```verilog
module shared_access_to_one_current_state_machine
#(
parameter N = 32,
parameter M = 8

)
(

output reg [(N-1):0] output_arguments,
output logic start_target_current_state_machine,
input target_current_state_machine_finished,
input sm_clk,
input logic start_request_a,
input logic start_request_b,
output logic finish_a,
output logic finish_b,
output logic reset_start_request_a,
output logic reset_start_request_b,
input [(N-1):0] input_argumnets_a,
input [(N-1):0] input_argumnets_b,
output reg [(M-1):0] received_data_a,
output reg [(M-1):0] received_data_b,
input reset,
input [M-1:0] in_received_data
);

logic register_data_a_enable;
logic register_data_b_enable;
logic select_b_output_parameters;

reg [12:0] state, current_state;

assign output_arguments = ( (select_b_output_parameters) ? input_argumnets_b :
input_argumnets_a);
assign received_data_a = ( (register_data_a_enable)? in_received_data: 0 ) ;
assign received_data_b = ( (register_data_b_enable)? in_received_data: 0 ) ;

parameter [ 11:0]       check_start_a = 12'b0000_00000000, // 0  // state_output
                        give_start_a= 12'b0001_00000110, //262
                        wait_for_finish_a= 12'b0010_00000000, //512
                        register_data_a= 12'b0011_01000000,// 832
                        give_finish_a= 12'b0100_00010000, //1040
                        check_start_b=12'b0101_00000001,//1281
                        give_start_b= 12'b0110_00001011,//1547
                        wait_for_finish_b= 12'b0111_00000001,//1793
                        register_data_b= 12'b1000_10000001, //2177
                        give_finish_b=12'b0001_00100001; // 289


            //  [0]select_b_output_parameters
        //      [1] start_target_current_state_machine //
            //  [2]reset_start_request_a //
            //  [3]reset_start_request_b //
            //  [4]finish_a//
            //  [5]finish_b //
            //  [6]register_data_a_enable
            //  [7]register_data_b_enable

always_ff@(posedge sm_clk, posedge reset) begin

    if(reset)  state<=check_start_a;
    else state <= current_state;

    end

always_comb begin

    case(state)

```

```
        check_start_a:

            if (start_request_a)  current_state = give_start_a;
            else current_state = check_start_b;

        give_start_a:  current_state = wait_for_finish_a;

        wait_for_finish_a:

            if ( target_current_state_machine_finished) current_state = register_data_a;
            else current_state = wait_for_finish_a;

        register_data_a:  current_state = give_finish_a;

        give_finish_a:  current_state = check_start_b;

        check_start_b:

            if ( start_request_b) current_state = give_start_b;
            else current_state = check_start_a;

        give_start_b:  current_state = wait_for_finish_b;

        wait_for_finish_b:

            if(target_current_state_machine_finished) current_state = register_data_b;
            else current_state = wait_for_finish_b;

        register_data_b:  current_state = give_finish_b;

        give_finish_b: current_state = check_start_a;

    endcase

end

assign {
register_data_b_enable, //  [7]
register_data_a_enable, //  [6]
finish_b ,//                [5]
finish_a,// [4]
reset_start_request_b, //   [3]
reset_start_request_a, // [2]
start_target_current_state_machine, //  [1]
select_b_output_parameters// [0]
 }  = state[7:0];



endmodule
```