

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

Câu 1	1
a)	1
b)	3
c)	4
d)	5
e)	6
 Câu 2	 15
a)	16
b)	17
c)	18
d)	20
 Câu 3	 24
a)	24
b)	28
c)	29
d)	32
e)	49
 Câu 4	 50
a)	51
b)	52
c)	53
d)	54

Danh sách hình vẽ

1	Lưu đồ giải thuật của thuật toán A/B	1
2	Sơ đồ khối tổng quát của module thực hiện thuật toán A/B	1
3	Thiết kế HighLevel FSM.	4
4	DataPath của bộ chia.	5
5	Ripple Carry Add/Sub 16 bit.	6
6	Control_Unit của bộ chia.	6
7	Dạng sóng của testbench mô phỏng khối thực hiện phép chia.	14
8	Giải thuật sử dụng.	15
9	Tổng quan kết nối thiết kế.	16
10	Máy trạng thái bậc cao.	17
11	Bộ so sánh nhỏ hơn.	18
12	Datapath.	19
13	Control Unit FSM.	20
14	Dạng sóng lúc bắt đầu.	23
15	Dạng sóng lúc hoàn thành.	23
16	Yêu cầu của bộ nhớ.	24
17	Flowchart của thuật toán Selection Sort.	25
18	Thiết kế tổng quan của module Selection_Sort	27
19	Lưu đồ trạng thái của module Selection_Sort	28
20	Thiết kế tổng quan của module DataPath_Unit	30
21	Sóng ngõ ra tổng quan của module test.	49
22	Ví dụ.	50
23	Giải thuật sử dụng.	50
24	Tổng quan kết nối của thiết kế.	51
25	Máy trạng thái bậc cao.	52

26	Datapath.	53
27	Control Unit FSM.	54
28	Dạng sóng lúc bắt đầu.	58
29	Dạng sóng lúc hoàn thành.	59

Danh sách bảng

1	Kết quả chạy giải thuật.	2
2	Bảng mô tả các tín hiệu.	3
3	Bảng các trường hợp đặc biệt (Special Cases).	7
4	Bảng tín hiệu I/O của module <code>max_min</code>	16
5	Bảng I/O của module <code>Selection_Sort</code>	27
6	I/O table for module <code>clear_even</code>	51

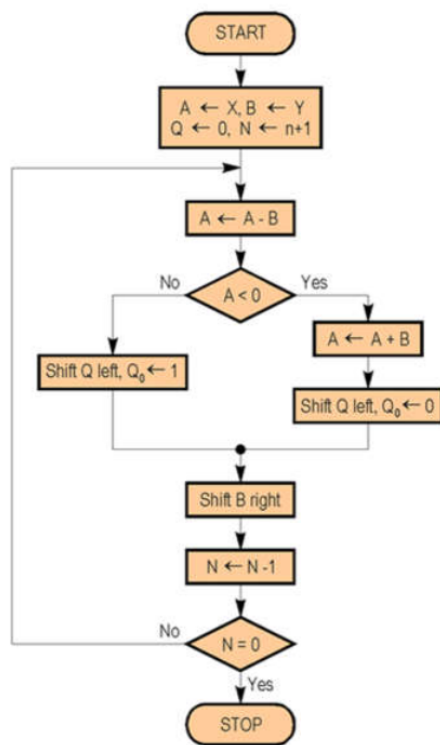
List of Listings

1	Module <code>divider8_top</code>	7
2	Module <code>div_ctrl</code>	8
3	Module <code>alu16</code>	9
4	Module <code>div_dp</code>	9
5	Module <code>eq8</code>	11
6	Kết quả test của module thực hiện phép chia.	11
7	HDL mô tả bộ so sánh bé hơn có dấu.	20
8	HDL mô tả thiết kế tìm số lớn nhất và nhỏ nhất.	21
9	Chương trình tạo giá trị ngẫu nhiên ban đầu cho bộ nhớ.	22
10	Chương trình kiểm định thiết kế.	23
11	Kết quả kiểm định cho thiết kế bộ tìm số lớn nhất và nhỏ nhất.	23
12	Đoạn chương trình C của giải thuật Selection Sort.	24
13	Đoạn code nguyên mẫu của giải thuật Selection Sort.	26
14	Đoạn code chỉnh sửa của giải thuật Selection Sort.	26
15	Kết quả so sánh 2 cách viết của Selection Sort.	26
16	Module <code>Selection_Sort</code>	33

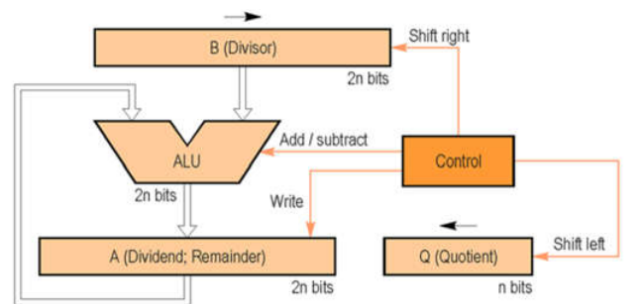
17	Module SinglePort_RAM.	34
18	Module Control_unit.	35
19	Module Data_path.	38
20	Module Block_Addr.	40
21	Module Block_Read_data.	41
22	Module Block_Write_data.	42
23	Module RAM_addr.	43
24	Module RAM_read_data.	43
25	Module RAM_write_data.	44
26	Module SS_detect_edge.	45
27	Module Update_I.	45
28	Module Update_J.	46
29	Module Update_MIN.	47
30	Kết quả sau khi kiểm định lại chức năng của module.	47
31	HDL mô tả thiết kế xóa các phần tử có giá trị chẵn trong một mảng dữ liệu. . .	54
32	Chương trình tạo giá trị ngẫu nhiên ban đầu cho bộ nhớ.	56
33	Chương trình kiểm định thiết kế.	56
34	Kết quả kiểm định cho thiết kế xóa các phần tử có giá trị chẵn trong một mảng dữ liệu.	56

Câu 1

Để thực hiện chia hai số A cho B , ta có lưu đồ giải thuật sau:



Hình 1: Lưu đồ giải thuật của thuật toán A/B .



Hình 2: Sơ đồ khối tổng quát của module thực hiện thuật toán A/B .

a) Mô tả các bước chạy của giải thuật với ngõ vào $A = 174$ và $B = 25$.

Bước 1: Khởi tạo và load số A (Dividend_16 bits) và B (Divisor_16 bits) vào thanh ghi. Thực hiện căn chỉnh cho MSB_B nằm cùng MSB_A . Đồng thời, bắt đầu với thanh ghi Q (8 bit) = 0 và khởi tạo số vòng lặp bằng số bit $Q + 1$ (9 vòng).

Bước 2: Thực hiện $A - B$ nếu $bit_borrow = 0$ và cập nhật $A \leftarrow A - B$. Hoặc thực hiện $A + B$ nếu $bit_borrow = 1$ để restore A . Với Q (8_bits) sẽ được cập nhật bằng cách shift left như sau: $Q = \{regQ[6:0], \sim bit_borrow\}$;

Bước 3: Shift right B và lặp lại quá trình $A - B$ đến khi đủ số vòng lặp thì đó là kết quả cuối cùng.

Theo đề, với $A = 174$ (1010_1110) và $B = 25$ (11001). Ta xác định được:

Vòng lặp	Reg_A	Reg_B	A - B	Borrow_flag	Q = {reg_Q[6:0], ~borrow_flag}
(Init)	Load A = 174	Load B = 25 \ll 256 = 6400	Chưa thực hiện		0000_0000
9	174	6400	$A - B < 0 \implies A = 174$	1	0000_0000
8	174	$B \gg 1 = 3200$	$A - B < 0 \implies A = 174$	1	0000_0000
7	174	$B \gg 1 = 1600$	$A - B < 0 \implies A = 174$	1	0000_0000
6	174	$B \gg 1 = 800$	$A - B < 0 \implies A = 174$	1	0000_0000
5	174	$B \gg 1 = 400$	$A - B < 0 \implies A = 174$	1	0000_0000
4	174	$B \gg 1 = 200$	$A - B < 0 \implies A = 174$	1	0000_0000
3	174	$B \gg 1 = 100$	$A - B > 0 \implies A = 74$	0	0000_0001
2	74	$B \gg 1 = 50$	$A - B > 0 \implies A = 24$	0	0000_0011
1	24	$B \gg 1 = 25$	$A - B < 0 \implies A = 24$	1	0000_0110

Bảng 1: Kết quả chạy giải thuật.

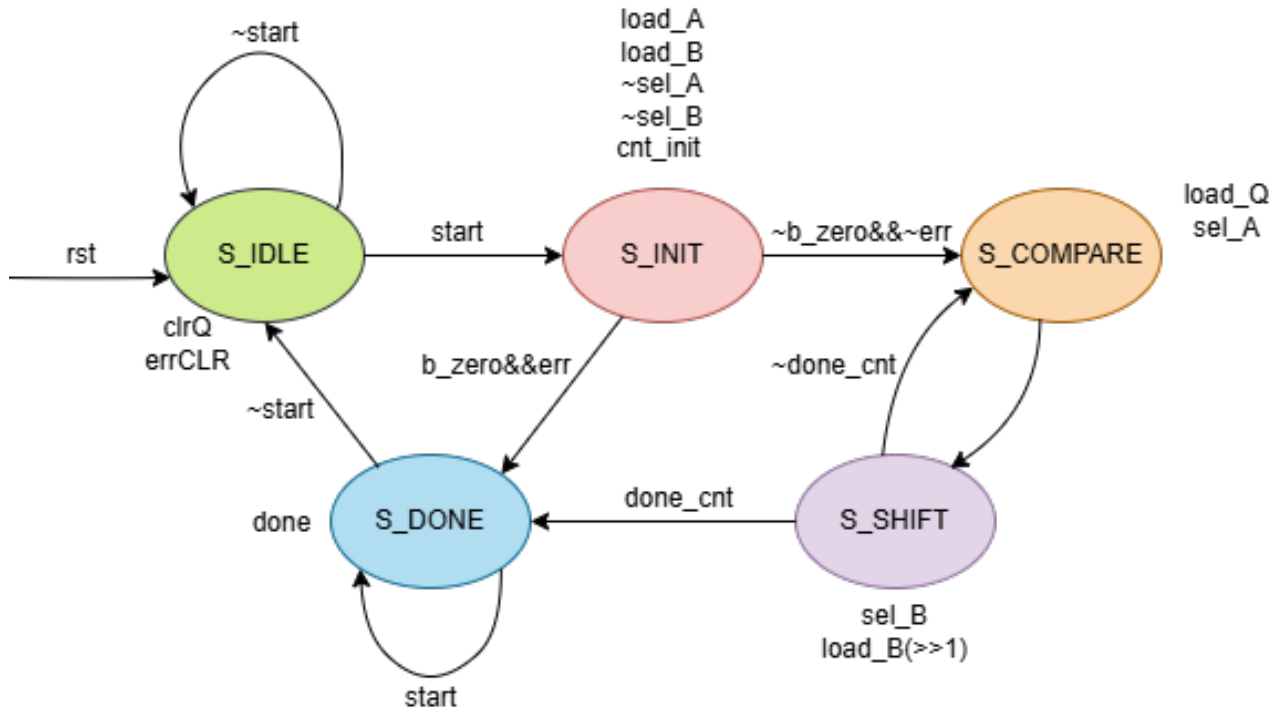
Vậy sau 9 vòng lặp kết quả của $A : B$ (174 : 25) là **Q = 0000_0110 = 6** và dư **R = 24**.

b) Định nghĩa ngõ vào và ngõ ra của thiết kế, giả sử A và B là 2 số 8 bit (Yêu cầu phải có chân start và reset).

Tên tín hiệu	I/O	Độ rộng bit	Mô tả tín hiệu
clk	Input	1	Tín hiệu clock.
rst	Input	1	Reset tích cực cao.
start	Input	1	Tín hiệu bắt đầu.
A	Input	8	Dividend (Số bị chia).
B	Input	8	Divisor (Số chia).
Q	Output	8	Quotient (Thương).
R	Output	8	Remainder (Dư).
done	Output	1	Tín hiệu báo phép chia hoàn tất.
err	Output	1	Tín hiệu lỗi khi Divisor = 0.

Bảng 2: Bảng mô tả các tín hiệu.

c) Thiết kế máy trạng thái bậc cao (High level FSM) của thiết kế.

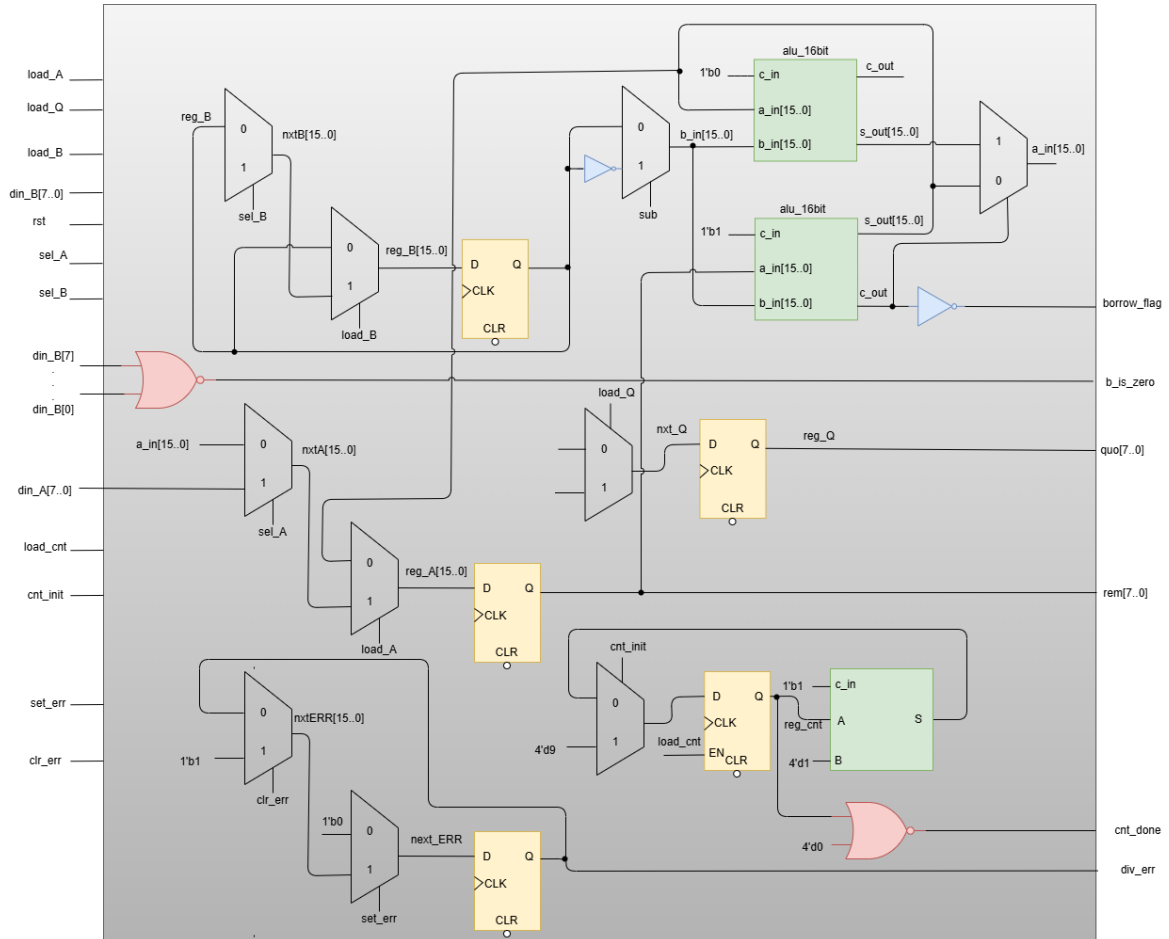


Hình 3: Thiết kế HighLevel FSM.

Trong đó:

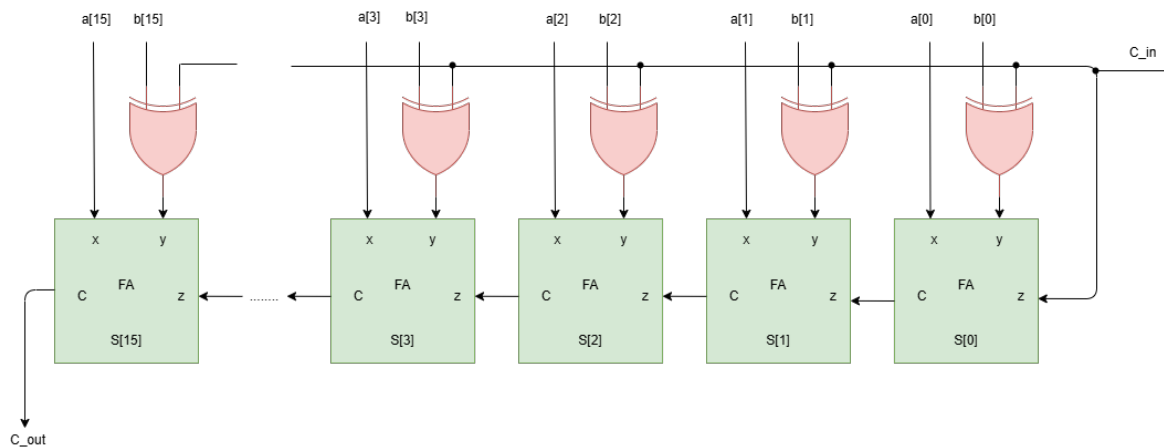
- S_IDLE: Là trạng thái chờ khi reset, khi có tín hiệu start thì module sẽ hoạt động.
- S_INIT: Trạng thái khởi tạo. Dùng để nạp dữ liệu ban đầu trước khi thực hiện tính toán và kiểm tra xem có error không.
- S_COMPARE: Trạng thái thực hiện so sánh giá trị A và B. Từ đó, quyết định `borrow_flag` để thực hiện ghi kết quả mới vào A và Q.
- S_SHIFT: Trạng thái này để Shift right B ($B \gg 1$), đồng thời giảm counter được khởi tạo ở S_INIT để chuẩn bị cho vòng lặp sau. Ngoài ra, kiểm tra xem đã thực hiện đủ vòng lặp hay chưa để quyết định chuyển đến S_COMPARE hay S_DONE.
- S_DONE: Trạng thái đã hoàn thành phép chia.

d) Thiết kế Datapath và Control Unit.



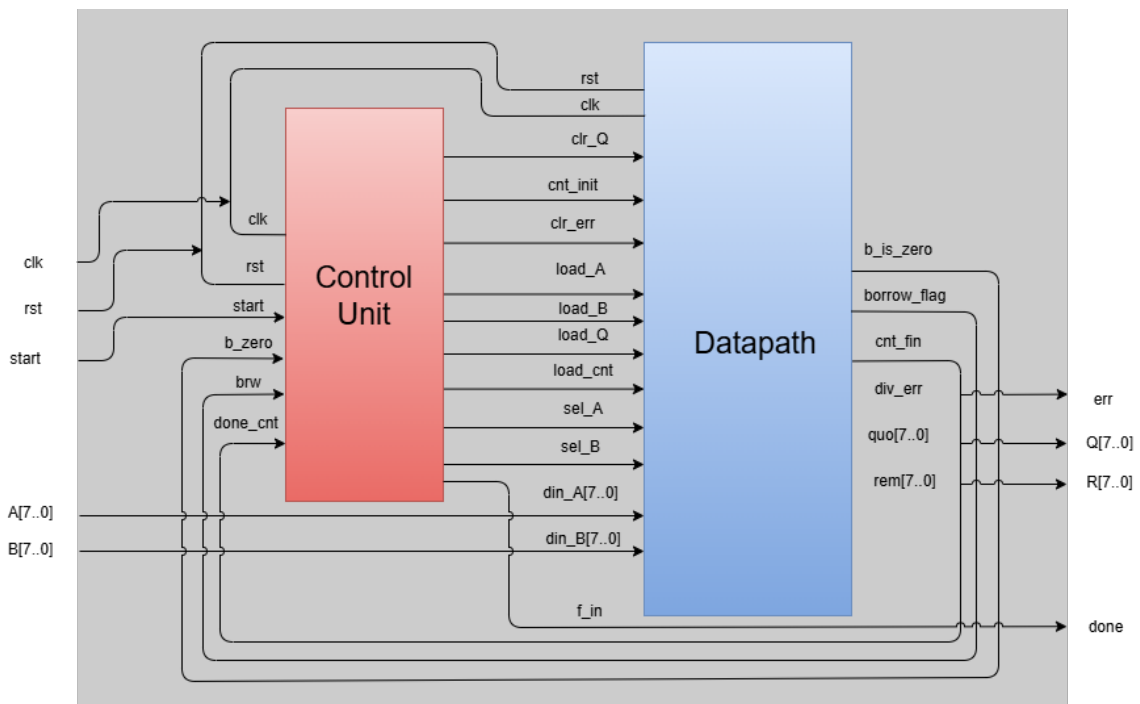
Hình 4: DataPath của bộ chia.

Trong đó, bộ alu_16 bit được thiết kế theo kiểu Ripple Carry Add/Sub 16 bit:



Hình 5: Ripple Carry Add/Sub 16 bit.

Dựa vào đó, thiết kế Control Unit điều khiển Datapath như sau:



Hình 6: Control_Unit của bộ chia.

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

Ngoài những trường hợp thực hiện phép chia bình thường, ta cần kiểm tra một vài đặc biệt để kiểm tra xem có đúng với thiết kế hay chưa.

Special cases	Q	R	error
$B = 0$	Don't care	Don't care	1
$A = 0 \ \& \ B \neq 0$	0	0	0
$A < B$	0	A	0

Bảng 3: Bảng các trường hợp đặc biệt (Special Cases).

1. Code thực hiện chương trình.

```

1  module divider8_top (
2      input  logic      clk,
3      input  logic      rst,
4      input  logic      start,
5      input  logic [7:0] A,
6      input  logic [7:0] B,
7
8      output logic      done,
9      output logic      err,
10     output logic [7:0] Q,
11     output logic [7:0] R
12 );
13
14     logic lA, sAinit;
15     logic lB, sBinit;
16     logic lQ, cQ;
17     logic lCNT, cINIT;
18     logic eSET, eCLR;
19
20     logic b_zero, brw, cnt_done;
21
22     div_ctrl ctrl(
23         .clk(clk), .rst(rst),
24         .go(start),
25         .b_zero(b_zero),
26         .brw(brw),
27         .done_cnt(cnt_done),
28         .ldA(lA), .selA_init(sAinit),
29         .ldB(lB), .selB_init(sBinit),
30         .ldQ(lQ), .clrQ(cQ),
31         .ldCNT(lCNT), .cntINIT(cINIT),
32         .errSET(eSET), .errCLR(eCLR),
33         .fin(done)
34     );
35
36     div_dp dp(
37         .clk(clk), .rst(rst),
38         .din_A(A), .din_B(B),
39         .ld_A(lA), .sel_A_init(sAinit),
40         .ld_B(lB), .sel_B_init(sBinit),
41         .ld_Q(lQ), .clr_Q(cQ),
42         .ld_cnt(lCNT), .cnt_init(cINIT),
43         .set_err(eSET),
44         .clr_err(eCLR),
45         .b_is_zero(b_zero),
46         .borrow_flag(brw),
47         .cnt_fin(cnt_done),

```

```

48         .quo(Q),
49         .rem(R),
50         .div_err(err)
51     );
52 endmodule

```

Listing 1: Module divider8_top.

```

1  module div_ctrl (
2      input  logic clk,
3      input  logic rst,
4      input  logic go,
5      input  logic b_zero,
6      input  logic brw,
7      input  logic done_cnt,
8
9      output logic ldA, selA_init,
10     output logic ldB, selB_init,
11     output logic ldQ, clrQ,
12     output logic ldCNT, cntINIT,
13     output logic errSET, errCLR,
14     output logic fin
15 );
16
17     typedef enum logic[2:0] {
18         S_IDLE = 3'd0,
19         S_INIT = 3'd1,
20         S_COMPARE = 3'd2,
21         S_SHIFT = 3'd3,
22         S_DONE = 3'd4
23     } st_t;
24
25     st_t st, nxt;
26
27     always_ff @(posedge clk or posedge rst)
28         if (rst) st <= S_IDLE;
29         else    st <= nxt;
30
31     always_comb begin
32         nxt = st;
33         ldA=0; selA_init=0;
34         ldB=0; selB_init=0;
35         ldQ=0; clrQ=0;
36         ldCNT=0; cntINIT=0;
37         errSET=0; errCLR=0;
38         fin=0;
39
40         case(st)
41
42             S_IDLE: begin
43                 clrQ = 1;
44                 errCLR = 1;
45                 if (go)
46                     nxt = S_INIT;
47             end
48
49             S_INIT: begin
50                 selA_init = 1; ldA = 1;
51                 selB_init = 1; ldB = 1;
52                 ldCNT = 1; cntINIT = 1;
53                 clrQ = 1;
54                 errCLR = 1;
55                 if (b_zero) begin

```

```

56         errSET = 1;
57         nxt = S_DONE;
58     end else begin
59         nxt = S_COMPARE;
60     end
61 end
62
63 S_COMPARE: begin
64     ldA = 1;
65     ldQ = 1;
66     nxt = S_SHIFT;
67 end
68
69 S_SHIFT: begin
70     ldB = 1;
71     ldCNT = 1;
72     if (done_cnt) nxt = S_DONE;
73     else nxt = S_COMPARE;
74 end
75
76 S_DONE: begin
77     fin = 1;
78     if (!go) nxt = S_IDLE;
79 end
80
81 endcase
82 end
83 endmodule

```

Listing 2: Module div_ctrl.

```

1  module alu16 (
2      input  logic [15:0] a_in,
3      input  logic [15:0] b_in,
4      input  logic      sub,
5      output logic [15:0] s_out,
6      output logic      carry_out
7  );
8      logic [15:0] b_sel;
9      logic      c_in;
10
11      assign b_sel = sub ? ~b_in : b_in;
12      assign c_in  = sub ? 1'b1 : 1'b0;
13
14      assign {carry_out, s_out} = a_in + b_sel + c_in;
15  endmodule

```

Listing 3: Module alu16.

```

1  module div_dp (
2      input  logic      clk,
3      input  logic      rst,
4      input  logic [7:0] din_A,
5      input  logic [7:0] din_B,
6
7
8      input  logic      ld_A,
9      input  logic      sel_A_init,
10     input  logic      ld_B,
11     input  logic      sel_B_init,
12     input  logic      ld_Q,
13     input  logic      clr_Q,

```

```

14     input logic      ld_cnt,
15     input logic      cnt_init,
16     input logic      set_err,
17     input logic      clr_err,
18
19
20     output logic      b_is_zero,
21     output logic      borrow_flag,
22     output logic      cnt_fin,
23
24
25     output logic [7:0] quo,
26     output logic [7:0] rem,
27     output logic      div_err
28 );
29
30
31     logic [15:0] regA;
32     logic [15:0] regB;
33     logic [7:0]  regQ;
34     logic [3:0]  regCNT;
35     logic        regERR;
36
37
38     eq8 chkB (.x(din_B), .zero(b_is_zero));
39
40
41     logic [15:0] alu_res;
42     logic        alu_cout;
43
44     alu16 alu1 (
45         .a_in(regA),
46         .b_in(regB),
47         .sub(1'b1),
48         .s_out(alu_res),
49         .carry_out(alu_cout)
50     );
51
52
53     assign borrow_flag = ~alu_cout;
54
55
56     logic [15:0] alu_restore;
57     alu16 alu2 (
58         .a_in(alu_res),
59         .b_in(regB),
60         .sub(1'b0),
61         .s_out(alu_restore),
62         .carry_out()
63     );
64
65
66     logic [15:0] nxtA;
67     assign nxtA =
68         sel_A_init ? {8'd0, din_A} :
69         (borrow_flag ? alu_restore : alu_res);
70
71
72     logic [15:0] nxtB;
73     assign nxtB =
74         sel_B_init ? {din_B, 8'd0} : (regB >> 1);
75
76
77     logic [7:0] nxtQ;

```

```

78     assign nxtQ = {regQ[6:0], ~borrow_flag};
79
80
81     logic [3:0] nxtCNT;
82     assign nxtCNT = regCNT - 1;
83     assign cnt_fin = (regCNT == 4'd1);
84
85
86     logic nxtERR;
87     assign nxtERR = set_err ? 1'b1 :
88                     clr_err ? 1'b0 :
89                     regERR;
90
91
92     always_ff @(posedge clk or posedge rst) begin
93         if (rst) begin
94             regA <= 0;
95             regB <= 0;
96             regQ <= 0;
97             regCNT <= 0;
98             regERR <= 0;
99         end else begin
100             if (ld_A) regA <= nxtA;
101             if (ld_B) regB <= nxtB;
102
103             if (clr_Q) regQ <= 8'd0;
104             else if (ld_Q) regQ <= nxtQ;
105
106             if (ld_cnt) begin
107                 if (cnt_init) regCNT <= 4'd9;
108                 else regCNT <= nxtCNT;
109             end
110
111             regERR <= nxtERR;
112         end
113     end
114
115     assign quo = regQ;
116     assign rem = regA[7:0];
117     assign div_err = regERR;
118
119 endmodule

```

Listing 4: Module div_dp.

```

1 module eq8 (
2     input logic [7:0] x,
3     output logic      zero
4 );
5     assign zero = (x == 8'd0);
6 endmodule

```

Listing 5: Module eq8.

- Để tạo giá trị ngẫu nhiên ($\$unrandom_range(0,255)$) trong bộ nhớ để mô phỏng.

```

# =====
#     STARTING ADVANCED DIVIDER TESTBENCH
#     =====
#

```



```
# [PASS] A=174 B=25 | Q=6 R=24
# [PASS] A=0 B=25 | Q=0 R=0
# [PASS][DIV0] A=37 B=0 detected
# [PASS][DIV0] A=0 B=0 detected
# [PASS] A=10 B=25 | Q=0 R=10
# [PASS] A=50 B=50 | Q=1 R=0
# [PASS] A=231 B=124 | Q=1 R=107
# [PASS] A=156 B=28 | Q=5 R=16
# [PASS] A=232 B=167 | Q=1 R=65
# [PASS] A=22 B=196 | Q=0 R=22
# [PASS] A=0 B=89 | Q=0 R=0
# [PASS] A=248 B=251 | Q=0 R=248
# [PASS] A=17 B=162 | Q=0 R=17
# [PASS] A=252 B=165 | Q=1 R=87
# [PASS] A=222 B=150 | Q=1 R=72
# [PASS] A=144 B=23 | Q=6 R=6
# [PASS] A=222 B=131 | Q=1 R=91
# [PASS] A=134 B=20 | Q=6 R=14
# [PASS] A=205 B=92 | Q=2 R=21
# [PASS] A=79 B=178 | Q=0 R=79
# [PASS] A=151 B=255 | Q=0 R=151
# [PASS] A=30 B=172 | Q=0 R=30
# [PASS] A=44 B=228 | Q=0 R=44
# [PASS] A=158 B=133 | Q=1 R=25
# [PASS] A=178 B=155 | Q=1 R=23
# [PASS] A=82 B=173 | Q=0 R=82
# [PASS] A=49 B=24 | Q=2 R=1
# [PASS] A=153 B=192 | Q=0 R=153
# [PASS] A=2 B=140 | Q=0 R=2
# [PASS] A=225 B=136 | Q=1 R=89
# [PASS] A=47 B=53 | Q=0 R=47
# [PASS] A=20 B=15 | Q=1 R=5
# [PASS] A=17 B=45 | Q=0 R=17
# [PASS] A=72 B=215 | Q=0 R=72
# [PASS] A=29 B=193 | Q=0 R=29
# [PASS] A=245 B=152 | Q=1 R=93
# [PASS] A=130 B=130 | Q=1 R=0
# [PASS] A=113 B=193 | Q=0 R=113
# [PASS] A=21 B=83 | Q=0 R=21
# [PASS] A=126 B=143 | Q=0 R=126
# [PASS] A=57 B=94 | Q=0 R=57
# [PASS] A=56 B=171 | Q=0 R=56
# [PASS] A=161 B=122 | Q=1 R=39
# [PASS] A=24 B=129 | Q=0 R=24
# [PASS] A=184 B=129 | Q=1 R=55
# [PASS] A=225 B=248 | Q=0 R=225
# [PASS] A=162 B=47 | Q=3 R=21
```

```
# [PASS] A=13 B=151 | Q=0 R=13
# [PASS] A=56 B=67 | Q=0 R=56
# [PASS] A=193 B=77 | Q=2 R=39
# [PASS] A=26 B=117 | Q=0 R=26
# [PASS] A=201 B=214 | Q=0 R=201
# [PASS] A=211 B=1 | Q=211 R=0
# [PASS] A=214 B=137 | Q=1 R=77
# [PASS] A=42 B=191 | Q=0 R=42
# [PASS] A=224 B=15 | Q=14 R=14
# [PASS] A=165 B=212 | Q=0 R=165
# [PASS] A=63 B=163 | Q=0 R=63
# [PASS] A=126 B=89 | Q=1 R=37
# [PASS] A=125 B=78 | Q=1 R=47
# [PASS] A=24 B=186 | Q=0 R=24
# [PASS] A=226 B=219 | Q=1 R=7
# [PASS] A=51 B=19 | Q=2 R=13
# [PASS] A=252 B=113 | Q=2 R=26
# [PASS] A=189 B=130 | Q=1 R=59
# [PASS] A=137 B=41 | Q=3 R=14
# [PASS] A=129 B=94 | Q=1 R=35
# [PASS] A=173 B=87 | Q=1 R=86
# [PASS] A=148 B=168 | Q=0 R=148
# [PASS] A=125 B=116 | Q=1 R=9
# [PASS] A=168 B=74 | Q=2 R=20
# [PASS] A=175 B=89 | Q=1 R=86
# [PASS] A=77 B=161 | Q=0 R=77
# [PASS] A=30 B=28 | Q=1 R=2
# [PASS] A=254 B=72 | Q=3 R=38
# [PASS] A=37 B=106 | Q=0 R=37
# [PASS] A=73 B=206 | Q=0 R=73
# [PASS] A=0 B=51 | Q=0 R=0
# [PASS] A=195 B=254 | Q=0 R=195
# [PASS] A=94 B=187 | Q=0 R=94
# [PASS] A=27 B=138 | Q=0 R=27
# [PASS] A=65 B=59 | Q=1 R=6
# [PASS] A=237 B=8 | Q=29 R=5
# [PASS] A=121 B=33 | Q=3 R=22
# [PASS] A=196 B=126 | Q=1 R=70
# [PASS] A=172 B=147 | Q=1 R=25
# [PASS] A=178 B=136 | Q=1 R=42
# [PASS] A=143 B=6 | Q=23 R=5
# [PASS] A=3 B=214 | Q=0 R=3
# [PASS] A=181 B=69 | Q=2 R=43
# [PASS] A=252 B=233 | Q=1 R=19
# [PASS] A=176 B=255 | Q=0 R=176
# [PASS] A=1 B=189 | Q=0 R=1
# [PASS] A=170 B=32 | Q=5 R=10
```

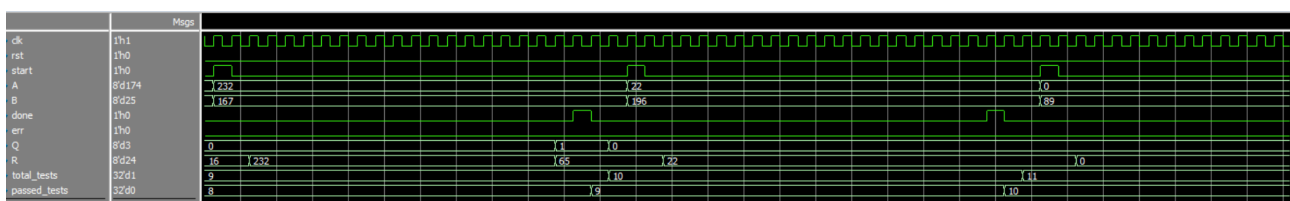
```

# [PASS] A=11 B=217 | Q=0 R=11
# [PASS] A=210 B=45 | Q=4 R=30
# [PASS] A=133 B=79 | Q=1 R=54
# [PASS] A=248 B=172 | Q=1 R=76
# [PASS][DIV0] A=196 B=0 detected
# [PASS] A=120 B=116 | Q=1 R=4
# [PASS] A=61 B=56 | Q=1 R=5
# [PASS] A=89 B=128 | Q=0 R=89
# [PASS] A=203 B=5 | Q=40 R=3
# [PASS] A=85 B=224 | Q=0 R=85
# [PASS] A=53 B=253 | Q=0 R=53
# [PASS] A=175 B=44 | Q=3 R=43
#
# =====
# SUMMARY
# Total : 106
# Passed : 106
# Failed : 0
# =====
# ALL TESTS PASSED
# ** Note: $finish : D:/Div/top/testbench.sv(134)
# Time: 23935 ns Iteration: 1 Instance: /tb_divider
# 1
# Break in Module tb_divider at D:/Div/top/testbench.sv line 134

```

Listing 6: Kết quả test của module thực hiện phép chia.

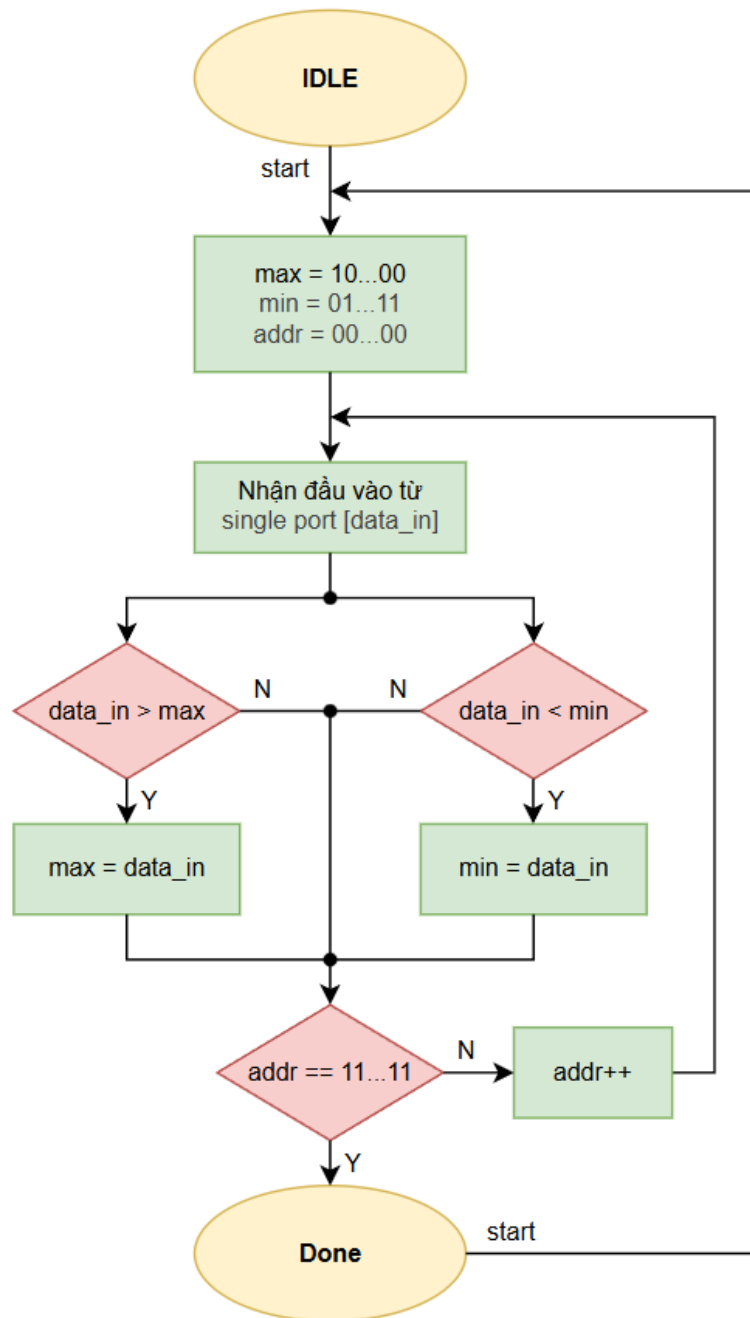
3. Waveform và kết quả chạy testbench.



Hình 7: Dạng sóng của testbench mô phỏng khối thực hiện phép chia.

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 8: 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).

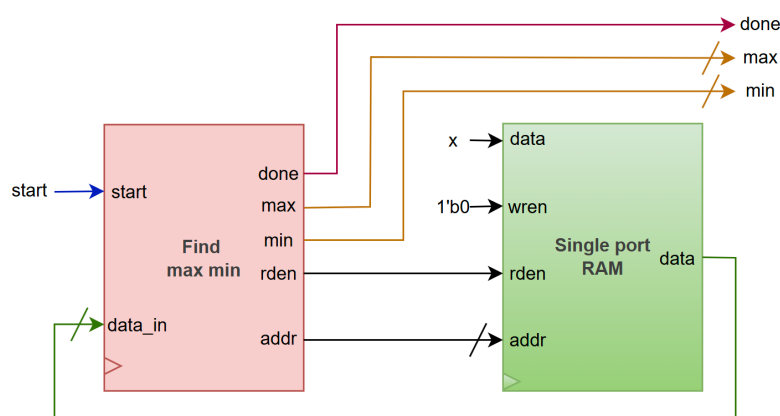
Đề bài không đề cập đến xử lý cho dạng dữ liệu có dấu hoặc không dấu nên nhóm chọn xử lý ở dạng có dấu bù 2.

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

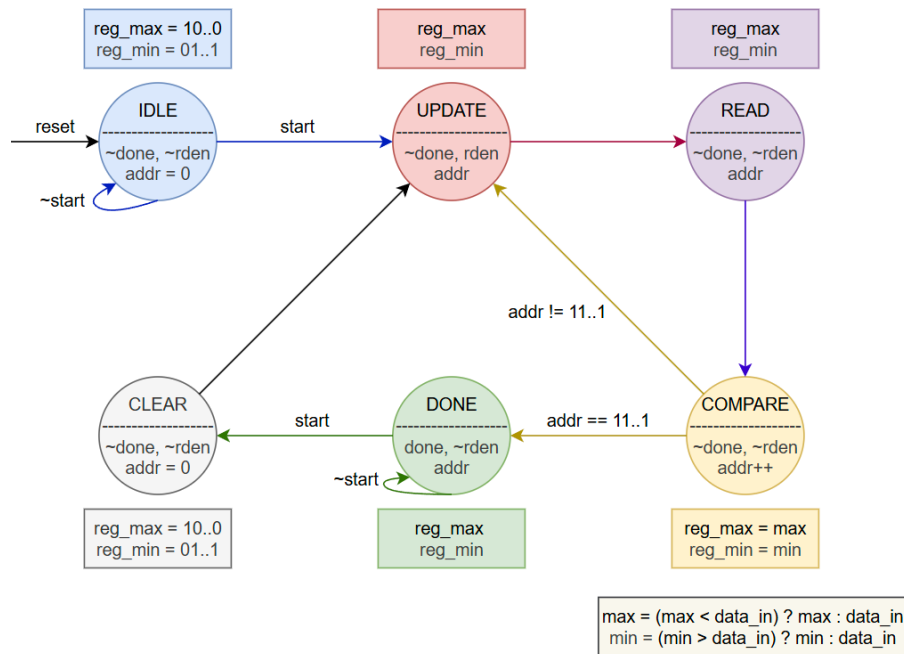
Bảng 4: Bảng tín hiệu I/O của module `max_min`

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 9: Tổng quan kết nối thiết kế.

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



Hình 10: Máy trạng thái bậc cao.

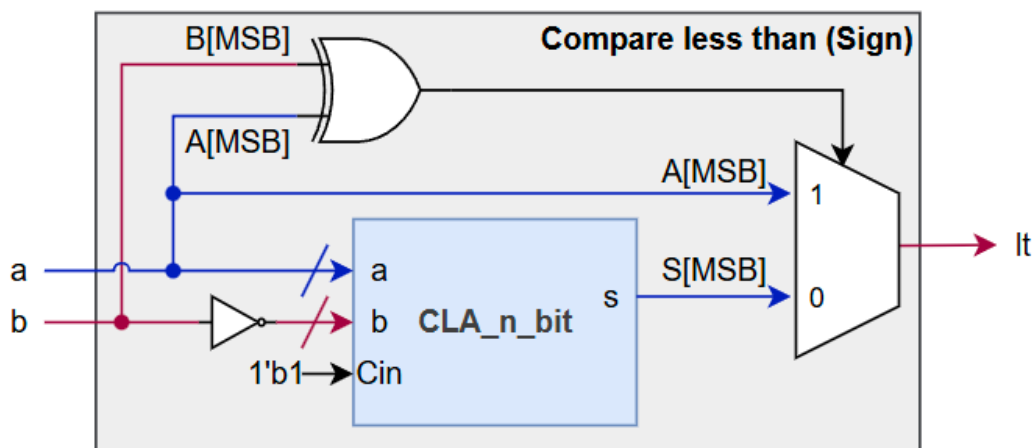
Thiết kế gồm 6 trạng thái chính:

- **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 tính 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 kỳ 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.

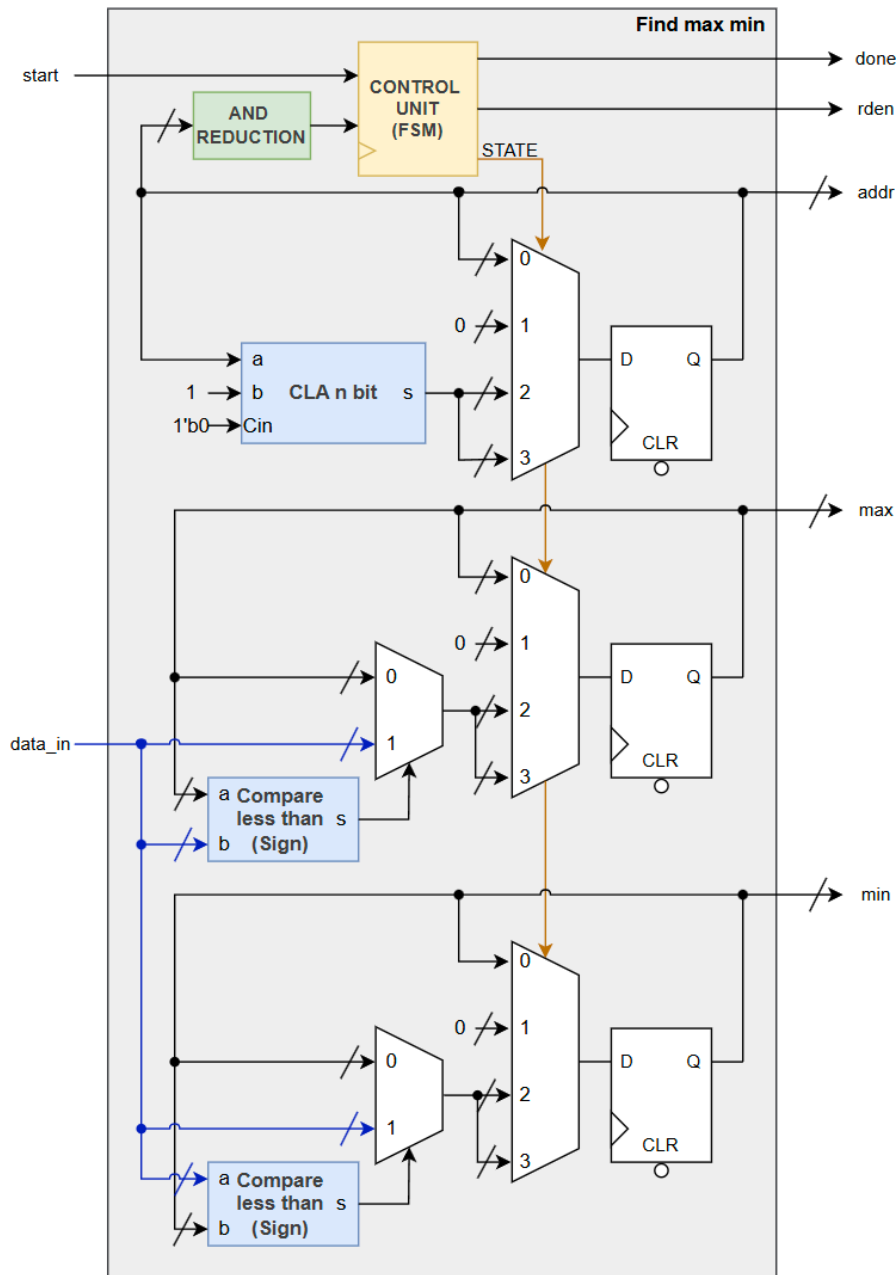
c) Thiết kế Datapath và Control Unit của thiết kế.

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 11: 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 12: 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:

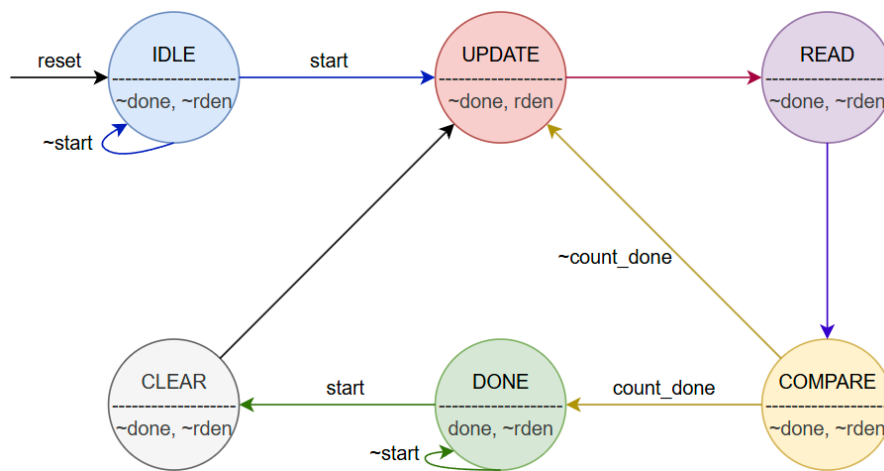
- Thanh ghi địa chỉ, đóng vai trò trở vào bộ nhớ, kết hợp với tín hiệu rden để lấy dữ liệu đầu vào.

- 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 13: Control Unit FSM.

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

```

1  module lt_compare_sign #(
2      parameter WIDTH = 8
3  )(
4      input  logic [WIDTH - 1:0] i_a ,
5      input  logic [WIDTH - 1:0] i_b ,
6      output logic               o_lt
7  );
8
9      logic [WIDTH - 1:0] s;
10
11     adder_flex_no_carry #(
12         .WIDTH(WIDTH)
13     ) compare_max (
14         .i_a   (i_a),
15         .i_b   (~i_b),
16         .i_cin (1'b1),
17         .o_s   (s)
18     );
19
20     assign o_lt = (i_a[WIDTH-1] ^ i_b[WIDTH-1]) ? i_a[WIDTH-1] : s[WIDTH-1];

```

```

21
22 endmodule

```

Listing 7: HDL mô tả bộ so sánh bé hơn có dấu.

```

1  module max_min #(
2      parameter DEPTH = 128,
3      parameter WIDTH = 8
4  )(
5      input logic i_clk,
6      input logic i_rst_n,
7      input logic i_start,
8      output logic o_rden,
9      output logic [$clog2(DEPTH) - 1:0] o_addr,
10     input logic signed [WIDTH - 1:0] i_data,
11     output logic o_done,
12     output logic signed [WIDTH - 1:0] o_max,
13     output logic signed [WIDTH - 1:0] o_min
14 );
15
16     localparam DEPTH_S = DEPTH - 1;
17     localparam WIDTH_S = WIDTH - 1;
18
19     logic [WIDTH - 1:0] max, min;
20     logic flag_max, flag_min;
21
22     typedef enum logic[2:0] {
23         IDLE,
24         READ,
25         COMPARE,
26         UPDATE,
27         DONE,
28         CLEAR
29     } e_state;
30
31     e_state pstate, nstate;
32     logic [$clog2(DEPTH) - 1:0] reg_count, count, count_add;
33
34     always_ff @(posedge i_clk, negedge i_rst_n) begin : blockName
35         if(~i_rst_n) pstate <= IDLE;
36         else pstate <= nstate;
37     end
38
39     always_comb begin
40         case(pstate)
41             IDLE : nstate = i_start ? UPDATE : pstate;
42             UPDATE : nstate = (&reg_count) ? DONE : READ;
43             READ : nstate = COMPARE;
44             COMPARE : nstate = UPDATE;
45             DONE : nstate = i_start ? CLEAR : pstate;
46             CLEAR : nstate = UPDATE;
47             default: nstate = IDLE;
48         endcase
49     end
50
51     assign count = (pstate == COMPARE) ? count_add : ((pstate == CLEAR) ? '0 : reg_count);
52     assign o_addr = count;
53
54     assign o_done = (pstate == DONE);
55     //assign o_rden = (pstate == UPDATE) | (pstate == IDLE);
56     assign o_rden = (pstate == UPDATE);
57
58     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'
81 b0}}} : o_max);
82
83 assign min = (pstate == COMPARE) ? (flag_min ? i_data : o_min) : ((pstate == CLEAR) ? {1'b0, {WIDTH_S{1'
84 b1}}} : o_min);
85
86 lt_compare_sign #(
87     .WIDTH(WIDTH)
88 ) compare_max (
89     .i_a (o_max),
90     .i_b (i_data),
91     .o_lt(flag_max) // a < b
92 );
93
94 lt_compare_sign #(
95     .WIDTH(WIDTH)
96 ) compare_min (
97     .i_a (i_data),
98     .i_b (o_min),
99     .o_lt(flag_min) // a < b
100 );
101
102 endmodule

```

Listing 8: 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] = $unrandom_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 9: Chương trình tạo giá trị ngẫu nhiên ban đầu cho bộ nhớ.

```

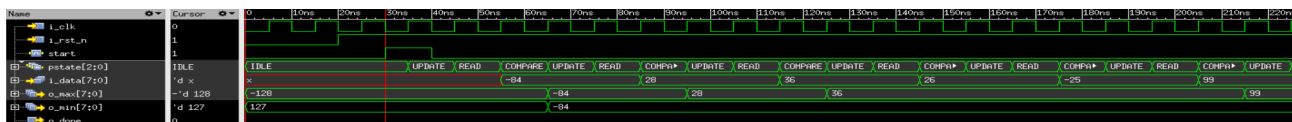
1 always @(posedge clk) begin
2     if (done) begin
3         $display("DUT Min = %0d, DUT Max = %0d", dut_min, dut_max);
4
5         if (dut_min === golden_min && dut_max === golden_max)
6             $display(">>> TEST PASS <<<");
7         else begin
8             $display(">>> TEST FAIL <<<");
9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

```

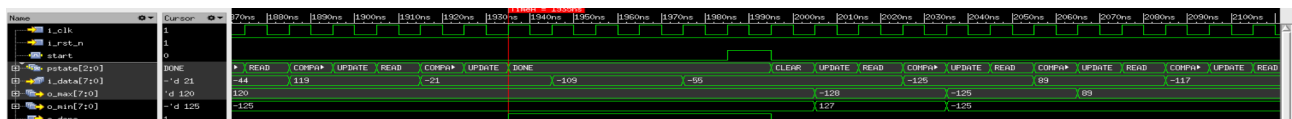
Listing 10: Chương trình kiểm định thiết kế.

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



Hình 14: Dạng sóng lúc bắt đầu.



Hình 15: Dạng sóng lúc hoàn thành.

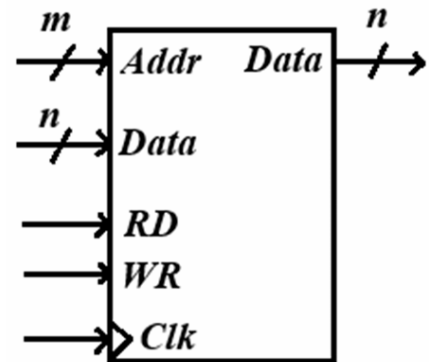
Kết luận: Máy trạng thái chuyển đúng so với dự tính, giá trị kết quả so sánh chính xác.

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 12: Đoạn chương trình C của giải thuật Selection Sort.

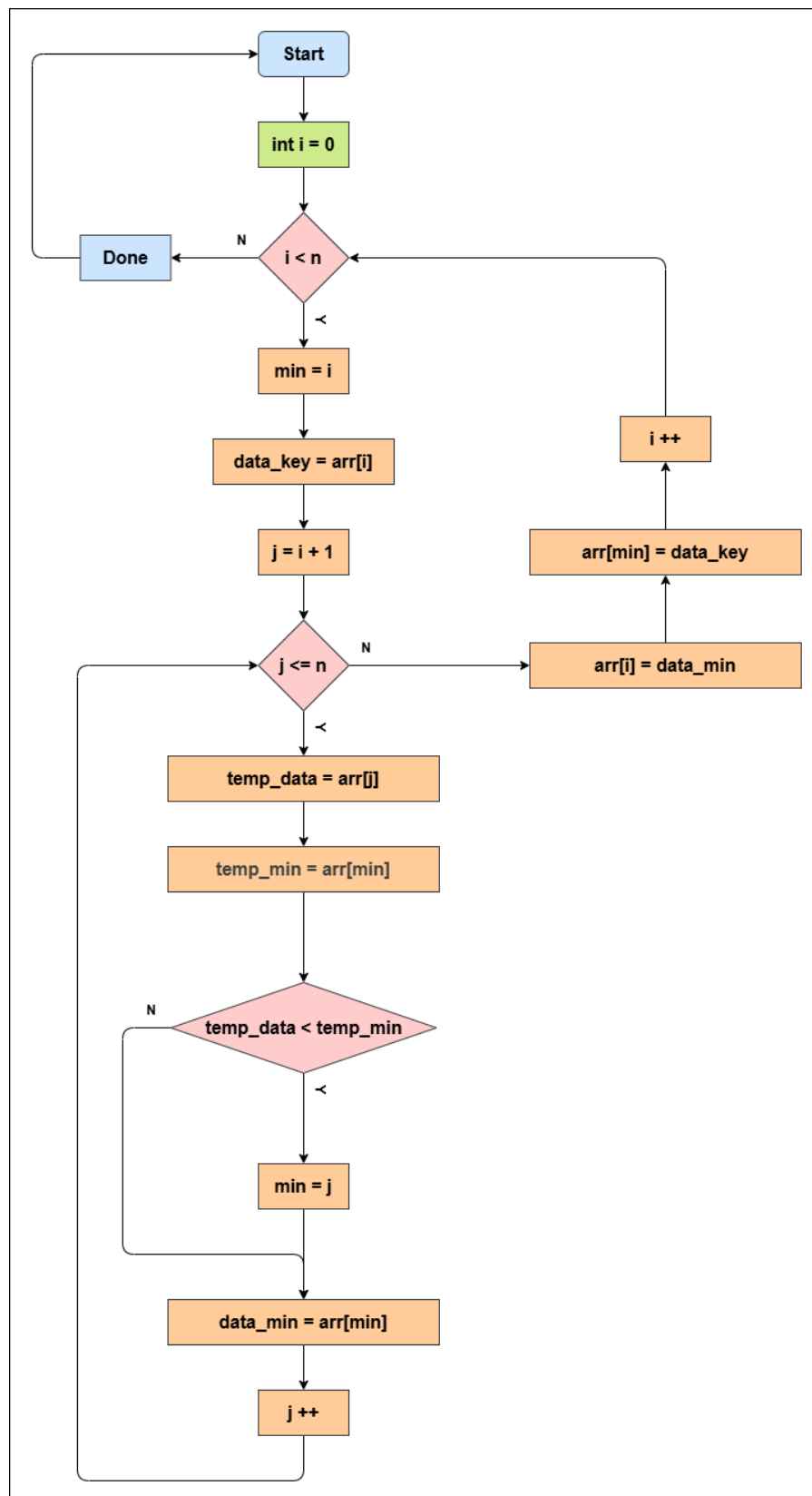


Hình 16: Yêu cầu của bộ nhớ.

Người ta muốn chuyển đổi giải thuật ở đoạn code 12 trên sang phần cứng để thực thi. Giả sử mảng được lưu trong bộ nhớ *Single Port* ở hình 16 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 12 trên dưới dạng flowchart như sau:



Hình 17: Flowchart của thuật toán Selection Sort.

Từ hình 17, 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 gốc ở chương trình 12.

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

Listing 13: Đoạn code nguyên mẫu của giải thuật Selection Sort.

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

Listing 14: Đoạn code chỉnh sửa của giải thuật Selection Sort.

Kết quả cho ra là:

```
Finished reading file: ./tools/unordered.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'.
```

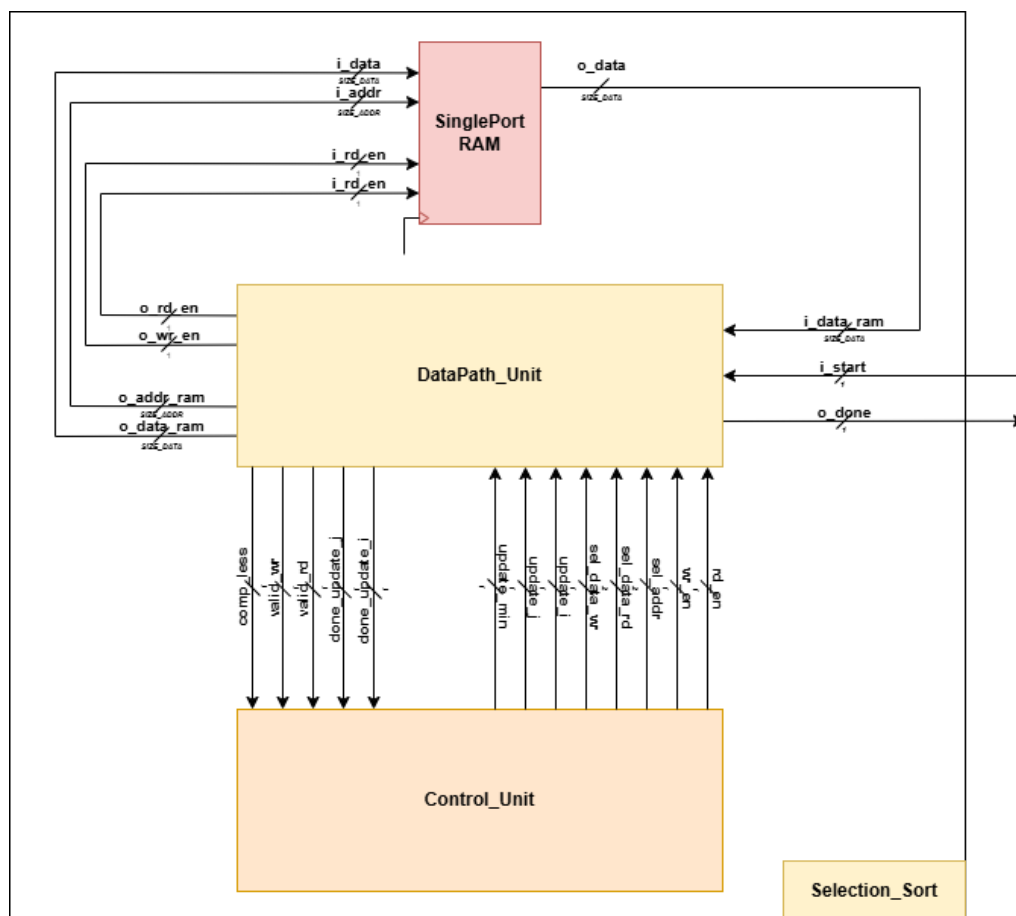
Listing 15: Kết quả so sánh 2 cách viết của Selection Sort.

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

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

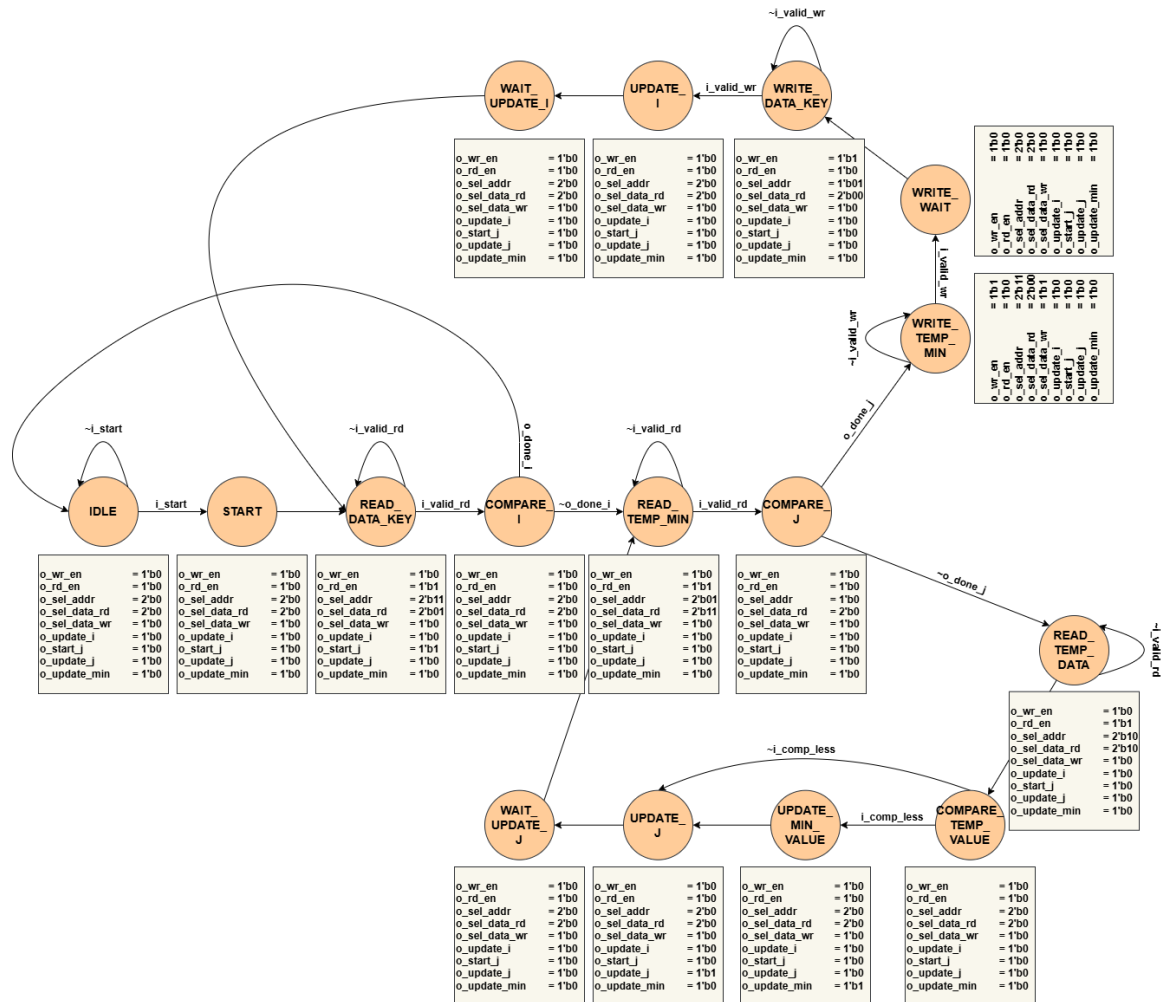
Bảng 5: Bảng I/O của module **Selection_Sort**.

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



Hình 18: Thiết kế tổng quan của module **Selection_Sort**.

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



Hình 19: Lưu đồ trạng thái của module `Selection_Sort`.

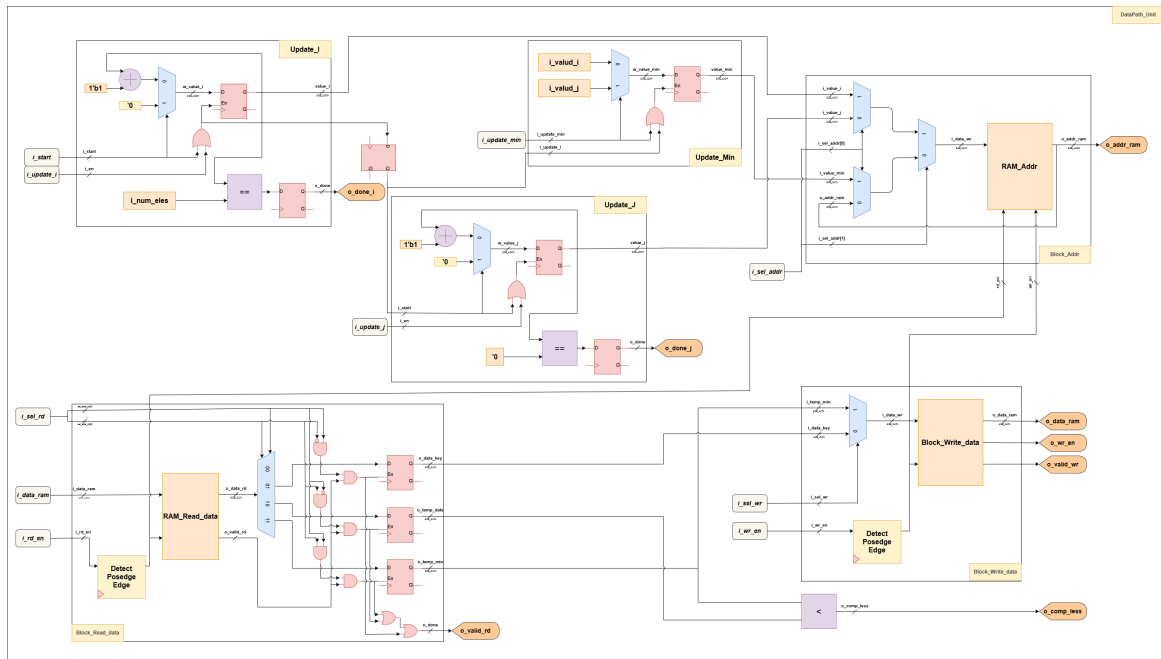
FSM gồm có 16 trạng thái như sau:

- 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`.

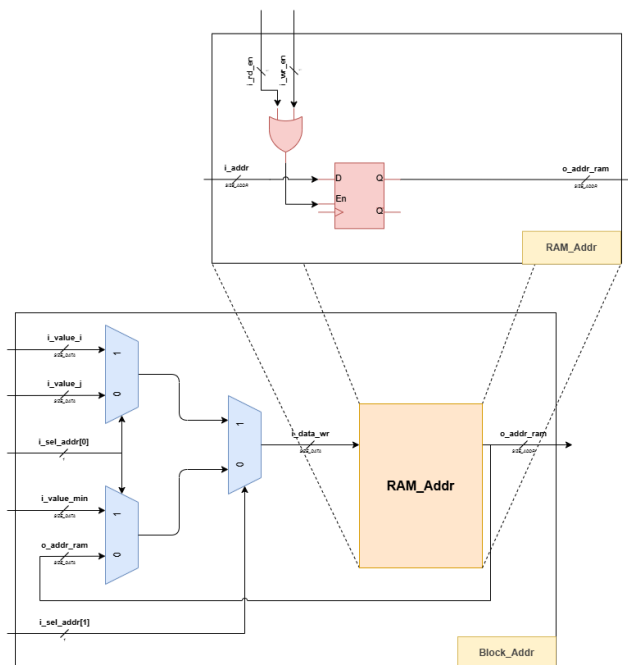
c) Thiết kế Datapath và Control Unit của thiết kế.

1. Thiết kế DataPath:

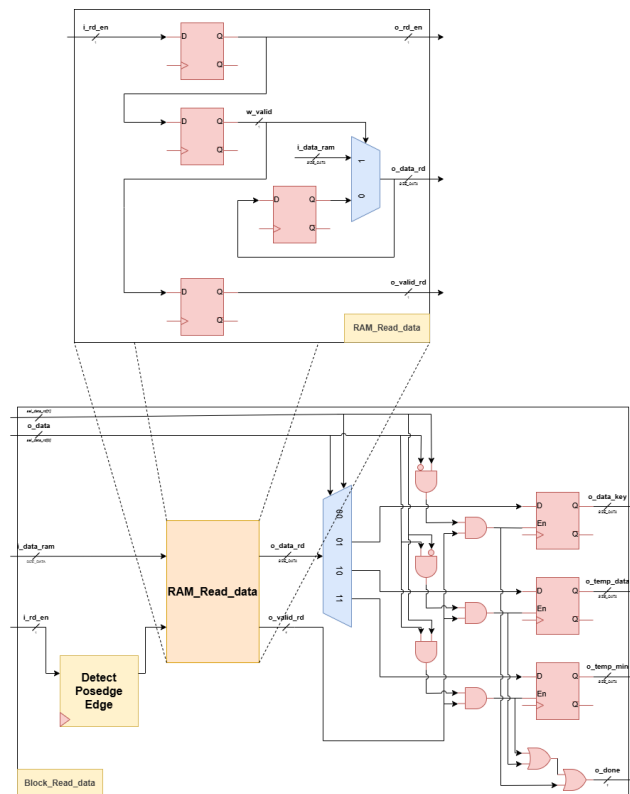


Hình 20: Thiết kế tổng quan của module DataPath_Unit.

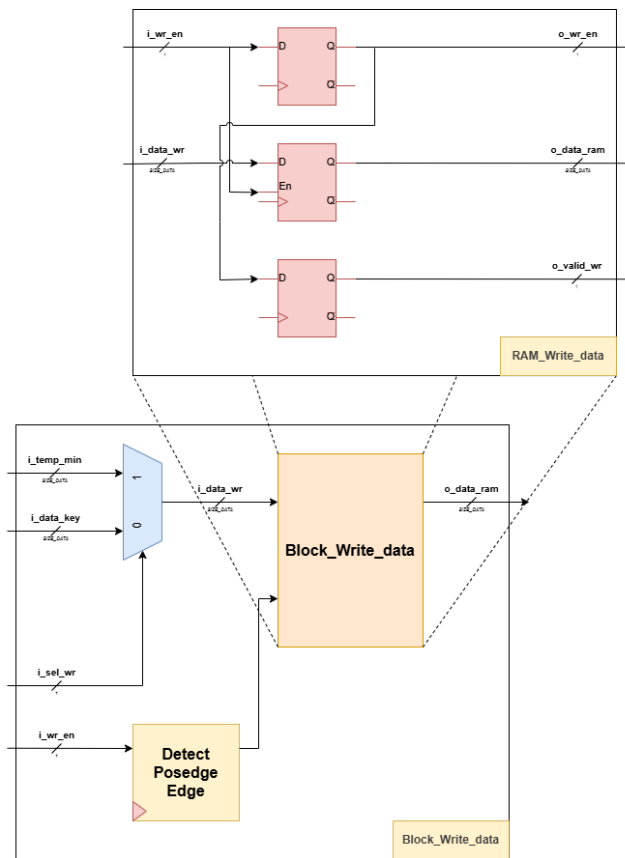
- RAM_Addr:



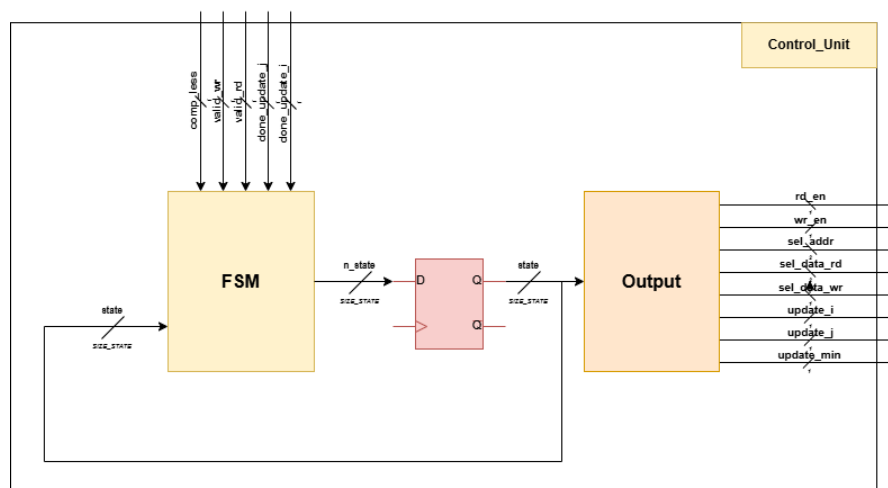
- RAM_Read_data:



- 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:

```

1  module Selection_Sort #(
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_start        ,
9      input logic [SIZE_ADDR-1:0] i_num_elems ,
10     output logic              o_done
11 );
12
13 logic w_rd_en;
14 logic w_wr_en;
15 logic [1:0] w_sel_addr;
16 logic [1:0] w_sel_data_rd;
17 logic      w_sel_data_wr;
18 logic w_update_i;
19 logic w_update_j;
20 logic w_update_min;
21
22 logic w_comp_less;
23 logic w_valid_wr;
24 logic w_valid_rd;
25 logic w_done_update_j;
26 logic w_done_update_i;
27
28 logic w_rd_en_ram;
29 logic w_wr_en_ram;
30 logic [SIZE_ADDR-1:0] w_addr_ram;
31 logic [SIZE_DATA-1:0] w_i_data_ram;
32 logic [SIZE_DATA-1:0] w_o_data_ram;
33
34 Data_path #(
35     .SIZE_ADDR      (SIZE_ADDR),
36     .SIZE_DATA      (SIZE_DATA)
37 ) DATA_PATH_UNIT (
38     .i_clk          (i_clk),
39     .i_rst_n        (i_rst_n),
40     .i_num_elems    (i_num_elems),
41     .i_rd_en        (w_rd_en),
42     .i_wr_en        (w_wr_en),
43     .i_sel_addr     (w_sel_addr),
44     .i_sel_data_rd   (w_sel_data_rd),
45     .i_sel_data_wr   (w_sel_data_wr),
46     .i_start_i      (i_start), //i_state=>topmodule
47     .i_start_j      (),
48     .i_update_i      (w_update_i),
49     .i_update_j      (w_update_j),
50     .i_update_min    (w_update_min),
51     .i_data_ram      (w_o_data_ram),
52     .o_wr_en        (w_wr_en_ram),
53     .o_rd_en        (w_rd_en_ram),
54     .o_addr_ram      (w_addr_ram),
55     .o_data_ram      (w_i_data_ram),
56     .o_comp_less     (w_comp_less),
57     .o_valid_wr      (w_valid_wr),
58     .o_valid_rd      (w_valid_rd),
59     .o_done_j        (w_done_update_j),
60     .o_done_sort     (w_done_update_i)
61 );
62
63 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 16: 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 17: 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 18: 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

```

```

31 );
32
33 logic [SIZE_ADDR-1:0] value_i;
34 logic                w_o_en_i;
35 logic [SIZE_ADDR-1:0] value_j;
36 logic                w_i_en_j;
37 logic [SIZE_ADDR-1:0] value_min;
38
39 logic                w_en_temp_min;
40 logic [SIZE_DATA-1:0] w_temp_min;
41 logic                w_en_temp_data;
42 logic [SIZE_DATA-1:0] w_temp_data;
43 logic                w_en_data_key;
44 logic [SIZE_DATA-1:0] w_data_key;
45
46 logic [SIZE_ADDR-1:0] w_addr_ram;
47 logic [SIZE_DATA-1:0] w_data_wr;
48 logic [SIZE_DATA-1:0] w_data_rd;
49
50 assign o_comp_less = w_temp_data < w_temp_min;
51
52 Update_I #(
53     .SIZE_ADDR      (SIZE_ADDR)
54 ) UPDATE_I_UNIT (
55     .i_clk           (i_clk),
56     .i_rst_n         (i_rst_n),
57     .i_num_elems     (i_num_elems),
58     .i_start         (i_start_i),
59     .i_en            (i_update_i),
60     .o_en            (w_o_en_i),
61     .o_value_i       (value_i),
62     .o_done          (o_done_sort)
63 );
64
65 always_ff @( posedge i_clk or negedge i_rst_n) begin
66     if(~i_rst_n)
67         w_i_en_j    <= '0;
68     else
69         w_i_en_j    <= w_o_en_i;
70 end
71
72 Update_J #(
73     .SIZE_ADDR      (SIZE_ADDR)
74 ) UPDATE_J_UNIT (
75     .i_clk           (i_clk),
76     .i_rst_n         (i_rst_n),
77     .i_num_elems     (i_num_elems),
78     .i_value_i       (value_i),
79     .i_start         (w_i_en_j),
80     .i_en            (i_update_j),
81     .o_en            (),
82     .o_value_j       (value_j),
83     .o_done          (o_done_j)
84 );
85
86 Update_MIN #(
87     .SIZE_ADDR      (SIZE_ADDR)
88 ) UPDATE_MIN_UNIT (
89     .i_clk           (i_clk),
90     .i_rst_n         (i_rst_n),
91     .i_value_i       (value_i),
92     .i_value_j       (value_j),
93     .i_update_i      (w_i_en_j),
94     .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 19: 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 20: Module Block_Addr.

```

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

```

```

45
46 RAM_read_data #(
47     .SIZE_DATA      (SIZE_DATA)
48 ) RAM_READ_DATAAA (
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 21: 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 ,

```

```

14         output logic                                o_done
15     );
16
17     logic w_i_wr_en;
18     logic [SIZE_DATA-1:0] w_i_data_ram;
19     SS_detect_edge #(
20         .POS_EDGE    (1)    // 1: posedge, 0: negedge
21     ) DETECT_EDGE (
22         .i_clk        (i_clk),
23         .i_rst_n      (i_rst_n),
24         .i_signal     (i_wr_en),
25         .o_signal     (w_i_wr_en)
26     );
27
28     assign w_i_data_ram = i_sel_wr ? i_temp_min : i_data_key;
29
30     RAM_write_data #(
31         .SIZE_DATA    (SIZE_DATA)
32     ) RAM_WRITE_DATA (
33         .i_clk        (i_clk),
34         .i_rst_n      (i_rst_n),
35         .i_wr_en      (w_i_wr_en),
36         .i_data_wr    (w_i_data_ram),
37         .o_wr_en      (o_wr_en),
38         .o_data_wr    (o_data_ram),
39         .o_done       (o_done)
40     );
41
42     endmodule

```

Listing 22: Module Block_Write_data.

```

1  module RAM_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 [SIZE_ADDR-1:0] i_addr_ram  ,
9      output logic [SIZE_ADDR-1:0] o_addr_ram
10 );
11 logic w_en;
12 assign w_en = i_rd_en | i_wr_en;
13 always_ff @( posedge i_clk or negedge i_rst_n ) begin
14     if(~i_rst_n) begin
15         o_addr_ram    <= '0;
16     end else if(w_en) begin
17         o_addr_ram    <= i_addr_ram;
18     end
19 end
20
21 endmodule

```

Listing 23: Module RAM_addr.

```

1  module RAM_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 [SIZE_DATA-1:0]    i_data_rd    ,
8      output logic                   o_rd_en      ,
9      output logic [SIZE_DATA-1:0]    o_data_rd    ,
10     output logic                   o_valid
11 );
12
13 logic w_valid;
14 always_ff @( posedge i_clk or negedge i_rst_n ) begin
15     if(~i_rst_n) begin
16         o_rd_en    <= '0;
17     end else begin
18         o_rd_en    <= i_rd_en;
19     end
20 end
21 always_ff @( posedge i_clk or negedge i_rst_n ) begin
22     if(~i_rst_n) begin
23         w_valid    <= '0;
24         o_valid    <= '0;
25     end else begin
26         w_valid    <= o_rd_en;
27         o_valid    <= w_valid;
28     end
29 end
30 logic [SIZE_DATA-1:0] w_save_data;
31 always_ff @( posedge i_clk or negedge i_rst_n ) begin
32     if(~i_rst_n) begin
33         w_save_data <= '0;
34     end else begin
35         w_save_data <= o_data_rd;
36     end
37 end
38 assign o_data_rd = (w_valid) ? i_data_rd : w_save_data;
39 endmodule

```

Listing 24: Module RAM_read_data.

```

1 module RAM_write_data #(
2     parameter SIZE_DATA = 8
3 )
4 (
5     input logic          i_clk      ,
6     input logic          i_rst_n    ,
7     input logic          i_wr_en    ,
8     input logic [SIZE_DATA-1:0] i_data_wr ,
9     output logic         o_wr_en    ,
10    output logic [SIZE_DATA-1:0] o_data_wr ,
11    output logic         o_done
12 );
13
14 always_ff @( posedge i_clk or negedge i_rst_n ) begin
15     if(~i_rst_n) begin
16         o_wr_en    <= '0;
17     end else begin
18         o_wr_en    <= i_wr_en;
19     end
20 end
21 always_ff @( posedge i_clk or negedge i_rst_n ) begin
22     if(~i_rst_n) begin
23         o_done    <= '0;
24     end else begin
25         o_done    <= o_wr_en;
26     end
27 end
28 always_ff @( posedge i_clk or negedge i_rst_n ) begin

```

```

28     if (~i_rst_n) begin
29         o_data_wr <= '0;
30     end else if (i_wr_en) begin
31         o_data_wr <= i_data_wr;
32     end
33 end
34 endmodule

```

Listing 25: Module RAM_write_data.

```

1  // POS_EDGE = 0 -> detect negedge edge
2  // POS_EDGE = 1 -> detect posedge edge
3  module SS_detect_edge #(
4      parameter POS_EDGE = 1    // 1: posedge, 0: negedge
5  )(
6      input  logic i_clk,
7      input  logic i_rst_n,
8      input  logic i_signal,
9      output logic o_signal
10 );
11
12 logic w_p_signal, w_n_signal;
13
14 always_ff @(posedge i_clk or negedge i_rst_n) begin
15     if (~i_rst_n)
16         w_p_signal <= 1'b0;
17     else
18         w_p_signal <= i_signal;
19 end
20
21 always_ff @(posedge i_clk or negedge i_rst_n) begin
22     if (~i_rst_n)
23         w_n_signal <= 1'b0;
24     else
25         w_n_signal <= w_p_signal;
26 end
27
28 generate
29     if (POS_EDGE) begin
30         assign o_signal = (~w_n_signal) & (w_p_signal);
31     end else begin
32         assign o_signal = (w_n_signal) & (~w_p_signal);
33     end
34 endgenerate
35
36 endmodule

```

Listing 26: Module SS_detect_edge.

```

1  module Update_I #(
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_num_elems ,
7
8      input logic          i_start        ,
9      input logic          i_en           ,
10
11     output logic          o_en           ,
12     output logic [SIZE_ADDR-1:0] o_value_i ,
13     output logic          o_done

```

```

14 );
15
16 logic w_start;
17 logic [SIZE_ADDR-1:0] w_pre_value_i;
18 logic [SIZE_ADDR-1:0] w_next_value_i;
19 logic w_pre_done;
20 logic w_enable;
21
22 assign w_start = i_start;
23 assign w_enable = w_start | (i_en & ~w_pre_done);
24 assign w_next_value_i = o_value_i + 1'b1;
25 assign w_pre_value_i = w_start ? ('0) : (w_next_value_i);
26
27 always_ff @( posedge i_clk or negedge i_rst_n ) begin : proc_output_value_i
28     if(~i_rst_n) begin
29         o_value_i <= '0;
30     end else if(w_enable) begin
31         o_value_i <= w_pre_value_i;
32     end
33 end
34
35 // assign w_pre_done = (w_pre_value_i == (i_num_elems));
36 assign w_pre_done = (o_value_i == (i_num_elems));
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 27: Module Update_I.

```

1 module Update_J #(
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_num_elems ,
7     input logic [SIZE_ADDR-1:0] i_value_i ,
8
9     input logic i_start ,
10    input logic i_en ,
11
12    output logic o_en ,
13    output logic [SIZE_ADDR-1:0] o_value_j ,
14    output logic o_done
15 );
16
17 logic w_start;
18 logic [SIZE_ADDR-1:0] w_pre_value_j;
19 logic [SIZE_ADDR-1:0] w_next_value_j;
20 logic w_pre_done;
21 logic w_enable;
22
23 assign w_pre_done = (o_value_j == ('0));
24 assign w_start = i_start;
25 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 28: 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 29: 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

```

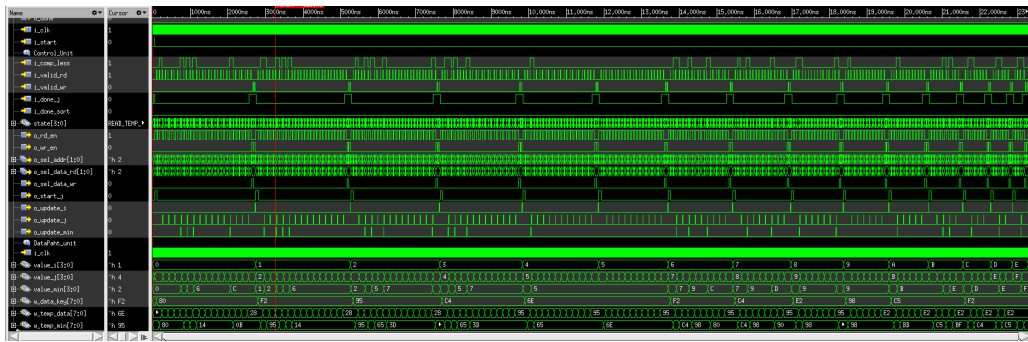
```

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)

```

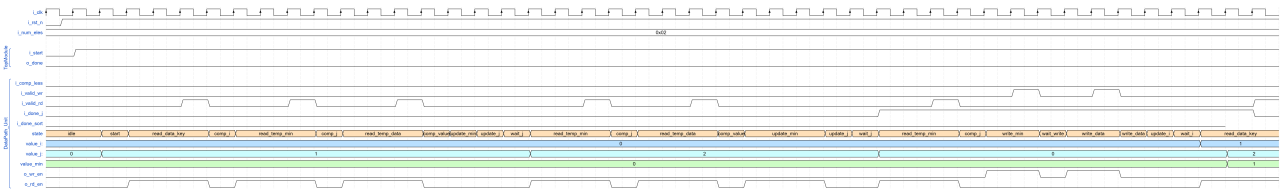
Listing 30: Kết quả sau khi kiểm định lại chức năng của module.



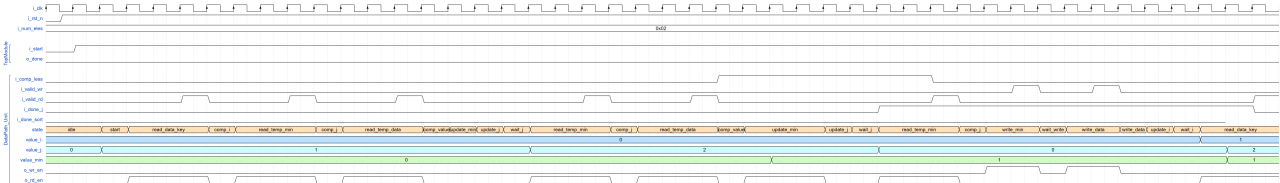
Hình 21: Sóng ngõ ra tổng quan của module test.

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)

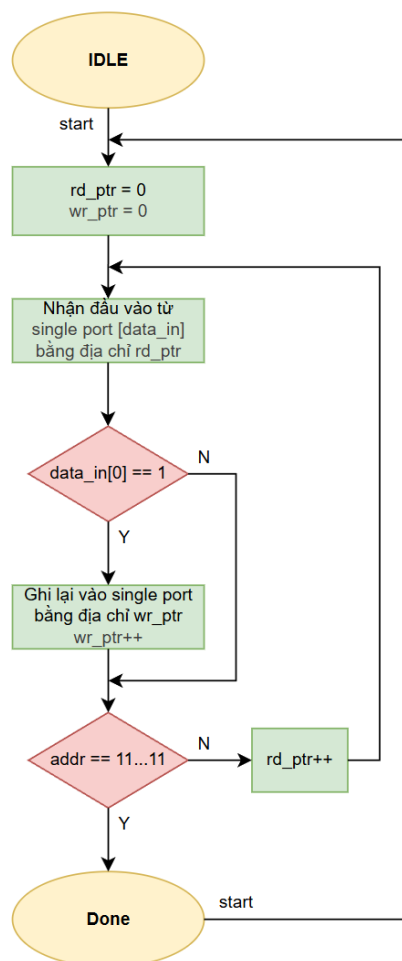
Địa chỉ	Nội dung
0x04	15
0x03	14
0x02	12
0x01	33
0x00	24

Trước khi thực hiện

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

Sau khi thực hiện

Hình 22: Ví dụ.



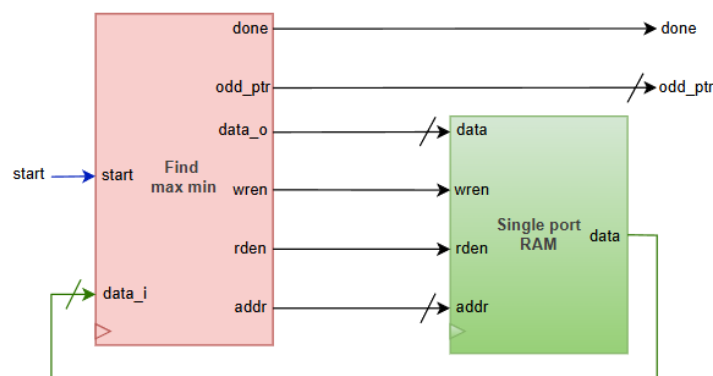
Hình 23: Giải thuật sử dụng.

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

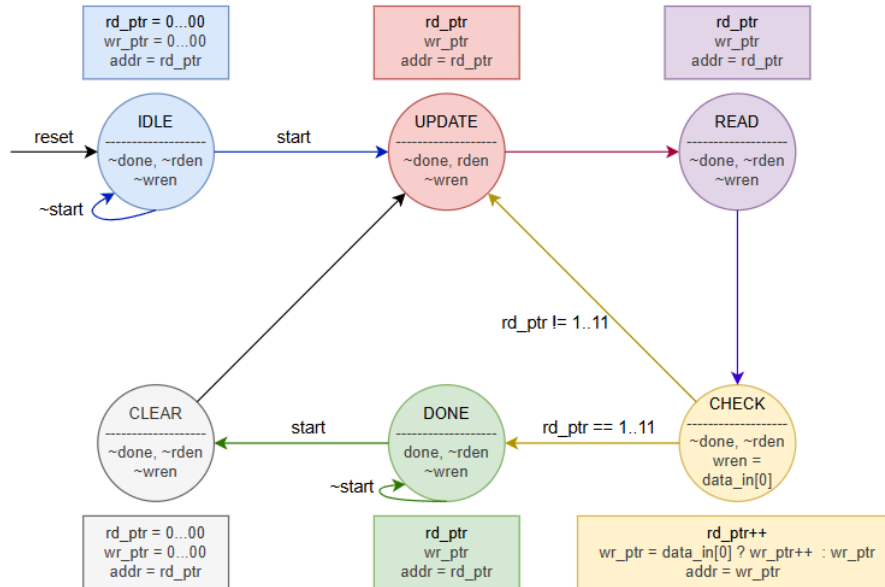
Bảng 6: I/O table for module `clear_even`

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 24: Tổng quan kết nối của thiết kế.

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

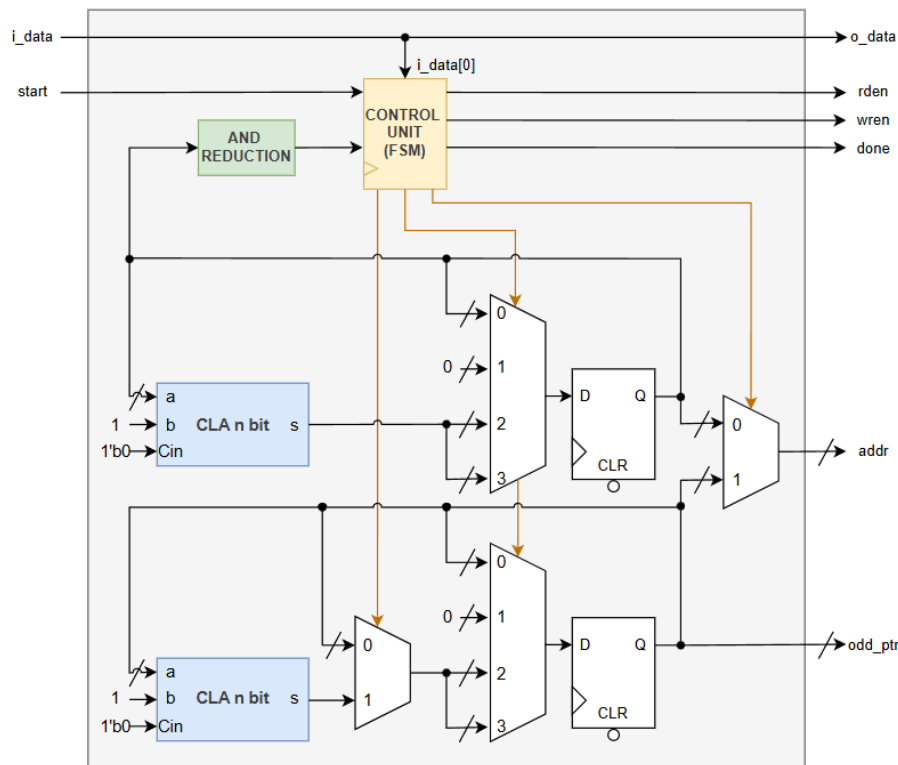


Hình 25: Máy trạng thái bậc cao.

- **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 tính 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 kỳ 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 kỳ 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 26: Datapath.

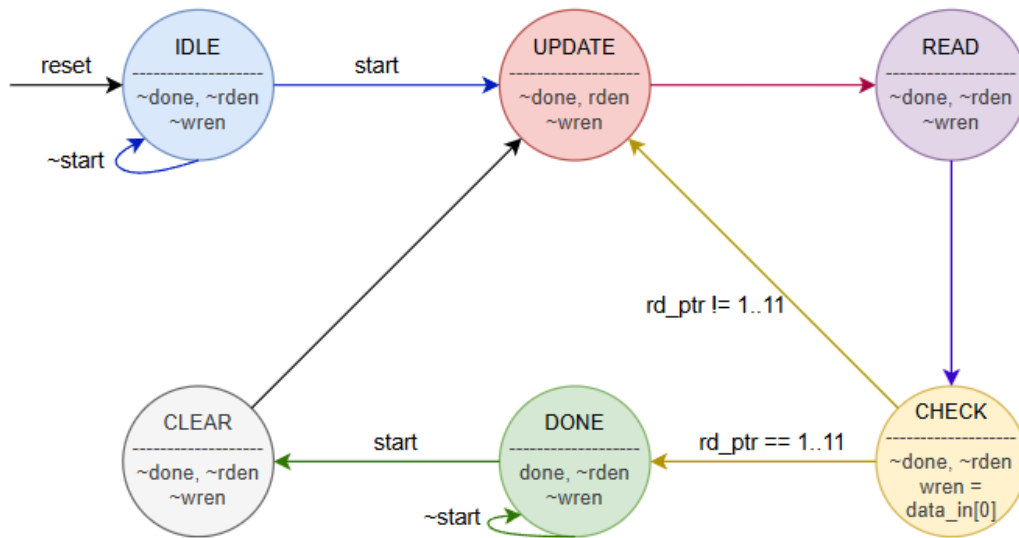
Datapath của thiết kế gồm 2 khối MUX 4-1 và 2 khối MUX 2-1 để xác định giá trị cập nhật cho 2 thanh ghi dựa vào trạng thái hiện tại và lựa chọn địa chỉ truy cập bộ nhớ.

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 27: Control Unit FSM.

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

```

1  module clear_even #(
2      parameter DEPTH = 32,
3      parameter WIDTH = 8
4  ) (
5      input logic          i_clk      ,
6      input logic          i_rst_n   ,
7      input logic          i_start   ,
8      output logic         o_rden     ,
9      output logic         o_wren     ,
10     output logic          [$clog2(DEPTH) - 1:0] o_addr ,
11     input logic signed    [WIDTH      - 1:0] i_data ,
12     output logic          o_done     ,
13     output logic signed    [WIDTH      - 1:0] o_data ,
14     output logic          [$clog2(DEPTH) - 1:0] o_odd_ptr
15 );
16
17     localparam DEPTH_S = DEPTH - 1;
18     localparam WIDTH_S = WIDTH - 1;
19
20     typedef enum logic[2:0] {
21         IDLE      ,
22         UPDATE    ,
23         READ      ,
24         CHECK     ,
25         DONE      ,
26         CLEAR     ,
27     } e_state;
28
29     e_state pstate, nstate;
30     logic [$clog2(DEPTH) - 1:0] reg_rd_ptr, rd_ptr, rd_ptr_add, reg_wr_ptr, wr_ptr, wr_ptr_add;
31

```

```

32 logic odd_flag, count_end;
33 assign odd_flag = i_data[0];
34 assign count_end = (&reg_rd_ptr);
35
36 always_ff @(posedge i_clk, negedge i_rst_n) begin
37     if(~i_rst_n) pstate <= IDLE ;
38     else pstate <= nstate;
39 end
40
41 always_comb begin
42     case(pstate)
43         IDLE : nstate = i_start ? UPDATE : pstate;
44         UPDATE : nstate = READ;
45         READ : nstate = CHECK;
46         CHECK : nstate = count_end ? DONE : UPDATE;
47
48         DONE : nstate = i_start ? CLEAR : pstate;
49         CLEAR : nstate = UPDATE;
50         default: nstate = IDLE;
51     endcase
52 end
53
54 always_ff @(posedge i_clk, negedge i_rst_n) begin
55     if(~i_rst_n) reg_rd_ptr <= '0;
56     else reg_rd_ptr <= rd_ptr;
57 end
58
59 always_ff @(posedge i_clk, negedge i_rst_n) begin
60     if(~i_rst_n) reg_wr_ptr <= '0;
61     else reg_wr_ptr <= wr_ptr;
62 end
63
64 assign o_rden = (pstate == UPDATE);
65 assign o_wren = (pstate == CHECK) ? odd_flag : 1'b0;
66 assign o_addr = (pstate == CHECK) ? reg_wr_ptr : reg_rd_ptr;
67
68 assign rd_ptr = (pstate == CHECK) ? rd_ptr_add : ((pstate == CLEAR) ? '0 : reg_rd_ptr);
69 assign wr_ptr = (pstate == CHECK) ? (odd_flag & ~count_end ? wr_ptr_add : reg_wr_ptr) : ((pstate ==
    CLEAR) ? '0 : reg_wr_ptr);
70
71 assign o_data = i_data;
72
73 assign o_done = (pstate == DONE);
74
75 assign o_odd_ptr = reg_wr_ptr;
76
77 adder_flex_no_carry #(
78     .WIDTH($clog2(DEPTH))
79 ) add_1 (
80     .i_a (reg_rd_ptr),
81     .i_b ({DEPTH_S[1'b0]},1'b1}),
82     .i_cin (1'b0),
83     .o_s (rd_ptr_add)
84 );
85
86 adder_flex_no_carry #(
87     .WIDTH($clog2(DEPTH))
88 ) add_2 (
89     .i_a (reg_wr_ptr),
90     .i_b ({DEPTH_S[1'b0]},1'b1}),
91     .i_cin (1'b0),
92     .o_s (wr_ptr_add)
93 );
94

```

```
95 endmodule
```

Listing 31: 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 32: 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 33: 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
```

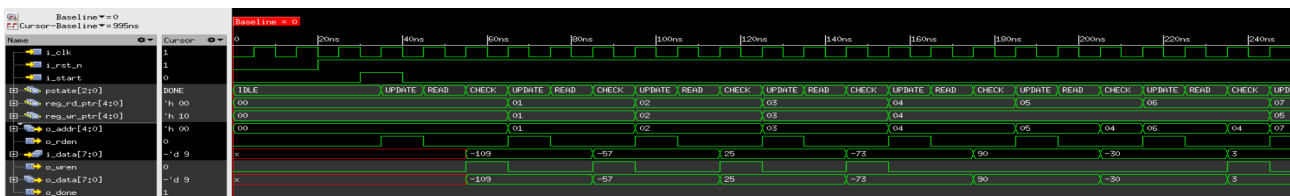
```
RAM[ 0] = -109
[TRUE] expect = -109
RAM[ 1] = -57
[TRUE] expect = -57
RAM[ 2] = 25
[TRUE] expect = 25
RAM[ 3] = -73
[TRUE] expect = -73
RAM[ 4] = 3
[TRUE] expect = 3
RAM[ 5] = -45
[TRUE] expect = -45
RAM[ 6] = -63
[TRUE] expect = -63
RAM[ 7] = -47
[TRUE] expect = -47
RAM[ 8] = 115
[TRUE] expect = 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 34: Kết quả kiểm định cho thiết kế xóa các phần tử có giá trị chẵn trong một mảng dữ liệu.



Hình 28: Dạng sóng lúc bắt đầu.

