Homework 3

1. Module trap\_edge

`default\_nettype none

module trap\_edge (

  input clk,

  input async\_sig,

  input reset,

  output logic trapped\_edge

);

logic ff1, ff2, ff3;

always\_ff @( posedge async\_sig or posedge reset ) begin

  if (reset) begin

    ff1 <= 1'b0;

  end else begin

    ff1 <= 1'b1;

  end

end

always\_ff @( posedge clk ) begin

  ff2 <= ff1;

end

always\_ff @( posedge clk ) begin

  ff3 <= ff2;

end

assign trapped\_edge = ff3;

endmodule

`default\_nettype wire

A diagram of a block diagram

Description automatically generated with low confidence

1. Module shared\_access\_to\_one\_state\_machine

`default\_nettype none

module shared\_access\_to\_one\_state\_machine #(

  parameter N = 32,

  parameter M = 8

) (

  output reg [(N-1):0] output\_arguments,

  output start\_target\_state\_machine,

  input target\_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\_arguments\_a,

  input [(N-1):0] input\_arguments\_b,

  output reg [(M-1):0] received\_data\_a,

  output reg [(M-1):0] received\_data\_b,

  input reset, // async active high reset

  input [M-1:0] in\_received\_data

);

logic select\_b\_output\_parameters;

logic register\_data\_a\_enable;

logic register\_data\_b\_enable;

logic [11:0] state;

localparam check\_start\_a      = 12'b0000\_0000\_0000;

localparam check\_start\_b      = 12'b0001\_0000\_0001;

localparam give\_start\_a       = 12'b0010\_0000\_0110;

localparam give\_start\_b       = 12'b0011\_0000\_1011;

localparam wait\_for\_finish\_a  = 12'b0100\_0000\_0000;

localparam wait\_for\_finish\_b  = 12'b0101\_0000\_0001;

localparam register\_data\_a    = 12'b0110\_0100\_0000;

localparam register\_data\_b    = 12'b0111\_1000\_0001;

localparam give\_finish\_a      = 12'b1000\_0001\_0000;

localparam give\_finish\_b      = 12'b1001\_0010\_0001;

assign select\_b\_output\_parameters = state[0];

assign start\_target\_state\_machine = state[1];

assign reset\_start\_request\_a      = state[2];

assign reset\_start\_request\_b      = state[3];

assign finish\_a                   = state[4];

assign finish\_b                   = state[5];

assign register\_data\_a\_enable     = state[6];

assign register\_data\_b\_enable     = state[7];

always\_ff @( posedge sm\_clk or posedge reset ) begin

  if (reset) begin

    state <= check\_start\_a;

  end else begin

    case (state)

      check\_start\_a:  if (start\_request\_a) state <= give\_start\_a;

                      else state <= check\_start\_b;

      check\_start\_b:  if (start\_request\_b) state <= give\_start\_b;

                      else state <= check\_start\_a;

      give\_start\_a: state <= wait\_for\_finish\_a;

      give\_start\_b: state <= wait\_for\_finish\_b;

      wait\_for\_finish\_a:  if (target\_state\_machine\_finished) state <= register\_data\_a;

                          else state <= wait\_for\_finish\_a;

      wait\_for\_finish\_b:  if (target\_state\_machine\_finished) state <= register\_data\_b;

                          else state <= wait\_for\_finish\_b;

      register\_data\_a: state <= give\_finish\_a;

      register\_data\_b: state <= give\_finish\_b;

      give\_finish\_a: state <= check\_start\_b;

      give\_finish\_b: state <= check\_start\_a;

      default: state <= check\_start\_a;

    endcase

  end

end

assign output\_arguments = (select\_b\_output\_parameters) ? input\_arguments\_b : input\_arguments\_a;

always\_ff @( posedge sm\_clk ) begin

  if (register\_data\_a\_enable) begin

    received\_data\_a <= in\_received\_data;

  end

end

always\_ff @( posedge sm\_clk ) begin

  if (register\_data\_b\_enable) begin

    received\_data\_b <= in\_received\_data;

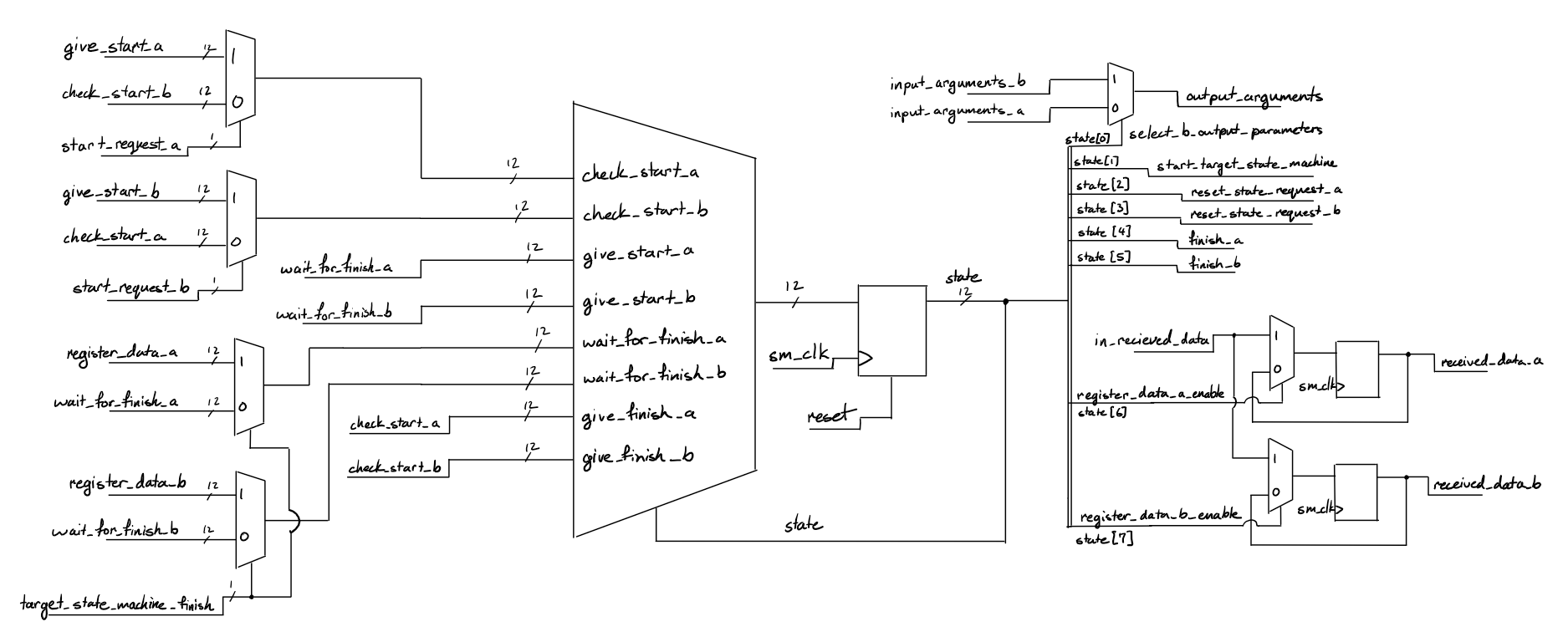
  end

end

endmodule

`default\_nettype wire

1. Module shared\_access\_to\_one\_state\_machine schematic



1. Simulation

start\_request\_a and start\_request\_b are both high but go to give\_start\_a because it is currently in state check\_start\_a. This asserts reset\_start\_request\_a to high and sends start\_target\_state\_machine signal.  
output\_arguments also shows input\_arguments\_a.

A screenshot of a computer

Description automatically generated with low confidence

Select\_b\_output\_parameters (state[0]) decides which output\_arguments are shown. Those of A if state[0] = 0. Those of B if state[0] = 1.

state[6] is register\_data\_a\_enable signal.  
This is high when in state register\_data\_a.  
in\_received\_data is registered to received\_data\_a if register\_data\_a\_enable is high. Here it takes on the value FF.

During register\_data\_b, in\_received\_data is registered to received\_data\_b because register\_data\_b\_enable (state[7]) is high.  
This causes it to take in value DD.

Just like a, b follows equivalent states.

After give\_finish\_a, the next state is check\_start\_b, so even though both start\_request\_a and start\_request\_b are both high. It will take inputs from fsm\_b.

After state register\_data\_a go to state give\_finish\_a, indicated by output finish\_a high.

After one clock cycle move to state wait\_for\_finish\_a. Stay in this state until target\_state\_machine\_finished signal is high.  
Then go to register\_data\_a state.

Reset will default to check\_start\_a