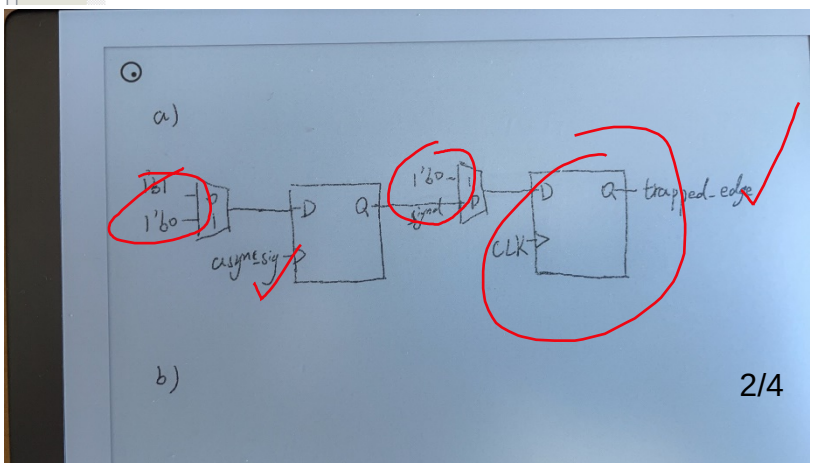
a)

```
module trap_edge(
    input logic async_sig, clk, reset,
    output logic trapped_edge);
logic Q;
always_ff @(posedge async_sig, posedge reset) begin
            if(reset) Q <= 1'b0;
            else Q <= 1'b1;
        end
always_ff @(posedge clk, posedge reset) begin
            if(reset) trapped_edge <= 1'b0;
            else trapped_edge <= Q;
        end
endmodule
```

5/6

```
1  module trap_edge(
2      input logic async_sig, clk, reset,
3      output logic trapped_edge
4  );
5   logic Q;
6
7  always_ff @(posedge async_sig, posedge reset) begin
8              if(reset) Q <= 1'b0;
9              else Q <= 1'b1;
10         end
11
12 always_ff @(posedge clk, posedge reset) begin
13             if(reset) trapped_edge <= 1'b0;
14             else trapped_edge <= Q;
15         end
16
17  endmodule
```



2/4

b)

```
module shared_access_to_one_state_machine
#(
parameter N = 32,
parameter M = 8
)
(

input target_state_machine_finished,
input sm_clk,
input logic start_request_a,
input logic start_request_b,
input [(N-1):0] input_arguments_a,
input [(N-1):0] input_arguments_b,
input reset,
input [M-1:0] in_received_data,

output logic finish_a,//
output logic finish_b,//
output logic reset_start_request_a,//
output logic reset_start_request_b,//
output reg [(M-1):0] received_data_a,
output reg [(M-1):0] received_data_b,
output reg [(N-1):0] output_arguments,
output start_target_state_machine//

);

logic [11:0] state;
parameter check_start_a =       12'b0000_00000000;
parameter give_start_a =        12'b0001_01100000;
parameter wait_for_finish_a = 12'b0010_00000000;
parameter register_data_a =   12'b0011_00000010;
parameter give_finish_a =     12'b0100_00001000;

parameter check_start_b =       12'b0101_10000000;
parameter give_start_b =        12'b0110_11010000;
parameter wait_for_finish_b = 12'b0111_10000000;
parameter register_data_b =   12'b1000_10000001;
parameter give_finish_b =     12'b1001_10000100;

logic            select_b_output_parameters,           register_data_a_enable,
register_data_b_enable;
```

```
//list the output with different state
assign register_data_b_enable = state[0];
assign register_data_a_enable = state[1];
assign finish_b = state[2];
assign finish_a = state[3];
assign reset_start_request_b = state[4];
assign reset_start_request_a = state[5];
assign start_target_state_machine = state[6];
assign select_b_output_parameters = state[7];

//select_b_output_parameters: the function of this state machine output is (in
pseudo-code):
assign output_arguments = (select_b_output_parameters) ? input_arguments_b :
input_arguments_a;

//register_data_a_enable & register_data_b_enable:
always_ff @(posedge sm_clk) begin
        if(register_data_a_enable) received_data_a <= in_received_data;
        if(register_data_b_enable) received_data_b <= in_received_data;
end

//state machine
always_ff @(posedge sm_clk, posedge reset)begin
    if(reset) state <= check_start_a;
    else
        case(state)

            check_start_a: begin
                if (!start_request_a) state <= check_start_b;
            else if (start_request_a) state <= give_start_a;
                else state <= check_start_a;
            end

         give_start_a: state <= wait_for_finish_a;

         wait_for_finish_a: begin
                if (!target_state_machine_finished) state <= wait_for_finish_a;
            else if (target_state_machine_finished) state <= register_data_a;
                else state <= wait_for_finish_a;
            end

         register_data_a: state <= give_finish_a;

         give_finish_a: state <= check_start_b;
```

```
        check_start_b: begin
                if (!start_request_b) state <= check_start_a;
           else if (start_request_b) state <= give_start_b;
                else state <= check_start_b;
           end

        give_start_b: state <= wait_for_finish_b;

        wait_for_finish_b: begin
                if (!target_state_machine_finished) state <= wait_for_finish_b;
           else if (target_state_machine_finished) state <= register_data_b;
                else state <= wait_for_finish_b;
           end

           register_data_b: state <= give_finish_b;

        give_finish_b: state <= check_start_a;

           default: state <= check_start_a;
        endcase
end

endmodule
```

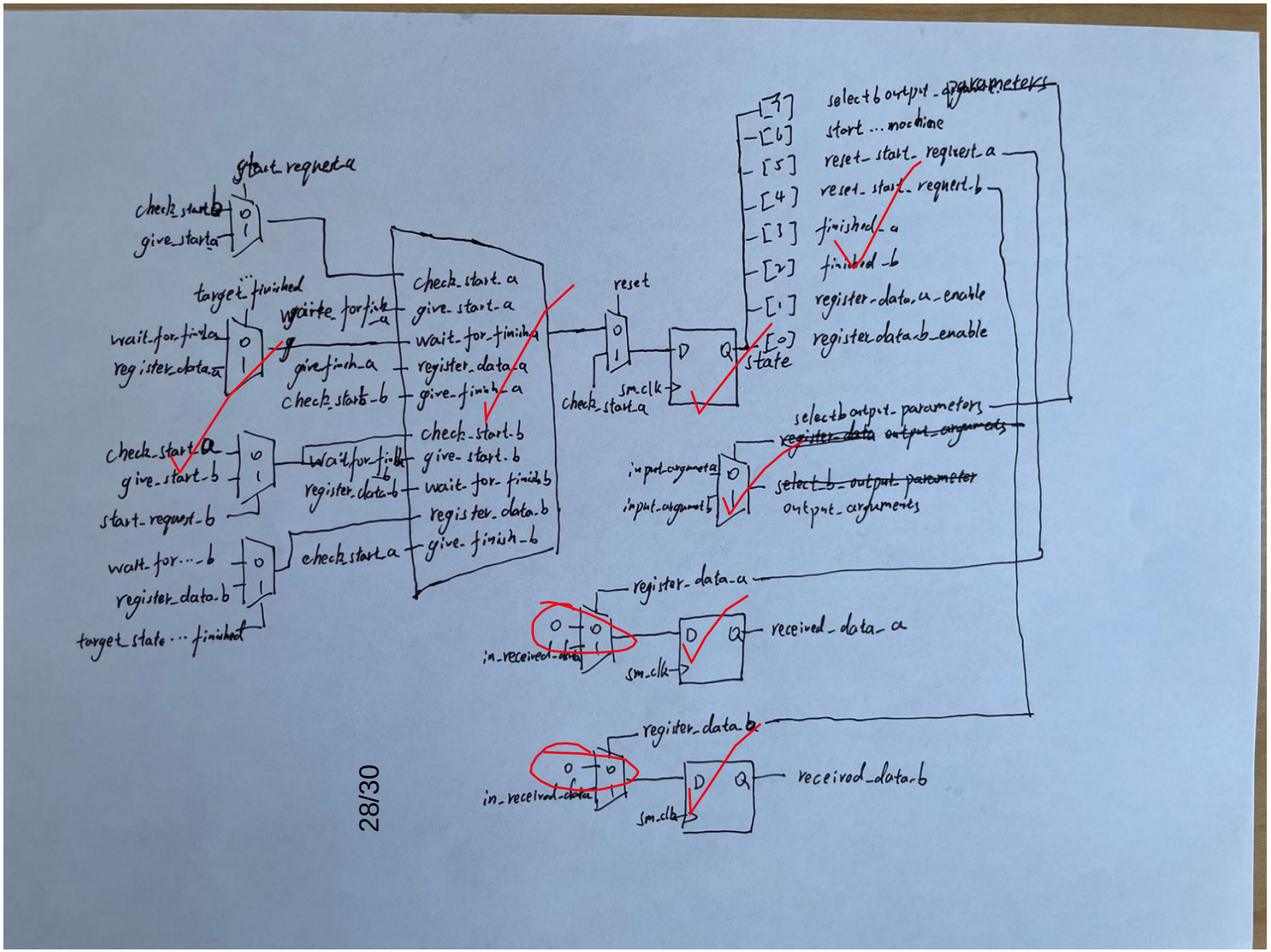


```
module shared_access_to_one_state_machine
#(
parameter N = 32,
parameter M = 8
)
(

input target_state_machine_finished,
input sm_clk,
input logic start_request_a,
input logic start_request_b,
input [(N-1):0] input_arguments_a,
input [(N-1):0] input_arguments_b,
input reset,
input [M-1:0] in_received_data,

output logic finish_a,//
output logic finish_b,//
output logic reset_start_request_a,//
output logic reset_start_request_b,//
output reg [(M-1):0] received_data_a,
output reg [(M-1):0] received_data_b,
output reg [(N-1):0] output_arguments,
output start_target_state_machine//

);

logic [11:0] state;
parameter check_start_a =     12'b0000_00000000;
parameter give_start_a =      12'b0001_01100000;
parameter wait_for_finish_a = 12'b0010_00000000;
parameter register_data_a =   12'b0011_00000010;
parameter give_finish_a =     12'b0100_00001000;

parameter check_start_b =     12'b0101_10000000;
parameter give_start_b =      12'b0110_11010000;
parameter wait_for_finish_b = 12'b0111_10000000;
parameter register_data_b =   12'b1000_10000001;
parameter give_finish_b =     12'b1001_10000100;

logic select_b_output_parameters, register_data_a_enable, register_data_b_enable;

//list the output with different state
assign register_data_b_enable = state[0];
assign register_data_a_enable = state[1];
```

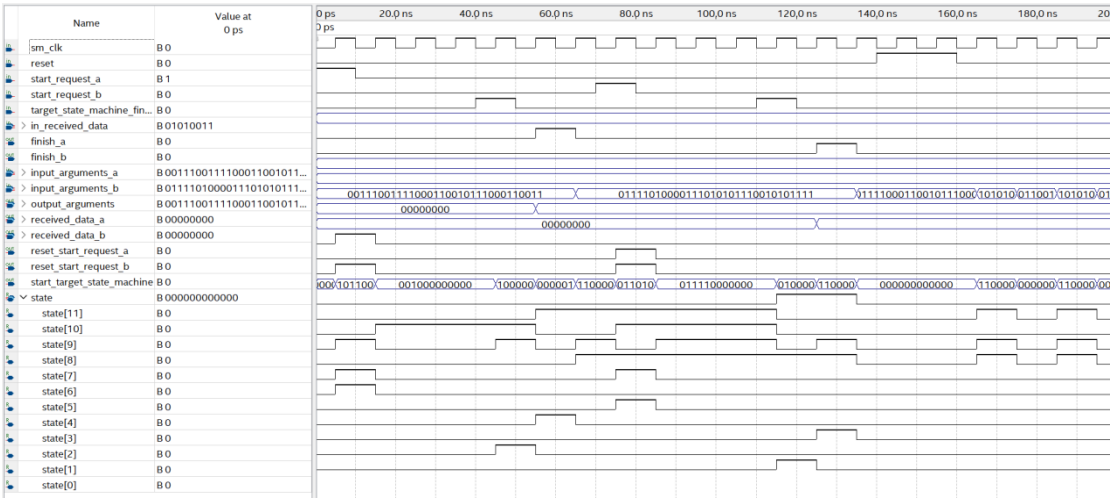
```
43  //list the output with different state
44  assign register_data_b_enable = state[0];
45  assign register_data_a_enable = state[1];
46  assign finish_b = state[2];
47  assign finish_a = state[3];
48  assign reset_start_request_b = state[4];
49  assign reset_start_request_a = state[5];
50  assign start_target_state_machine = state[6];
51  assign select_b_output_parameters = state[7];
52
53  //select_b_output_parameters: the function of this state machine output is (in pseudo-code):
54  assign output_arguments = (select_b_output_parameters) ? input_arguments_b : input_arguments_a;
55
56  //register_data_a_enable & register_data_b_enable:
57  always_ff @(posedge sm_clk) begin
58        if(register_data_a_enable) received_data_a <= in_received_data;
59        if(register_data_b_enable) received_data_b <= in_received_data;
60  end
61
62  //state machine
63  always_ff @(posedge sm_clk, posedge reset)begin
64     if(reset) state <= check_start_a;
65     else
66        case(state)
67
68           check_start_a: begin
69              if (!start_request_a) state <= check_start_b;
70              else if (start_request_a) state <= give_start_a;
71              else state <= check_start_a;
72           end
73
74           give_start_a: state <= wait_for_finish_a;
75
76           wait_for_finish_a: begin
77              if (!target_state_machine_finished) state <= wait_for_finish_a;
78              else if (target_state_machine_finished) state <= register_data_a;
79              else state <= wait_for_finish_a;
80           end
81
82           register_data_a: state <= give_finish_a;
83
84           give_finish_a: state <= check_start_b;
85
86           check_start_b: begin
87              if (!start_request_b) state <= check_start_a;
```

```
86           check_start_b: begin
87              if (!start_request_b) state <= check_start_a;
88              else if (start_request_b) state <= give_start_b;
89              else state <= check_start_b;
90           end
91
92           give_start_b: state <= wait_for_finish_b;
93
94           wait_for_finish_b: begin
95              if (!target_state_machine_finished) state <= wait_for_finish_b;
96              else if (target_state_machine_finished) state <= register_data_b;
97              else state <= wait_for_finish_b;
98           end
99
100          register_data_b: state <= give_finish_b;
101
102          give_finish_b: state <= check_start_a;
103
104          default: state <= check_start_a;
105       endcase
106 end
107
108 endmodule
```

c)


start_request_a
check_start_b
give_start_a
target_finished
wait_for_finish_a
wait_for_finish_a
register_data_a
give_finish_a
check_start_b
check_start_a
give_start_b
start_request_b
wait_for_finish_b
register_data_b
wait_for_..._b
register_data_b
target_state ... finished
check_start_a
check_start_a
give_start_a
wait_for_finish_a
register_data_a
give_finish_a
check_start_b
give_start_b
wait_for_finish_b
register_data_b
give_finish_b
reset
check_start_a
sm_clk
D
Q
state
[7]
[6]
[5]
[4]
[3]
[2]
[1]
[0]
select_b_output_parameters
start_..._machine
reset_start_request_a
reset_start_request_b
finished_a
finished_b
register_data_a_enable
register_data_b_enable
select_b_output_parameters
register_data
output_arguments
input_argument_a
input_argument_b
select_b_output_parameter
output_arguments
register_data_a
in_received_data_a
sm_clk
received_data_a
register_data_b
in_received_data_b
sm_clk
received_data_b

d)



when start_request_a is 1, the state chenge to give_start_a(0001/output:01100000) from check_start_a(0000/output:00000000)

then the state change to wait_for_finish_a(0010/output:00000000)

when target_state_machine_finished is 0, then the state change to wait_for_finish_a(0010/output:00000000)

when target_state_machine_finished is 1, then the state change to register_data_a(0011/output:00000010)

Then the state change to give_finish_a(0100/output:00001000), and check_start_b(0101/output:10000000)

when start_request_b is 1, the state chenge to give_start_b(0111/output:10000000) from check_start_b(0110/output:11010000)

then the state change to wait_for_finish_b(0111/output:10000000)

when target_state_machine_finished is 0, then the state change to wait_for_finish_b(0111/output:10000000)

when target_state_machine_finished is 1, then the state change to register_data_b(1000/output:10000001)

Then the state change to give_finish_b(0101/output:10000100), and check_start_a(0000/output:00000000)

If reset is 1, the state is check_start_a,

If (select_b_output_parameters) = 1, output_arguments = input_arguments_b, if (select_b_output_parameters) = 0, output_arguments = input_arguments_a

e)