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

—o0o—

BÀI TẬP CHƯƠNG 5

Thiết Kế Hệ Tuần Tự

SUPERVISOR: Nguyễn Trung Hiếu

SUBJECT: Digital design and verification

GROUP: 08

List of Members

STT	MSSV	Họ Và Tên	Lớp
1	2213874	Nguyễn Thanh Tùng	L01
2	2210780	Nguyễn Đại Đồng	L01
3	2213496	Nguyễn Quốc Tín	L01

Ho Chi Minh City, 01/11/2025

Mục lục

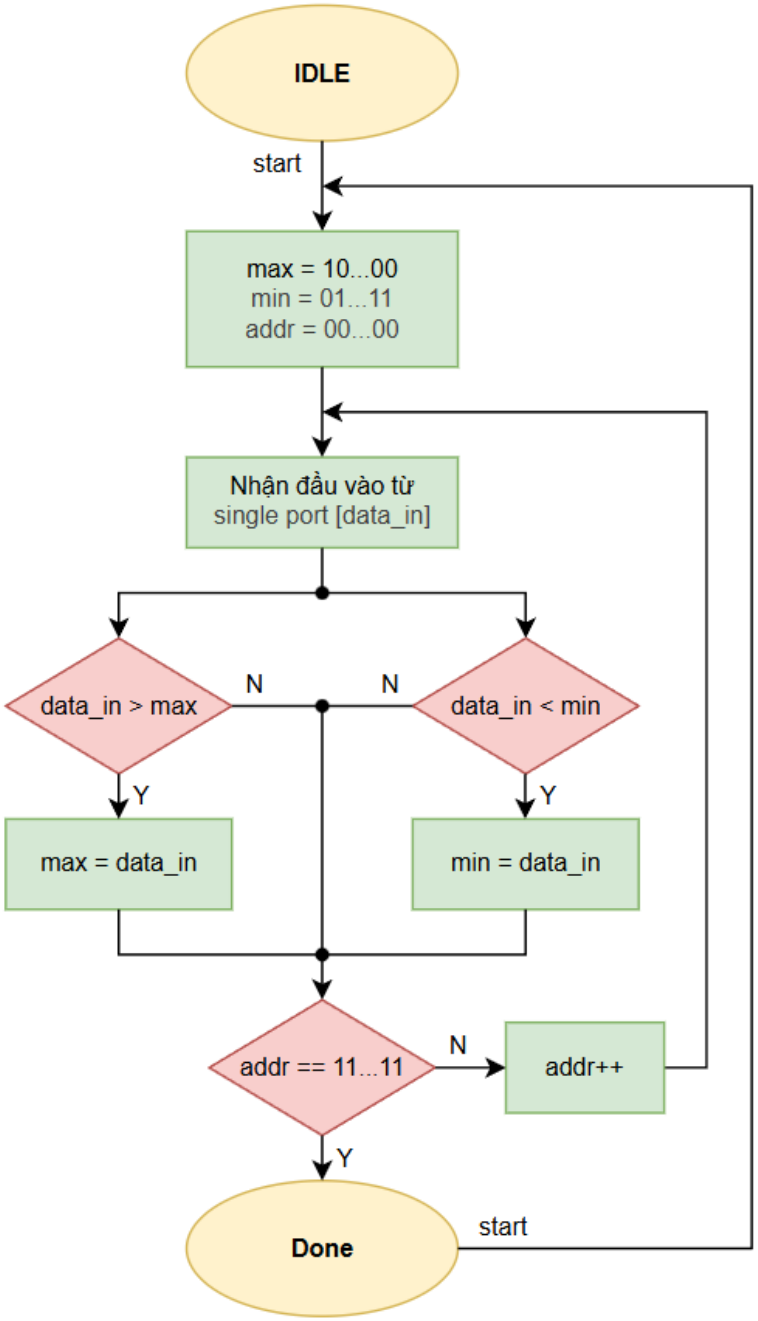
Danh sách hình vẽ

Danh sách bảng

List of Listings

Câu 2

Thiết kế phần cứng dùng để tìm giá trị lớn nhất và nhỏ nhất trong một mảng dữ liệu. Giả sử mảng được lưu trong bộ nhớ Single Port và quá trình đọc/ghi diễn ra đồng bộ theo Clk và hoàn thành trong 1 Clk.



Hình 1: Giải thuật sử dụng.

a) Định nghĩa ngõ vào và ra của thiết kế, vẽ kết nối của thiết kế với bộ nhớ (Yêu cầu phải có chân start và reset).

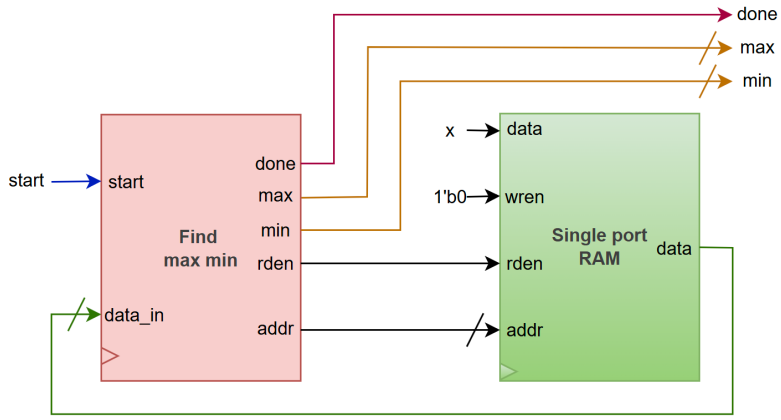
Đề bài không đề cập đến xử lí cho dạng dữ liệu có dấu hoặc không dấu nên nhóm chọn xử lí ở dạng có dấu bù 2.

Hệ thống được thiết kế cần có các tín hiệu:

Tên tín hiệu	IO	Độ rộng	Mô tả
clk	Input	1	Tín hiệu clock
rst_n	Input	1	Reset tích cực mức thấp
start	Input	1	Tín hiệu bắt đầu hoạt động
rden	Output	1	Tín hiệu cho phép đọc dữ liệu từ bộ nhớ
addr	Output	$\lceil \log_2(\text{DEPTH}) \rceil$	Địa chỉ đọc dữ liệu
data_in	Input	signed [WIDTH-1:0]	Dữ liệu đầu vào từ bộ nhớ
done	Output	1	Tín hiệu kết thúc quá trình tìm max/min
max	Output	signed [WIDTH-1:0]	Giá trị lớn nhất tìm được
min	Output	signed [WIDTH-1:0]	Giá trị nhỏ nhất tìm được

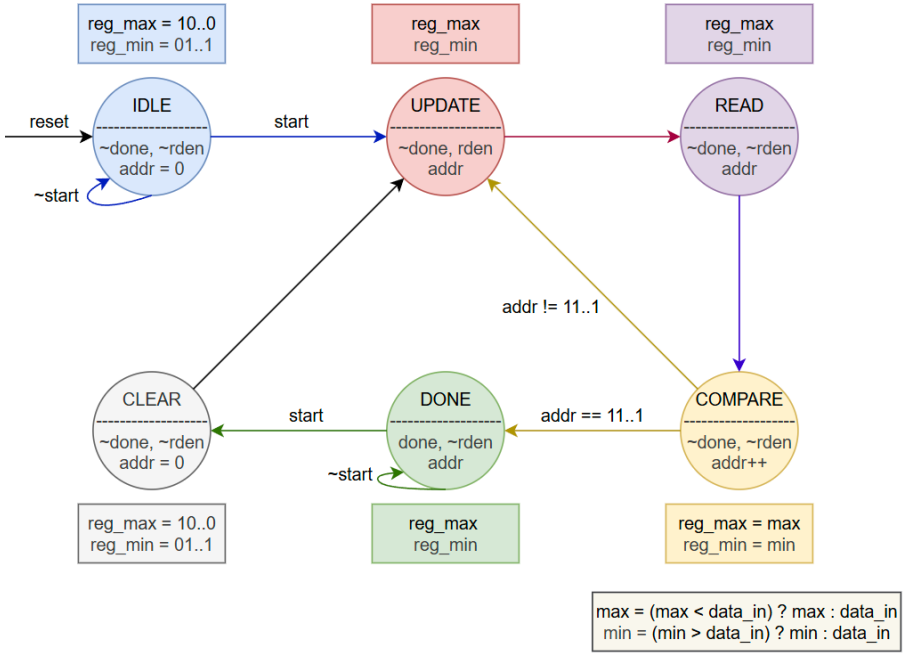
Bảng 1: Bảng tín hiệu I/O của module `max_min`

Các tín hiệu ngõ ra từ khối Find max min được kết nối như hình với các ngõ vào của khối bộ nhớ, riêng ngõ vào wren luôn để mức thấp vì không cập nhật dữ liệu mới vào bộ nhớ.



Hình 2: Tổng quan kết nối thiết kế.

b) Thiết kế máy trạng thái bậc cao của thiết kế.



Hình 3: Máy trạng thái bậc cao.

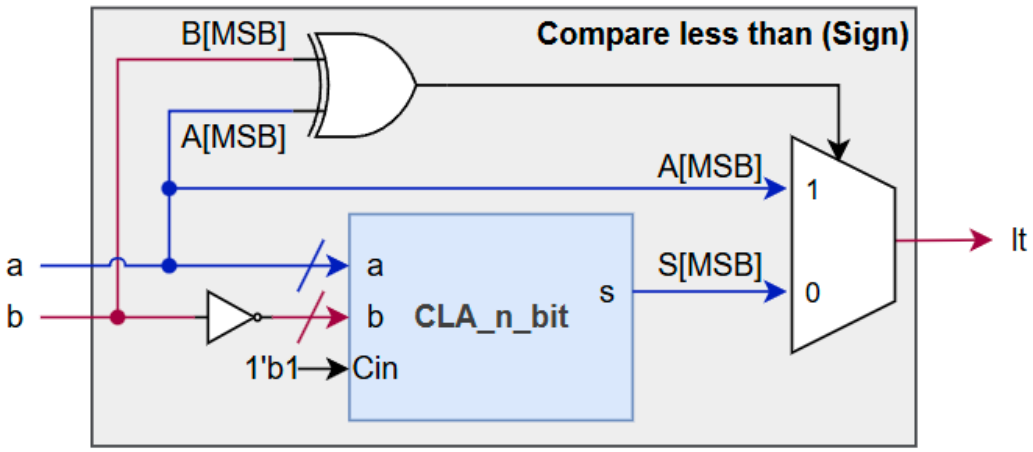
Thiết kế gồm 6 trạng thái chính:

- IDLE: là trạng thái ban đầu, khi reset sẽ luôn trở về trạng thái này, giá trị thanh ghi max sẽ được đặt là cực tiểu và thanh ghi min sẽ được đặt là cực đại, đây là trạng thái chờ ban đầu của hệ thống, khi có tín hiệu start sẽ chuyển sang trạng thái UPDATE.
- UPDATE: ở trạng thái này sẽ cập nhật các giá trị max, min và địa chỉ truy cập bộ nhớ mới được tín toán từ trạng thái COMPARE, đồng thời đưa ngõ ra rden lên mức cao để chuẩn bị nhận dữ liệu ngõ vào mới từ bộ nhớ.
- READ: đây là trạng thái chờ đọc vì ở đây bộ nhớ đọc đồng bộ, do đó cần phải đợi một chu kì để nhận dữ liệu đầu vào.
- COMPARE: ở trạng thái này, sẽ lấy tín hiệu đầu vào đem so sánh với các dữ liệu trong thanh ghi max và min, nếu dữ liệu thỏa sẽ được cập nhật ở trạng thái kế tiếp, đồng thời cũng cập nhật địa chỉ mới (addr + 1). Nếu đã so sánh hết dữ liệu trong bộ nhớ sẽ chuyển sang trạng thái DONE, ngược lại sẽ sang trạng thái UPDATE để chuẩn bị nhận dữ liệu mới từ bộ nhớ.

- DONE: là trạng thái thông báo việc tìm giá trị lớn nhất, nhỏ nhất hoàn tất, lúc này tín hiệu done sẽ tích cực mức cao (done chỉ mức cao ở trạng thái này). Nếu nhận được tín hiệu start mức cao sẽ chuyển sang trạng thái CLEAR, ngược lại sẽ giữ trạng thái hiện tại.
- CLEAR: trạng thái này đóng vai trò đặt lại giá trị thanh ghi max là cực tiểu và thanh ghi min là cực đại để chuẩn bị cho lần tìm giá trị lớn nhất, nhỏ nhất kế tiếp.

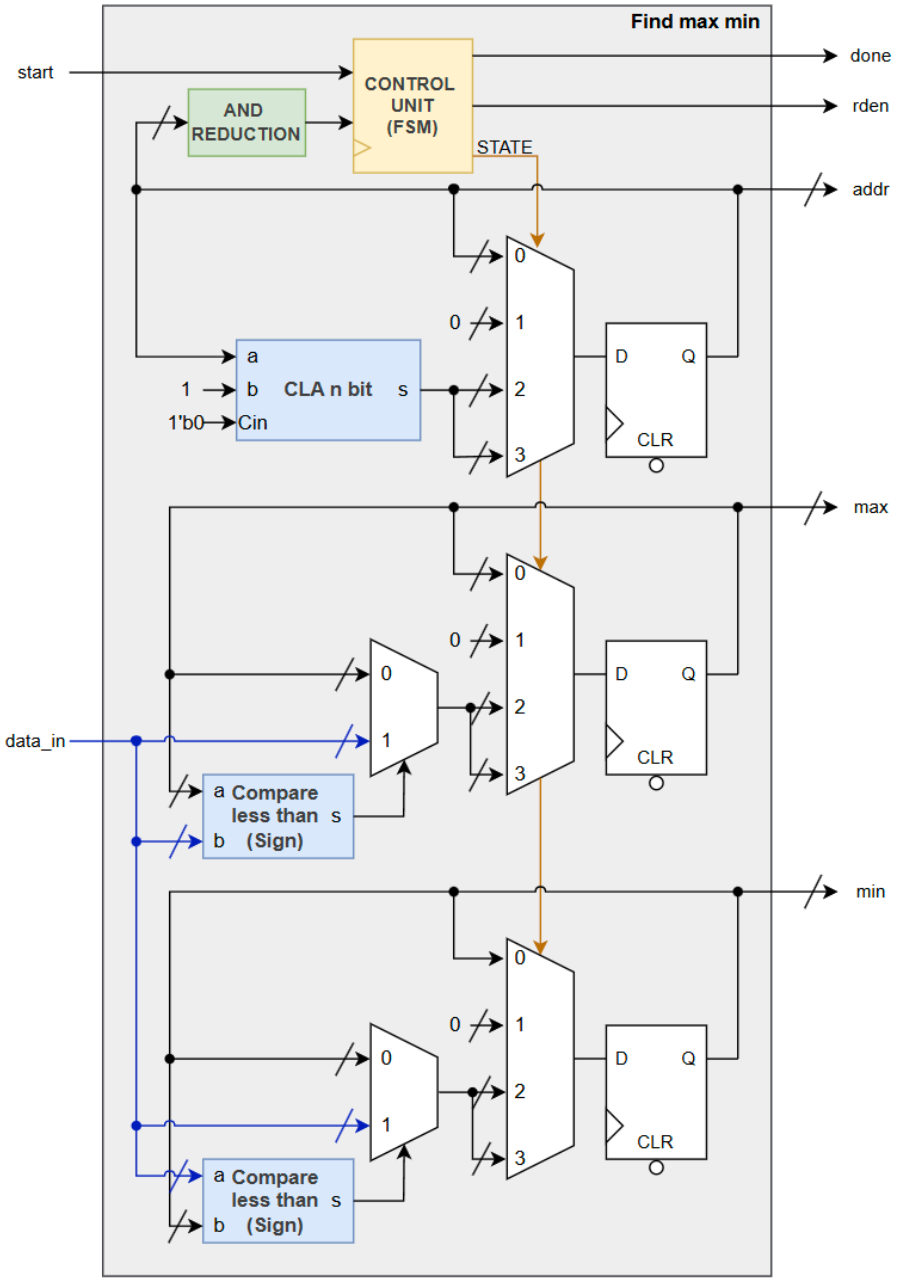
c) Thiết kế Datapath và Control Unit của thiết kế.

Đầu tiên, cần phải thiết kế khối so sánh với ngõ ra nhỏ hơn tích cực cao để so sánh các giá trị của ngõ vào so với giá trị trong 2 thanh ghi max, min.



Hình 4: Bộ so sánh nhỏ hơn.

Bộ so sánh gồm 1 bộ CLA để làm phép trừ tính toán giá trị chênh lệch giữa hai giá trị đầu vào a và b. Một cổng logic XOR 2 ngõ vào (là bit MSB của hai ngõ vào a, b) để xác định a, b có cùng dấu không. Tín hiệu ngõ ra cổng XOR sẽ được nối với tín hiệu lựa chọn của một bộ MUX 2 sang 1 để lựa chọn nếu a, b cùng dấu → sử dụng bit MSB của ngõ ra CLA, nếu a, b khác dấu thì sử dụng bit MSB của a làm cờ so sánh a $<$ b.



Hình 5: Datapath.

Datapath của thiết kế gồm 3 khối MUX 4-1 để xác định giá trị cập nhật cho 3 thanh ghi dựa vào trạng thái hiện tại.

Ba thanh ghi gồm:

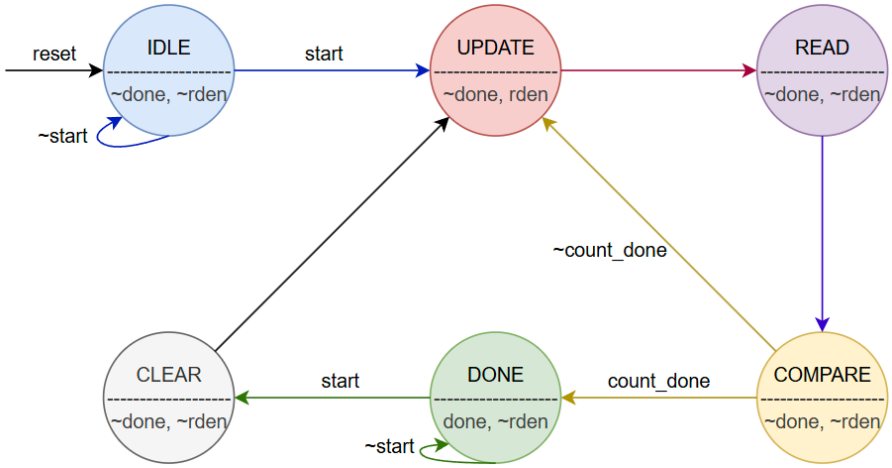
- Thanh ghi địa chỉ, đóng vai trò trỏ vào bộ nhớ, kết hợp với tín hiệu rden để lấy dữ liệu đầu vào.

- Thanh ghi max, lưu giá trị tối đa.
- Thanh ghi min, lưu giá trị cực tiểu.

Một bộ cộng để tăng địa chỉ truy cập bộ nhớ.

Hai bộ so sánh nhỏ hơn để đưa ra tín hiệu lựa chọn giữ giá trị thanh ghi max, min hoặc thay thế bằng giá trị đầu vào data_in.

Một bộ AND Reduction để xác định địa chỉ đạt đến giá trị cuối của bộ nhớ, là điều kiện để chuyển từ trạng thái COMPARE sang trạng thái DONE.



Hình 6: Control Unit FSM.

d) Viết chương trình mô phỏng hoạt động của thiết kế.

```
module lt_compare_sign #(
    parameter WIDTH = 8
)(
    input  logic [WIDTH - 1:0] i_a ,
    input  logic [WIDTH - 1:0] i_b ,
    output logic               o_lt
);

    logic [WIDTH - 1:0] s;

    adder_flex_no_carry #(
        .WIDTH(WIDTH)
    ) compare_max (
        .i_a   (i_a),
        .i_b   (~i_b),
        .i_cin (1'b1),
        .o_s   (s)
    );

    assign o_lt = (i_a[WIDTH-1] ^ i_b[WIDTH-1]) ? i_a[WIDTH-1] : s[WIDTH-1];
```

```

endmodule
```

Listing 1: HDL mô tả bộ so sánh bé hơn có dấu.

```
module max_min #(
    parameter DEPTH = 128,
    parameter WIDTH = 8
)(
    input  logic                              i_clk    ,
    input  logic                              i_rst_n  ,
    input  logic                              i_start  ,
    output logic                              o_rden   ,
    output logic        [$clog2(DEPTH) - 1:0] o_addr   ,
    input  logic signed [WIDTH         - 1:0] i_data   ,
    output logic                              o_done   ,
    output logic signed [WIDTH         - 1:0] o_max    ,
    output logic signed [WIDTH         - 1:0] o_min
);

    localparam DEPTH_S = DEPTH - 1;
    localparam WIDTH_S = WIDTH - 1;

    logic [WIDTH - 1:0] max, min;
    logic flag_max, flag_min;

    typedef enum logic[2:0] {
        IDLE    ,
        READ    ,
        COMPARE ,
        UPDATE  ,
        DONE    ,
        CLEAR
    } e_state;

    e_state pstate, nstate;
    logic [$clog2(DEPTH) - 1:0] reg_count, count, count_add;

    always_ff @(posedge i_clk, negedge i_rst_n) begin : blockName
        if(~i_rst_n) pstate <= IDLE  ;
        else         pstate <= nstate;
    end

    always_comb begin
        case(pstate)
            IDLE   : nstate = i_start ? UPDATE : pstate;
            UPDATE : nstate = (&reg_count) ? DONE : READ;
            READ   : nstate = COMPARE;
            COMPARE: nstate = UPDATE ;
            DONE   : nstate = i_start ? CLEAR : pstate;
            CLEAR  : nstate = UPDATE;
            default: nstate = IDLE;
        endcase
    end

    assign count  = (pstate == COMPARE) ? count_add : ((pstate == CLEAR) ? '0 : reg_count);
    assign o_addr = count;

    assign o_done = (pstate == DONE);
    //assign o_rden = (pstate == UPDATE) | (pstate == IDLE);
    assign o_rden = (pstate == UPDATE);

    always_ff @(posedge i_clk, negedge i_rst_n) begin
```

```
59          if(~i_rst_n) begin
60              reg_count <= '0;
61              o_max     <= {1'b1, {WIDTH_S{1'b0}}};
62              o_min     <= {1'b0, {WIDTH_S{1'b1}}};
63          end
64          else begin
65              reg_count <= count;
66              o_max     <= max  ;
67              o_min     <= min  ;
68          end
69      end
70
71      adder_flex_no_carry #(
72          .WIDTH($clog2(DEPTH))
73      ) add_1 (
74          .i_a   (reg_count),
75          .i_b   ({{DEPTH_S{1'b0}},1'b1}),
76          .i_cin (1'b0),
77          .o_s   (count_add)
78      );
79
80      assign max = (pstate == COMPARE) ? (flag_max ? i_data : o_max) : ((pstate == CLEAR) ? {1'b1, {WIDTH_S{1'
        b0}}} : o_max);
81      assign min = (pstate == COMPARE) ? (flag_min ? i_data : o_min) : ((pstate == CLEAR) ? {1'b0, {WIDTH_S{1'
        b1}}} : o_min);
82
83      lt_compare_sign #(
84          .WIDTH(WIDTH)
85      ) compare_max (
86          .i_a (o_max),
87          .i_b (i_data),
88          .o_lt(flag_max) // a < b
89      );
90
91      lt_compare_sign #(
92          .WIDTH(WIDTH)
93      ) compare_min (
94          .i_a (i_data),
95          .i_b (o_min),
96          .o_lt(flag_min) // a < b
97      );
98
99  endmodule
```

Listing 2: HDL mô tả thiết kế tìm số lớn nhất và nhỏ nhất.

Để tạo giá trị ngẫu nhiên trong bộ nhớ để mô phỏng, nhóm chọn sử dụng $\$unrandom\_range(,)$; mà System Verilog cung cấp.

```
1     localparam WIDTH_S = DW - 1;
2
3     golden_min = {1'b0, {WIDTH_S{1'b1}}};   // max  2^(n-1)
4     golden_max = {1'b1, {WIDTH_S{1'b0}}};   // min -2^(m-1)
5
6     for (i = 0; i < DEPTH; i++) begin
7        ram.mem[i] = $urandom_range(-2**(WIDTH_S), 2**(WIDTH_S) - 1);
8
9        if (ram.mem[i] < golden_min) golden_min = ram.mem[i];
10
11       if (ram.mem[i] > golden_max) golden_max = ram.mem[i];
12    end
13
```

```
14      $display("Golden Min = %0d, Golden Max = %0d",
15      golden_min, golden_max);
```

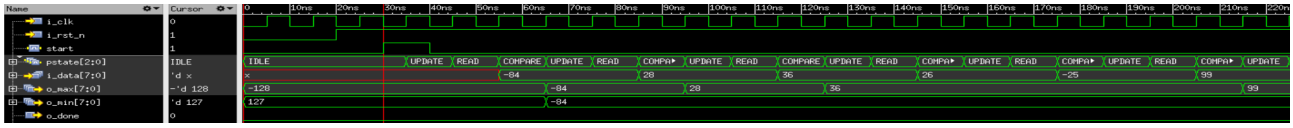
Listing 3: Chương trình tạo giá trị ngẫu nhiên ban đầu cho bộ nhớ.

```
1       always @(posedge clk) begin
2          if (done) begin
3             $display("DUT Min = %0d, DUT Max = %0d", dut_min, dut_max);
4
5             if (dut_min === golden_min && dut_max === golden_max)
6             $display(">>> TEST PASS <<<");
7             else begin
8                $display(">>> TEST FAIL <<<");
9                $display("Expected Min=%0d Max=%0d", golden_min, golden_max);
10            end
11         end
12      end
```

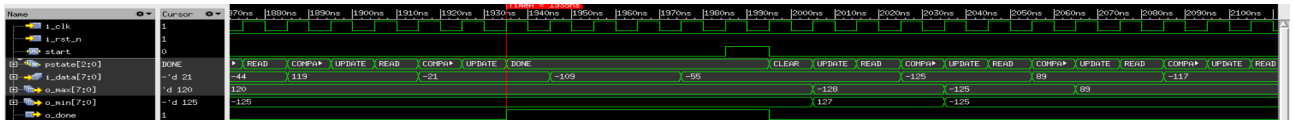
Listing 4: Chương trình kiểm định thiết kế.

```
xcelium> run
Golden Min = -125, Golden Max = 120
DUT Min = -125, DUT Max = 120
>>> TEST PASS <<<
Simulation complete via $finish(1) at time 995 NS + 0
../01_tb/max_min_tb.sv:106              $finish;
```

Listing 5: Kết quả kiểm định cho thiết kế bộ tìm số lớn nhất và nhỏ nhất.



Hình 7: Dạng sóng lúc bắt đầu.



Hình 8: Dạng sóng lúc hoàn thành.

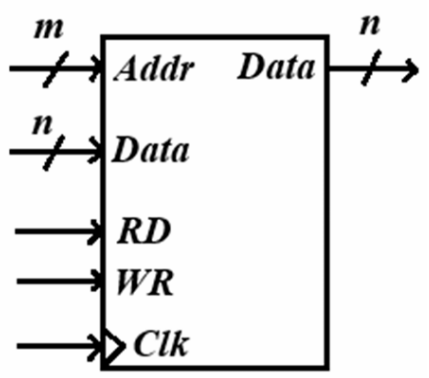
Kết luận: Máy trạng thái chuyển đúng so với dự tính, giá trị kết quả so sánh chính xác.

Câu 3

Cho đoạn code C sau dùng để sắp xếp một mảng n phần tử theo thứ tự tăng dần sử dụng giải thuật ***Selection Sort***:

```
int n = arr.size() - 1;
for(int i = 0; i < n; i++) {
    min = i;
    for(int j = i+1; j <= n; j++){
        if(arr[j] < arr[min]){
            min = j;
        }
    }
    swap(arr[i], arr[min]);
}

```

Listing 6: Đoạn chương trình C của giải thuật Selection Sort.

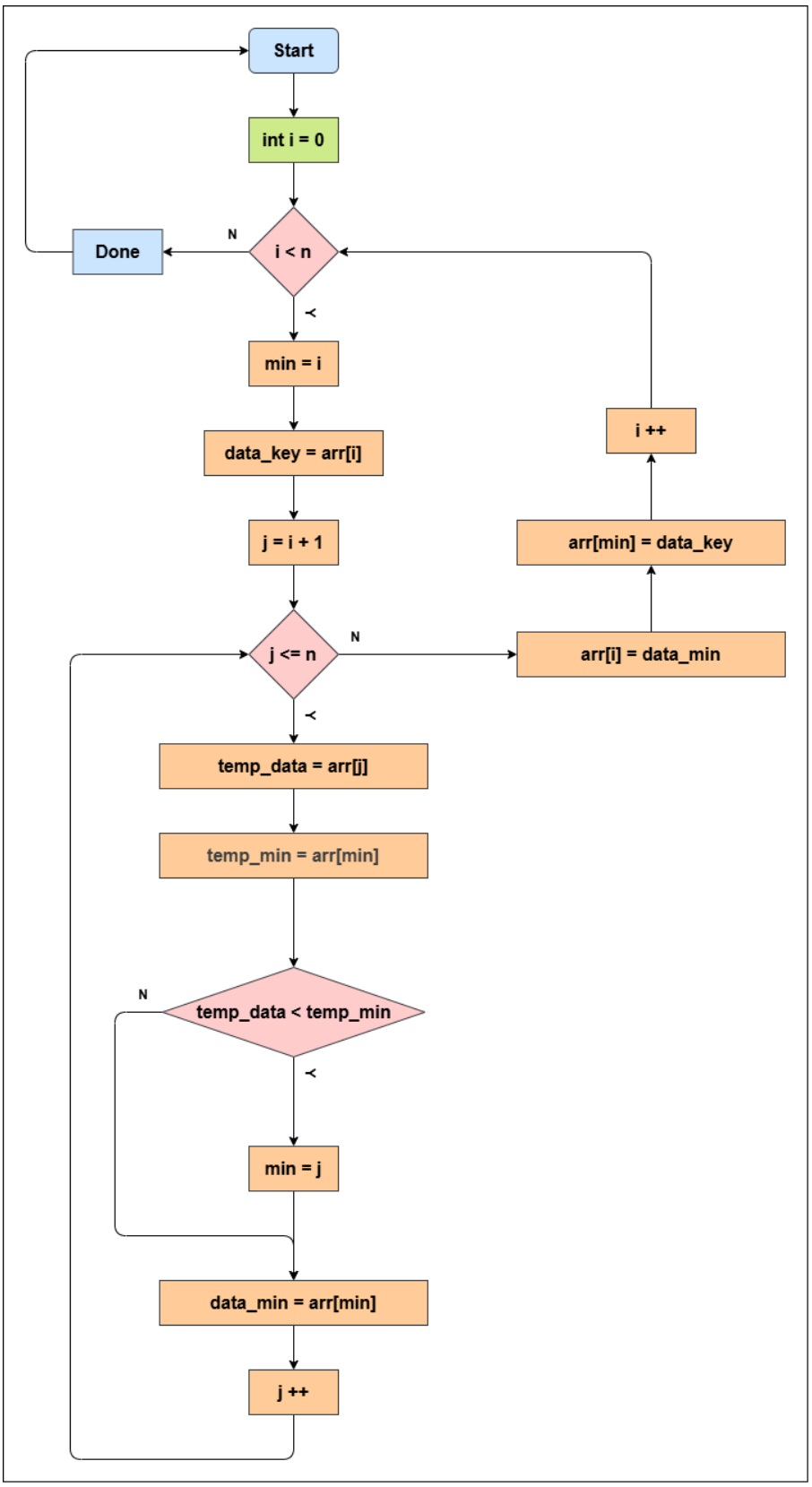


Hình 9: Yêu cầu của bộ nhớ.

Người ta muốn chuyển đổi giải thuật ở đoạn code 6 trên sang phần cứng để thực thi. Giả sử mảng được lưu trong bộ nhớ *Single Port* ở hình 9 và quá trình đọc/ghi diễn ra đồng bộ theo Clock và hoàn thành trong 1 Clock.

a) Định nghĩa ngõ vào ra của thiết kế, vẽ kết nối của thiết kế của bộ nhớ (Yêu cầu phải có chân Start và Reset).

Ta tiến hành biểu diễn đoạn code 6 trên dưới dạng flowchart như sau:



Hình 10: Flowchart của thuật toán Selection Sort.

Từ hình 10, cho ta thấy được các dữ liệu được đọc trước và các dữ liệu có sẵn để có thể dễ dàng trong việc chuyển đổi từ giải thuật phần mềm qua phần cứng. Thực hiện kiểm chứng cách hoạt động của giải thuật theo flowchart và giải thuật góc ở chương trình 6.

```
1   void selection_sort_standard(std::
    vector<int> &arr){
2       int n = arr.size() - 1;
3       for(int i = 0; i < n; i++){
4          int min = i;
5          for(int j = i+1; j <= n; j++){
6             if(arr[j] < arr[min]){
7                min = j;
8             }
9          }
10         std::swap(arr[i], arr[min]);
11      }
12   }
13
14
15
16
17
18
19
20
21
22
23
24
```

Listing 7: Đoạn code nguyên mẫu của giải thuật Selection Sort.

```
1   void selection_sort_cus(std::vector<
    int> &arr){
2       int n = arr.size() - 1;
3       int min          = 0;
4       int temp_data    = 0;
5       int temp_min     = 0;
6       int data_key     = 0;
7       int data_min     = 0;
8       for(int i = 0; i < n; i++){
9          data_key = arr[i];
10         min = i;
11         for(int j = i + 1; j <= n; j++)
    {
12            temp_data = arr[j];
13            temp_min  = arr[min];
14            if(temp_data < temp_min){
15               min = j;
16            }
17            data_min = arr[min];
18         }
19         arr[i] = data_min;
20         arr[min] = data_key;
21      }
22   }
23
```

Listing 8: Đoạn code chỉnh sửa của giải thuật Selection Sort.

Kết quả cho ra là:

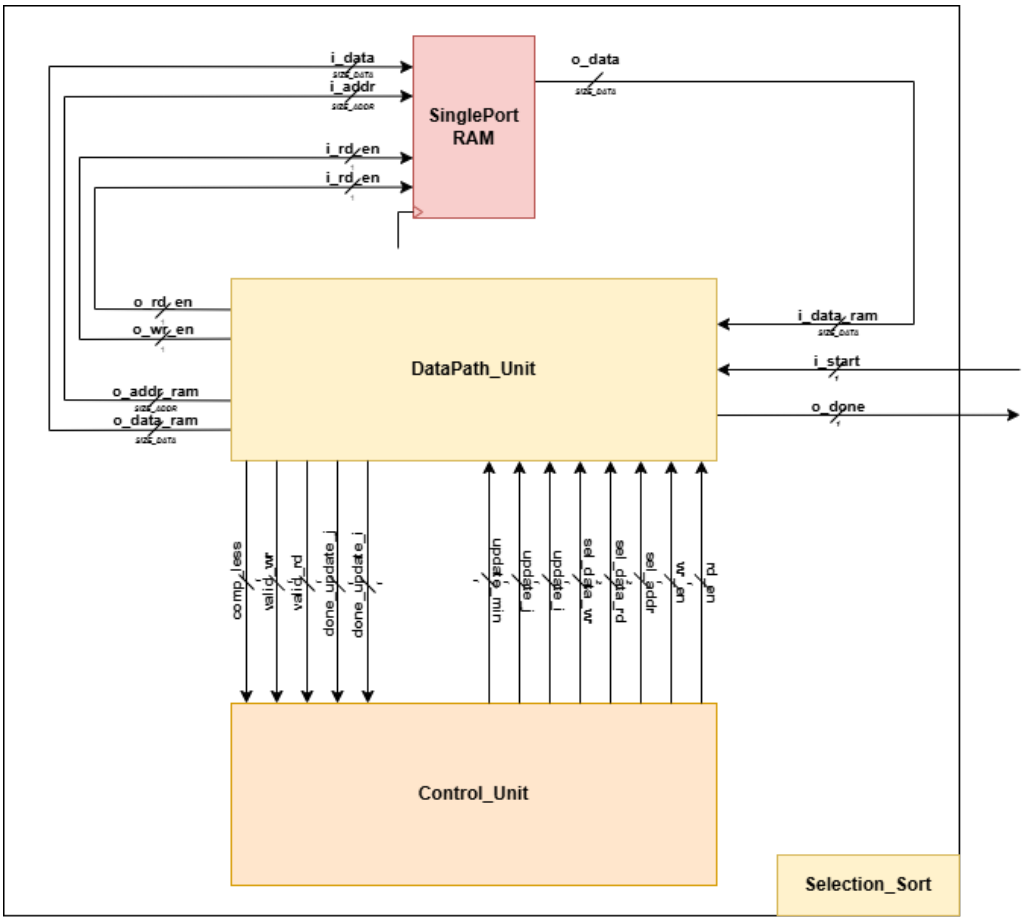
```
Finished reading file: ./tools/unsorted.txt
Check Selection_Sort Standard: PASS
Finish write file './Reports/COMPILE_REPORT/sorted_standard.txt'.
Check Selection_Sort Cus: PASS
Finish write file './Reports/COMPILE_REPORT/sorted_cus.txt'.
```

Listing 9: Kết quả so sánh 2 cách viết của Selection Sort.

Sau khi đã viết lại đoạn code cho dễ nhìn, tiếp đến sẽ định nghĩa lại ngõ vào về ra của module top chính ra `Selection_Sort` như sau:

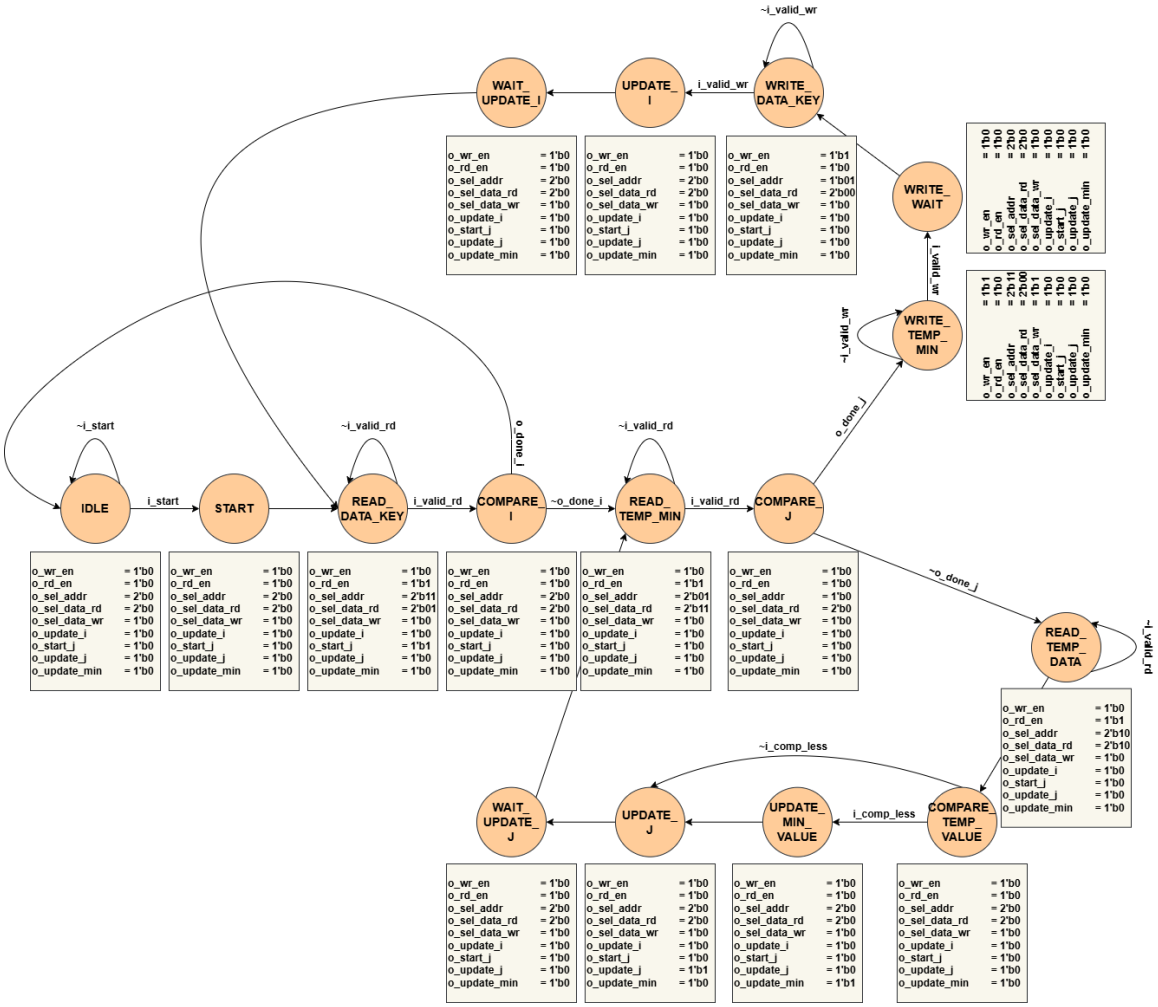
Signal	Size	Functional
i_clk	1	Clock của toàn hệ thống.
i_rst_t	1	Tín hiệu reset của hệ thống với tích cực thấp.
i_start	1	Tín hiệu bắt đầu hoạt động của bộ.
o_done	1	Tín hiệu cho biết được đã sắp xếp xong mảng.

Từ đó, ta có thiết kế tổng quan của module `Selection_Sort`:



Hình 11: Thiết kế tổng quan của module `Selection_Sort`.

b) Thiết kế máy trạng thái bậc cao của thiết kế.



Hình 12: Lưu đồ trạng thái của module `Selection_Sort`.

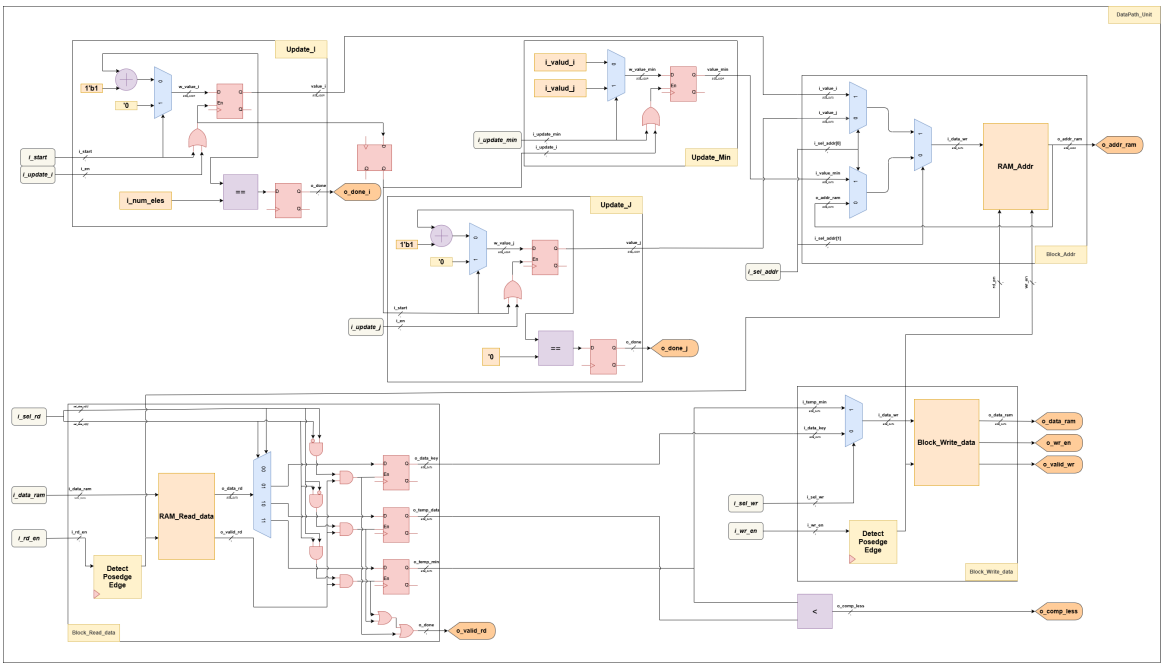
FSM gồm có 16 trạng thái như sau:

- IDLE: là trạng thái khởi tạo của module sau khi có reset, và chờ có tín hiệu `i_start` để module bắt đầu hoạt động.
- START: là trạng thái khởi tạo giá trị khởi tạo cho `value_i`.
- READ_DATA_KEY: là đợi đọc dữ liệu tại vị trí đang xét (theo giá trị `value_i`).
- COMPARE_I: là trạng thái kiểm tra vòng lặp (for) cho biến `value_i` xem đã kết thúc quá trình sắp xếp dữ liệu trong mảng.

- READ_TEMP_MIN: là trạng thái đọc dữ liệu `temp_min` có mục đích để kiểm tra ra giá trị nhỏ nhất từ vị trí `value_i` đến cuối mảng.
- COMPARE_J: là trạng thái kiểm tra vòng lặp (for) cho biến `value_j` xem đã kết thúc quá trình dò giá trị trong mảng.
- READ_TEMP_DATA: là trạng thái đọc dữ liệu theo giá trị của `value_j`.
- COMPARE_TEMP_VALUE: là kiểm tra thử xem có giá trị nào nhỏ hơn tại vị trí tại giá trị nhỏ nhất hiện tại.
- UPDATE_MIN_VALUE: là việc cập nhật vị trí cho vị trí min của dữ liệu.
- UPDATE_J: là trạng thái thực hiện việc tăng giá trị của `value_j`.
- WAIT_UPDATE_J: là trạng thái chờ giá trị `value_j` cập nhật.
- WRITE_TEMP_MIN: thực hiện ghi lại dữ liệu nhỏ nhất lại vị trí đang xét tại `value_i`.
- WRITE_WAIT: là trạng thái chờ dữ liệu ghi.
- WRITE_DATA_KEY: thực hiện ghi lại giá trị ở tại `value_i` vào thế chỗ của `value_min`.
- UPDATE_I: là trạng thái thực hiện việc tăng giá trị của `value_i`.
- WAIT_UPDATE_I: là trạng thái chờ cập nhật cho giá trị `value_i`.

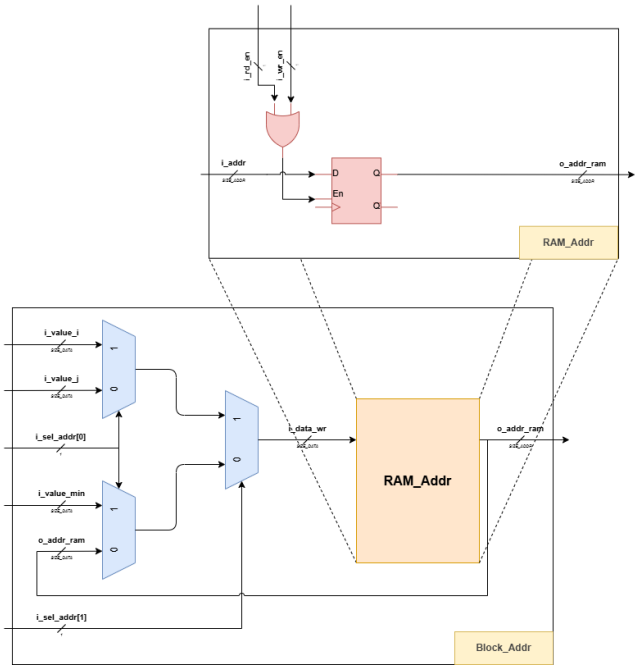
c) Thiết kế Datapath và Control Unit của thiết kế.

1. Thiết kế DataPath:

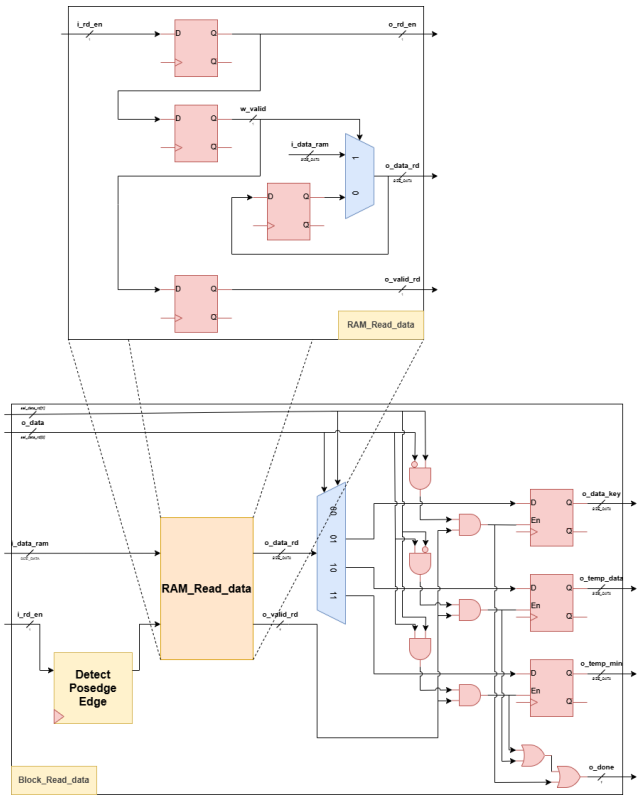


Hình 13: Thiết kế tổng quan của module `DataPath_Unit`.

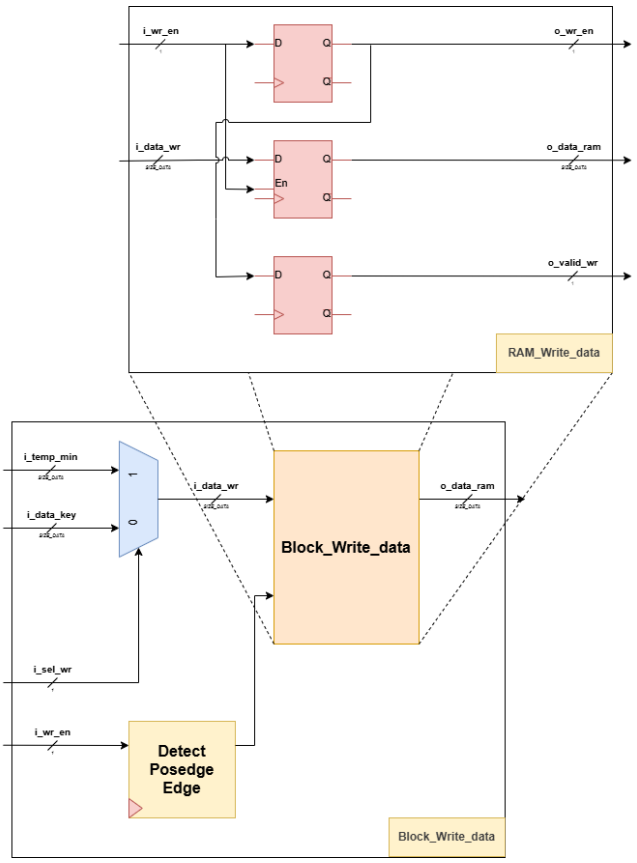
- RAM_Addr:



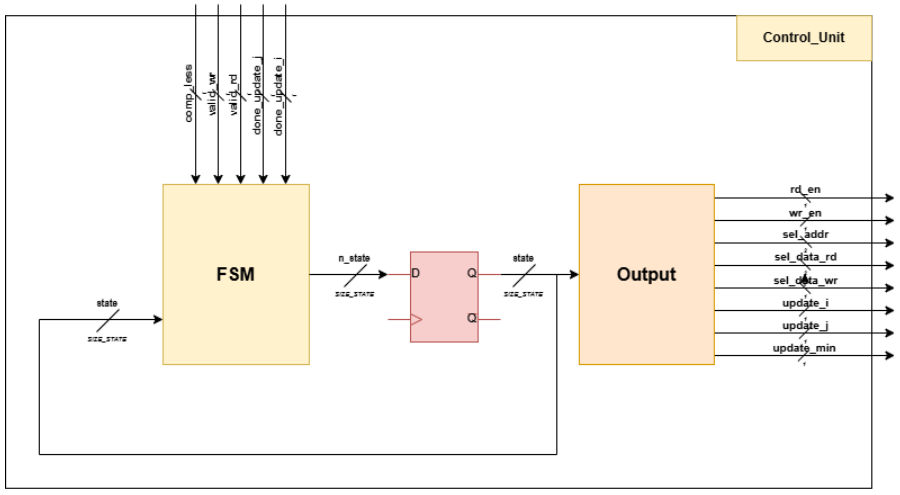
- RAM_Read_data:



- RAM_Write_data:



2. Thiết kế của Control_Unit:



d) Viết chương trình mô phỏng hoạt động của thiết kế.

1. Thiết kế tổng quan:

```
module Selection_Sort #(
    parameter SIZE_ADDR     = 8 ,
    parameter PATH_RAM  = "./../03_verif/Topmodule/tools/random_hex.txt",
    parameter SIZE_DATA     = 8
)(
    input logic                     i_clk       ,
    input logic                     i_rst_n     ,
    input logic                     i_start     ,
    input logic [SIZE_ADDR-1:0]     i_num_elems ,
    output logic                    o_done
);

logic w_rd_en;
logic w_wr_en;
logic [1:0] w_sel_addr;
logic [1:0] w_sel_data_rd;
logic       w_sel_data_wr;
logic w_update_i;
logic w_update_j;
logic w_update_min;

logic w_comp_less;
logic w_valid_wr;
logic w_valid_rd;
logic w_done_update_j;
logic w_done_update_i;

logic w_rd_en_ram;
logic w_wr_en_ram;
logic [SIZE_ADDR-1:0] w_addr_ram;
logic [SIZE_DATA-1:0] w_i_data_ram;
logic [SIZE_DATA-1:0] w_o_data_ram;

Data_path #(
    .SIZE_ADDR          (SIZE_ADDR),
    .SIZE_DATA          (SIZE_DATA)
) DATA_PATH_UNIT (
    .i_clk              (i_clk),
    .i_rst_n            (i_rst_n),
    .i_num_elems        (i_num_elems),
    .i_rd_en            (w_rd_en),
    .i_wr_en            (w_wr_en),
    .i_sel_addr         (w_sel_addr),
    .i_sel_data_rd      (w_sel_data_rd),
    .i_sel_data_wr      (w_sel_data_wr),
    .i_start_i          (i_start), //i_state=>topmodule
    .i_start_j          (),
    .i_update_i         (w_update_i),
    .i_update_j         (w_update_j),
    .i_update_min       (w_update_min),
    .i_data_ram         (w_o_data_ram),
    .o_wr_en            (w_wr_en_ram),
    .o_rd_en            (w_rd_en_ram),
    .o_addr_ram         (w_addr_ram),
    .o_data_ram         (w_i_data_ram),
    .o_comp_less        (w_comp_less),
    .o_valid_wr         (w_valid_wr),
    .o_valid_rd         (w_valid_rd),
    .o_done_j           (w_done_update_j),
    .o_done_sort        (w_done_update_i)
);

Control_unit CONTROL_UNIT (
```

```
64      .i_clk              (i_clk),
65      .i_rst_n            (i_rst_n),
66      .i_start            (i_start),
67      .i_comp_less        (w_comp_less),
68      .i_valid_wr         (w_valid_wr),
69      .i_valid_rd         (w_valid_rd),
70      .i_done_j           (w_done_update_j),
71      .i_done_sort        (w_done_update_i),
72      .o_wr_en            (w_wr_en),
73      .o_rd_en            (w_rd_en),
74      .o_sel_addr         (w_sel_addr),
75      .o_sel_data_rd      (w_sel_data_rd),
76      .o_sel_data_wr      (w_sel_data_wr),
77      .o_update_i         (w_update_i),
78      .o_start_j          (),
79      .o_update_j         (w_update_j),
80      .o_update_min       (w_update_min)
81  );
82
83  SinglePort_RAM #(
84      .SIZE_ADDR          (SIZE_ADDR),
85      .PATH_RAM           (PATH_RAM),
86      .SIZE_DATA          (SIZE_DATA)
87  ) RAM_UNIT (
88      .i_clk              (i_clk),
89      .i_rst_n            (i_rst_n),
90      .i_rd_en            (w_rd_en_ram),
91      .i_wr_en            (w_wr_en_ram),
92      .i_addr             (w_addr_ram),
93      .i_data             (w_i_data_ram),
94      .o_data             (w_o_data_ram)
95  );
96
97  always_ff @( posedge i_clk or negedge i_rst_n ) begin
98      if(~i_rst_n)
99          o_done  <= '0;
100     else
101         o_done  <= w_done_update_i;
102 end
103
104 endmodule
```

Listing 10: Module `Selection_Sort`.

```
1   module SinglePort_RAM #(
2       parameter SIZE_ADDR = 8 ,
3       parameter PATH_RAM  = "./../03_verif/Topmodule/tools/random_hex.txt",
4       parameter SIZE_DATA = 8
5   )(
6       input logic                         i_clk       ,
7       input logic                         i_rst_n     ,
8       input logic                         i_rd_en     ,
9       input logic                         i_wr_en     ,
10      input logic [SIZE_ADDR-1:0]         i_addr      ,
11      input logic [SIZE_DATA-1:0]         i_data      ,
12      output logic [SIZE_DATA-1:0]        o_data
13  );
14
15  localparam DEPTH     = 1 << SIZE_ADDR;   // 2^SIZE_ADDR
16  logic [SIZE_DATA-1:0] mem_unit [0:DEPTH-1];
17  initial begin
18      $readmemh(PATH_RAM, mem_unit);
19  end
```

```
20
21  always_ff @( posedge i_clk) begin : proc_write_data
22      if(i_wr_en) begin
23          mem_unit[i_addr]     <= i_data;
24      end
25  end
26  always_ff @( posedge i_clk or negedge i_rst_n) begin : proc_read_data
27      if(~i_rst_n) begin
28          o_data               <= '0;
29      end else if(i_rd_en) begin
30          o_data               <= mem_unit[i_addr];
31      end
32  end
33
34  endmodule
```

Listing 11: Module `SinglePort_RAM`.

```
1   module Control_unit (
2       input logic                     i_clk       ,
3       input logic                     i_rst_n     ,
4       input logic                     i_start     ,
5       input logic                     i_comp_less ,
6       input logic                     i_valid_wr  ,
7       input logic                     i_valid_rd  ,
8       input logic                     i_done_j    ,
9       input logic                     i_done_sort ,
10      output logic                    o_wr_en     ,
11      output logic                    o_rd_en     ,
12      output logic [1:0]              o_sel_addr  ,
13      output logic [1:0]              o_sel_data_rd,
14      output logic                    o_sel_data_wr,
15      output logic                    o_update_i  ,
16      output logic                    o_start_j,
17      output logic                    o_update_j  ,
18      output logic                    o_update_min
19  );
20
21  typedef enum logic [3:0] {
22      IDLE             = 4'd0,
23      START            = 4'd1,
24      READ_DATA_KEY    = 4'd2,
25      COMPARE_I        = 4'd3,
26      READ_TEMP_MIN    = 4'd4,
27      COMPARE_J        = 4'd5,
28      READ_TEMP_DATA   = 4'd6,
29      COMP_TEMP_VALUE  = 4'd7,
30      UPDATE_MIN_VALUE = 4'd8,
31      UPDATE_J         = 4'd9,
32      WAIT_J           = 4'd10,
33      WRITE_TEMP_MIN   = 4'd11,
34      WAIT_WRITE       = 4'd12,
35      WRITE_DATA_KEY   = 4'd13,
36      UPDATE_I         = 4'd14,
37      WAIT_I           = 4'd15
38  } state_e;
39   state_e state, n_state;
40
41  always_comb begin : proc_next_state
42      case (state)
43          IDLE: begin
44              n_state = i_start ? START : IDLE;
45          end
```

```
46          START: begin
47              n_state = READ_DATA_KEY;
48          end
49          READ_DATA_KEY: begin
50              n_state = i_valid_rd ? COMPARE_I : READ_DATA_KEY;
51          end
52          COMPARE_I: begin
53              n_state = i_done_sort ? IDLE : READ_TEMP_MIN;
54          end
55          READ_TEMP_MIN: begin
56              n_state = i_valid_rd ? COMPARE_J : READ_TEMP_MIN;
57          end
58          COMPARE_J: begin
59              n_state = i_done_j ? WRITE_TEMP_MIN : READ_TEMP_DATA;
60          end
61          READ_TEMP_DATA: begin
62              n_state = i_valid_rd ? COMP_TEMP_VALUE : READ_TEMP_DATA;
63          end
64          COMP_TEMP_VALUE: begin
65              n_state = i_comp_less ? UPDATE_MIN_VALUE : UPDATE_J;
66          end
67          UPDATE_MIN_VALUE: begin
68              n_state = UPDATE_J;
69          end
70          UPDATE_J: begin
71              n_state = WAIT_J;
72          end
73          WAIT_J: begin
74              n_state = READ_TEMP_MIN;
75          end
76          WRITE_TEMP_MIN: begin
77              n_state = i_valid_wr ? WAIT_WRITE : WRITE_TEMP_MIN;
78          end
79          WAIT_WRITE: begin
80              n_state = WRITE_DATA_KEY;
81          end
82          WRITE_DATA_KEY: begin
83              n_state = i_valid_wr ? UPDATE_I : WRITE_DATA_KEY;
84          end
85          UPDATE_I: begin
86              n_state = WAIT_I;
87          end
88          WAIT_I: begin
89              n_state = READ_DATA_KEY;
90          end
91          default: begin
92              n_state = IDLE;
93          end
94      endcase
95  end
96  always_ff @( posedge i_clk or negedge i_rst_n ) begin : proc_update_state
97      if(~i_rst_n)
98          state   <= IDLE;
99      else
100         state   <= n_state;
101 end
102
103 always_comb begin : proc_output
104     // default values
105     o_wr_en         = '0;
106     o_rd_en         = '0;
107     o_sel_addr      = '0;
108     o_sel_data_rd   = '0;
109     o_sel_data_wr   = '0;
```

```
110         o_update_i      = '0;
111         o_start_j       = '0;
112         o_update_j      = '0;
113         o_update_min    = '0;
114
115         case (state)
116             READ_DATA_KEY: begin
117                 o_start_j     = 1'b1;
118                 o_rd_en       = 1'b1;
119                 o_sel_addr    = 2'b11;
120                 o_sel_data_rd = 2'b01;
121             end
122
123             COMPARE_I: begin
124                 o_sel_addr    = 2'b00;
125                 o_sel_data_rd = 2'b00;
126             end
127
128             READ_TEMP_MIN: begin
129                 o_rd_en       = 1'b1;
130                 o_sel_addr    = 2'b01;
131                 o_sel_data_rd = 2'b11;
132             end
133
134             COMPARE_J: begin
135                 o_sel_addr    = 2'b00;
136                 o_sel_data_rd = 2'b00;
137             end
138
139             READ_TEMP_DATA: begin
140                 o_rd_en       = 1'b1;
141                 o_sel_addr    = 2'b10;
142                 o_sel_data_rd = 2'b10;
143             end
144
145             COMP_TEMP_VALUE: begin
146                 o_sel_addr    = 2'b00;
147                 o_sel_data_rd = 2'b00;
148             end
149
150             UPDATE_MIN_VALUE: begin
151                 o_sel_addr    = 2'b00;
152                 o_sel_data_rd = 2'b00;
153                 o_update_min  = 1'b1;
154             end
155
156             UPDATE_J: begin
157                 o_sel_addr    = 2'b00;
158                 o_sel_data_rd = 2'b00;
159                 o_update_j    = 1'b1;
160             end
161
162             WAIT_J: begin
163                 o_sel_addr    = 2'b00;
164                 o_sel_data_rd = 2'b00;
165             end
166
167             WRITE_TEMP_MIN: begin
168                 o_wr_en       = 1'b1;
169                 o_sel_addr    = 2'b11;
170                 o_sel_data_rd = 2'b00;
171                 o_sel_data_wr = 1'b1;
172             end
173
```

```
174         UPDATE_I: begin
175             o_sel_addr    = 2'b00;
176             o_sel_data_rd = 2'b00;
177             o_update_i    = 1'b1;
178         end
179
180         WRITE_DATA_KEY: begin
181             o_wr_en       = 1'b1;
182             o_sel_addr    = 2'b01;
183             o_sel_data_rd = 2'b00;
184             o_sel_data_wr = 1'b0;
185         end
186
187         default: begin
188             o_wr_en         = '0;
189             o_rd_en         = '0;
190             o_sel_addr      = '0;
191             o_sel_data_rd   = '0;
192             o_sel_data_wr   = '0;
193             o_update_i      = '0;
194             o_start_j       = '0;
195             o_update_j      = '0;
196             o_update_min    = '0;
197         end
198     endcase
199 end
200
201
202
203 endmodule
```

Listing 12: Module `Control_unit`.

```
1  module Data_path #(
2      parameter SIZE_ADDR = 8,
3      parameter SIZE_DATA = 8
4  )(
5      input logic                     i_clk           ,
6      input logic                     i_rst_n         ,
7      input logic [SIZE_ADDR-1:0]     i_num_elems     ,
8
9      input logic                     i_rd_en         ,
10     input logic                     i_wr_en         ,
11     input logic [1:0]               i_sel_addr      ,
12     input logic [1:0]               i_sel_data_rd   ,
13     input logic                     i_sel_data_wr   ,
14     input logic                     i_start_i       , //i_state=>topmodule
15     input logic                     i_start_j       ,
16     input logic                     i_update_i      ,
17     input logic                     i_update_j      ,
18     input logic                     i_update_min    ,
19
20     input logic [SIZE_DATA-1:0]     i_data_ram      ,
21     output logic                    o_wr_en         ,
22     output logic                    o_rd_en         ,
23     output logic [SIZE_ADDR-1:0]    o_addr_ram      ,
24     output logic [SIZE_DATA-1:0]    o_data_ram      ,
25
26     output logic                    o_comp_less     ,
27     output logic                    o_valid_wr      ,
28     output logic                    o_valid_rd      ,
29     output logic                    o_done_j        ,
30     output logic                    o_done_sort
```

```
);

logic [SIZE_ADDR-1:0] value_i;
logic                 w_o_en_i;
logic [SIZE_ADDR-1:0] value_j;
logic                 w_i_en_j;
logic [SIZE_ADDR-1:0] value_min;

logic                 w_en_temp_min;
logic [SIZE_DATA-1:0] w_temp_min;
logic                 w_en_temp_data;
logic [SIZE_DATA-1:0] w_temp_data;
logic                 w_en_data_key;
logic [SIZE_DATA-1:0] w_data_key;

logic [SIZE_ADDR-1:0] w_addr_ram;
logic [SIZE_DATA-1:0] w_data_wr;
logic [SIZE_DATA-1:0] w_data_rd;

assign o_comp_less = w_temp_data < w_temp_min;

Update_I #(
    .SIZE_ADDR      (SIZE_ADDR)
) UPDATE_I_UNIT (
    .i_clk          (i_clk),
    .i_rst_n        (i_rst_n),
    .i_num_elems    (i_num_elems),
    .i_start        (i_start_i),
    .i_en           (i_update_i),
    .o_en           (w_o_en_i),
    .o_value_i      (value_i),
    .o_done         (o_done_sort)
);

always_ff @( posedge i_clk or negedge i_rst_n) begin
    if(~i_rst_n)
        w_i_en_j    <= '0;
    else
        w_i_en_j    <= w_o_en_i;
end

Update_J #(
    .SIZE_ADDR      (SIZE_ADDR)
) UPDATE_J_UNIT (
    .i_clk          (i_clk),
    .i_rst_n        (i_rst_n),
    .i_num_elems    (i_num_elems),
    .i_value_i      (value_i),
    .i_start        (w_i_en_j),
    .i_en           (i_update_j),
    .o_en           (),
    .o_value_j      (value_j),
    .o_done         (o_done_j)
);

Update_MIN #(
    .SIZE_ADDR      (SIZE_ADDR)
) UPDATE_MIN_UNIT (
    .i_clk          (i_clk),
    .i_rst_n        (i_rst_n),
    .i_value_i      (value_i),
    .i_value_j      (value_j),
    .i_update_i     (w_i_en_j),
    .i_update_min   (i_update_min),
```

```
95      .o_addr_min     (value_min)
96  );
97
98  Block_Addr #(
99      .SIZE_ADDR      (SIZE_ADDR)
100 ) ADDR_RAM (
101     .i_clk          (i_clk),
102     .i_rst_n        (i_rst_n),
103     .i_rd_en        (i_rd_en),
104     .i_wr_en        (i_wr_en),
105     .i_sel_addr     (i_sel_addr),
106     .i_value_i      (value_i),
107     .i_value_j      (value_j),
108     .i_value_min    (value_min),
109     .o_addr_ram     (o_addr_ram)
110 );
111 Block_Read_data #(
112     .SIZE_DATA      (SIZE_DATA)
113 ) READ_RAM (
114     .i_clk          (i_clk),
115     .i_rst_n        (i_rst_n),
116     .i_rd_en        (i_rd_en),
117     .i_sel_data_rd  (i_sel_data_rd),
118     .i_data_ram     (i_data_ram),
119     .o_rd_en        (o_rd_en),
120     .o_temp_min     (w_temp_min),
121     .o_temp_data    (w_temp_data),
122     .o_data_key     (w_data_key),
123     .o_done         (o_valid_rd)
124 );
125 Block_Write_data #(
126     .SIZE_DATA      (SIZE_DATA)
127 ) WRITE_DATA (
128     .i_clk          (i_clk),
129     .i_rst_n        (i_rst_n),
130     .i_wr_en        (i_wr_en),
131     .i_sel_wr       (i_sel_data_wr),
132     .i_data_key     (w_data_key),
133     .i_temp_min     (w_temp_min),
134     .o_wr_en        (o_wr_en),
135     .o_data_ram     (o_data_ram),
136     .o_done         (o_valid_wr)
137 );
138
139 endmodule
```

Listing 13: Module `Data_path`.

```
1   module Block_Addr #(
2       parameter SIZE_ADDR = 8
3   )(
4       input logic                     i_clk       ,
5       input logic                     i_rst_n     ,
6       input logic                     i_rd_en     ,
7       input logic                     i_wr_en     ,
8       input logic [1:0]               i_sel_addr  ,
9
10      input logic [SIZE_ADDR-1:0]     i_value_i   ,
11      input logic [SIZE_ADDR-1:0]     i_value_j   ,
12      input logic [SIZE_ADDR-1:0]     i_value_min ,
13      output logic [SIZE_ADDR-1:0]    o_addr_ram
14  );
15
```

```
16  logic [SIZE_ADDR-1:0] w_addr_ram;
17
18  assign w_addr_ram = i_sel_addr[1] ? (i_sel_addr[0] ? i_value_i : i_value_j) : (i_sel_addr[0] ?
        i_value_min : o_addr_ram);
19  RAM_addr #(
20      .SIZE_ADDR      (SIZE_ADDR)
21  ) RAM_ADDR_UNIT (
22      .i_clk          (i_clk),
23      .i_rst_n        (i_rst_n),
24      .i_rd_en        (i_rd_en),
25      .i_wr_en        (i_wr_en),
26      .i_addr_ram     (w_addr_ram),
27      .o_addr_ram     (o_addr_ram)
28  );
29
30  endmodule
```

Listing 14: Module `Block_Addr`.

```
1   module Block_Read_data #(
2       parameter SIZE_DATA = 8
3   )(
4       input logic                     i_clk           ,
5       input logic                     i_rst_n         ,
6       input logic                     i_rd_en         ,
7       input logic [1:0]               i_sel_data_rd   ,
8       input logic [SIZE_DATA-1:0]     i_data_ram      ,
9
10      output logic                    o_rd_en         ,
11      output logic [SIZE_DATA-1:0]    o_temp_min      ,
12      output logic [SIZE_DATA-1:0]    o_temp_data     ,
13      output logic [SIZE_DATA-1:0]    o_data_key      ,
14      output logic                    o_done
15  );
16
17  logic [SIZE_DATA-1:0] w_data_ram;
18  logic w_valid_rd;
19
20  logic w_rst_sel_data_rd;
21  logic [1:0] w_sel_data_rd;
22
23  logic w_en_temp_min;
24  logic w_en_temp_data;
25  logic w_en_data_key;
26
27  logic w_i_rd_en;
28  SS_detect_edge #(
29      .POS_EDGE   (1)   // 1: posedge, 0: negedge
30  ) DETECT_EDGE (
31      .i_clk      (i_clk),
32      .i_rst_n    (i_rst_n),
33      .i_signal   (i_rd_en),
34      .o_signal   (w_i_rd_en)
35  );
36
37  assign w_rst_sel_data_rd = (i_rst_n & ~o_done);
38  always_ff @( posedge i_clk or negedge w_rst_sel_data_rd ) begin
39      if(~w_rst_sel_data_rd) begin
40          w_sel_data_rd   <= '0;
41      end else if(w_i_rd_en) begin
42          w_sel_data_rd   <= i_sel_data_rd;
43      end
44  end
```

```
45
46  RAM_read_data #(
47      .SIZE_DATA      (SIZE_DATA)
48  ) RAM_READ_DATAA (
49      .i_clk          (i_clk),
50      .i_rst_n        (i_rst_n),
51      .i_rd_en        (w_i_rd_en),
52      .i_data_rd      (i_data_ram),
53      .o_rd_en        (o_rd_en),
54      .o_data_rd      (w_data_ram),
55      .o_valid        (w_valid_rd)
56  );
57
58  assign w_en_temp_min    = (w_sel_data_rd[1] & w_sel_data_rd[0]) & w_valid_rd;
59  assign w_en_temp_data   = (w_sel_data_rd[1] & ~w_sel_data_rd[0]) & w_valid_rd;
60  assign w_en_data_key    = (~w_sel_data_rd[1] & w_sel_data_rd[0]) & w_valid_rd;
61
62  always_ff @( posedge i_clk or negedge i_rst_n ) begin
63      if(~i_rst_n) begin
64          o_temp_min      <= '0;
65      end else if(w_en_temp_min) begin
66          o_temp_min      <= w_data_ram;
67      end
68  end
69  always_ff @( posedge i_clk or negedge i_rst_n ) begin
70      if(~i_rst_n) begin
71          o_temp_data      <= '0;
72      end else if(w_en_temp_data) begin
73          o_temp_data      <= w_data_ram;
74      end
75  end
76  always_ff @( posedge i_clk or negedge i_rst_n ) begin
77      if(~i_rst_n) begin
78          o_data_key      <= '0;
79      end else if(w_en_data_key) begin
80          o_data_key      <= w_data_ram;
81      end
82  end
83  always_ff @( posedge i_clk or negedge i_rst_n ) begin
84      if(~i_rst_n) begin
85          o_done      <= '0;
86      end else begin
87          o_done      <= (w_en_temp_min | w_en_temp_data | w_en_data_key);
88      end
89  end
90
91  endmodule
```

Listing 15: Module `Block_Read_data`.

```
1   module Block_Write_data #(
2       parameter SIZE_DATA = 8
3   )(
4       input logic                     i_clk       ,
5       input logic                     i_rst_n     ,
6       input logic                     i_wr_en     ,
7       input logic                     i_sel_wr    ,
8
9       input logic [SIZE_DATA-1:0]     i_data_key  ,
10      input logic [SIZE_DATA-1:0]     i_temp_min  ,
11
12      output logic                    o_wr_en     ,
13      output logic [SIZE_DATA-1:0]    o_data_ram  ,
```

```
    output logic                    o_done
);

logic w_i_wr_en;
logic [SIZE_DATA-1:0] w_i_data_ram;
SS_detect_edge #(
    .POS_EDGE   (1)   // 1: posedge, 0: negedge
) DETECT_EDGE (
    .i_clk      (i_clk),
    .i_rst_n    (i_rst_n),
    .i_signal   (i_wr_en),
    .o_signal   (w_i_wr_en)
);

assign w_i_data_ram = i_sel_wr ? i_temp_min : i_data_key;

RAM_write_data #(
    .SIZE_DATA  (SIZE_DATA)
) RAM_WRITE_DATA (
    .i_clk      (i_clk),
    .i_rst_n    (i_rst_n),
    .i_wr_en    (w_i_wr_en),
    .i_data_wr  (w_i_data_ram),
    .o_wr_en    (o_wr_en),
    .o_data_wr  (o_data_ram),
    .o_done     (o_done)
);

endmodule
```

Listing 16: Module `Block_Write_data`.

```
module RAM_addr #(
    parameter SIZE_ADDR = 8
)(
    input logic                     i_clk       ,
    input logic                     i_rst_n     ,
    input logic                     i_rd_en     ,
    input logic                     i_wr_en     ,
    input logic [SIZE_ADDR-1:0]     i_addr_ram  ,
    output logic [SIZE_ADDR-1:0]    o_addr_ram
);
logic w_en;
assign w_en = i_rd_en | i_wr_en;
always_ff @( posedge i_clk or negedge i_rst_n ) begin
    if(~i_rst_n) begin
        o_addr_ram      <= '0;
    end else if(w_en) begin
        o_addr_ram          <= i_addr_ram;
    end
end

endmodule
```

Listing 17: Module `RAM_addr`.

```
module RAM_read_data #(
    parameter SIZE_DATA = 8
)(
    input logic                     i_clk       ,
    input logic                     i_rst_n     ,
    input logic                     i_rd_en     ,
```

```
    input logic [SIZE_DATA-1:0]     i_data_rd   ,
    output logic                    o_rd_en     ,
    output logic [SIZE_DATA-1:0]    o_data_rd   ,
    output logic                    o_valid
);

logic w_valid;
always_ff @( posedge i_clk or negedge i_rst_n ) begin
    if(~i_rst_n) begin
        o_rd_en     <= '0;
    end else begin
        o_rd_en     <= i_rd_en;
    end
end
always_ff @( posedge i_clk or negedge i_rst_n ) begin
    if(~i_rst_n) begin
        w_valid     <= '0;
        o_valid     <= '0;
    end else begin
        w_valid     <= o_rd_en;
        o_valid     <= w_valid;
    end
end
logic [SIZE_DATA-1:0] w_save_data;
always_ff @( posedge i_clk or negedge i_rst_n ) begin
    if(~i_rst_n) begin
        w_save_data   <= '0;
    end else begin
        w_save_data   <= o_data_rd;
    end
end
assign o_data_rd = (w_valid) ? i_data_rd : w_save_data;
endmodule
```

Listing 18: Module `RAM_read_data`.

```
module RAM_write_data #(
    parameter SIZE_DATA = 8
)(
    input logic                     i_clk       ,
    input logic                     i_rst_n     ,
    input logic                     i_wr_en     ,
    input logic [SIZE_DATA-1:0]     i_data_wr   ,
    output logic                    o_wr_en     ,
    output logic [SIZE_DATA-1:0]    o_data_wr   ,
    output logic                    o_done
);

always_ff @( posedge i_clk or negedge i_rst_n ) begin
    if(~i_rst_n) begin
        o_wr_en     <= '0;
    end else begin
        o_wr_en     <= i_wr_en;
    end
end
always_ff @( posedge i_clk or negedge i_rst_n ) begin
    if(~i_rst_n) begin
        o_done      <= '0;
    end else begin
        o_done      <= o_wr_en;
    end
end
always_ff @( posedge i_clk or negedge i_rst_n ) begin
```

```
    if(~i_rst_n) begin
        o_data_wr   <= '0;
    end else if (i_wr_en) begin
        o_data_wr   <= i_data_wr;
    end
end
endmodule
```

Listing 19: Module `RAM_write_data`.

```
// POS_EDGE = 0 -> detect negedge edge
// POS_EDGE = 1 -> detect posedge edge
module SS_detect_edge #(
    parameter POS_EDGE = 1   // 1: posedge, 0: negedge
)(
    input  logic i_clk,
    input  logic i_rst_n,
    input  logic i_signal,
    output logic o_signal
);

logic w_p_signal, w_n_signal;

always_ff @(posedge i_clk or negedge i_rst_n) begin
    if (~i_rst_n)
        w_p_signal <= 1'b0;
    else
        w_p_signal <= i_signal;
end

always_ff @(posedge i_clk or negedge i_rst_n) begin
    if (~i_rst_n)
        w_n_signal <= 1'b0;
    else
        w_n_signal <= w_p_signal;
end

generate
    if (POS_EDGE) begin
        assign o_signal = (~w_n_signal) & (w_p_signal);
    end else begin
        assign o_signal = (w_n_signal) & (~w_p_signal);
    end
endgenerate

endmodule
```

Listing 20: Module `SS_detect_edge`.

```
module Update_I #(
    parameter SIZE_ADDR = 8
)(
    input logic                     i_clk       ,
    input logic                     i_rst_n     ,
    input logic [SIZE_ADDR-1:0]     i_num_elems ,

    input logic                     i_start     ,
    input logic                     i_en        ,

    output logic                    o_en        ,
    output logic [SIZE_ADDR-1:0]    o_value_i   ,
    output logic                    o_done
```

```
);

logic w_start;
logic [SIZE_ADDR-1:0]   w_pre_value_i;
logic [SIZE_ADDR-1:0]   w_next_value_i;
logic w_pre_done;
logic w_enable;

assign w_start          = i_start;
assign w_enable         = w_start | (i_en & ~w_pre_done);
assign w_next_value_i   = o_value_i + 1'b1;
assign w_pre_value_i    = w_start ? ('0) : (w_next_value_i);

always_ff @( posedge i_clk or negedge i_rst_n ) begin : proc_output_value_i
    if(~i_rst_n) begin
        o_value_i       <= '0;
    end else if(w_enable) begin
        o_value_i       <= w_pre_value_i;
    end
end

// assign w_pre_done       = (w_pre_value_i == (i_num_elems));
assign w_pre_done       = (o_value_i == (i_num_elems));
always_ff @( posedge i_clk or negedge i_rst_n ) begin : proc_done
    if(~i_rst_n) begin
        o_done          <= '0;
    end else begin
        o_done          <= w_pre_done;
    end
end
// assign o_done          = w_pre_done;

assign o_en = w_enable;

endmodule
```

Listing 21: Module `Update_I`.

```
module Update_J #(
    parameter SIZE_ADDR = 8
)(
    input logic                     i_clk       ,
    input logic                     i_rst_n     ,
    input logic [SIZE_ADDR-1:0]     i_num_elems ,
    input logic [SIZE_ADDR-1:0]     i_value_i   ,

    input logic                     i_start     ,
    input logic                     i_en        ,

    output logic                    o_en        ,
    output logic [SIZE_ADDR-1:0]    o_value_j   ,
    output logic                    o_done
);

logic w_start;
logic [SIZE_ADDR-1:0]   w_pre_value_j;
logic [SIZE_ADDR-1:0]   w_next_value_j;
logic w_pre_done;
logic w_enable;

assign w_pre_done       = (o_value_j == ('0));
assign w_start          = i_start;
assign w_enable         = w_start | (i_en & ~w_pre_done);
```

```
26  assign w_next_value_j    = o_value_j + 1'b1;
27  assign w_pre_value_j     = w_start ? (i_value_i+1'b1) : (w_next_value_j);
28
29  always_ff @( posedge i_clk or negedge i_rst_n ) begin : proc_output_value_j
30      if(~i_rst_n) begin
31          o_value_j        <= '0;
32      end else if(w_enable) begin
33          o_value_j        <= w_pre_value_j;
34      end
35  end
36
37  always_ff @( posedge i_clk or negedge i_rst_n ) begin : proc_done
38      if(~i_rst_n) begin
39          o_done           <= '0;
40      end else begin
41          o_done           <= w_pre_done;
42      end
43  end
44  // assign o_done           = w_pre_done;
45
46  assign o_en = w_enable;
47
48  endmodule
```

Listing 22: Module `Update_J`.

```
1   module Update_MIN #(
2       parameter SIZE_ADDR = 8
3   )(
4       input logic                     i_clk       ,
5       input logic                     i_rst_n     ,
6       input logic [SIZE_ADDR-1:0]     i_value_i   ,
7       input logic [SIZE_ADDR-1:0]     i_value_j   ,
8
9       input logic                     i_update_i  ,
10      input logic                     i_update_min,
11
12      output logic [SIZE_ADDR-1:0]    o_addr_min
13  );
14
15  logic w_en;
16  assign w_en = i_update_i | i_update_min;
17  always_ff @( posedge i_clk or negedge i_rst_n ) begin : proc_output_addr_min
18      if(~i_rst_n) begin
19          o_addr_min      <= '0;
20      end else if(w_en) begin
21          o_addr_min      <= (i_update_min) ? i_value_j : i_value_i;
22      end
23  end
24
25  endmodule
```

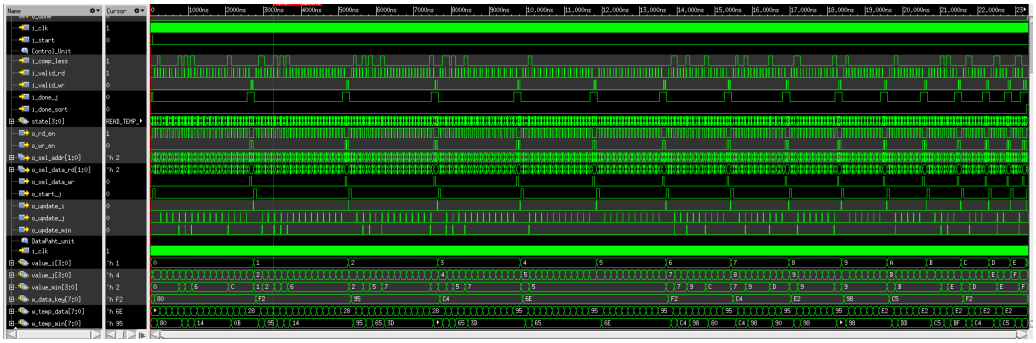
Listing 23: Module `Update_MIN`.

2. Kiểm định lại hoạt động của module:

```
      Building instance specific data structures.
    Loading native compiled code:     ..................... Done
    Design hierarchy summary:
    Instances  Unique
```

```
 Modules:                16      15
 Registers:             116     113
 Scalar wires:           92       -
 Vectored wires:         55       -
 Always blocks:          29      27
 Initial blocks:          3       3
 Cont. assignments:      24      24
 Pseudo assignments:     75      75
 Simulation timescale:  1ps
 Writing initial simulation snapshot: worklib.tb_Selection_Sort:sv
 Loading snapshot worklib.tb_Selection_Sort:sv ...................
Done
 xmsim: *W,DSEM2009: This SystemVerilog design is simulated as per
IEEE 1800-2009 SystemVerilog simulation semantics. Use -
disable_sem2009 option for turning off SV 2009 simulation semantics.
 xcelium> source /opt/cadence/XCELIUM2009/tools/xcelium/files/xmsimrc
 xcelium> run
 >>> Sending START...
 >>> DONE received at time 23305000

 ===== RAM AFTER SORT =====
 RAM[0] = 0b
 RAM[1] = 14
 RAM[2] = 28
 RAM[3] = 3d
 RAM[4] = 65
 RAM[5] = 6e
 RAM[6] = 80
 RAM[7] = 90
 RAM[8] = 95
 RAM[9] = 98
 RAM[10] = bd
 RAM[11] = bf
 RAM[12] = c4
 RAM[13] = c5
 RAM[14] = e2
 RAM[15] = f2
 >>> Saved sorted RAM to sorted_output.txt
 Simulation complete via $finish(1) at time 23325 NS + 0
 ../Topmodule/tb_Selection_Sort.sv:71         $finish;
 xcelium> exit
 TOOL: xrun(64)  20.09-s001: Exiting on Dec 12, 2025 at 08:31:02 EST
 (total: 00:00:01)
```

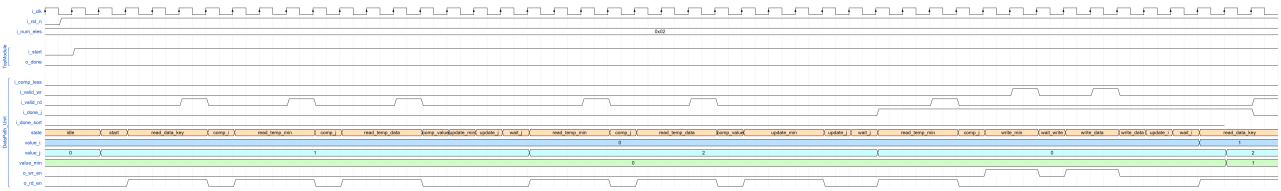
Listing 24: Kết quả sau khi kiểm định lại chức năng của module.



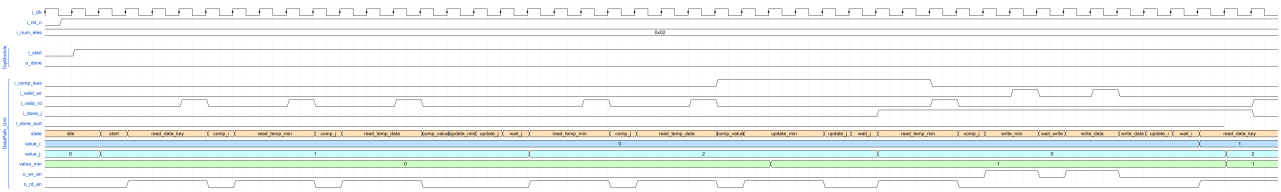
Hình 14: Sóng ngõ ra tổng quan của module test.

e) Giả sử vòng lặp $i = 0$, không có swap (hoán đổi) và vòng lặp $i = 1$, có swap. Vẽ dạng sóng từ lúc $start = 1$ đến lúc hoàn thành vòng lặp $i = 1$ của thiết kế.

1. Vòng lặp $i = 0$ với không có swap:



2. Vòng lặp $i = 0$ với có swap:



Câu 4

Thiết kế phần cứng dùng xóa đi các phần tử có giá trị chẵn trong một mảng dữ liệu (mô tả ở ví dụ). Giả sử mảng được lưu trong bộ nhớ Single Port và quá trình đọc/ghi diễn ra đồng bộ theo Clk và hoàn thành trong 1 Clk.

Ví dụ: (... là dữ liệu từ các ô nhớ khác được chuyển xuống)

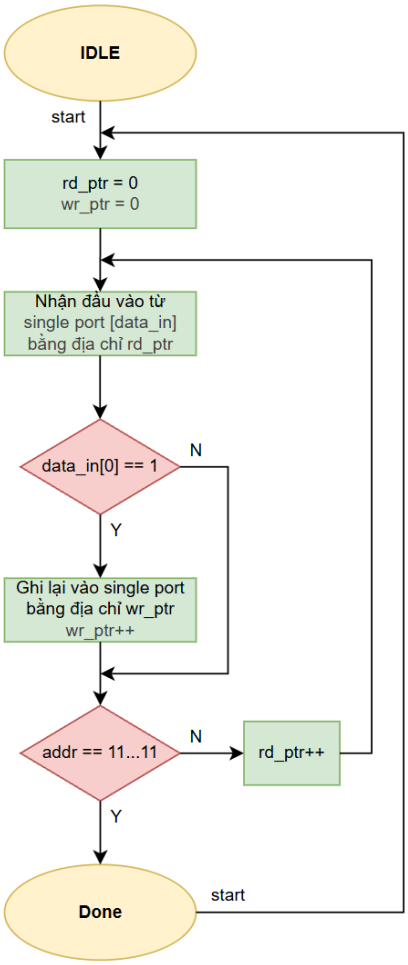
Địa chỉ	Nội dung
0x04	15
0x03	14
0x02	12
0x01	33
0x00	24

Trước khi thực hiện

Địa chỉ	Nội dung
0x04	..
0x03	..
0x02	...
0x01	15
0x00	33

Sau khi thực hiện

Hình 15: Ví dụ.



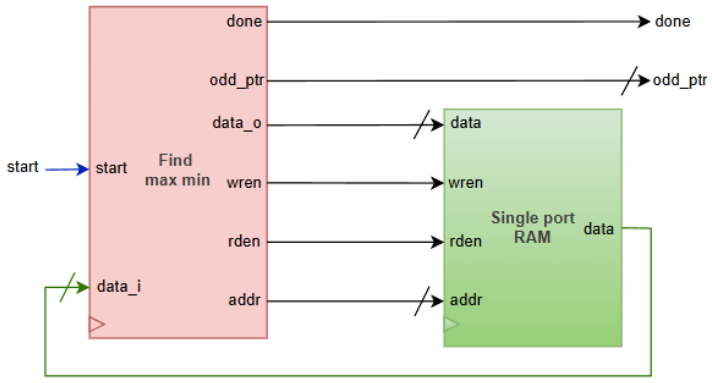
Hình 16: Giải thuật sử dụng.

a) Định nghĩa ngõ vào và ra của thiết kế, vẽ kết nối của thiết kế với bộ nhớ (Yêu cầu phải có chân start và reset).

Tên tín hiệu	IO	Độ rộng	Mô tả	
clk	Input	logic	1	Tín hiệu clock
rst_n	Input	logic	1	Reset tích cực mức thấp
start	Input	logic	1	Tín hiệu bắt đầu hoạt động
rden	Output	logic	1	Tín hiệu cho phép đọc dữ liệu từ bộ nhớ
wren	Output	logic	1	Tín hiệu cho phép ghi dữ liệu từ bộ nhớ
addr	Output	logic	$\lceil \log_2(\text{DEPTH}) \rceil$	Địa chỉ đọc/ghi dữ liệu
i_data	Input	logic signed	WIDTH	Dữ liệu đầu vào từ bộ nhớ
done	Output	logic	1	Tín hiệu kết thúc quá trình
o_data	Output	logic signed	WIDTH	Dữ liệu đầu ra, ghi lại số lẻ vào bộ nhớ
odd_ptr	Output	logic	$\lceil \log_2(\text{DEPTH}) \rceil$	Khoảng dữ liệu hợp lệ trong bộ nhớ

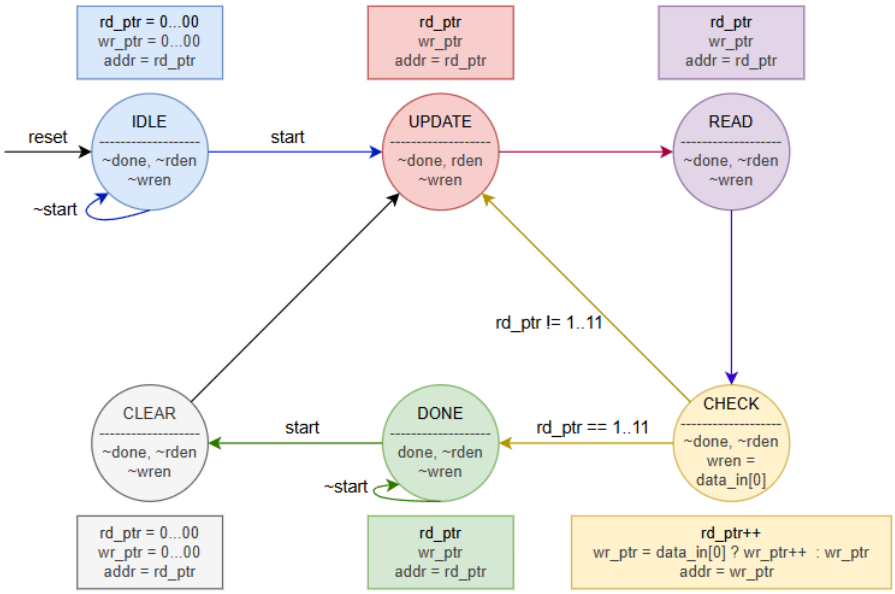
Bảng 2: I/O table for module `clear_even`

Khi lọc dữ liệu chẵn ra khỏi bộ nhớ, đồng thời dồn các giá trị lẻ về các địa chỉ thấp, điều này khiến một khoảng dữ liệu phía trên sẽ không còn hợp lệ sau khi thực hiện xong, tín hiệu odd_ptr đóng vai trò giới hạn khoảng dữ liệu hợp lệ trong bộ nhớ (từ địa chỉ 0 đến odd_ptr là các giá trị lẻ hợp lệ sau khi xóa các giá trị chẵn ra khỏi bộ nhớ).



Hình 17: Tổng quan kết nối của thiết kế.

b) Thiết kế máy trạng thái bậc cao của thiết kế.

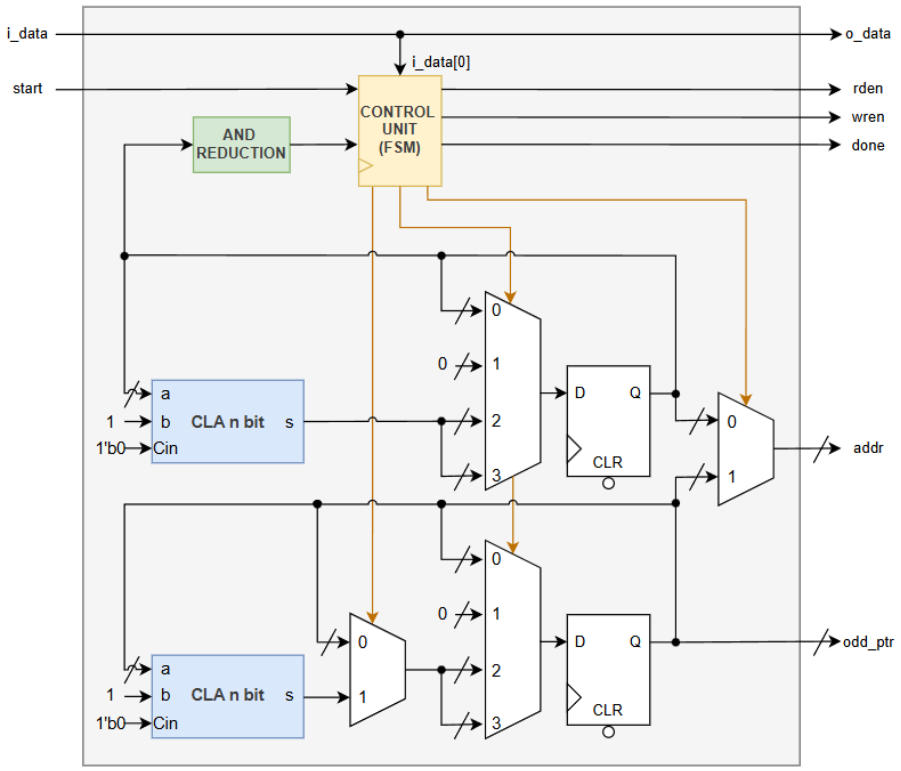


Hình 18: Máy trạng thái bậc cao.

- IDLE: là trạng thái ban đầu, khi reset sẽ luôn trở về trạng thái này, giá trị các thanh ghi mặc định là 0, đây là trạng thái chờ ban đầu của hệ thống, khi có tín hiệu start sẽ chuyển sang trạng thái UPDATE.
- UPDATE: ở trạng thái này sẽ cập nhật các giá trị hai thanh ghi con trỏ đọc/ghi mới được tín toán từ trạng thái CHECK, đồng thời đưa ngõ ra rden lên mức cao để chuẩn bị nhận dữ liệu ngõ vào mới từ bộ nhớ, địa chỉ đọc chứa trong thanh ghi rd_ptr.
- READ: đây là trạng thái chờ đọc vì ở đây bộ nhớ đọc đồng bộ, do đó cần phải đợi một chu kì để nhận dữ liệu đầu vào.
- CHECK: ở trạng thái này, sẽ lấy LSB của tín hiệu đầu vào để xác định chẵn/lẻ. Khi là số lẻ sẽ tiến hành đưa ngõ ra wren lên mức cao để ghi vào bộ nhớ với địa chỉ chứa trong thanh ghi wr_ptr đồng thời sẽ tăng giá trị wr_ptr lên 1 ở trạng thái kế. Khi là số chẵn thì sẽ không ghi, không cập nhật giá trị thanh ghi ở chu kì kế. Thanh ghi rd_ptr sẽ tăng thêm 1 ở trạng thái kế. Nếu đã xét hết dữ liệu trong bộ nhớ sẽ chuyển sang trạng thái DONE, ngược lại sẽ sang trạng thái UPDATE để chuẩn bị nhận dữ liệu mới từ bộ nhớ.
- DONE: là trạng thái thông báo việc hoàn tất, lúc này tín hiệu done sẽ tích cực mức cao (done chỉ mức cao ở trạng thái này). Nếu nhận được tín hiệu start mức cao sẽ chuyển sang trạng thái CLEAR, ngược lại sẽ giữ trạng thái hiện tại.

- CLEAR: trạng thái này đóng vai trò đặt lại giá trị các thanh ghi con trỏ là 0, chuẩn bị cho lần hoạt động kế tiếp.

c) Thiết kế Datapath và Control Unit của thiết kế.



Hình 19: Datapath.

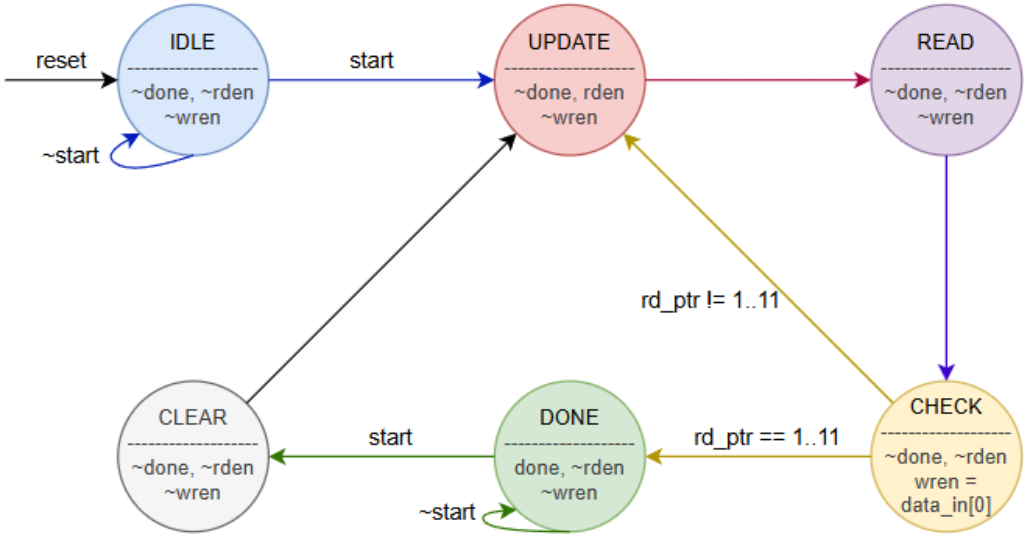
Datapath của thiết kế gồm 2 khối MUX 4-1 và 2 khối MUX 2-1 để xác định giá trị cập nhật cho 2 thanh ghi dựa vào trạng thái hiện tại và lựa chọn địa chỉ truy cập bộ nhớ.

Hai thanh ghi gồm:

- Thanh ghi địa chỉ con trỏ đọc rd_ptr, đóng vai trò trỏ vào bộ nhớ, kết hợp với tín hiệu rden để lấy dữ liệu đầu vào.
- Thanh ghi địa chỉ con trỏ ghi wr_ptr, đóng vai trò trỏ vào bộ nhớ, kết hợp với tín hiệu wren để lưu lại các phần tử lẻ.

Hai bộ cộng để cập nhật địa chỉ con trỏ đọc/ghi.

Một bộ AND Reduction để xác định địa chỉ đạt đến giá trị cuối của bộ nhớ, là điều kiện để chuyển từ trạng thái CHECK sang trạng thái DONE.



Hình 20: Control Unit FSM.

d) Viết chương trình mô phỏng hoạt động của thiết kế.

```
module clear_even #(
    parameter DEPTH = 32,
    parameter WIDTH = 8
)(
    input  logic                              i_clk    ,
    input  logic                              i_rst_n  ,
    input  logic                              i_start  ,
    output logic                              o_rden   ,
    output logic                              o_wren   ,
    output logic        [$clog2(DEPTH) - 1:0] o_addr   ,
    input  logic signed [WIDTH         - 1:0] i_data   ,
    output logic                              o_done   ,
    output logic signed [WIDTH         - 1:0] o_data   ,
    output logic        [$clog2(DEPTH) - 1:0] o_odd_ptr
);

    localparam DEPTH_S = DEPTH - 1;
    localparam WIDTH_S = WIDTH - 1;

    typedef enum logic[2:0] {
        IDLE    ,
        UPDATE  ,
        READ    ,
        CHECK   ,
        DONE    ,
        CLEAR
    } e_state;

    e_state pstate, nstate;
    logic [$clog2(DEPTH) - 1:0] reg_rd_ptr, rd_ptr, rd_ptr_add, reg_wr_ptr, wr_ptr, wr_ptr_add;

```

```
    logic  odd_flag, count_end;
    assign odd_flag  = i_data[0];
    assign count_end = (&reg_rd_ptr);

    always_ff @(posedge i_clk, negedge i_rst_n) begin
        if(~i_rst_n) pstate <= IDLE  ;
        else         pstate <= nstate;
    end

    always_comb begin
        case(pstate)
            IDLE   : nstate = i_start ? UPDATE : pstate;
            UPDATE : nstate = READ;
            READ   : nstate = CHECK;
            CHECK  : nstate = count_end ? DONE : UPDATE;

            DONE   : nstate = i_start ? CLEAR : pstate;
            CLEAR  : nstate = UPDATE;
            default: nstate = IDLE;
        endcase
    end

    always_ff @(posedge i_clk, negedge i_rst_n) begin
        if(~i_rst_n) reg_rd_ptr <= '0;
        else         reg_rd_ptr <= rd_ptr;
    end

    always_ff @(posedge i_clk, negedge i_rst_n) begin
        if(~i_rst_n) reg_wr_ptr <= '0;
        else         reg_wr_ptr <= wr_ptr;
    end

    assign o_rden = (pstate == UPDATE);
    assign o_wren = (pstate == CHECK) ? odd_flag : 1'b0;
    assign o_addr = (pstate == CHECK) ? reg_wr_ptr : reg_rd_ptr;

    assign rd_ptr = (pstate == CHECK) ? rd_ptr_add : ((pstate == CLEAR) ? '0 : reg_rd_ptr);
    assign wr_ptr = (pstate == CHECK) ? (odd_flag & ~count_end  ? wr_ptr_add : reg_wr_ptr) : ((pstate ==
    CLEAR) ? '0 : reg_wr_ptr);

    assign o_data = i_data;

    assign o_done = (pstate == DONE);

    assign o_odd_ptr = reg_wr_ptr;

    adder_flex_no_carry #(
        .WIDTH($clog2(DEPTH))
    ) add_1 (
        .i_a   (reg_rd_ptr),
        .i_b   ({{DEPTH_S{1'b0}},1'b1}),
        .i_cin (1'b0),
        .o_s   (rd_ptr_add)
    );

    adder_flex_no_carry #(
        .WIDTH($clog2(DEPTH))
    ) add_2 (
        .i_a   (reg_wr_ptr),
        .i_b   ({{DEPTH_S{1'b0}},1'b1}),
        .i_cin (1'b0),
        .o_s   (wr_ptr_add)
    );

```

```
95 endmodule
```

Listing 25: HDL mô tả thiết kế xóa các phần tử có giá trị chẵn trong một mảng dữ liệu.

Để tạo giá trị ngẫu nhiên trong bộ nhớ để mô phỏng, nhóm chọn sử dụng $\$unrandom\_range(,);$ mà System Verilog cung cấp.

```
1  localparam WIDTH_S = DW - 1;
2  logic signed [WIDTH-1:0] buffer [DEPTH-1:0];
3
4  for (int i = 0; i < DEPTH; i++) begin
5     ram.mem[i] = $urandom_range(-2**(WIDTH_S), 2**(WIDTH_S) - 1);
6     buffer[i] = '0;
7     if(ram.mem[i][0] == 0) $display("RAM[%5d] DATA_EVEN = %5d", i, ram.mem[i]);
8  end
9
10 $display("\n");
11
12 for (int i = 0; i < DEPTH; i++) begin
13    if(ram.mem[i][0]) begin
14       $display("RAM[%5d] DATA_ODD = %5d", i, ram.mem[i]);
15       buffer[odd] = ram.mem[i];
16       odd++;
17    end
18 end
19
20 $display("Odd number of numbers:  %5d", odd);
```

Listing 26: Chương trình tạo giá trị ngẫu nhiên ban đầu cho bộ nhớ.

```
1  for(int i = 0; i < DEPTH; i++) begin
2     $display("RAM[%5d] = %5d", i, ram.mem[i]);
3     if(i >= odd_ptr) $display("[%4s] expect = %5d", (ram.mem[i] == buffer[i]) ? "TRUE" : "FAIL", buffer[i
   ]);
4  end
5
6  $display("Odd address use: %5d", odd_ptr);
```

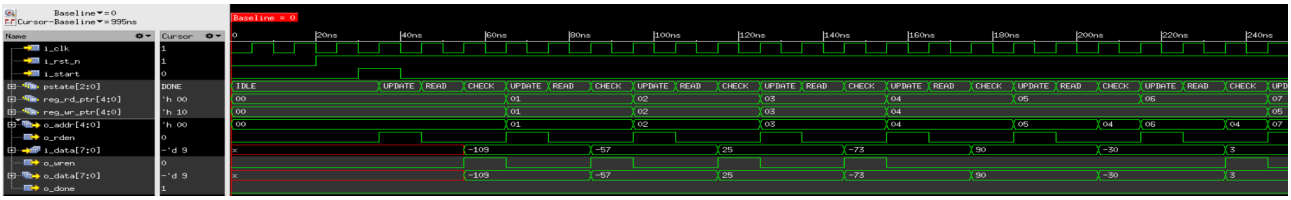
Listing 27: Chương trình kiểm định thiết kế.

```
xcelium> run
RAM[    4] DATA_EVEN =    90
RAM[    5] DATA_EVEN =   -30
RAM[    8] DATA_EVEN =    96
RAM[   10] DATA_EVEN =    50
RAM[   11] DATA_EVEN =    58
RAM[   12] DATA_EVEN =   -60
RAM[   13] DATA_EVEN =   -20
RAM[   16] DATA_EVEN =   -66
RAM[   18] DATA_EVEN =   -88
RAM[   19] DATA_EVEN =    76
RAM[   20] DATA_EVEN =   -80
RAM[   24] DATA_EVEN =    54
RAM[   26] DATA_EVEN =   -76
```

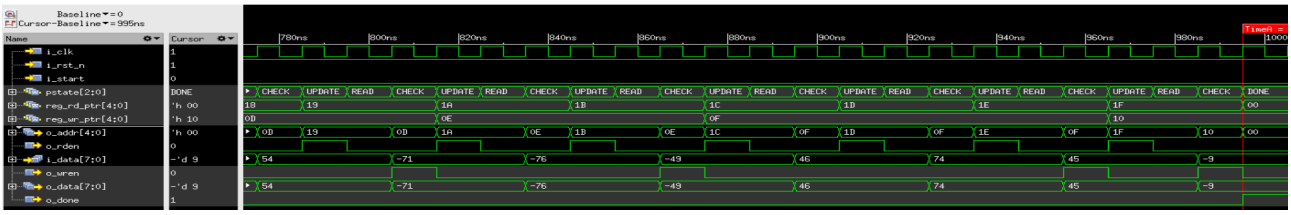
```
RAM[   28] DATA_EVEN =    46
RAM[   29] DATA_EVEN =    74


RAM[    0] DATA_ODD =  -109
RAM[    1] DATA_ODD =   -57
RAM[    2] DATA_ODD =    25
RAM[    3] DATA_ODD =   -73
RAM[    6] DATA_ODD =     3
RAM[    7] DATA_ODD =   -45
RAM[    9] DATA_ODD =   -63
RAM[   14] DATA_ODD =   -47
RAM[   15] DATA_ODD =   115
RAM[   17] DATA_ODD =    49
RAM[   21] DATA_ODD =   -91
RAM[   22] DATA_ODD =    53
RAM[   23] DATA_ODD =   -23
RAM[   25] DATA_ODD =   -71
RAM[   27] DATA_ODD =   -49
RAM[   30] DATA_ODD =    45
RAM[   31] DATA_ODD =    -9
Odd number of numbers:   17

RAM[    0] =  -109
[TRUE] expect =  -109
RAM[    1] =   -57
[TRUE] expect =   -57
RAM[    2] =    25
[TRUE] expect =    25
RAM[    3] =   -73
[TRUE] expect =   -73
RAM[    4] =     3
[TRUE] expect =     3
RAM[    5] =   -45
[TRUE] expect =   -45
RAM[    6] =   -63
[TRUE] expect =   -63
RAM[    7] =   -47
[TRUE] expect =   -47
RAM[    8] =   115
[TRUE] expect =   115
RAM[    9] =    49
[TRUE] expect =    49
RAM[   10] =   -91
[TRUE] expect =   -91
RAM[   11] =    53
[TRUE] expect =    53
```

```
RAM[   12] =   -23
[TRUE] expect =   -23
RAM[   13] =   -71
[TRUE] expect =   -71
RAM[   14] =   -49
[TRUE] expect =   -49
RAM[   15] =    45
[TRUE] expect =    45
RAM[   16] =    -9
[TRUE] expect =    -9
RAM[   17] =    49
RAM[   18] =   -88
RAM[   19] =    76
RAM[   20] =   -80
RAM[   21] =   -91
RAM[   22] =    53
RAM[   23] =   -23
RAM[   24] =    54
RAM[   25] =   -71
RAM[   26] =   -76
RAM[   27] =   -49
RAM[   28] =    46
RAM[   29] =    74
RAM[   30] =    45
RAM[   31] =    -9
Odd address use:16
Simulation complete via $finish(1) at time 20040 NS + 0
../01_tb/clear_even_tb.sv:103         $finish;
```

Listing 28: Kết quả kiểm định cho thiết kế xóa các phần tử có giá trị chẵn trong một mảng dữ liệu.



Hình 21: Dạng sóng lúc bắt đầu.



Hình 22: Dạng sóng lúc hoàn thành.

Kết luận: Máy trạng thái chuyển đúng so với dự tính, các giá trị chẵn đã được lọc chính xác.