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

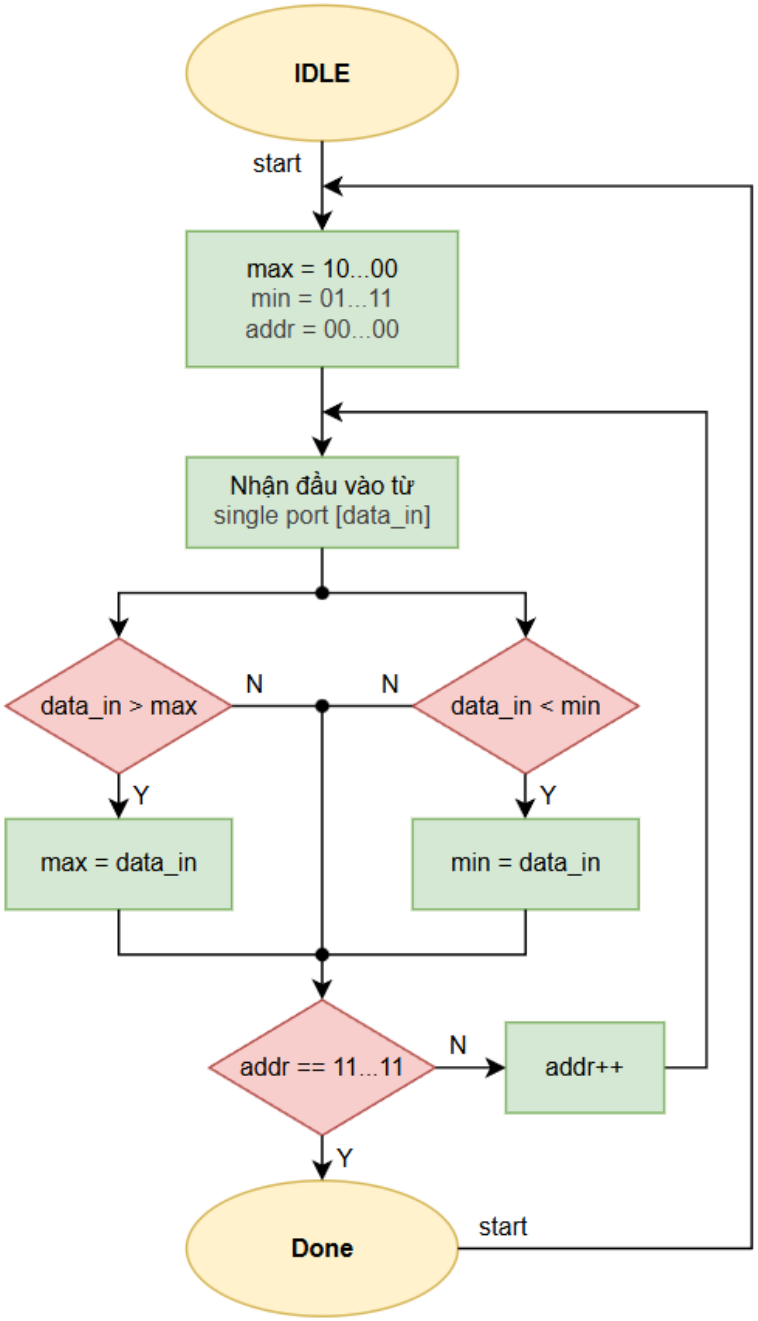
Danh sách hình vẽ

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).

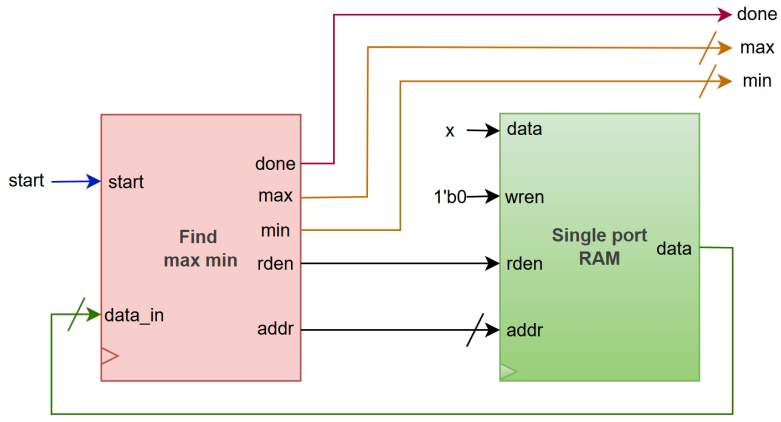
Đề bài không đề cập đến xử lí cho dạng dữ liệu có dấu hoặc không dấu nên nhóm chọn xử lí ở dạng có dấu bù 2.

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

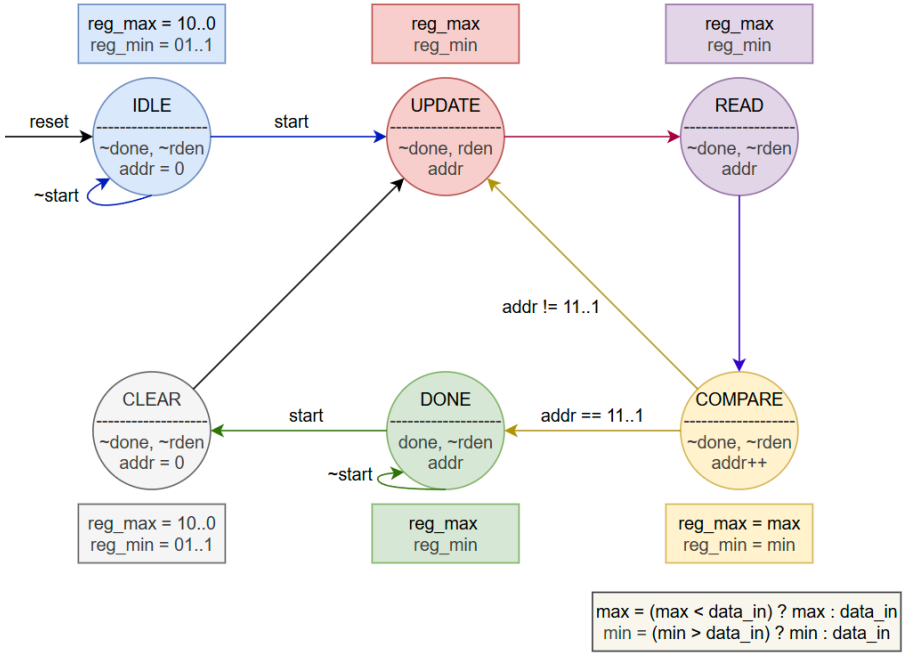
Bảng 1: Bảng tín hiệu I/O của module `max_min`

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.

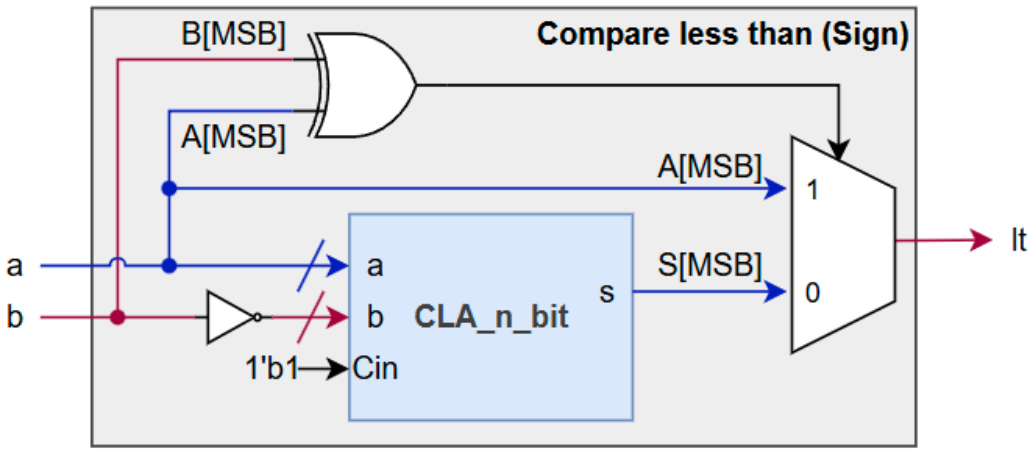
Thiết kế gồm 6 trạng thái chính:

- 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.

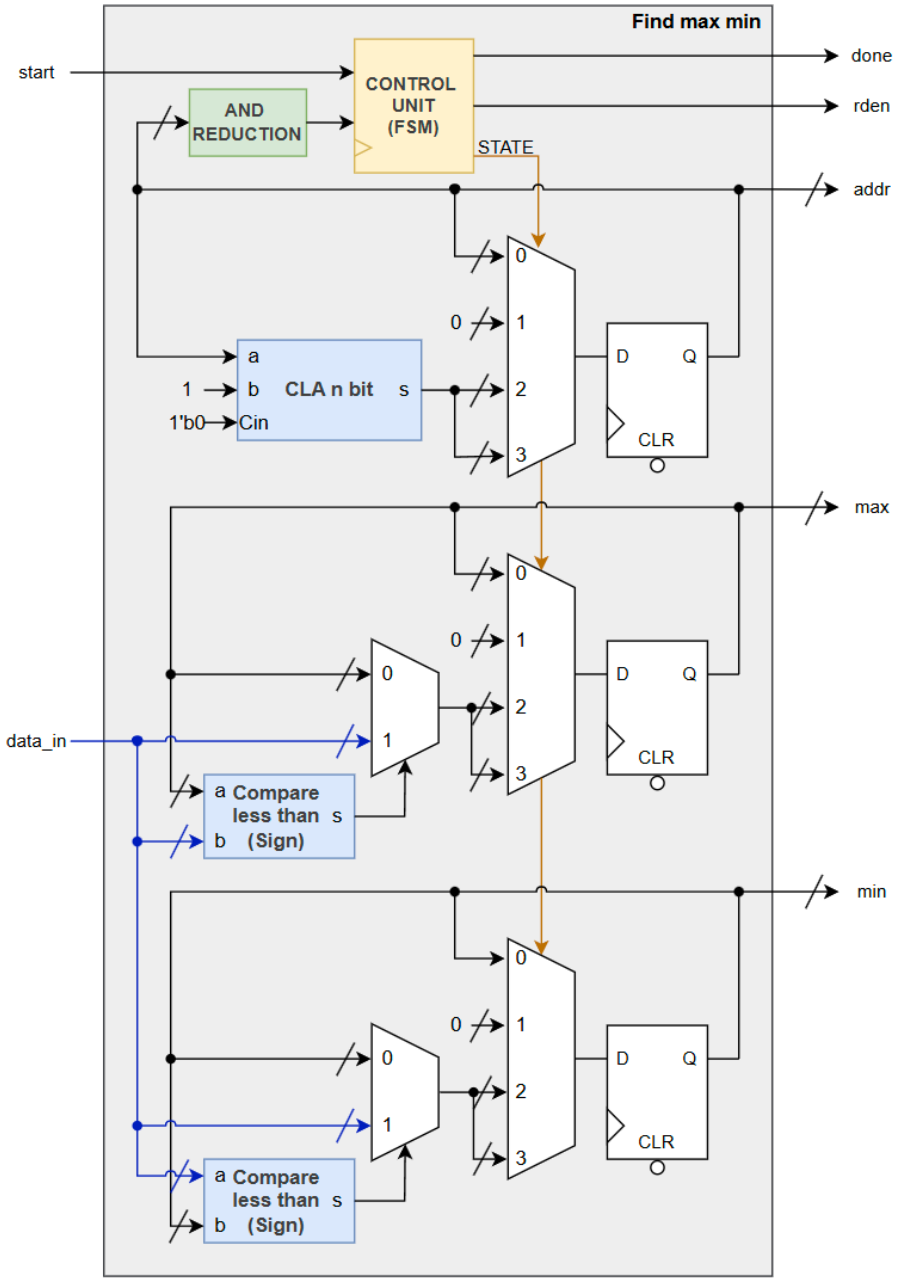
c) Thiết kế Datapath và Control Unit của thiết kế.

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:

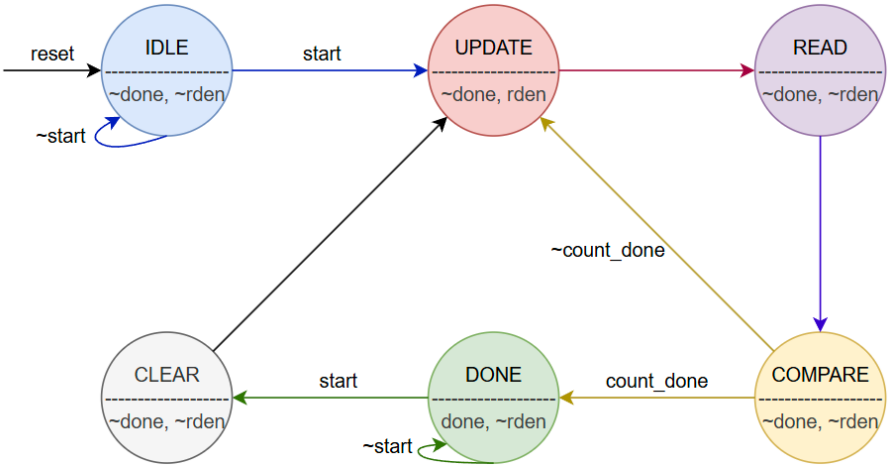
- Thanh ghi địa chỉ, đóng vai trò trỏ vào bộ nhớ, kết hợp với tín hiệu rden để lấy dữ liệu đầu vào.

- 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);
```

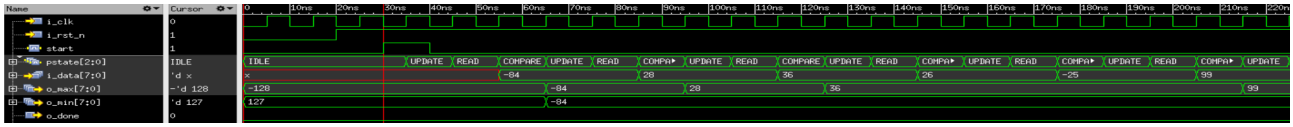
Listing 3: Chương trình tạo giá trị ngẫu nhiên ban đầu cho bộ nhớ.

```
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
```

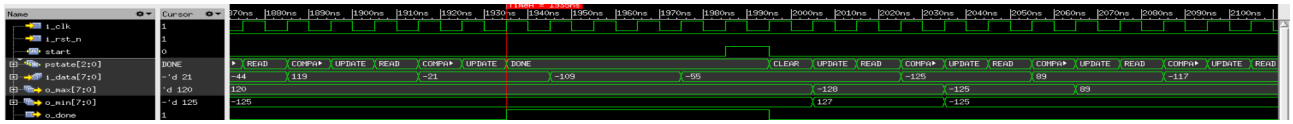
Listing 4: Chương trình kiểm định thiết kế.

```
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.

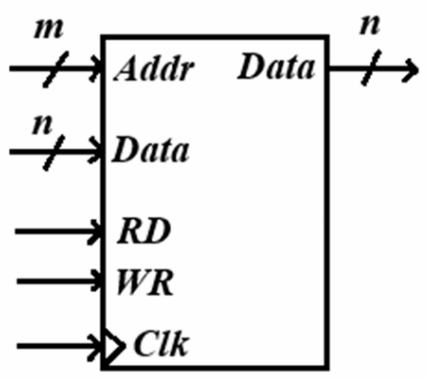
Kết luận: Máy trạng thái chuyển đúng so với dự tính, giá trị kết quả so sánh chính xác.

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***:

```
1        int n = arr.size() - 1;
2        for(int i = 0; i < n; i++) {
3            min = i;
4            for(int j = i+1; j <= n; j++){
5                if(arr[j] < arr[min]){
6                    min = j;
7                }
8            }
9            swap(arr[i], arr[min]);
10       }
11
```

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

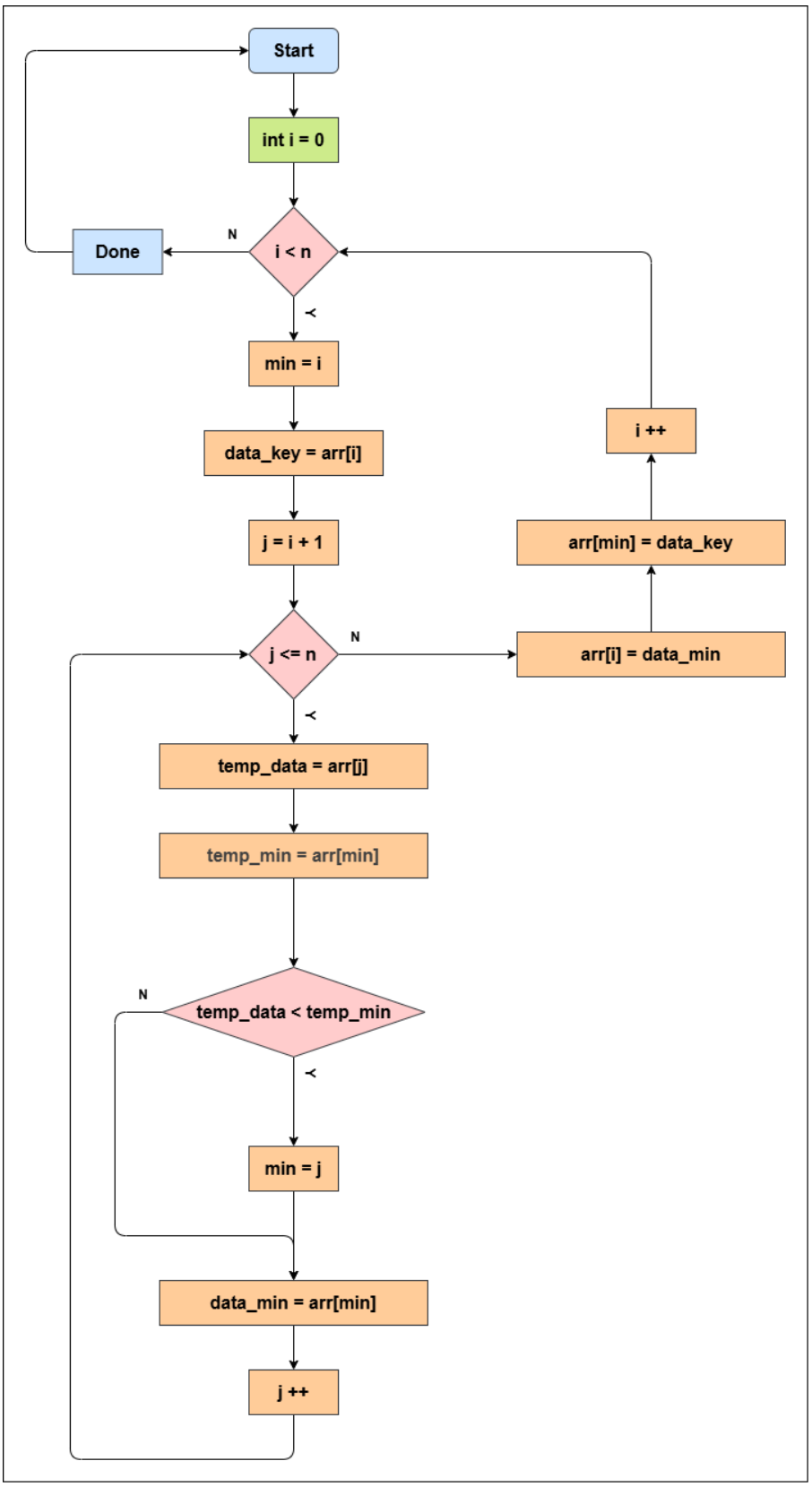


Hình 9: Yêu cầu của bộ nhớ.

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:

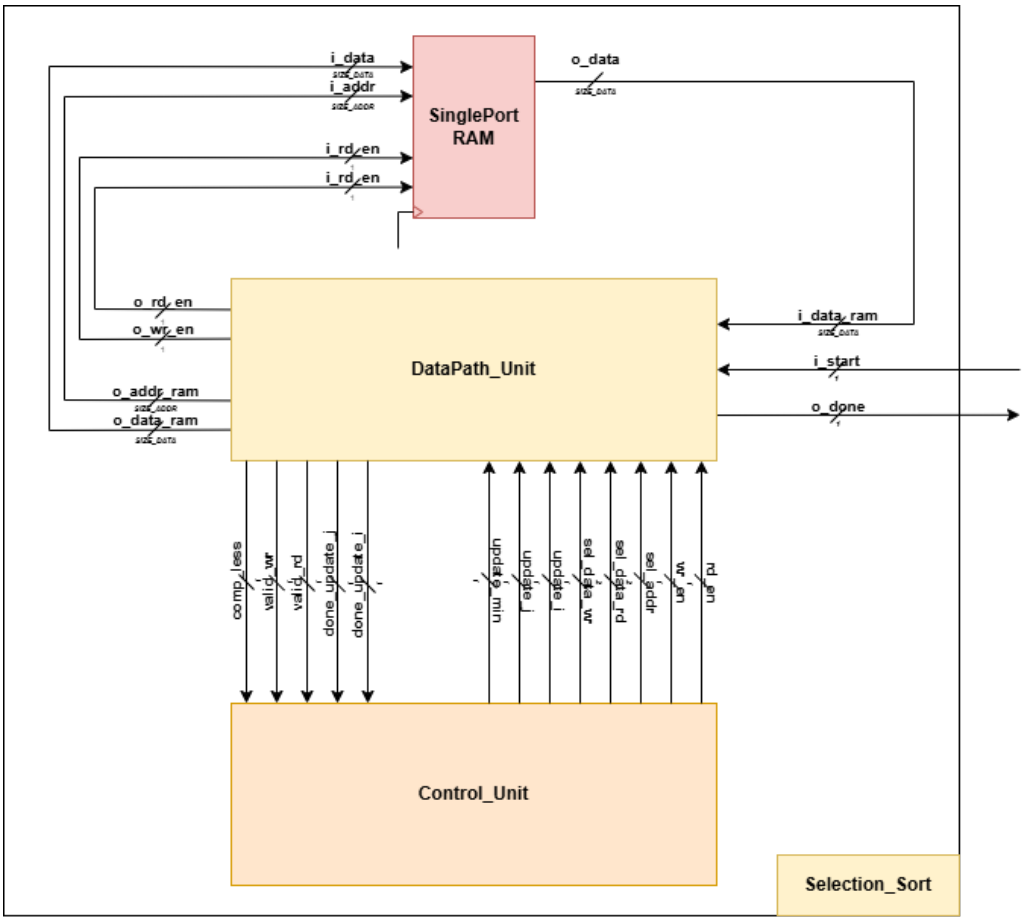
```
Finished reading file: ./tools/unsorted.txt
Check Selection_Sort Standard: PASS
Finish write file './Reports/COMPILE_REPORT/sorted_standard.txt'.
Check Selection_Sort Cus: PASS
Finish write file './Reports/COMPILE_REPORT/sorted_cus.txt'.
```

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:

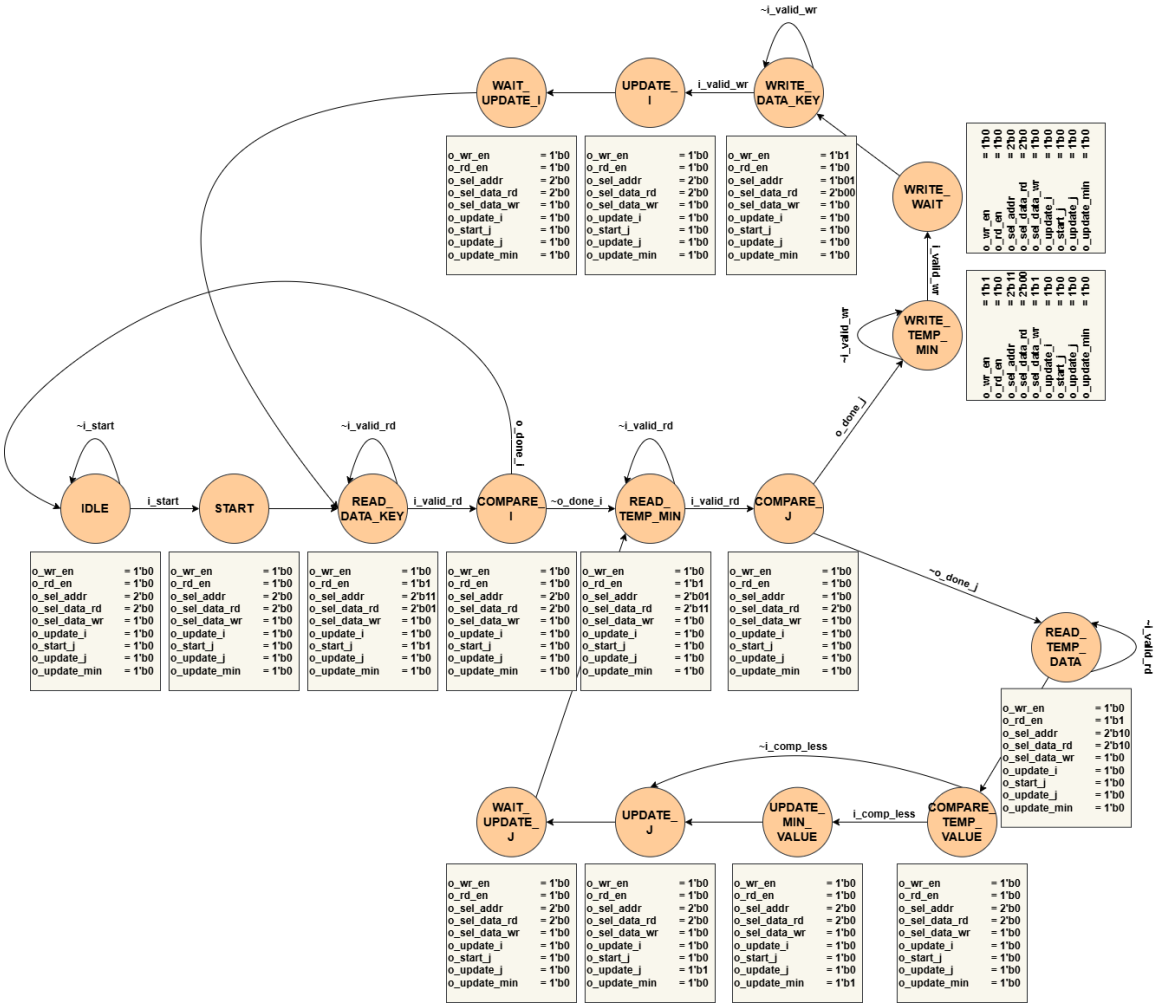
Signal	Size	Functional
i_clk	1	Clock của toàn hệ thống.
i_rst_t	1	Tín hiệu reset của hệ thống với tích cực thấp.
i_start	1	Tín hiệu bắt đầu hoạt động của bộ.
o_done	1	Tín hiệu cho biết được đã sắp xếp xong mảng.

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: