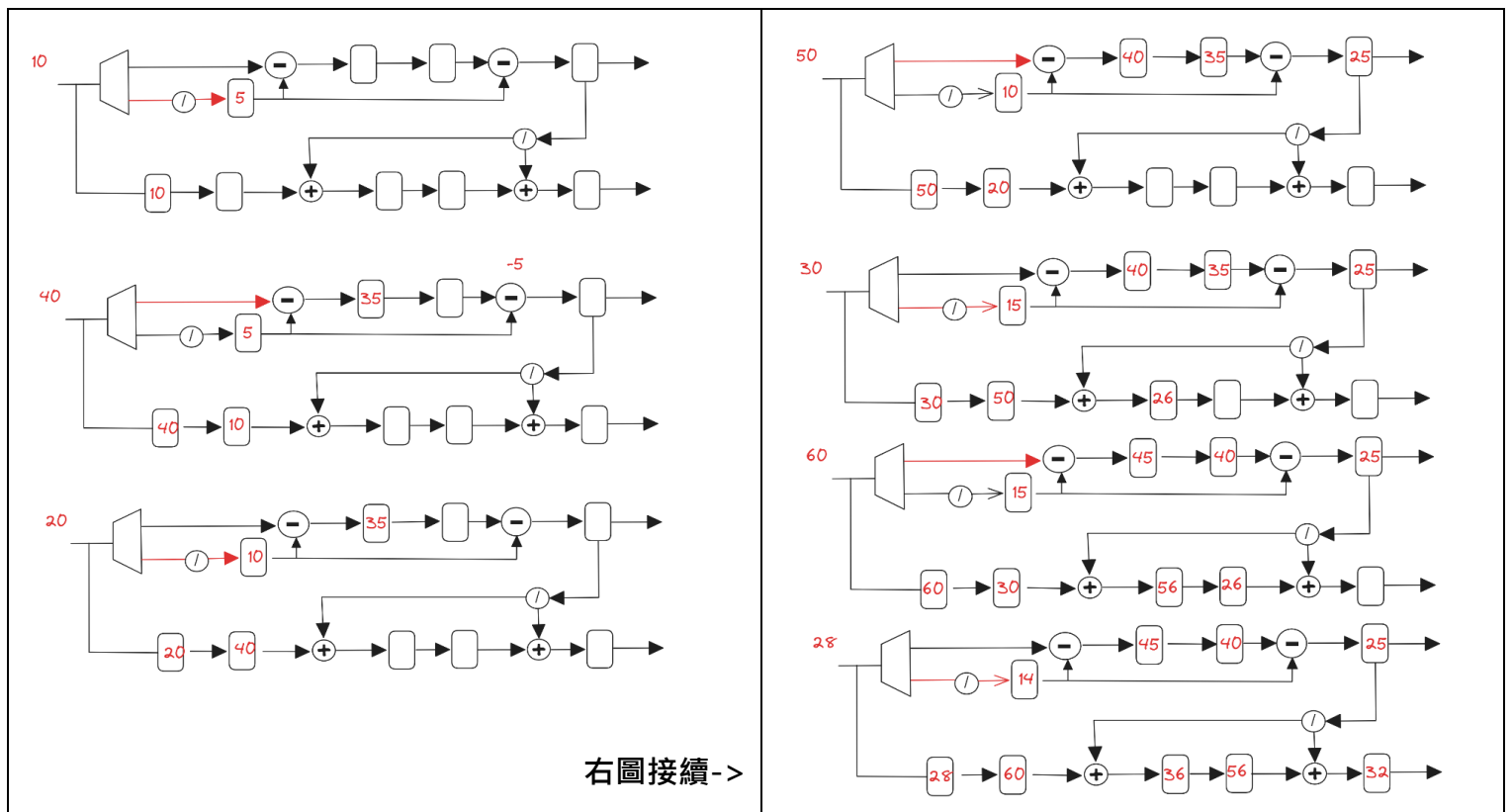
```

endmodule

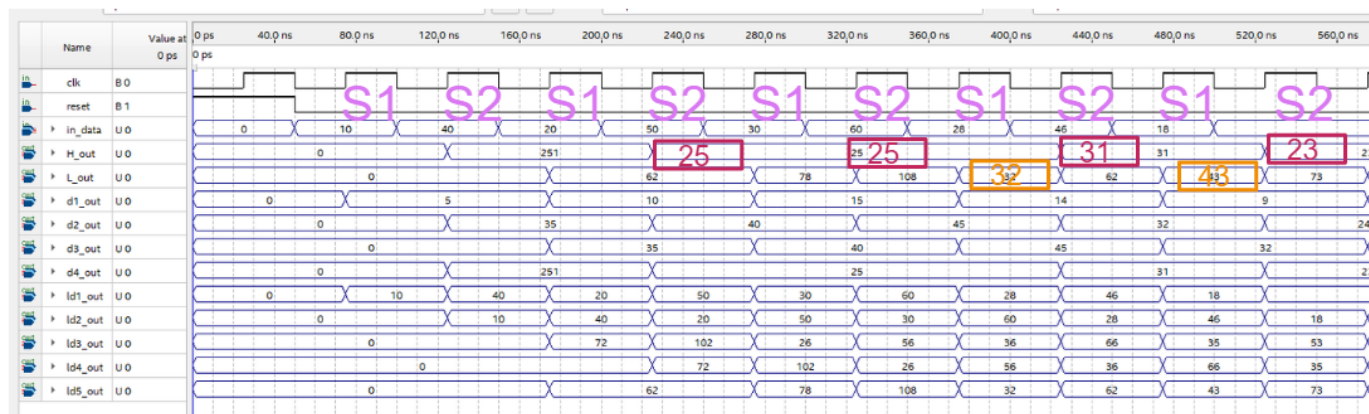
電路 RTL 架構圖



時序架構圖



波形圖



狀態
 H1
 L1

一維二階分解

程式碼

```
module TOP(clk, reset, in_data, d1_out, d2_out, d3_out, d4_out, ld1_out,
ld2_out, ld3_out, ld4_out, ld5_out, L_out, H_out);
    input clk;
    input reset;
    input [7:0] in_data;
    output [7:0] d1_out, d2_out, d3_out, d4_out;
    output [7:0] ld1_out, ld2_out, ld3_out, ld4_out, ld5_out;
    output [7:0] L_out;
    output [7:0] H_out;

    parameter S1 = 2'd0;
    parameter S2 = 2'd1;
    parameter S3 = 2'd2;
    parameter S4 = 2'd3;

    reg [7:0] d_out[3:0];
    reg [7:0] ld_out[4:0];

    assign H_out = d_out[3];
```

```

assign L_out = ld_out[4];
assign d1_out = d_out[0];
assign d2_out = d_out[1];
assign d3_out = d_out[2];
assign d4_out = d_out[3];
assign ld1_out = ld_out[0];
assign ld2_out = ld_out[1];
assign ld3_out = ld_out[2];
assign ld4_out = ld_out[3];
assign ld5_out = ld_out[4];
wire [7:0] mux_a;
wire [7:0] mux_b;
wire [7:0] mux_c;
wire [7:0] mux_d;
wire [7:0] mux_e;
reg [1:0] state, nstate;
reg [15:0] reg_b;
reg [15:0] reg_c;
reg [31:0] reg_d;

assign mux_a = (state == S1) ? in_data :
               (state == S3) ? in_data :
               reg_d[15:8];
assign mux_b = (state == S2) ? in_data :
               (state == S4) ? in_data :
               reg_d[7:0];
assign mux_c = (state == S4) ? d_out[2] :
               (state == S2) ? d_out[2] :
               reg_b[7:0];
assign mux_d = (state == S3) ? ld_out[1] :
               (state == S1) ? ld_out[1] :
               reg_d[31:24];

assign mux_e = (state == S4) ? reg_c[7:0] :
               (state == S2) ? reg_c[7:0] :
               ld_out[3];

always@(posedge clk, posedge reset)begin

```

```

    if(reset)begin
        state <= S1;
    end
    else begin
        state <= nstate;
    end
end

always@(*)begin
    case(state)
        S1 :    nstate = S2;
        S2 :    nstate = S3;
        S3 :    nstate = S4;
        S4 :    nstate = S1;
        default: nstate = S1;
    endcase
end

integer x;
always@(posedge clk, posedge reset)begin
    if(reset)begin
        for (x = 0; x < 4; x = x + 1)begin
            d_out[x] <= 8'd0;
        end
    end
    else begin
        d_out[0] <= mux_a >> 1;
        d_out[1] <= mux_b - d_out[0];
        d_out[2] <= d_out[1];
        d_out[3] <= mux_c - d_out[0];
    end
end

always@(posedge clk, posedge reset)begin
    if(reset)begin
        for (x = 0; x < 5; x = x + 1)begin
            ld_out[x] <= 8'd0;
        end
    end
end

```

```

    end
    else begin
        ld_out[0] <= in_data;
        ld_out[1] <= ld_out[0];
        ld_out[2] <= (d_out[3] >> 2) + mux_d;
        ld_out[3] <= ld_out[2];
        ld_out[4] <= (d_out[3] >> 2) + mux_e;
    end
end

always@(posedge clk, posedge reset)begin
    if(reset)begin
        reg_b <= 16'd0;
    end
    else begin
        reg_b <= {d_out[2], reg_b[15:8]};
    end
end

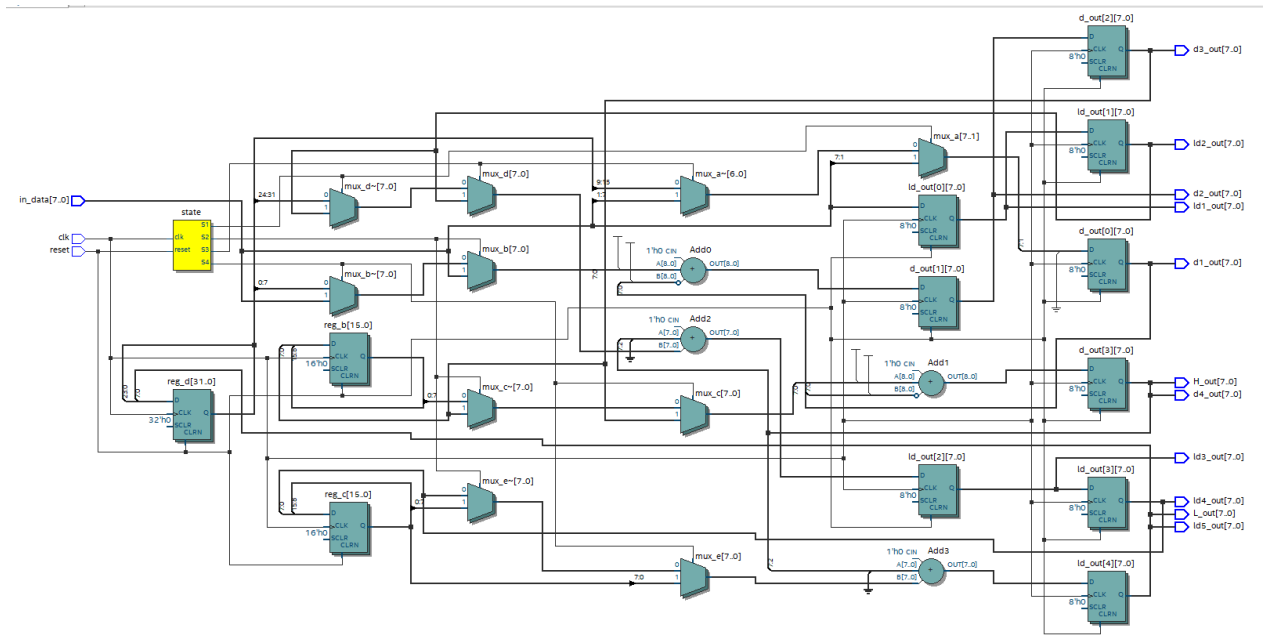
always@(posedge clk, posedge reset)begin
    if(reset)begin
        reg_c <= 16'd0;
    end
    else begin
        reg_c <= {ld_out[3], reg_c[15:8]};
    end
end

always@(posedge clk, posedge reset)begin
    if(reset)begin
        reg_d <= 31'd0;
    end
    else begin
        reg_d <= {reg_d[23:0], ld_out[4]};
    end
end

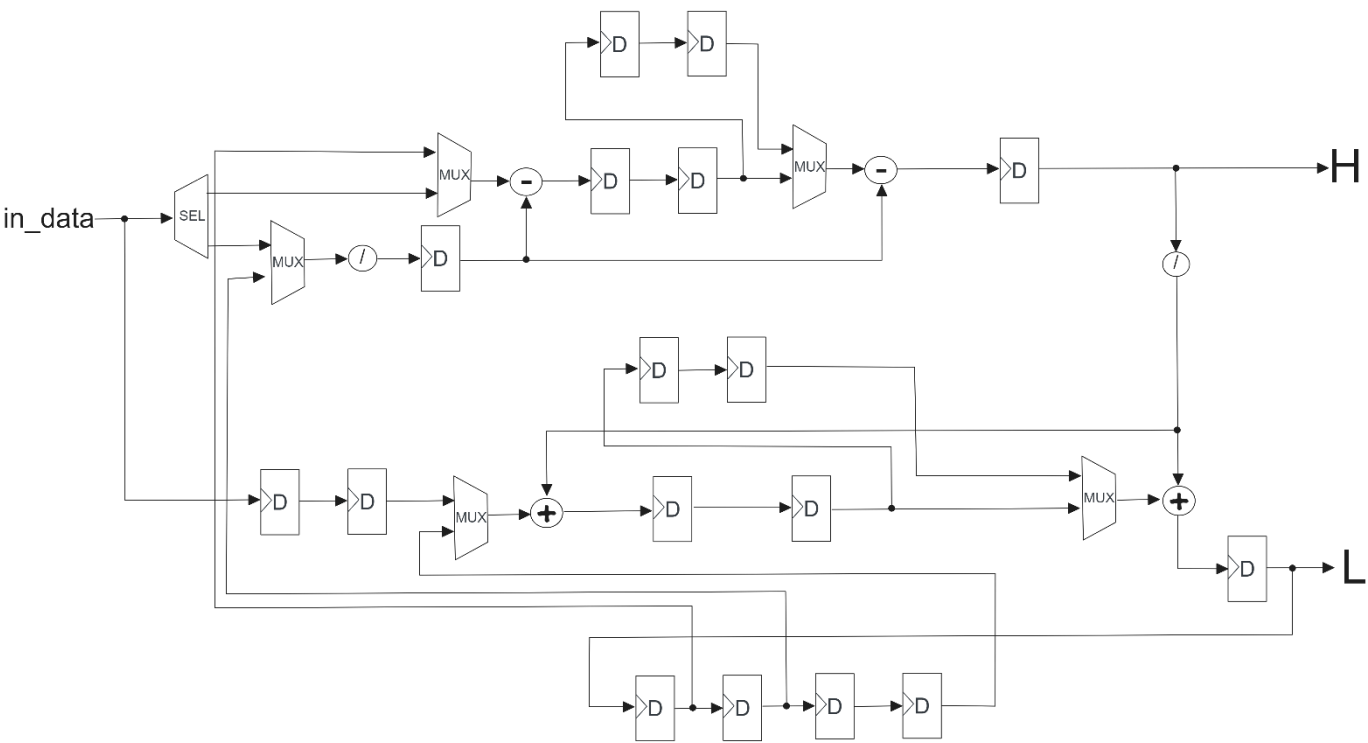
endmodule

```

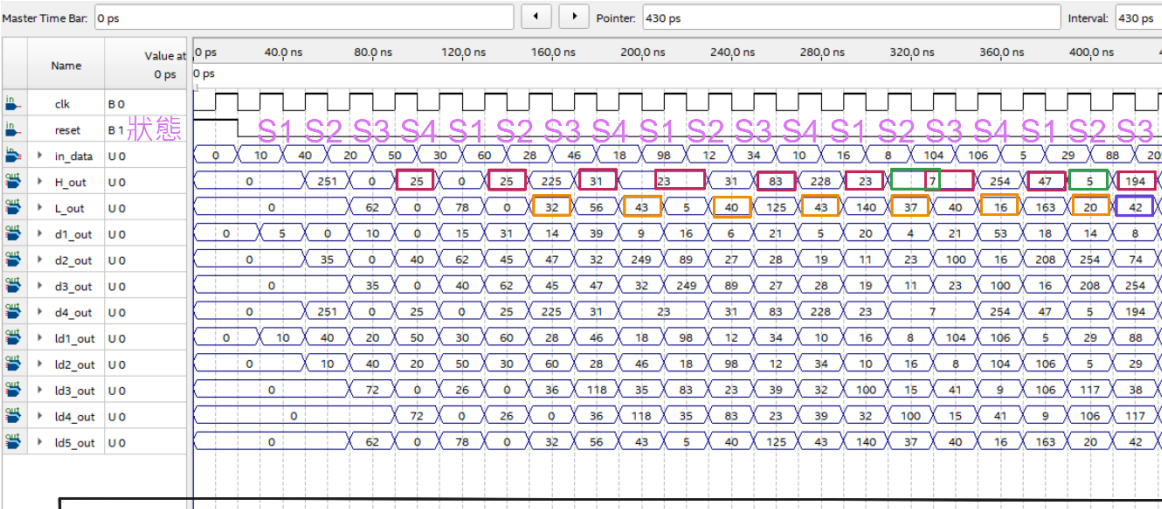
電路 RTL 架構圖



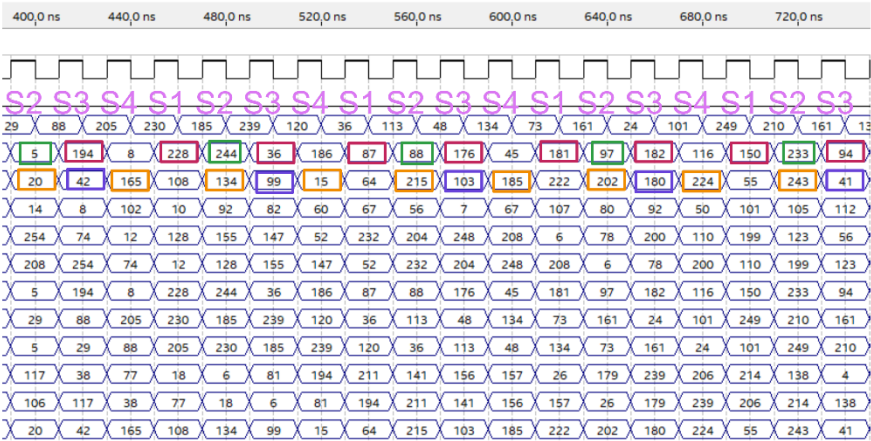
電路設計圖



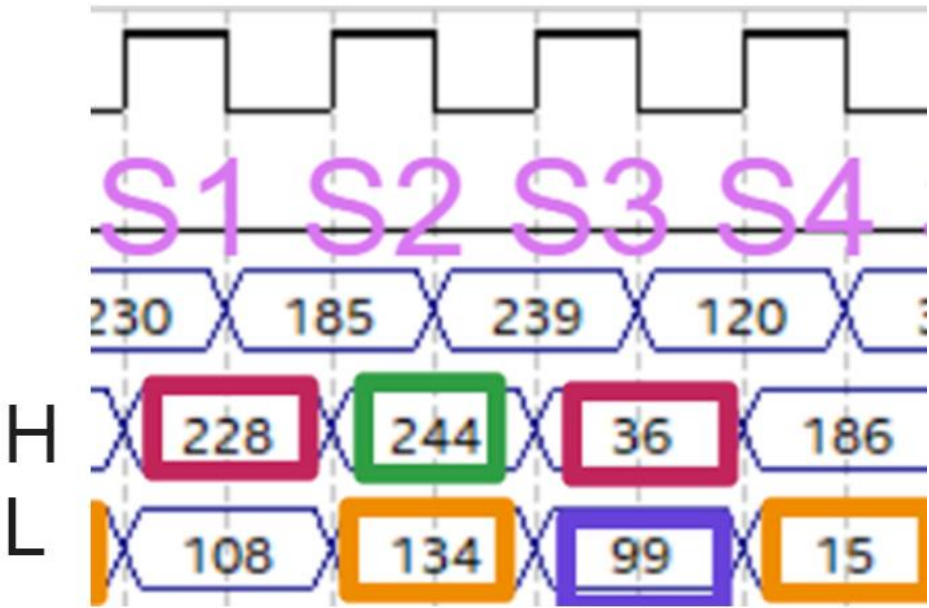
波形圖



H1
L1
H2
L2



狀態



一維三階分解

程式碼

```
module TOP(clk, reset, in_data, d1_out, d2_out, d3_out, d4_out, ld1_out, ld2_out,
ld3_out, ld4_out, ld5_out, L_out, H_out, state, mux_a, mux_b, mux_c, mux_d, mux_e);
    output [2:0] state;
    output [7:0] mux_a, mux_b, mux_c, mux_d, mux_e;
    input clk;
    input reset;
    input [7:0] in_data;
    output [7:0] d1_out, d2_out, d3_out, d4_out;
    output [7:0] ld1_out, ld2_out, ld3_out, ld4_out, ld5_out;
    output [7:0] L_out, H_out;
    parameter S1 = 3'd0;
    parameter S2 = 3'd1;
    parameter S3 = 3'd2;
    parameter S4 = 3'd3;
    parameter S5 = 3'd4;
    parameter S6 = 3'd5;
    parameter S7 = 3'd6;
    parameter S8 = 3'd7;
    reg [7:0] d_out [3:0];
    reg [7:0] ld_out [4:0];
    assign H_out = d_out [3];
    assign L_out = ld_out [4];
    assign d1_out = d_out [0];
    assign d2_out = d_out [1];
    assign d3_out = d_out [2];
    assign d4_out = d_out [3];
    assign ld1_out = ld_out [0];
    assign ld2_out = ld_out [1];
    assign ld3_out = ld_out [2];
    assign ld4_out = ld_out [3];
    assign ld5_out = ld_out [4];
    wire [7:0] mux_a, mux_b, mux_c, mux_d, mux_e;
    reg [2:0] state, nstate;
    reg [47:0] reg_b, reg_c;
    reg [39:0] reg_d;
    assign mux_a = (state == S1 | state == S3 | state == S5 | state == S7) ?
in_data :
                (state == S4 | state == S8) ? reg_d [15:8] :
                (state == S6) ? reg_d [23:16] :
                8'd0;

    assign mux_b = (state == S2 | state == S4 | state == S6 | state == S8) ?
in_data :
```

```

        (state == S1 | state == S5) ? reg_d[7:0] :
        (state == S7) ? ld_out[4] :
        8'd0;

    assign mux_c = (state == S2 | state == S4 | state == S6 | state == S8) ?
d_out[2] :
        (state == S1 | state == S5) ? reg_b[39:32] :
        (state == S7) ? reg_b[7:0] :
        8'd0;

    assign mux_d = (state == S1 | state == S3 | state == S5 | state == S7) ?
ld_out[1] :
        (state == S2 | state == S6) ? reg_d[31:24] :
        (state == S8) ? reg_d[39:32] :
        8'd0;

    assign mux_e = (state == S1 | state == S3 | state == S5 | state == S7) ?
ld_out[3] :
        (state == S2 | state == S6) ? reg_c[39:32] :
        (state == S8) ? reg_c[7:0] :
        8'd0;

    always@(posedge clk, posedge reset)begin
        if(reset)begin
            state <= S1;
        end
        else begin
            state <= nstate;
        end
    end
    always@(*)begin
        case(state)
            S1 :    nstate = S2;
            S2 :    nstate = S3;
            S3 :    nstate = S4;
            S4 :    nstate = S5;
            S5 :    nstate = S6;
            S6 :    nstate = S7;
            S7 :    nstate = S8;
            S8 :    nstate = S1;
            default: nstate = S1;
        endcase
    end
    integer x;
    always@(posedge clk, posedge reset)begin
        if(reset)begin
            for (x = 0; x < 4; x = x + 1)begin

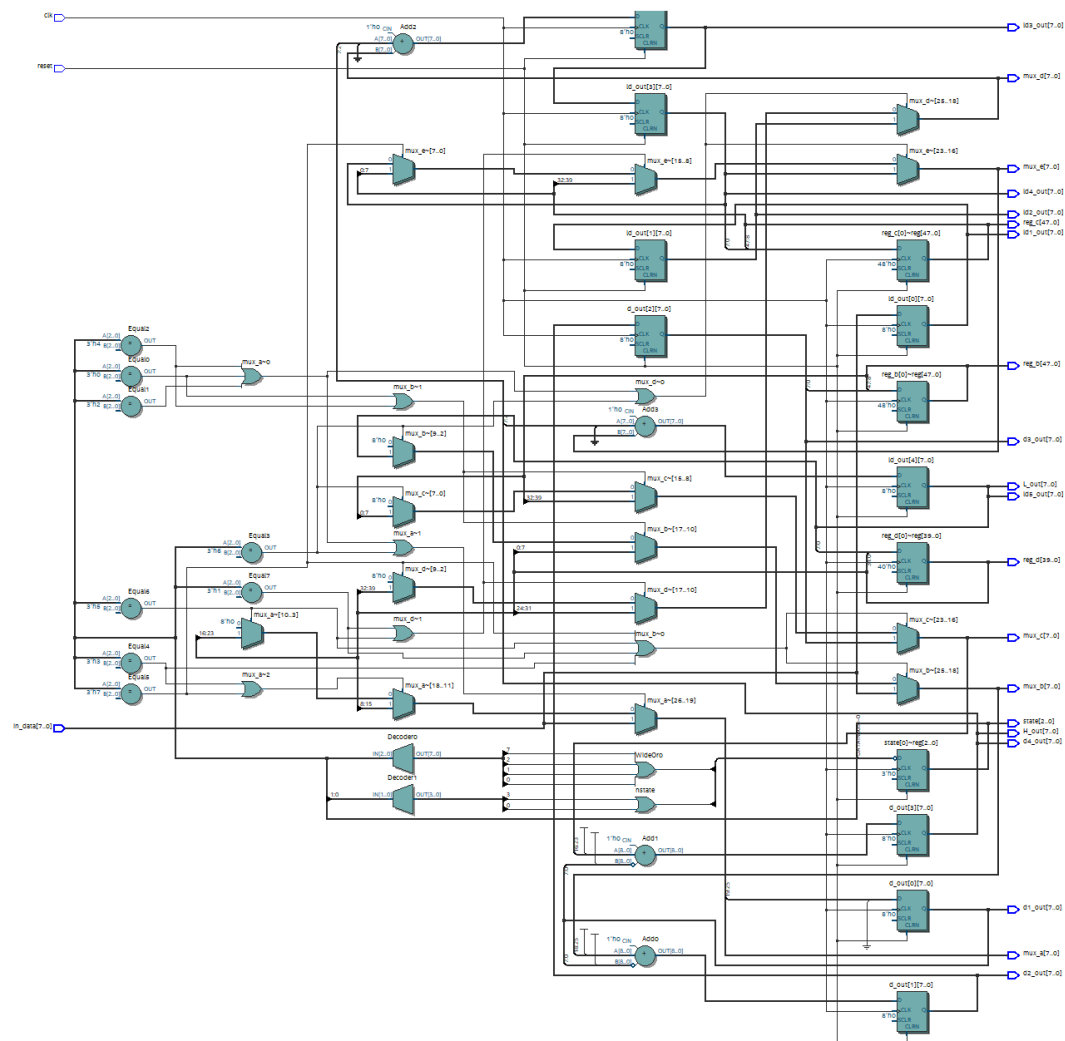
```

```

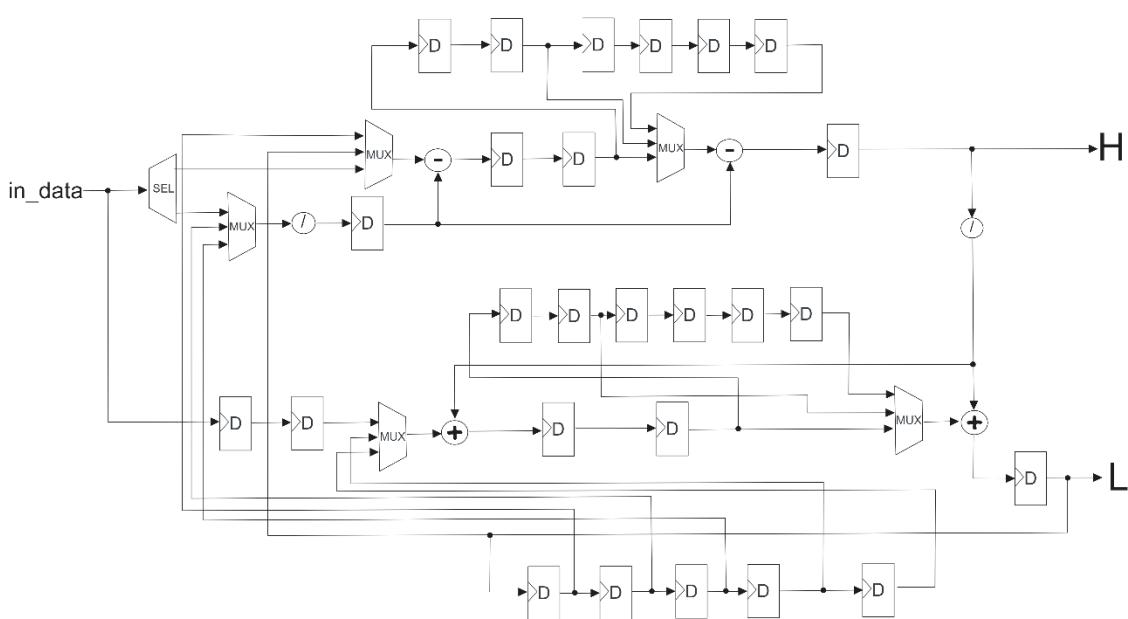
        d_out[x] <= 8'd0;
    end
end
else begin
    d_out[0] <= mux_a >> 1;
    d_out[1] <= mux_b - d_out[0];
    d_out[2] <= d_out[1];
    d_out[3] <= mux_c - d_out[0];
end
end
always@(posedge clk, posedge reset)begin
    if(reset)begin
        for (x = 0; x < 5; x = x + 1)begin
            ld_out[x] <= 8'd0;
        end
    end
    else begin
        ld_out[0] <= in_data;
        ld_out[1] <= ld_out[0];
        ld_out[2] <= (d_out[3] >> 2) + mux_d;
        ld_out[3] <= ld_out[2];
        ld_out[4] <= (d_out[3] >> 2) + mux_e;
    end
end
always@(posedge clk, posedge reset)begin
    if(reset)begin
        reg_b <= 48'd0;
    end
    else begin
        reg_b <= {d_out[2], reg_b[47:8]};
    end
end
always@(posedge clk, posedge reset)begin
    if(reset)begin
        reg_c <= 48'd0;
    end
    else begin
        reg_c <= {ld_out[3], reg_c[47:8]};
    end
end
always@(posedge clk, posedge reset)begin
    if(reset)begin
        reg_d <= 40'd0;
    end
    else begin
        reg_d <= {reg_d[31:0], ld_out[4]};
    end
end
endmodule

```

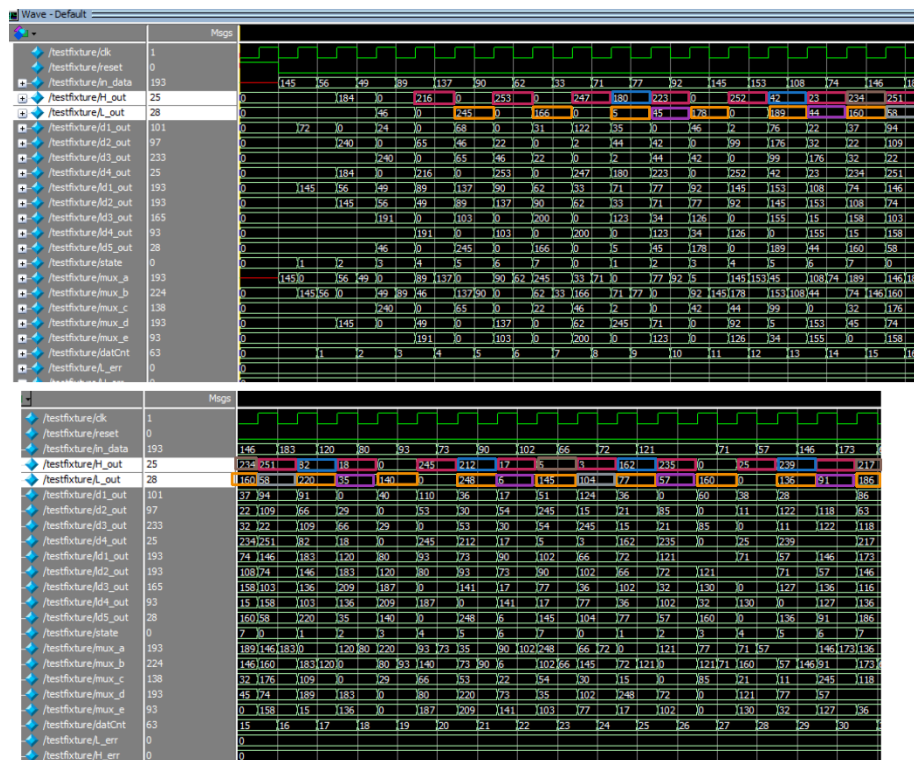
電路 RTL 架構圖



電路設計圖



波形圖



透過 testbench 去跑模擬

```

Transcript
VSIM 8> run -all
# Start Simulation
#
# \      \
# (o>    (o>
# //     //
# _(( )_  V/_
# ||     ||
#
# -----
# Simulation is done
#
#
# *****
#
#           \
# Successful !! \      \
#           \ (o>  (o>
#           //   //
# Simulation PASS!! _(( )_  V/_
#           ||     ||
#           ||     ||
# *****
#
#
# ** Note: $finish      : C:/Users/User/Desktop/tb/modelsim/tb.v(104)
# Time: 650 ns Iteration: 0 Instance: /testfixture
# 1
# Break in Module testfixture at C:/Users/User/Desktop/tb/modelsim/tb.v line 104

```

圖像二維三階分解

程式碼

```
module image(clk, reset, read_valid, r_adder, in_data, out_valid, w_adder,
out_data, csel, done, state);
    input clk;
    input reset;
    input [7:0] in_data;
    output read_valid;
    output out_valid;
    output reg [5:0] r_adder;
    output [5:0] w_adder;
    output [7:0] out_data;
    output [2:0] csel;
    output done;
    output [3:0] state;
    wire [7:0] TOP_in_data;
    wire H_valid, L_valid;
    wire [7:0] H_out, L_out;
    reg reset_data;
    reg [3:0] state, nstate;
    reg [7:0] r_cnt;
    reg [7:0] w_cnt;
    reg [5:0] reg_w_adder;
    parameter IDLE = 4'd0;
    parameter L1R = 4'd1;
    parameter R1R = 4'd2;
    parameter L1C = 4'd3;
    parameter R1C = 4'd4;
    parameter L2R = 4'd5;
    parameter R2R = 4'd6;
    parameter L2C = 4'd7;
    parameter R2C = 4'd8;
    parameter L3R = 4'd9;
    parameter R3R = 4'd10;
    parameter L3C = 4'd11;
    parameter DONE = 4'd12;
```

```

assign w_adder = (state == L1R && L_valid) ? reg_w_adder - 6'd5 :
                (state == L1C && L_valid) ? reg_w_adder - 6'd40 :
                (state == L2R && L_valid) ? reg_w_adder - 6'd3 :
                (state == L2C && L_valid) ? reg_w_adder - 6'd24 :
                (state == L3R && L_valid) ? reg_w_adder - 6'd2 :
                (state == L3C && L_valid) ? reg_w_adder - 6'd16 :
                reg_w_adder;

assign done = (state == DONE);
assign out_data = (L_valid) ? L_out :
                 (H_valid) ? H_out :
                 8'd0;

assign out_valid = ((L_valid || H_valid) && w_cnt < 8'd64 && (state == L1R
|| state == L1C)) ? 1'b1 :
                 ((L_valid || H_valid) && w_cnt < 8'd16 && (state == L2R ||
state == L2C)) ? 1'b1 :
                 ((L_valid || H_valid) && w_cnt < 8'd4 && (state == L3R ||
state == L3C)) ? 1'b1 :
                 1'b0;
assign TOP_in_data = (r_cnt < 8'd64) ? in_data : 8'd0;

TOP u1(clk, reset_data, TOP_in_data, L_valid, H_valid, L_out, H_out);

always@(posedge clk, posedge reset)begin
    if(reset)begin
        state <= IDLE;
    end
    else begin
        state <= nstate;
    end
end

always@(*)begin
    case(state)
        IDLE : nstate = L1R;
        L1R : nstate = w_cnt >= 8'd64 ? R1R : L1R;

```



```

    R1R : nstate = L1C;
    L1C : nstate = w_cnt >= 8'd64 ? R1C : L1C;
    R1C : nstate = L2R;
    L2R : nstate = w_cnt >= 8'd16 ? R2R : L2R;
    R2R : nstate = L2C;
    L2C : nstate = w_cnt >= 8'd16 ? R2C : L2C;
    R2C : nstate = L3R;
    L3R : nstate = w_cnt >= 8'd4 ? R3R : L3R;
    R3R : nstate = L3C;
    L3C : nstate = w_cnt >= 8'd4 ? DONE : L3C;
    DONE : nstate = DONE;
    default: nstate = IDLE;
endcase
end

assign csel = (state == L1R) ? 3'd1 :
    (state == L1C) ? 3'd2 :
    (state == L2R) ? 3'd3 :
    (state == L2C) ? 3'd4 :
    (state == L3R) ? 3'd5 :
    (state == L3C) ? 3'd6 :
    3'd0;

assign read_valid = (state == L1R & r_cnt < 8'd64) ? 1'b1 :
    (state == L1C & r_cnt < 8'd64) ? 1'b1 :
    (state == L2R & r_cnt < 8'd16) ? 1'b1 :
    (state == L2C & r_cnt < 8'd16) ? 1'b1 :
    (state == L3R & r_cnt < 8'd4) ? 1'b1 :
    (state == L3C & r_cnt < 8'd4) ? 1'b1 :
    1'b0;

always@(posedge clk, posedge reset)begin
    if(reset)begin
        reg_w_adder <= 6'd4;
    end
    else begin

```

```

        case(state)
            L1R : reg_w_adder <= (H_valid) ? reg_w_adder + 6'd1 :
(&w_cnt[2:0]) ? reg_w_adder + 6'd4 : reg_w_adder;
            R1R : reg_w_adder <= 6'd32;
            L1C : reg_w_adder <= (H_valid) ? reg_w_adder + 6'd8 :
(&w_cnt[2:0]) ? reg_w_adder + 6'd33 : reg_w_adder;
            R1C : reg_w_adder <= 6'd2;

            L2R : reg_w_adder <= (H_valid) ? reg_w_adder + 6'd1 :
(&w_cnt[1:0]) ? reg_w_adder + 6'd6 : reg_w_adder;
            R2R : reg_w_adder <= 6'd16;
            L2C : reg_w_adder <= (H_valid) ? reg_w_adder + 6'd8 :
(&w_cnt[1:0]) ? reg_w_adder + 6'd49 : reg_w_adder;
            R2C : reg_w_adder <= 6'd1;

            L3R : reg_w_adder <= (H_valid) ? reg_w_adder + 6'd1 :
(&w_cnt[0]) ? reg_w_adder + 6'd7 : reg_w_adder;
            R3R : reg_w_adder <= 6'd8;
            L3C : reg_w_adder <= (H_valid) ? reg_w_adder + 6'd8 :
(&w_cnt[0]) ? reg_w_adder + 6'd57 : reg_w_adder;

            default : reg_w_adder <= 6'd4;
        endcase
    end
end

always@(posedge clk, posedge reset)begin
    if(reset)begin
        r_adder <= 6'd0;
    end
    else begin
        case(state)
            L1R:begin
                r_adder <= r_adder + 6'd1;
            end
            L1C:begin
                r_adder <= (&r_cnt[2:0]) ? r_adder + 6'd9 : r_adder + 6'd8;
            end
        end
    end
end

```

```

        L2R:begin
            r_adder <= (&r_cnt[1:0]) ? r_adder + 6'd5 : r_adder + 6'd1;
        end
        L2C:begin
            r_adder <= (&r_cnt[1:0]) ? r_adder + 6'd41 : r_adder + 6'd8;
        end

        L3R:begin
            r_adder <= (&r_cnt[0]) ? r_adder + 6'd7 : r_adder + 6'd1;
        end
        L3C:begin
            r_adder <= (&r_cnt[0]) ? r_adder + 6'd57 : r_adder + 6'd8;
        end
        default:begin
            r_adder <= 6'd0;
        end
    endcase
end
end

always@(posedge clk, posedge reset)begin
    if(reset)begin
        w_cnt <= 8'd0;
    end
    else begin
        case(out_valid)
            1'b0 : w_cnt <= 8'd0;
            1'b1 : w_cnt <= w_cnt + 8'd1;
        endcase
    end
end

always@(posedge clk, posedge reset)begin
    if(reset)begin
        r_cnt <= 8'd0;
    end
end

```

```

        else begin
            case(state[0])
                1'b0 : r_cnt <= 8'd0;
                1'b1 : r_cnt <= r_cnt + 8'd1;
            endcase
        end
    end
end
always@(posedge clk, posedge reset)begin
    if(reset)begin
        reset_data <= 1'b1;
    end
    else begin
        case(state[0])
            1'b0 : reset_data <= 1'b1;
            1'b1 : reset_data <= 1'b0;
        endcase
    end
end
end

```

endmodule

```

module TOP(clk, reset, in_data, L_valid, H_valid, L_out, H_out);
    input clk;
    input reset;
    input [7:0] in_data;
    output L_valid;
    output H_valid;
    output [7:0] L_out;
    output [7:0] H_out;

    parameter P1    = 3'd0;
    parameter P2    = 3'd1;
    parameter P3    = 3'd2;
    parameter EVEN  = 3'd3;
    parameter ODD   = 3'd4;
    parameter EVEN2 = 3'd5;
    parameter ODD2  = 3'd6;

```

```

reg [7:0] save_first;
reg flag_first;
reg [7:0] d_out[3:0];
reg [7:0] ld_out[4:0];
wire [7:0] sub1;
wire [7:0] sub2;
reg [2:0] state, nstate;
assign sub1 = in_data - d_out[0];
assign sub2 = d_out[2] - d_out[0];

assign H_out = (state == ODD || state == ODD2) ? d_out[3] : 8'd0;
assign L_out = (state == EVEN2) ? ((flag_first) ? save_first + (d_out[3] >>
2) : ld_out[4]) : 8'd0;

assign H_valid = (state == ODD || state == ODD2) && ~reset;
assign L_valid = (state == EVEN2) && ~reset;

always@(posedge clk, posedge reset)begin
    if (reset)begin
        save_first <= 8'd0;
    end
    else begin
        case(state)
            P1 : save_first <= in_data;
        endcase
    end
end

always@(posedge clk, posedge reset)begin
    if (reset)begin
        flag_first <= 1'b1;
    end
    else begin
        case(state)
            EVEN2: flag_first <= 1'b0;
        endcase
    end
end

```

```

end

always@(posedge clk, posedge reset)begin
    if(reset)begin
        state <= P1;
    end
    else begin
        state <= nstate;
    end
end

always@(*)begin
    case(state)
        P1      : nstate = P2;
        P2      : nstate = P3;
        P3      : nstate = EVEN;
        EVEN    : nstate = ODD;
        ODD     : nstate = EVEN2;
        EVEN2   : nstate = ODD2;
        ODD2    : nstate = EVEN2;
        default : nstate = P1;
    endcase
end

always@(posedge clk, posedge reset)begin
    if(reset)begin
        d_out[0] <= 8'd0;
        d_out[1] <= 8'd0;
    end
    else begin
        case(state[0])
            1'b0 : d_out[0] <= in_data >> 1;
            1'b1 : d_out[1] <= sub1;
        endcase
    end
end
end

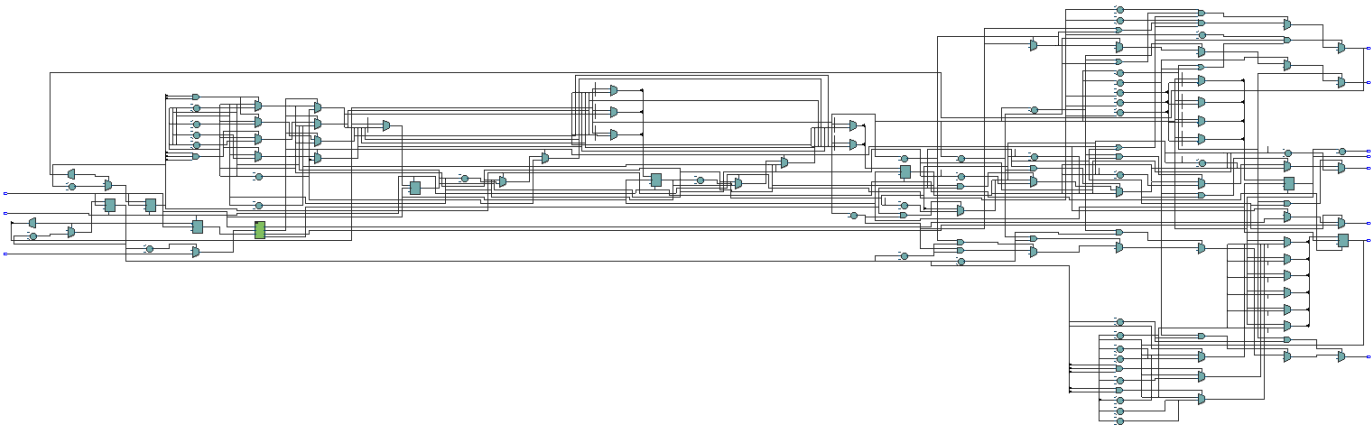
```

```
always@(posedge clk, posedge reset)begin
    if(reset)begin
        d_out[2] <= 8'd0;
        d_out[3] <= 8'd0;
    end
    else begin
        d_out[2] <= d_out[1];
        d_out[3] <= sub2;
    end
end
end
```

```
always@(posedge clk, posedge reset)begin
    if(reset)begin
        ld_out[0] <= 8'd0;
        ld_out[1] <= 8'd0;
        ld_out[2] <= 8'd0;
        ld_out[3] <= 8'd0;
        ld_out[4] <= 8'd0;
    end
    else begin
        ld_out[0] <= in_data;
        ld_out[1] <= ld_out[0];
        ld_out[2] <= (d_out[3] >> 2) + ld_out[1];
        ld_out[3] <= ld_out[2];
        ld_out[4] <= (d_out[3] >> 2) + ld_out[3];
    end
end
end
```

```
endmodule
```

電路 RTL 架構圖



電路設計架構圖

L1R

讀取

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

L1R

寫出

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

L1C

讀取

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

L1C

寫出

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

L2R

讀取

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

L2R

寫出

0	2	1	3				
8							
16							
24							
32							
40							
48							
56							

L2C

讀取

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

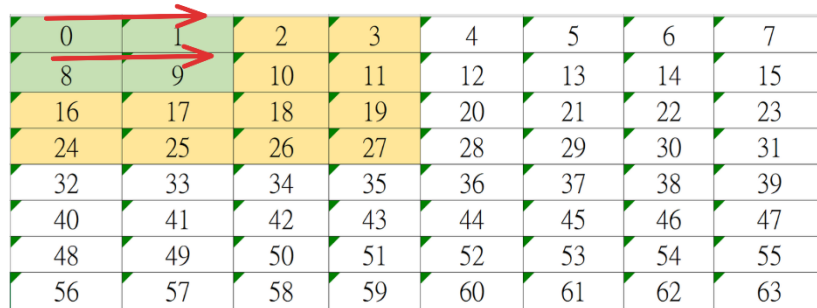
L2C

寫出

0	2	1	3	4	5	6	7
8				12	13	14	15
16				20	21	22	23
24				28	29	30	31
32				36	37	38	39
40				44	45	46	47
48				52	53	54	55
56				60	61	62	63

L3R

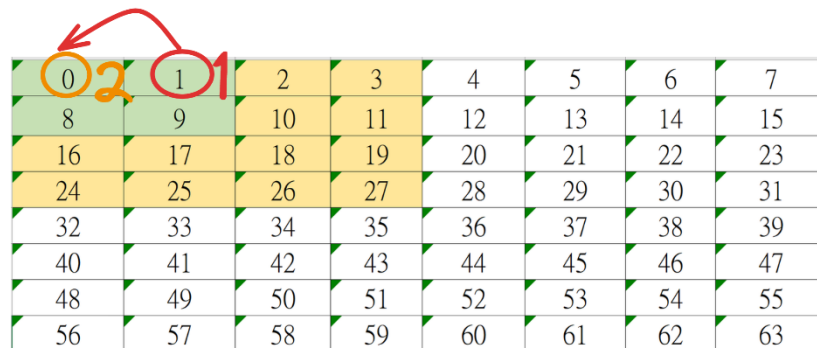
讀取



0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

L3R

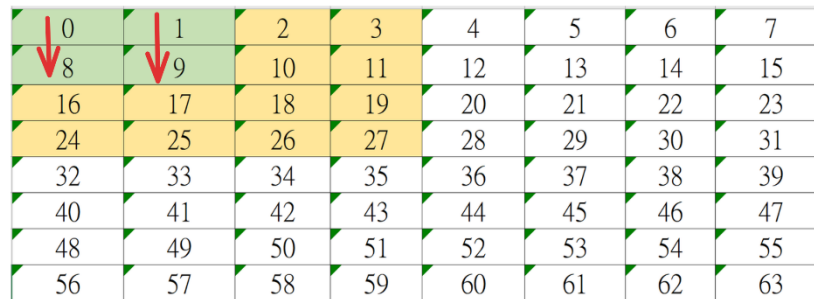
寫出



0	2	1	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15		
16	17	18	19	20	21	22	23		
24	25	26	27	28	29	30	31		
32	33	34	35	36	37	38	39		
40	41	42	43	44	45	46	47		
48	49	50	51	52	53	54	55		
56	57	58	59	60	61	62	63		

L3C

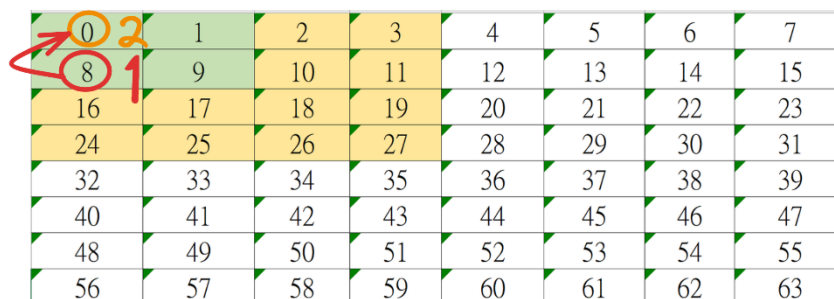
讀取



0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

L3C

寫出

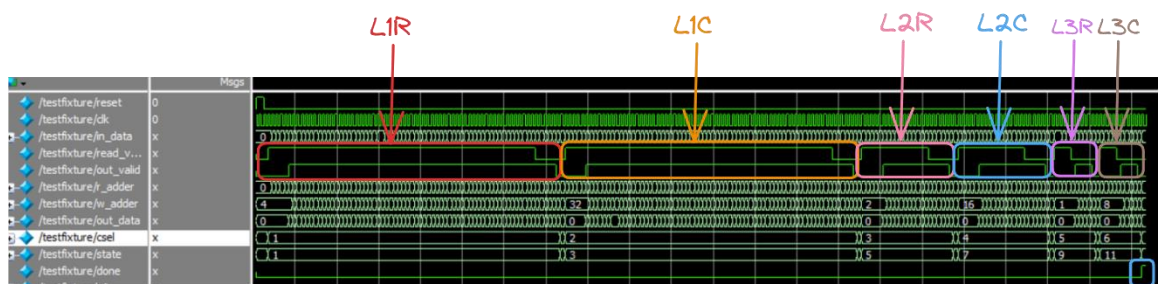


0	2	1	3	4	5	6	7
8	1	9	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

波形輸出

L1R 寫出

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63



結束