VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRICAL AND ELECTRONICS ENGINEERING

Laboratory Report

EE3201: Introduction to IC Design

Laboratory 1:

RTL DESIGN

Group 19_Class L02, Semester HK251

Supervisor: MSc. Nguyen Trung Hieu

Performed by students	MSSV	Done
Nguyễn Trọng Lộc	2311959	100%
Huỳnh Trung Kiên	2311729	100%
Nguyễn Đình Trọng Khôi	2311682	100%

Ho Chi Minh City, October 2025

Mục lục

RTL DESIGN 2

I.EXPERIMENT 1 4

1.1. Mục tiêu 5

1.2. Thiết kế mạch 6

1.3. System Verilog HDL code 7

1.4. Testbench kiểm thử 8

1.5. Mô tả hoạt động 9

1.6. Kết quả mô phỏng 10

1.7. Kết luận

II.EXPERIMENT 2 11

2.1. Mục tiêu 11

2.2. Thiết kế mạch 11

2.3. System Verilog HDL code 12

2.4. Testbench kiểm thử 13

2.5. Mô tả hoạt động 16

2.6. Kết quả mô phỏng 17

2.7. Kết luận

III.EXPERIMENT 3 20

3.1. Mục tiêu 20

3.2. Thiết kế mạch 21

3.3. System Verilog HDL code 22

3.4. Testbench kiểm thử 23

3.5. Mô tả hoạt động 24

3.6. Kết quả mô phỏng 25

IV. EXPERIMENT 4 26

4.1. Yêu cầu thiết kế ex4 26

4.2. Design circuit ex4 27

4.3. System Verilog HDL code to describe circuit ex4 27

4.4. Testbench ex4 29

4.5. Mô tả hoạt động thiết kế ex4 31

4.6. Kết quả ex4 32

I . EXPERIMENT 1

1.1. Mục tiêu

Description: The circuit in Figure 1, which is often called an *accumulator*, is used to add the value of an input *A* to itself repeatedly. The circuit includes a carry out from the adder, as well as an *overflow* output signal. If the input *A* is considered as a 2's-complement number, then *overflow* should be set to 1 in the case where the output *sum* produced does not represent a correct 2's complement result.

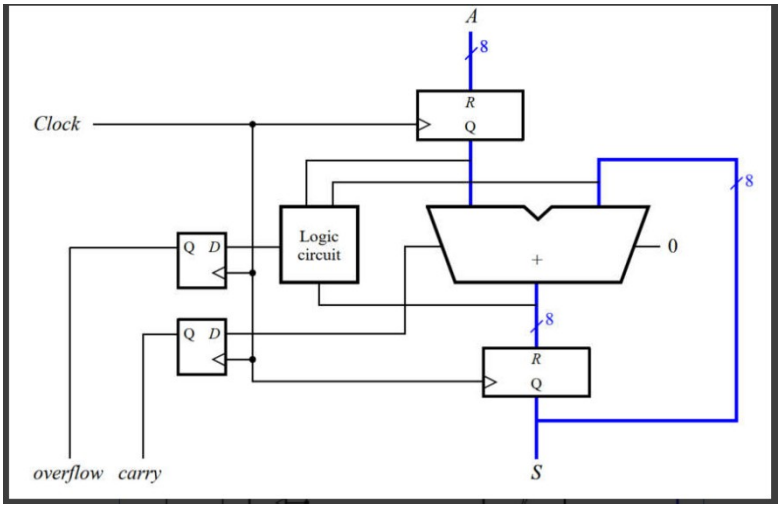
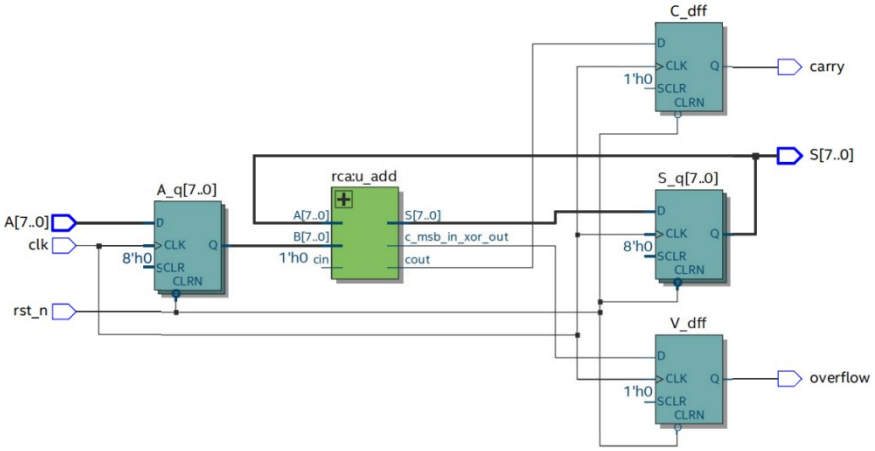


Figure 1: An eight-bit accumulator circuit.

1.2. Thiết kế mạch



1.3. System Verilog HDL code

```
module fa1(input logic a,b,cin, output logic s,cout);
  logic p; assign p=a^b; assign s=p^cin;
  assign cout = (a&b)|(a&cin)|(b&cin);
endmodule
module rca #(parameter N=8)(
  input  logic [N-1:0] A,B,
  input  logic         cin,
  output logic [N-1:0] S,
  output logic         cout,
  output logic         c_msb_in_xor_out
);
  logic [N:0] C; assign C[0]=cin;
  genvar i; generate
    for (i=0;i<N;i++) begin: G
      fa1 u(.a(A[i]),.b(B[i]),.cin(C[i]),.s(S[i]),.cout(C[i+1]));
    end
```

```
  endgenerate
  assign cout = C[N];
  assign c_msb_in_xor_out = C[N-1]^C[N]; // ovf 2's comp
endmodule
module ex1(                     // ===== EXPERIMENT 1 ===
  input  logic       clk,
  input  logic       rst_n,     //
  input  logic [7:0] A,         // vào thanh ghi A (R/Q)
  output logic [7:0] S,         // ra từ thanh ghi S (R/Q)
  output logic       overflow,  // qua DFF
  output logic       carry      // qua DFF
);
  logic [7:0] A_q;        // Thanh ghi A (R/Q ở góc trên)
  always_ff @(posedge clk or negedge rst_n) begin
    if (!rst_n) A_q <= 8'b0;
    else        A_q <= A;
  end
  logic [7:0] S_q, S_next; // Thanh ghi S + vòng hồi tiếp
  logic C_out, V_raw, V_dff, C_dff;
  rca #(8) u_add(
    .A(S_q), .B(A_q), .cin(1'b0),
    .S(S_next), .cout(C_out), .c_msb_in_xor_out(V_raw)
  );
  wire logic_carry    = C_out; //tạo tín hiệu cờ
  wire logic_overflow = V_raw;
  always_ff @(posedge clk or negedge rst_n) begin // 2 DFF cho carry &
overflow
    if (!rst_n) begin
      C_dff <= 1'b0;
      V_dff <= 1'b0;
    end else begin
      C_dff <= logic_carry;
      V_dff <= logic_overflow;
    end
  end
  always_ff @(posedge clk or negedge rst_n) begin  // --- Thanh ghi kết quả S
    if (!rst_n) S_q <= 8'b0;
    else        S_q <= S_next;
  end
  assign S        = S_q;  // xuất S
  assign carry    = C_dff;
  assign overflow = V_dff;
endmodule
```

1.4. Testbench kiểm thử

```
`timescale 1ns/1ps
module ex1_tb;
    logic        clk;
    logic        rst_n;
    logic [7:0]  A;
    logic [7:0]  S;
    logic        carry;
    logic        overflow;
    // Kết nối DUT
    ex1 dut(
        .clk(clk),
        .rst_n(rst_n),
        .A(A),
        .S(S),
        .carry(carry),
        .overflow(overflow)
    );
    // Clock 10ns
    always #5 clk = ~clk;

    initial begin
        clk = 0;
        rst_n = 0;
        A = 8'h00;
        repeat(2) @(posedge clk);
        rst_n = 1;

        A = 8'h05;
        repeat(3) @(posedge clk);
        A = 8'hFF;
        @(posedge clk);
        A = 8'h7F;
        @(posedge clk);
        A = 8'h80;
        @(posedge clk);
        repeat(5) @(posedge clk);

        rst_n = 0;
        @(posedge clk);
        rst_n = 1;

        A = 8'h03;
        repeat(3) @(posedge clk);
        #10 $finish;
    end

    initial begin
```

```
        $display("Thời gian\t A\t S\t carry\t overflow");
        $monitor("%t\t %h\t %h\t %b\t %b", $time, A, S, carry, overflow);
    end

    initial begin
        $dumpfile("ex1.vcd");
        $dumpvars(0, ex1_tb);
    end
endmodule
```

1.5. Mô tả hoạt động của thiết kế ex1 :

Mạch thiết kế là bộ cộng - tích lũy 8 bit có khả năng cộng dồn giá trị đầu vào theo từng chu kỳ xung clock , đồng thời sinh ra cờ trạng thái là cờ carry và overflow.

*** cấu trúc tổng thể thiết kế:**

● Khối thanh ghi đầu vào (A_q):

+ Chức năng: lưu trữ giá trị 8-bit đầu vào A tại sườn dương của xung clock.

+Khi rst_n = 0, thanh ghi được reset về 0.Còn khi rst_n = 1, giá trị của A được chốt và giữ ổn định trong suốt một chu kỳ clock.

● Khối cộng 8-bit (RCA – Ripple Carry Adder)

Gồm 8 full adder (FA) mắc nối tiếp nhau.

-Thực hiện phép cộng giữa:

+ S_q: tổng được lưu từ chu kỳ trước,

+A_q: giá trị đầu vào mới đã được chốt,

+cin = 0 (không có bit nhớ vào ban đầu).

-Đầu ra của RCA gồm:

+S_next[7:0]: tổng của phép cộng (đầu vào cho thanh ghi S),

+C_out: bit nhớ ra cuối (carry flag),

+ V_raw = C7 ⊕ C8: tín hiệu phát hiện tràn trong biểu diễn bù hai (overflow flag).

- Khối thanh ghi kết quả và cờ (S_q, C_dff, V_dff)

+Thanh ghi S_q: lưu kết quả cộng (S_next) để sử dụng cho chu kỳ kế tiếp (cộng dồn).

+Thanh ghi C_dff và V_dff: lưu hai tín hiệu cờ carry và overflow, giúp ổn định tín hiệu đầu ra.

Các thanh ghi đều có reset bất đồng bộ, đưa về 0 khi rst_n = 0.

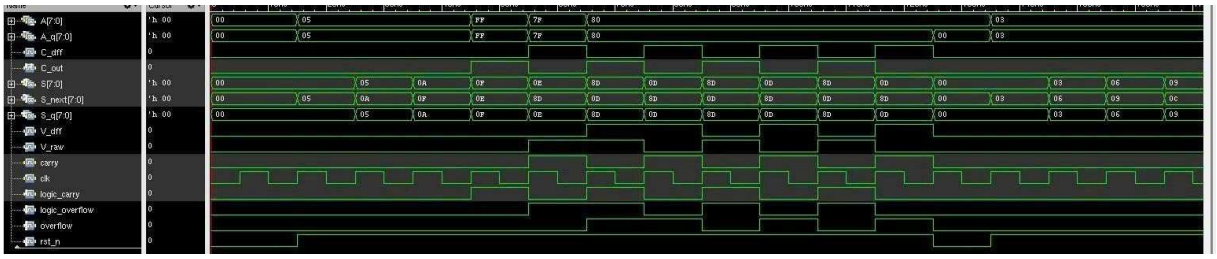
*** Hoạt động của thiết kế:**

+Khi rst_n=0, tất cả thanh ghi (A_q, S_q, C_dff, V_dff) được xóa về 0. kết quả làm cho cờ S=0, carry=0, overflow=0.

+Khi rst_n=1, tại sườn dương đầu tiên của clk: A_q nhận giá trị đầu vào A, rồi dữ liệu A_q đưa vào bộ cộng RCA cùng với S_q (tổng trước đó). Bộ cộng thực hiện phép S next = S_q+A_q . Trong đó có C_out là cờ tràn không dấu và V_raw=C7⊕C8 cờ tràn bù hai. Tại sườn dương kế tiếp clk S_next được lưu vào lại S_q để trở thành tổng hiện tại cho chu kỳ sau. Khi này 2 cờ carry và overflow cũng được chốt qua D-FF và carry bật khi có bit thứ 8 bị tràn và overflow bật khi tràn dấu trong biểu diễn bù hai.Ở mỗi chu kỳ clock sau S_q được hồi tiếp về đầu vào RCA.Lúc này S_next =S_q+A_q mới , nên mạch tiếp tục cộng dồn các giá trị đầu vào liên tiếp .

1.6. Kết quả mô phỏng

```
Loading snapshot worklib.ex1_tb:sv ..................... Done
xcelium> source ../tools/eda/cadence/XCELIUM2409/tools/xcelium/files/xmsimrc
xcelium> run
Thời gian        A       S       carry   overflow
                   0     00      00      0       0
                  15     05      00      0       0
                  25     05      05      0       0
                  35     05      0a      0       0
                  45     ff      0f      0       0
                  55     7f      0e      1       0
                  65     80      8d      0       1
                  75     80      0d      1       1
                  85     80      8d      0       0
                  95     80      0d      1       1
                 105     80      8d      0       0
                 115     80      0d      1       1
                 125     80      00      0       0
                 135     03      00      0       0
                 145     03      03      0       0
                 155     03      06      0       0
                 165     03      09      0       0
Simulation complete via $finish(1) at time 175 NS + 0
./01_tb/ex1_tb.sv:47        #10 $finish;
xcelium> exit
```



Nhận xét:

- Bảng log (console) và waveform đều cho thấy mạch cập nhật S chỉ ở cạnh lên của clock — giá trị S tiến theo chu kỳ như mong đợi (ví dụ: A=5 → S: 0 → 5 → 10 → 15).
- Ví dụ cụ thể trong log: tại ~45–55kns khi A=0xFF (−1) và S=0x0F (15) thì sau phép cộng S = 0x0E (14) với carry=1 và overflow=0 — đúng với quy tắc unsigned carry và signed overflow.
- Trường hợp gây overflow signed cũng hiển thị rõ: sau khi cộng 0x7F (127) vào một số dương nhỏ dẫn tới S=0x8D (kết quả nhị phân có MSB=1) thì overflow = 1; waveform thể hiện V_raw đổi tương ứng và V_dff (cờ đã được lưu) giữ giá trị ổn định giữa các chu kỳ.
- Waveform cho thấy có hai tín hiệu liên quan đến S (S_next và S_q): S_next là kết quả tổ hợp của adder, còn S_q được chốt vào cạnh clock — điều này khớp mô tả pipeline (compute → latch). Cả carry và overflow đều được ghim qua D-FF nên không bị glitch và hiển thị ổn định.

- Reset hoạt động đúng: khi rst_n active-low thì A_q, S_q, C_dff, V_dff về 0; sau release reset hệ bắt đầu hoạt động bình thường.
- Nhìn chung, cả hai nguồn (console và waveform) nhất quán với nhau: các phép thử điển hình (cộng nhỏ, tạo carry, tạo signed overflow, reset giữa chừng) đều cho kết quả đúng về S, carry và overflow.

1.7. Kết luận

Trong Experiment 1, chúng tôi đã thiết kế và mô phỏng một bộ cộng-tích lũy 8-bit theo kiến trúc trong hình 1 (RCA + thanh ghi hồi tiếp + DFF cho cờ). Kết quả mô phỏng cho thấy mạch thực hiện chính xác phép cộng dồn S(n+1) = S(n) + A sau mỗi cạnh lên của clock; đồng thời hai cờ carry và overflow được cập nhật chính xác cho các trường hợp tràn unsigned và tràn signed. Testbench đã kiểm tra các trường hợp tiêu biểu (cộng nhỏ, gây carry, gây overflow, reset giữa chu trình) và các waveform (.vcd) xác nhận hành vi mong đợi. Để nâng cao độ tin cậy, testbench đã bổ sung các thao tác prime trước khi đo và các kiểm tra tự động, đồng thời đề xuất kiểm tra thêm các vector biên để bao phủ mọi trường hợp cực trị.
Nhìn chung, thiết kế đạt yêu cầu và chức năng.

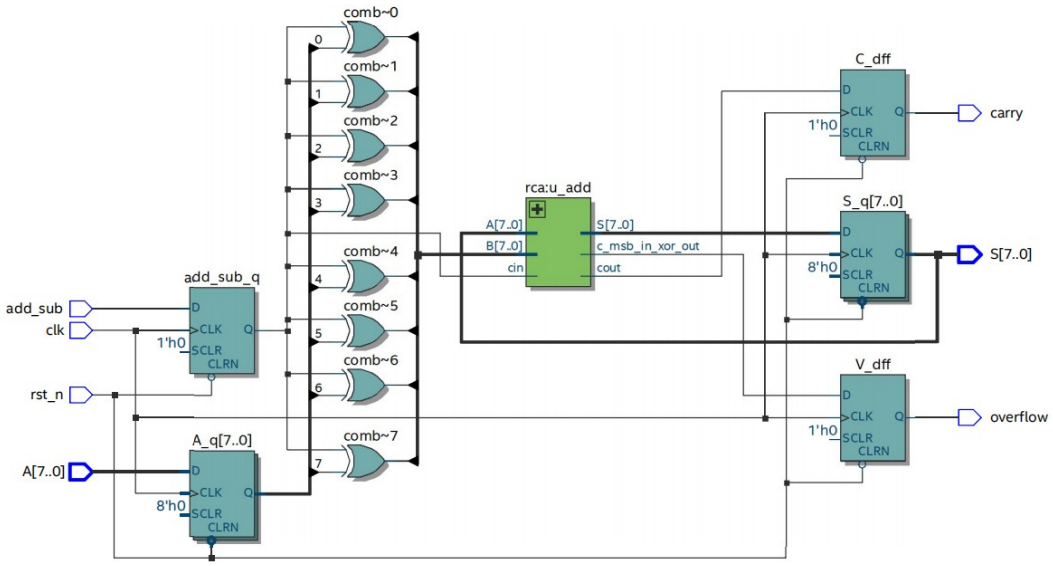
II . EXPERIMENT 2

2.1. Mục tiêu

Requirements:

- Extend the circuit from Experiment 1 to be able to both add and subtract numbers. To do so, add an *add_sub* input to your circuit. When *add_sub* is 1, your circuit should subtract *A* from register *S*, and when *add_sub* is 0 your circuit should add *A* to register *S* as in experiment 1.
- Draw your design circuit.
- Write System Verilog HDL code to describe your circuit.
- Write a testbench for the logic circuit with the requirement to test all typical (especially special) cases of the circuit.

2.2. Thiết kế mạch



2.3. System Verilog HDL code

```
module fa1(input logic a,b,cin, output logic s,cout);
  logic p;
  assign p    = a ^ b;
  assign s    = p ^ cin;
  assign cout = (a & b) | (a & cin) | (b & cin);
endmodule
// RCA
module rca #(parameter N=8)(
  input  logic [N-1:0] A, B,
  input  logic         cin,
  output logic [N-1:0] S,
  output logic         cout,
  output logic         c_msb_in_xor_out
);
  logic [N:0] C;
  assign C[0] = cin;

  genvar i;
  generate
    for (i=0;i<N;i++) begin : G
      fa1 u(.a(A[i]), .b(B[i]), .cin(C[i]), .s(S[i]), .cout(C[i+1]));
    end
  endgenerate

  assign cout             = C[N];
  assign c_msb_in_xor_out = C[N-1] ^ C[N];
endmodule
// ===== EXPERIMENT 2: Add/Sub Accumulator =====
```

```
// add_sub=0 -> S := S + A
// add_sub=1 -> S := S - A
module ex2(
  input  logic       clk,
  input  logic       rst_n,
  input  logic       add_sub,   // 0: add, 1: subtract
  input  logic [7:0] A,
  output logic [7:0] S,
  output logic       carry,
  output logic       overflow
);
  // --- Input reg ---
  logic [7:0] A_q;
  always_ff @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
      A_q       <= 8'b0;
    end else begin
      A_q       <= A;
    end
  end

  // --- Accumulator reg ---
  logic [7:0] S_q;
  // adder_result is computed combinationally below and then registered into
S_q
  always_ff @(posedge clk or negedge rst_n)
    if (!rst_n) S_q <= 8'b0;
    else        S_q <= adder_result;

  logic [7:0] adder_result;
  logic       C_out, V_raw;

  // IMPORTANT: use continuous assigns so Bx/cin always track A_q/add_sub_q
  logic [7:0] Bx;
  logic       cin;
  assign Bx  = A_q ^ {8{add_sub}}; // subtract -> ~A_q, add -> A_q
  assign cin = add_sub;

  rca #(8) u_add(
    .A(S_q), .B(Bx), .cin(cin),
    .S(adder_result), .cout(C_out), .c_msb_in_xor_out(V_raw)
  );
    // --- Flag reg  --
  logic C_dff, V_dff;
  always_ff @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
      C_dff <= 1'b0;
      V_dff <= 1'b0;
    end else begin
```

```
      C_dff <= C_out;  // carry out of the adder
      V_dff <= V_raw;  // two's complement ovf
    end
  end
  // --- Out -----
  assign S        = S_q;
  assign carry    = C_dff;
  assign overflow = V_dff;
endmodule
```

2.4. Testbench kiểm thử

```
`timescale 1ns/1ps

module ex2_tb;
  // ===== Cấu hình kiểm cờ =====
  // 0: không kiểm carry/overflow (chỉ kiểm S)
  // 1: carry_expect = Cout (adder carry-out)
  // 2: carry_expect = ~Cout (một số flow mapping C = no-borrow)
  localparam int FLAG_MODE        = 0;
  localparam int FLAG_LAT_STAGES  = 1;

  // ===== Tín hiệu =====
  logic        clk;
  logic        rst_n;
  logic        add_sub;
  logic [7:0]  A;
  logic [7:0]  S;
  logic        carry;
  logic        overflow;

  // ===== DUT =====
  ex2 dut(
    .clk(clk),
    .rst_n(rst_n),
    .add_sub(add_sub),
    .A(A),
    .S(S),
    .carry(carry),
    .overflow(overflow)
  );

  // ===== Clock =====
  initial clk = 1'b0;
  always #5 clk = ~clk;
  task tick; begin @(posedge clk); #1; end endtask

  // ===== Kéo rộng xung để nhìn rõ trên sóng =====
```

```
  logic carry_wide, overflow_wide; int cw=0, vw=0;
  always @(posedge clk) begin
    if (carry)    cw <= 3; else if (cw>0) cw <= cw-1;
    if (overflow) vw <= 3; else if (vw>0) vw <= vw-1;
    carry_wide    <= (cw>0);
    overflow_wide <= (vw>0);
  end

  // ===== Mô hình tham chiếu: pipeline =====
  byte ref_Sq;
  byte in_Aq;
  bit  in_addsub_q;

  // pipeline cờ tham chiếu (Cout/ ~Cout) và overflow
  bit co_pipe [0:3];
  bit co_inv_pipe [0:3];
  bit ov_pipe [0:3];

  // Hàm 1 chu kỳ adder/sub
  function automatic void adder_cycle(
    input  byte Sq_i, input byte Aq_i, input bit addsub_i,
    output byte Snext_o, output bit cout_o, output bit ovf_o
  );
    // Bx = A  hoặc ~A (khi trừ), cin = addsub
    byte Bx = addsub_i ? ~Aq_i : Aq_i;
    int  sum = (Sq_i & 8'hFF) + (Bx & 8'hFF) + (addsub_i ? 1 : 0);
    Snext_o = sum[7:0];
    cout_o  = (sum >> 8) & 1;                // Cout unsigned
    // Overflow hai’s-complement: cộng hai số cùng dấu ra khác dấu
    ovf_o   = ((Sq_i[7] == Bx[7]) && (Snext_o[7] != Sq_i[7]));
  endfunction

  // In trạng thái (không access net nội bộ)
  task display_status(input string note);
    $display("Time=%0t | %s | add_sub=%0b  A=%0d (0x%02h)  S=%0d
(0x%02h)  carry=%0b overflow=%0b",
             $time, note, add_sub, $signed(A), A, $signed(S), S, carry,
overflow);
  endtask

  // So sánh theo mode
  task automatic check_flags(string tag);
    if (FLAG_MODE == 0) begin
      $display("[%s] (SKIP) CARRY=%0b OVERFLOW=%0b", tag, carry, overflow);
    end
    else if (FLAG_MODE == 1) begin
      if (carry !== co_pipe[FLAG_LAT_STAGES])
        $error("[%s] CARRY mismatch: got %0b exp %0b", tag, carry,
co_pipe[FLAG_LAT_STAGES]);
```

```
      else
        $display("[%s] PASS CARRY=%0b", tag, carry);

      if (overflow !== ov_pipe[FLAG_LAT_STAGES])
        $error("[%s] OVERFLOW mismatch: got %0b exp %0b", tag, overflow,
ov_pipe[FLAG_LAT_STAGES]);
      else
        $display("[%s] PASS OVERFLOW=%0b", tag, overflow);
    end
    else begin // FLAG_MODE == 2  (so ~Cout)
      if (carry !== co_inv_pipe[FLAG_LAT_STAGES])
        $error("[%s] CARRY mismatch (~Cout): got %0b exp %0b", tag, carry,
co_inv_pipe[FLAG_LAT_STAGES]);
      else
        $display("[%s] PASS CARRY=%0b (~Cout)", tag, carry);

      if (overflow !== ov_pipe[FLAG_LAT_STAGES])
        $error("[%s] OVERFLOW mismatch: got %0b exp %0b", tag, overflow,
ov_pipe[FLAG_LAT_STAGES]);
      else
        $display("[%s] PASS OVERFLOW=%0b", tag, overflow);
    end
  endtask

  // Một bước kiểm tra
  task automatic step_and_check(string tag, bit set_addsub, byte set_A);
    byte s_exp; bit c_now, v_now;
    // 1) Tính s_exp, c_now, v_now cho chu kỳ hiện tại
    adder_cycle(ref_Sq, in_Aq, in_addsub_q, s_exp, c_now, v_now);

    // 2) Áp input cho chu kỳ kế
    add_sub = set_addsub; A = set_A;

    // 3) Tick + in trạng thái
    tick(); display_status(tag);

    // 4) Check S
    if (S !== s_exp) $error("[%s] S mismatch: got 0x%0h exp 0x%0h", tag, S,
s_exp);
    else             $display("[%s] PASS S=0x%0h", tag, S);

    // 5) Check cờ theo mode
    check_flags(tag);

    // 6) Cập nhật mô hình & pipeline cho chu kỳ kế
    ref_Sq      = s_exp;
    in_Aq       = set_A;
    in_addsub_q = set_addsub;
```

```
    co_pipe[3]     <= co_pipe[2];
    co_pipe[2]     <= co_pipe[1];
    co_pipe[1]     <= co_pipe[0];
    co_pipe[0]     <= c_now;

    co_inv_pipe[3] <= co_inv_pipe[2];
    co_inv_pipe[2] <= co_inv_pipe[1];
    co_inv_pipe[1] <= co_inv_pipe[0];
    co_inv_pipe[0] <= ~c_now;

    ov_pipe[3]     <= ov_pipe[2];
    ov_pipe[2]     <= ov_pipe[1];
    ov_pipe[1]     <= ov_pipe[0];
    ov_pipe[0]     <= v_now;
  endtask

  // Reset + prime
  task automatic do_reset_and_prime(input string why);
    $display("=== RESET: %s ===", why);
    add_sub = 1'b0; A = 8'h00;
    rst_n = 1'b0; repeat(3) @(posedge clk);
    display_status("ĐANG RESET");
    #2 rst_n = 1'b1; #1;
    tick(); display_status("NHẢ RESET");

    // Clear ref
    ref_Sq = 0; in_Aq = 0; in_addsub_q = 0;
    co_pipe      = '{default:0};
    co_inv_pipe  = '{default:0};
    ov_pipe      = '{default:0};

    // prime 2 NOP
    step_and_check("PRIME#1", 1'b0, 8'h00);
    step_and_check("PRIME#2", 1'b0, 8'h00);
  endtask

  // ===== Test plan (giữ nguyên như bạn đang dùng) =====
  initial begin
    clk=0; rst_n=0; add_sub=0; A=8'h00;
    $display("=== KIỂM TRA BỘ TÍCH LŨY VỚI CHỨC NĂNG CỘNG VÀ TRỪ ===");

    // Cụm 1
    do_reset_and_prime("Đầu bài");
    $display("\n=== KIỂM TRA PHÉP CỘNG CƠ BẢN ===");
    step_and_check("ADD +10",          1'b0, 8'd10);
    step_and_check("ADD +15",          1'b0, 8'd15);
    step_and_check("ADD +240(carry)", 1'b0, 8'd240);
    step_and_check("ADD +100",         1'b0, 8'd100);
    step_and_check("ADD +70",          1'b0, 8'd70);
```

```
// Cụm 2
$display("\n=== RESET VÀ KIỂM TRA PHÉP CỘNG LỚN ===");
do_reset_and_prime("ADD lớn");
step_and_check("ADD +112",        1'b0, 8'd112);
step_and_check("ADD +112",        1'b0, 8'd112);
step_and_check("ADD +32",         1'b0, 8'd32);

// Cụm 3
$display("\n=== KIỂM TRA PHÉP TRỪ ===");
do_reset_and_prime("SUB cơ bản");
step_and_check("PREP +50",        1'b0, 8'd50);
step_and_check("SUB  -20",        1'b1, 8'd20);
step_and_check("SUB  -40",        1'b1, 8'd40);

// Cụm 4
$display("\n=== KIỂM TRA OVERFLOW KHI TRỪ ===");
do_reset_and_prime("OVF SUB");
step_and_check("LOAD -128",       1'b0, 8'h80);
step_and_check("MIN-1 (OVF)",     1'b1, 8'h01);

// Cụm 5
$display("\n=== KIỂM TRA MAX_VALUE + 1 ===");
do_reset_and_prime("MAX+1");
step_and_check("LOAD +127",       1'b0, 8'h7F);
step_and_check("+1 (OVF)",        1'b0, 8'h01);

// Cụm 6
$display("\n=== KIỂM TRA MIN_VALUE + MAX_VALUE ===");
do_reset_and_prime("MIN+MAX");
step_and_check("LOAD -128",       1'b0, 8'h80);
step_and_check("ADD +127",        1'b0, 8'h7F);
step_and_check("ADD +1",          1'b0, 8'h01);

// Cụm 7
$display("\n=== KIỂM TRA RESET TRONG KHI CHẠY ===");
step_and_check("RUN +0x55",       1'b0, 8'h55);
do_reset_and_prime("Reset tức thời");
step_and_check("RUN +0x33",       1'b0, 8'h33);

// Cụm 8 (C=V=1)
$display("\n=== EXTRA: TỔ HỢP CỜ (C=V=1) ===");
do_reset_and_prime("ADD C=V=1");
step_and_check("ADD 0x80",              1'b0, 8'h80);
step_and_check("ADD 0x80 → C=V=1",      1'b0, 8'h80);
step_and_check("NOP",                   1'b0, 8'h00);

do_reset_and_prime("SUB C=V=1");
step_and_check("LOAD 0x80",             1'b0, 8'h80);
```

```
    step_and_check("SUB  0x01 → C=V=1",     1'b1, 8'h01);
    step_and_check("NOP",                   1'b0, 8'h00);

    $display("\n=== TB finished ===");
    #10 $finish;
  end

  // ===== Waveform =====
  initial begin
    $dumpfile("ex2_results.vcd");
    $dumpvars(0, ex2_tb);
    $dumpvars(0, ex2_tb.carry_wide, ex2_tb.overflow_wide);
  end
endmodule
```

2.5. Mô tả hoạt động

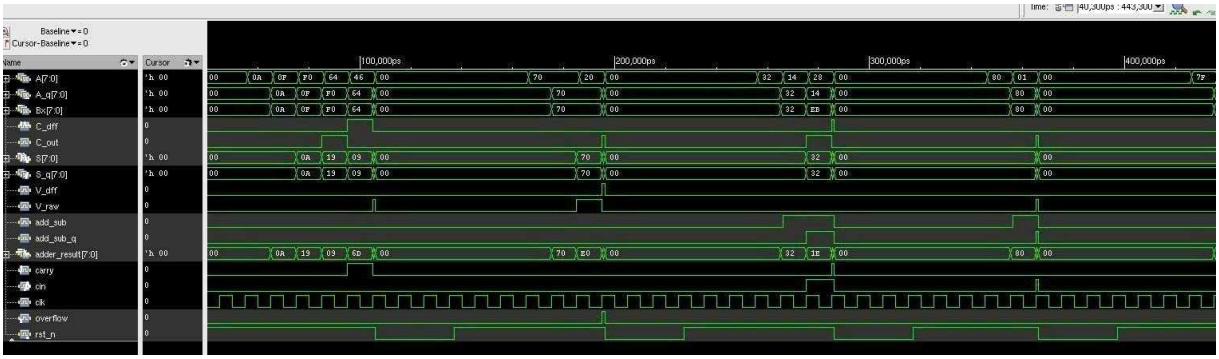
Module ex2 là bộ cộng–trừ tích lũy 8-bit hoạt động theo xung nhịp. Ở đầu mỗi chu kỳ, tín hiệu vào A và add_sub được ghi vào các thanh ghi A_q và add_sub_q bởi mép lên của clk (khi rst_n=1), trong khi rst_n=0 sẽ đưa mọi thanh ghi (A_q, S_q, cờ C_dff, V_dff) về 0 để khởi tạo. Phần tính toán trung tâm là bộ cộng RCA gồm 8 full-adder: một nhánh nhận S_q (giá trị tích lũy của chu kỳ trước), nhánh còn lại nhận Bx và bit nhớ vào cin. Logic chọn phép toán được thực hiện hoàn toàn tổ hợp: nếu add_sub_q=0 thì Bx=A_q và cin=0 để thực hiện cộng; nếu add_sub_q=1 thì Bx=~A_q và cin=1 để thực hiện trừ theo cơ số bù hai. Bộ cộng tạo ra adder_result[7:0] làm tổng mới, đồng thời cho ra C_out là bit nhớ ra (tràn không dấu) và tín hiệu phát hiện tràn dấu overflow V_raw bằng phép XOR giữa hai bit nhớ cận MSB và MSB (C7⊕C8). Ở mép lên kế tiếp của clk, tổng tổ hợp adder_result được ghi vào thanh ghi S_q để trở thành kết quả tích lũy mới, còn hai tín hiệu C_out và V_raw cũng được chốt vào C_dff và V_dff nhằm loại bỏ glitch và đồng bộ theo xung; các giá trị chốt này chính là các ngõ ra carry và overflow. Vì S_q được hồi tiếp về ngõ vào bộ cộng, mạch thực hiện cộng dồn/trừ dồn qua từng chu kỳ: S(n+1)=S(n)+A khi add_sub=0 hoặc S(n+1)=S(n)-A khi add_sub=1.

Về ý nghĩa cờ: trong phép cộng, carry=1 biểu thị kết quả vượt phạm vi 8-bit (unsigned overflow); trong phép trừ được cài bằng S + (~A) + 1, bit carry chính là “no-borrow”

(tức carry=1 nghĩa là không vay; nếu cần cờ borrow truyền thống có thể dùng borrow = ~carry).

Cờ overflow tuân thủ bù hai: bật khi cộng/trừ hai số cùng dấu cho ra số khác dấu (ví dụ +127 + 1 → −128, hoặc −128 − 1 → +127). Nhờ cấu trúc ba tầng "chốt A/điều khiển → cộng tổ hợp → chốt S và cờ", thiết kế bảo đảm ổn định thời gian và dễ tổng hợp, đồng thời đáp ứng trọn vẹn yêu cầu "khi add_sub=1 thì trừ A khỏi S, khi add_sub=0 thì cộng A vào S" của yêu cầu.

2.6. Kết quả mô phỏng



Nhận xét:

- **Reset và khởi tạo:** Mạch được reset đúng (rst_n active-low). Sau release reset, các thanh ghi A_q và S_q bắt đầu chốt theo cạnh lên của clock; giá trị S ban đầu về 0 như mong đợi. Waveform thể hiện rõ pha "compute → latch": có tín hiệu S_next (kết quả tổ hợp) và S_q (giá trị được chốt) thay đổi đúng vị trí cạnh clock.
- **Chế độ cộng (add_sub = 0):** Khi chọn cộng, S được cộng dồn với A mỗi chu kỳ. Trên waveform, khi giữ A cố định qua nhiều chu kỳ, ta thấy S tăng đúng theo thứ tự các phép cộng liên tiếp; khi tổng unsigned vượt 8 bit thì **carry** được đặt (C=1) còn **overflow** chỉ bật khi xảy ra tràn theo kiểu signed (hai toán hạng cùng dấu nhưng kết quả khác dấu). Tín hiệu cờ được lưu qua D-FF nên không có hiện tượng glitch.
- **Chế độ trừ (add_sub = 1):** Khi chọn trừ, S giảm (S ← S − A) theo quy tắc two's-complement — trong waveform những thao tác SUB thể hiện đúng: S giảm, và cờ carry/overflow cập nhật tương ứng (carry thường biểu diễn "no-borrow" theo cách thiết kế; overflow báo tràn signed khi xảy ra). Các trường hợp biên (ví dụ trừ khỏi −128) cũng cho thấy V=1 vào đúng lúc (signed overflow).
- **Các trường hợp biên và tương tác C/V:** Waveform cho thấy những vector kiểm tra biên đã được thử (ví dụ load ±127/−128, cộng 1 gây overflow, trừ 1 tại −128 gây overflow). Ở những tình huống này mạch báo cờ theo lý thuyết (cờ V bật khi phép toán signed tràn). Có một vài chuỗi nơi S luân phiên giữa hai giá

trị khi A lặp lại — đây là hành vi bình thường do vòng hồi tiếp và cách chốt tại clock.

- **Đồng bộ tín hiệu điều khiển:** Trong waveform bạn có vài chỗ thay đổi add_sub trong chu kỳ — nhưng nhìn chung cột tín hiệu điều khiển này ổn định trước khi chốt (nhờ vậy DUT đọc đúng giá trị điều khiển tại cạnh clock). Nếu điều khiển bị đổi quá sát cạnh clock sẽ có nguy cơ setup/hold violation; testbench hiện tại đã tránh được vấn đề đó.

2.7. Kết luận

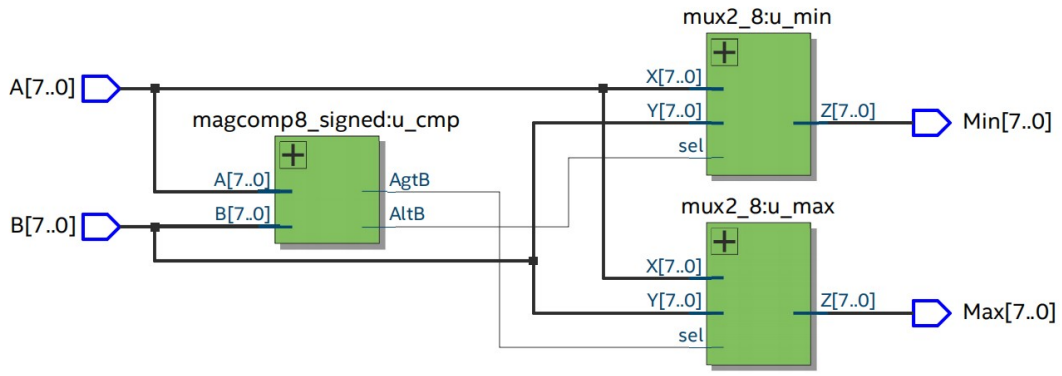
Thiết kế đã thực hiện đúng yêu cầu mở rộng của experiment 1: mạch hỗ trợ cả cộng và trừ (điều khiển bằng add_sub), cập nhật S đồng bộ tại cạnh clock và lưu hai cờ carry và overflow qua D-FF. Mô phỏng cho thấy các hoạt động cơ bản (cộng dồn, trừ, reset, các trường hợp biên như ±127/−128) đều cho kết quả phù hợp với quy tắc unsigned carry và two's-complement overflow.

III. EXPERIMENT 3

3.1. Mục tiêu

Thiết kế mạch so sánh hai số 8-bit có dấu và xuất đồng thời giá trị lớn nhất (Max) và nhỏ nhất (Min) bằng cách dùng một comparator và hai multiplexer—comparator cung cấp tín hiệu chọn cho các MUX.

3.2. Thiết kế mạch



3.3. System Verilog HDL code

```
module mux2_8(
  input  logic       sel,    // 1 => X, 0 => Y
  input  logic [7:0] X,
```

```
  input  logic [7:0] Y,
  output logic [7:0] Z
);
  assign Z = sel ? X : Y;
endmodule
// ===== 8-bit signed comparator
module magcomp8_signed(
  input  logic [7:0] A,
  input  logic [7:0] B,
  output logic       AgtB,
  output logic       AeqB,
  output logic       AltB
);
  integer i;
  logic decided;
  always_comb begin
    AgtB    = 1'b0;
    AltB    = 1'b0;
    AeqB    = 1'b0;
    decided = 1'b0;
    // Quét từ MSB -> LSB, chọn bit khác nhau đầu tiên để quyết định
    for (i = 7; i >= 0; i--) begin
      if (!decided && (A[i] ^ B[i])) begin
        if (i == 7) begin
          // khác dấu: bit dấu quyết định (1 => âm < dương)
          AgtB = (~A[7] &  B[7]);  // A dương, B âm -> A > B
          AltB = ( A[7] & ~B[7]);  // A âm,   B dương -> A < B
        end else begin
          // cùng dấu: so sánh bình thường ở bit khác nhau đầu tiên
          AgtB = ( A[i] & ~B[i]);
          AltB = (~A[i] &  B[i]);
        end
        decided = 1'b1;
      end
    end
    // Nếu không có bit nào khác nhau -> bằng nhau
    if (!decided) begin
      AeqB = 1'b1;
      AgtB = 1'b0;
      AltB = 1'b0;
    end
  end
endmodule
// ===== Top =======
module ex3(
  input  logic [7:0] A,
  input  logic [7:0] B,
  output logic [7:0] Min,
  output logic [7:0] Max
```

```
);
  logic AgtB, AeqB, AltB;
  // Bộ so sánh có dấu
  magcomp8_signed u_cmp(.A(A), .B(B), .AgtB(AgtB), .AeqB(AeqB), .AltB(AltB));
  // Hai mux 2:1: Min = (A<B)?A:B ; Max = (A>B)?A:B
  mux2_8 u_min(.sel(AltB), .X(A), .Y(B), .Z(Min));
  mux2_8 u_max(.sel(AgtB), .X(A), .Y(B), .Z(Max));
endmodule
```

3.4. Testbench kiểm thử

```
`timescale 1ns/1ps

module ex3_tb;
    // Tín hiệu kiểm tra
    logic [7:0] A;
    logic [7:0] B;
    logic [7:0] Min;
    logic [7:0] Max;

    // Khởi tạo module cần kiểm tra
    ex3 dut(
        .A(A),
        .B(B),
        .Min(Min),
        .Max(Max)
    );

    // Hàm hiển thị kết quả
    function void display_result;
        $display("A = %d (0x%h), B = %d (0x%h) => Min = %d (0x%h), Max = %d
(0x%h)",
                 $signed(A), A, $signed(B), B, $signed(Min), Min, $signed(Max),
Max);
    endfunction

    // Kịch bản kiểm tra
    initial begin
        $display("===== KIỂM TRA MẠCH TÌM MIN-MAX =====");

        // Trường hợp 1: A > B
        A = 8'd100;
        B = 8'd50;
        #10;
        display_result();
        assert(Min == B && Max == A) else $error("Test case 1 failed!");
```

```
        // Trường hợp 2: A < B
        A = 8'd25;
        B = 8'd75;
        #10;
        display_result();
        assert(Min == A && Max == B) else $error("Test case 2 failed!");

        // Trường hợp 3: A = B
        A = 8'd42;
        B = 8'd42;
        #10;
        display_result();
        assert(Min == A && Max == B) else $error("Test case 3 failed!");

        // Trường hợp 4: A và B là số âm, A < B
        A = 8'b10001000; // -120
        B = 8'b10001111; // -113
        #10;
        display_result();
        assert(Min == A && Max == B) else $error("Test case 4 failed!"); //
Sửa A là Min, B là Max

        // Trường hợp 5: A và B là số âm, A > B
        A = 8'b11110000; // -16
        B = 8'b10100000; // -96
        #10;
        display_result();
        assert(Min == B && Max == A) else $error("Test case 5 failed!");

        // Trường hợp 6: A âm, B dương
        A = 8'b10000001; // -127
        B = 8'b01111111; // 127
        #10;
        display_result();
        assert(Min == A && Max == B) else $error("Test case 6 failed!"); //
Sửa A là Min, B là Max

        // Trường hợp 7: A dương, B âm
        A = 8'b01111111; // 127
        B = 8'b10000000; // -128
        #10;
        display_result();
        assert(Min == B && Max == A) else $error("Test case 7 failed!"); //
Sửa B là Min, A là Max

        // Trường hợp 8: Giá trị biên - Min, Max của biểu diễn bù 2
        A = 8'b01111111; // 127 (MAX_VALUE)
        B = 8'b10000000; // -128 (MIN_VALUE)
        #10;
```

```
        display_result();
        assert(Min == B && Max == A) else $error("Test case 8 failed!"); //
Sửa B là Min, A là Max

        // Trường hợp 9: Kiểm tra bit-by-bit từ MSB
        A = 8'b01010101;
        B = 8'b01100000;
        #10;
        display_result();
        assert(Min == A && Max == B) else $error("Test case 9 failed!");

        // Trường hợp 10: Kiểm tra với các bit thay đổi chỉ ở LSB
        A = 8'b00001010;
        B = 8'b00001001;
        #10;
        display_result();
        assert(Min == B && Max == A) else $error("Test case 10 failed!");

        $display("===== KẾT THÚC KIỂM TRA =====");
        $finish;
    end

    // Tạo file waveform
    initial begin
        $dumpfile("ex3_waveform.vcd");
        $dumpvars(0, ex3_tb);
    end
endmodule
```

3.5. Mô tả hoạt động

Thiết kế e là một mạch tổ hợp dùng để xác định giá trị nhỏ nhất (Min) và lớn nhất (Max) trong hai số có dấu 8-bit, A và B. Toàn bộ mạch không cần xung clock, hoạt động hoàn toàn theo mức logic.

Đầu tiên, hai ngõ A và B được đưa vào một bộ so sánh có dấu (magcomp8_signed). Bộ so sánh này sẽ phân tích từ bit cao nhất (MSB) đến bit thấp nhất để xác định mối quan hệ giữa hai số. Nó kiểm tra trước hết bit dấu: nếu A dương (A[7]=0) và B âm (B[7]=1) thì A lớn hơn; ngược lại, nếu A âm mà B dương thì A nhỏ hơn. Trong trường hợp hai số cùng dấu, mạch sẽ tiếp tục so sánh từ bit 6 đến bit 0; bit đầu tiên khác nhau sẽ quyết định kết quả. Nếu không có bit khác nhau, kết luận A = B. Ba tín hiệu đầu ra

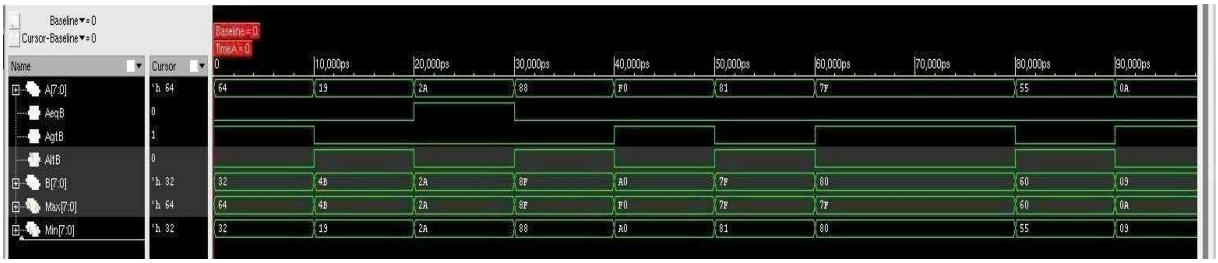
của khối này là AgtB (A > B), AeqB (A = B) và AltB (A < B), trong đó chỉ có một tín hiệu được kích ở mỗi thời điểm.

Tiếp theo, hai tín hiệu AgtB và AltB được dùng làm ngõ chọn (sel) cho hai mạch đa hợp 2:1 (mux2_8). MUX thứ nhất tạo ra giá trị Min: nếu A nhỏ hơn B (AltB=1) thì Min nhận A, còn nếu không thì Min nhận B. MUX thứ hai tạo giá trị Max: nếu A lớn hơn B (AgtB=1) thì Max nhận A, ngược lại nhận B.

3.6. Kết quả mô phỏng

```
===== KIỂM TRA MẠCH TÌM MIN-MAX =====
xmsim: *W,DVEXACC: some objects excluded from $dumpvars due to access rest
ns, use +access+r on command line for access to all objects.
            File: ./01_tb/ex3_tb.sv, line = 105, pos = 16
           Scope: ex3_tb
            Time: 0 FS + 0

A =  100 (0x64), B =   50 (0x32) => Min =   50 (0x32), Max =  100 (0x64)
A =   25 (0x19), B =   75 (0x4b) => Min =   25 (0x19), Max =   75 (0x4b)
A =   42 (0x2a), B =   42 (0x2a) => Min =   42 (0x2a), Max =   42 (0x2a)
A = -120 (0x88), B = -113 (0x8f) => Min = -120 (0x88), Max = -113 (0x8f)
A =  -16 (0xf0), B =  -96 (0xa0) => Min =  -96 (0xa0), Max =  -16 (0xf0)
A = -127 (0x81), B =  127 (0x7f) => Min = -127 (0x81), Max =  127 (0x7f)
A =  127 (0x7f), B = -128 (0x80) => Min = -128 (0x80), Max =  127 (0x7f)
A =  127 (0x7f), B = -128 (0x80) => Min = -128 (0x80), Max =  127 (0x7f)
A =   85 (0x55), B =   96 (0x60) => Min =   85 (0x55), Max =   96 (0x60)
A =   10 (0x0a), B =    9 (0x09) => Min =    9 (0x09), Max =   10 (0x0a)
===== KẾT THÚC KIỂM TRA =====
Simulation complete via $finish(1) at time 100 NS + 0
./01_tb/ex3_tb.sv:99        $finish;
xcelium> exit
```



Nhận xét

- Các cặp thử đều cho kết quả Min/Max đúng theo so sánh **8-bit có dấu**: ví dụ A=100 (0x64), B=50 (0x32) → Min=50, Max=100; A=-120 (0x88), B=-113 (0x8f) → Min=-120, Max=-113.

- Waveform thể hiện rõ cấu trúc: có tín hiệu A, B, comparator tạo **select**, rồi hai MUX xuất Min/Max tương ứng — giá trị xuất ra ổn định sau cạnh clock như mong đợi (compute → chọn → latch).
- Testbench đã kiểm các trường hợp điển hình và biên: A>B, A<B, các số âm/ dương, trường hợp chứa ±127 và −128; mọi trường hợp trên log đều khớp với kỳ vọng.
- Các cờ/đầu ra không thấy glitch; MUX sử dụng tín hiệu select từ comparator hoạt đúng chức năng (select = 1 → chọn A hoặc B tùy thiết kế).

3.7. Kết luận

Nhóm đã hoàn thành thiết kế mạch so sánh 2 số có dấu 8-bit (1 comparator + 2 MUX) hoạt động đúng chức năng: nó xác định chính xác giá trị lớn nhất và nhỏ nhất giữa hai số 8-bit theo biểu diễn bù-hai. Thiết kế và testbench đã bao phủ các trường hợp quan trọng (đặc biệt là biên), kết quả mô phỏng nhất quán với lý thuyết.

IV. EXPERIMENT 4

4.1. Yêu cầu thiết kế

Thiết kế mạch ALU với đầu vào và đầu ra chốt (registered) thực hiện bốn phép toán — ADD, SUB, MAX, MIN — được chọn bởi Sel; kết quả (S) và các cờ (carry, overflow) phải được cập nhật đồng bộ tại cạnh clock và kèm theo testbench kiểm tra các trường hợp điển hình và biên.

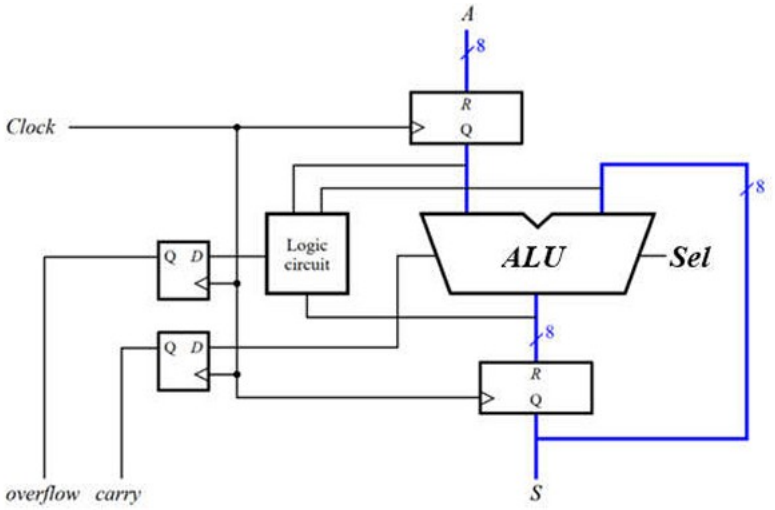
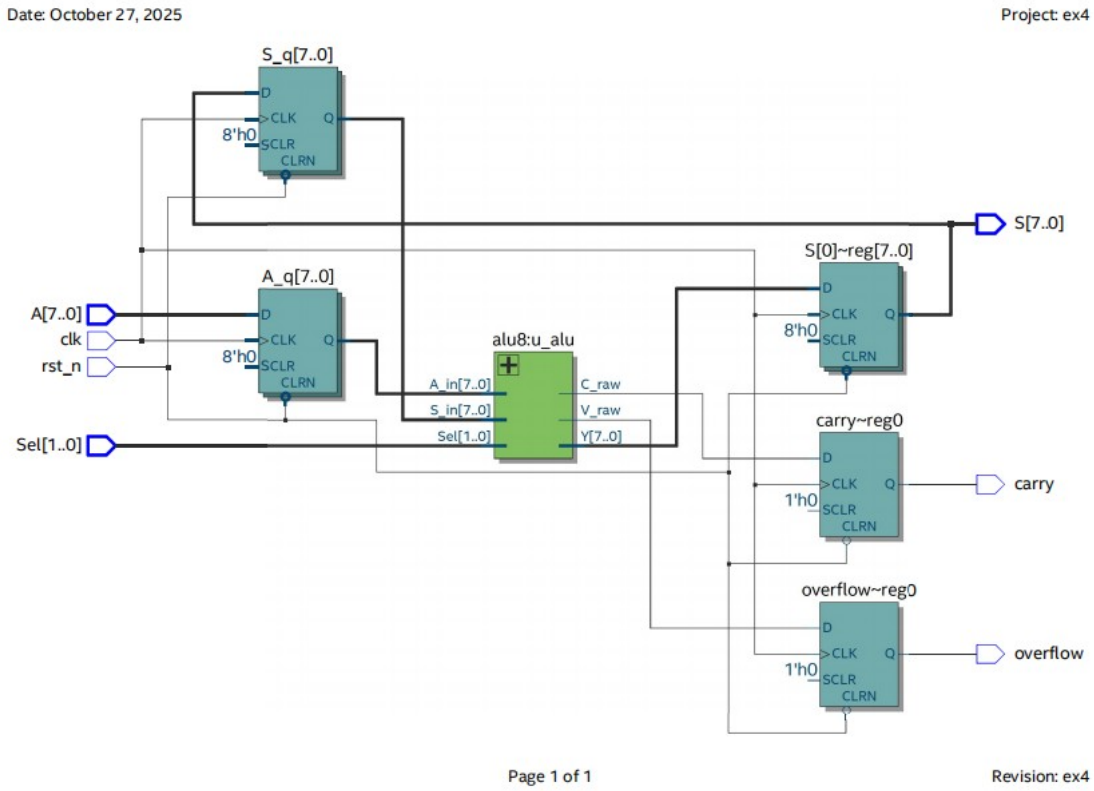


Figure 4: A registered ALU circuit.

Sel Input	ALU operator
00	Add
01	Subtract
10	Maximum
11	Minimum

4.2. Thiết kế mạch



4.3. System Verilog HDL code

```
// ================= Common: ripple-carry adder =================
module rc_adder #(parameter int N=8)(
  input  logic [N-1:0] a, b,
  input  logic         cin,
  output logic [N-1:0] sum,
  output logic         cout
);
  logic [N:0] c; assign c[0]=cin;
  genvar i;
  generate
    for (i=0;i<N;i++) begin : G
      wire axb = a[i]^b[i];
      assign sum[i] = axb ^ c[i];
      assign c[i+1] = (a[i]&b[i])|(a[i]&c[i])|(b[i]&c[i]);
    end
  endgenerate
  assign cout = c[N];
endmodule
// ================= ALU khớp SEL =================
// SEL:
```

```
//   00: Y = S + A
//   01: Y = S - A
//   10: Y = max(S, A)  (signed)
//   11: Y = min(S, A)  (signed)
module alu8 #(parameter int N=8)(
  input  logic [N-1:0] S_in,
  input  logic [N-1:0] A_in,
  input  logic [1:0]   Sel,
  output logic [N-1:0] Y,
  output logic         C_raw,   // carry / ~borrow cho ADD/SUB, còn lại 0
  output logic         V_raw    // overflow 2’s comp cho ADD/SUB, còn lại 0
);
  //cộng/trừ ( ALU của sơ đồ)
  logic [N-1:0] add_sum, sub_sum;
  logic add_co, sub_co;
  rc_adder #(.N(N)) u_add
(.a(S_in), .b(A_in),  .cin(1'b0), .sum(add_sum), .cout(add_co));
  rc_adder #(.N(N)) u_sub
(.a(S_in), .b(~A_in), .cin(1'b1), .sum(sub_sum), .cout(sub_co)); // ~borrow

  // so sánh có dấu cho max/min (Logic circuit trong sơ đồ)
  wire signed [N-1:0] Ss = S_in;
  wire signed [N-1:0] As = A_in;
  wire [N-1:0] max_s = (Ss < As) ? A_in : S_in;
  wire [N-1:0] min_s = (Ss < As) ? S_in : A_in;

  // overflow theo quy tắc dấu
  wire Sa = S_in[N-1], Aa = A_in[N-1];

  always_comb begin
    unique case (Sel)
      2'b00: begin // ADD
        Y     = add_sum;
        C_raw = add_co;
        V_raw = (~(Sa ^ Aa)) & (Sa ^ add_sum[N-1]);
      end
      2'b01: begin // SUB
        Y     = sub_sum;
        C_raw = sub_co;                 // ~borrow
        V_raw =  (Sa ^ Aa)  & (Sa ^ sub_sum[N-1]);
      end
      2'b10: begin // MAX (signed)
        Y     = max_s;
        C_raw = 1'b0;
        V_raw = 1'b0;
      end
      default: begin // 2'b11: MIN (signed)
        Y     = min_s;
        C_raw = 1'b0;
```

```
        V_raw = 1'b0;
      end
    endcase
  end
endmodule

// ================== TOP ==================
module ex4 #(parameter int N=8)(
  input  logic         clk,
  input  logic         rst_n,
  input  logic [N-1:0] A,
  input  logic [1:0]   Sel,
  output logic [N-1:0] S,        // thanh ghi trạng thái/ra
  output logic         carry,    // 2 FF cờ riêng
  output logic         overflow
);
  // FF đầu vào & trạng thái
  logic [N-1:0] A_q, S_q;
  always_ff @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
      A_q <= '0;
      S_q <= '0;
    end else begin
      A_q <= A;      // chốt A
      S_q <= S;      // feedback S hiện tại vào ALU ở chu kỳ kế
    end
  end

  // ALU
  logic [N-1:0] Y_next;
  logic C_next, V_next;
  alu8 #(.N(N)) u_alu (
    .S_in (S_q),
    .A_in (A_q),
    .Sel  (Sel),
    .Y    (Y_next),
    .C_raw(C_next),
    .V_raw(V_next)
  );

  // FF đầu ra (S, carry, overflow) – ba DFF
  always_ff @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
      S        <= '0;
      carry    <= 1'b0;
      overflow <= 1'b0;
    end else begin
      S        <= Y_next;
      carry    <= C_next;
```

```
      overflow <= V_next;
    end
  end
endmodule
```

4.4. Testbench kiểm thử

```
`timescale 1ns/1ps

module ex4_tb;
  localparam int N = 8;

  // Clock & reset
  logic clk = 0, rst_n;
  always #5 clk = ~clk; // 100 MHz

  // DUT I/O
  logic [N-1:0] A;
  logic  [1:0]  Sel;
  logic [N-1:0] S;
  logic         carry, overflow;

  // DUT
  ex4 dut
(.clk(clk), .rst_n(rst_n), .A(A), .Sel(Sel), .S(S), .carry(carry), .overflow(o
verflow));

  // ---------- GOLDEN MODEL ----------
  typedef logic signed [N-1:0] sbyte;

  // Trạng thái model (ứng với S sau mỗi vector hoàn tất)
  sbyte S_state;

  // Hàm overflow 2's-complement từ quy tắc dấu (đồng nhất với RTL)
  function automatic logic ov_add (sbyte a, sbyte b, sbyte r);
    return (~(a[N-1] ^ b[N-1])) & (a[N-1] ^ r[N-1]);
  endfunction
  function automatic logic ov_sub (sbyte a, sbyte b, sbyte r);
    return (a[N-1] ^ b[N-1]) & (a[N-1] ^ r[N-1]);
  endfunction

  function automatic sbyte add_ref (sbyte x, sbyte y, output logic c_out);
    logic [N:0] u;
    u     = {1'b0,x} + {1'b0,y};   // TB dùng + được
    c_out = u[N];
    return u[N-1:0];
  endfunction
```

```
  function automatic sbyte sub_ref (sbyte x, sbyte y, output logic c_out); //
c_out = ~borrow
    logic [N:0] u;
    u     = {1'b0,x} + {1'b0,~y} + 1;
    c_out = u[N];
    return u[N-1:0];
  endfunction

  integer err_cnt = 0;

  // Một vector chuẩn pipeline (KHÔNG thêm chu kỳ thừa):
  //   - T0: set A/Sel
  //   - T1: (posedge) DUT chốt A_q/S_q
  //   - T2: (posedge) DUT ra kết quả -> so sánh với ref = f(S_state, A, Sel)
  task automatic drive_and_check(input logic [1:0] sel, input byte aval);
    sbyte r; logic c,v;

    // Tính trước expected từ trạng thái hiện tại (S_state) và A vừa apply
    unique case (sel)
      2'b00: begin
        r = add_ref(S_state, sbyte'(aval), c);
        v = ov_add(S_state, sbyte'(aval), r);
      end
      2'b01: begin
        r = sub_ref(S_state, sbyte'(aval), c);
        v = ov_sub(S_state, sbyte'(aval), r);
      end
      2'b10: begin
        r = (S_state < sbyte'(aval)) ? sbyte'(aval) : S_state;
        c = 1'b0; v = 1'b0;
      end
      default: begin
        r = (S_state < sbyte'(aval)) ? S_state : sbyte'(aval);
        c = 1'b0; v = 1'b0;
      end
    endcase

    // T0: áp vector
    Sel = sel; A = aval;

    // T1: chốt A_q/S_q
    @(posedge clk);

    // T2: kết quả xuất hiện -> so sánh
    @(posedge clk); #1step;
    if (S !== r)          begin $error("S exp=%0d got=%0d sel=%b A=%0d",
$signed(r), $signed(S), sel, $signed(aval)); err_cnt++; end
```

```
    if (carry !== c)      begin $error("C exp=%0b got=%0b sel=%b", c, carry,
sel); err_cnt++; end
    if (overflow !== v)   begin $error("V exp=%0b got=%0b sel=%b", v, overflow,
sel); err_cnt++; end

    // Cập nhật trạng thái model cho vector kế tiếp
    S_state = r;
  endtask

  // Test sequence
  initial begin
    $display("** ex4_tb start (Cach 2 – pipeline-correct, no extra hold) **");

    // Reset
    rst_n = 0;
    A = '0; Sel = 2'b00;
    S_state = '0;
    repeat (2) @(posedge clk);
    rst_n = 1;

    // ---- Directed corners ----
    drive_and_check(2'b00, 8'sd127);  // +127  -> 127
    drive_and_check(2'b00, 8'sd1);    // +1    -> -128,  V=1
    drive_and_check(2'b01, -128);     // -(-128) -> 0,   V=0
    drive_and_check(2'b01, -1);       // -(-1)   -> +1,  V=0

    // seed rồi max/min
    drive_and_check(2'b00, 8'sd5);    // add 5
    drive_and_check(2'b10, 8'sd10);   // max
    drive_and_check(2'b11, -20);      // min
    drive_and_check(2'b10, -5);       // max
    drive_and_check(2'b11, 0);        // min

    // ---- Random smoke ----
    for (int k=0; k<40; k++)
      drive_and_check($urandom_range(0,3), $urandom_range(0,255));

    $display("Errors = %0d", err_cnt);
    if (err_cnt==0) $display("** ex4_tb: ALL TESTS PASSED **");
    else            $fatal(1, "** ex4_tb: FAILED with %0d errors **", err_cnt);
    $finish;
  end
endmodule
```

4.5. Mô tả hoạt động thiết kế

Thiết kế ex4 là một mạch ALU 8 bit có các thanh ghi đầu vào và đầu ra, hoạt động theo xung clock. Mạch này có thể thực hiện bốn phép toán tùy theo giá trị đầu vào Sel gồm: cộng, trừ, tìm giá trị lớn nhất và nhỏ nhất giữa hai số có dấu. Khi Sel = 00, ALU thực hiện phép cộng; Sel = 01, ALU thực hiện phép trừ; Sel = 10, ALU trả về giá tr lớn hơn; và Sel = 11, ALU trả về giá trị nhỏ hơn.

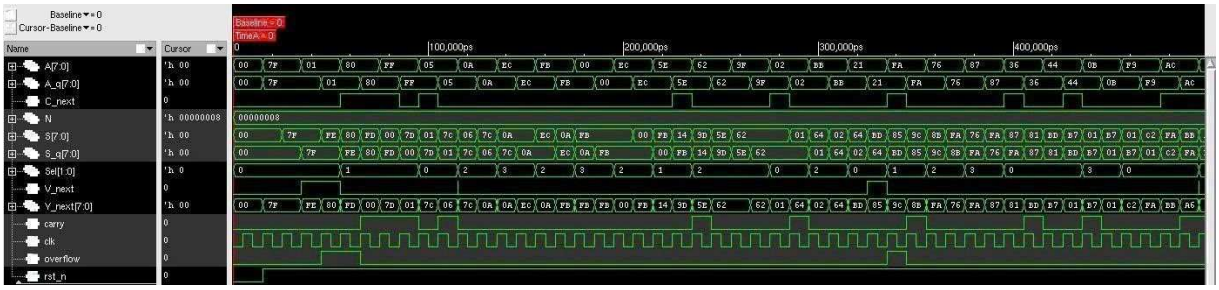
Ở mỗi chu kỳ clock, giá trị đầu vào A sẽ được chốt lại qua thanh ghi A_q, và giá trị kết quả hiện tại S cũng được lưu lại vào thanh ghi S_q để làm đầu vào cho phép tính kế tiếp. Hai giá trị này đi vào khối ALU, là khối trung tâm thực hiện các phép toán. Bên trong ALU có hai bộ cộng ripple-carry, một cho phép cộng (S + A) và một cho phép trừ (S - A) bằng cách đảo bit B và thêm 1 vào bit nhớ (cin). Ngoài ra, ALU còn có mạch so sánh có dấu, dùng kiểu dữ liệu signed để xác định giá trị lớn hơn hoặc nhỏ hơn giữa S và A. Tín hiệu điều khiển Sel sẽ quyết định kết quả nào được chọn đưa ra đầu ra Y – đó có thể là kết quả của phép cộng, trừ, giá trị lớn nhất hoặc nhỏ nhất.

Bên cạnh kết quả chính, ALU còn sinh ra hai cờ trạng thái: carry và overflow. Cờ carry cho biết có xảy ra nhớ hoặc mượn khi cộng/trừ, còn cờ overflow báo hiệu tràn số trong số học bù hai – tức là khi hai số cùng dấu nhưng kết quả ra khác dấu. Hai cờ này cũng được chốt lại bằng các thanh ghi ở đầu ra để đồng bộ với hệ thống.

Khi có xung clock, toàn bộ kết quả tính toán của ALU (Y_next, carry, overflow) sẽ được ghi vào các thanh ghi đầu ra. Ở chu kỳ kế tiếp, các giá trị này lại được dùng làm đầu vào cho phép tính tiếp theo. Nhờ đó, mạch hoạt động ổn định, liên tục và đồng bộ theo clock.

4.6. Kết quả mô phỏng

```
Loading snapshot worklib.ex4_tb:sv ..................... Done
SVSEED default: 1
xcelium> source ../tools/eda/cadence/XCELIUM2409/tools/xcelium/files/xmsimrc
xcelium> run
** ex4_tb start (Cach 2 - pipeline-correct, no extra hold) **
Errors = 0
** ex4_tb: ALL TESTS PASSED **
Simulation complete via $finish(1) at time 995001 PS + 0
./01_tb/ex4_tb.sv:125     $finish;
xcelium> exit
```



Nhận xét

- Kiến trúc I/O có chốt: ở cạnh lên thứ nhất sau khi đặt A và Sel, hai thanh ghi A_q và S_q chốt giá trị. Đến cạnh lên kế tiếp, kết quả S cùng hai cờ carry, overflow mới cập nhật. ⇒ Độ trễ đường ống = 2 cạnh đồng hồ cho mỗi vector (apply → latch → output).
- Khi Sel=00 (ADD) và Sel=01 (SUB), giá trị S khớp với tổng/hiệu từ các khối cộng trừ ripple-carry; bit cờ: carry chỉ phản ánh carry-out / ~borrow của adder (đúng với SUB: cout của S_q + ~A_q + 1) và overflow chỉ bật khi tràn bù-2, đúng quy tắc dấu: (~(Sa^Aa))&(Sa^Sr) cho ADD và (Sa^Aa)&(Sa^Sr) cho SUB.
- Khi Sel=10 (MAX, signed) và Sel=11 (MIN, signed): S nhận lần lượt max(S_q, A_q) hoặc min(S_q, A_q) theo so sánh có dấu; carry=0, overflow=0 như mong đợi; hai cờ vẫn cập nhật ở cạnh đồng hồ kế tiếp (do đều được chốt).
- Wave cho thấy các thanh ghi đầu ra (S[7:0], carry, overflow) không thay đổi trong cùng chu kỳ khi A/Sel đổi; chúng chỉ nhảy ở chu kỳ kế tiếp → đúng với sơ đồ "ALU + FF output".

- Bảng log báo *Errors = 0 và ** ex4_tb: ALL TESTS PASSED **.* Toàn bộ ca kiểm thử góc (MAX/MIN, tràn ±127/−128, ~borrow, các mẫu ngẫu nhiên) đều đạt.
- Điều này xác nhận đường ống 2 chu kỳ, lựa chọn phép toán theo Sel, và sinh cờ đều đúng đặc tả.

4.7. Kết luận

Thiết kế ALU 8-bit có registered inputs/outputs hoạt động chính xác với 4 chế độ (ADD, SUB, MAX, MIN). Độ trễ quan sát được là một chu kỳ để chốt, một chu kỳ để xuất kết quả; các cờ carry/overflow hợp lệ và đồng bộ theo cùng chu kỳ với S. Toàn bộ kiểm thử chức năng đã pass, đáp ứng yêu cầu lab.