```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: 
// Engineer: 
// 
// Create Date:    12:47:20 06/25/2021 
// Design Name: 
// Module Name:    I2C_Master 
// Project Name: 
// Target Devices: 
// Tool versions: 
// Description: 
//
// Dependencies: 
//
// Revision: 
// Revision 0.01 - File Created
// Additional Comments: 
//
//////////////////////////////////////////////////////////////////////////////////
module I2C_Master(
   input wire clk,
   input wire rst,
   input wire [6:0] addr,
   input wire [7:0] data_in,
   input wire enable,
   input wire rw,

   //output reg [7:0] data_out,
   output wire ready,

   inout i2c_sda,
   inout wire i2c_scl
   );

   localparam IDLE = 0;
   localparam START = 1;
   localparam ADDRESS = 2;
   localparam READ_ACK = 3;
   localparam WRITE_DATA = 4;
   localparam WRITE_ACK = 5;
   localparam READ_DATA = 6;
   localparam READ_ACK2 = 7;
   localparam STOP = 8;

   localparam DIVIDE_BY = 4;

   reg [7:0] state;
   reg [7:0] saved_addr;
   reg [7:0] saved_data;
   reg [7:0] counter;
   reg [7:0] counter2 = 0;
   reg write_enable;
   reg sda_out;
   reg i2c_scl_enable = 0;
   reg i2c_clk = 1;

```

```
    assign ready = ((rst == 0) && (state == IDLE)) ? 1 : 0;
    assign i2c_scl = (i2c_scl_enable == 0 ) ? 1 : i2c_clk;
    assign i2c_sda = (write_enable == 1) ? sda_out : 'bz;

    always @(posedge clk) begin
       if (counter2 == (DIVIDE_BY/2) - 1) begin
          i2c_clk <= ~i2c_clk;
          counter2 <= 0;
       end
       else counter2 <= counter2 + 1;
    end

    always @(negedge i2c_clk, posedge rst) begin
       if(rst == 1) begin
          i2c_scl_enable <= 0;
       end else begin
          if ((state == IDLE) || (state == START) || (state == STOP)) begin
             i2c_scl_enable <= 0;
          end else begin
             i2c_scl_enable <= 1;
          end
       end

    end


    always @(posedge i2c_clk, posedge rst) begin
       if(rst == 1) begin
          state <= IDLE;
       end
       else begin
          case(state)

             IDLE: begin
                if (enable) begin
                   state <= START;
                   saved_addr <= {addr, rw};
                   saved_data <= data_in;
                end
                else state <= IDLE;
             end

             START: begin
                counter <= 7;
                state <= ADDRESS;
             end

             ADDRESS: begin
                if (counter == 0) begin
                   state <= READ_ACK;
                end else counter <= counter - 1;
             end

             READ_ACK: begin
                if (i2c_sda == 0) begin
                   counter <= 7;
                   if(saved_addr[0] == 0) state <= WRITE_DATA;
```

```
                  else state <= READ_DATA;
               end else state <= STOP;
            end

            WRITE_DATA: begin
               if(counter == 0) begin
                  state <= READ_ACK2;
               end else counter <= counter - 1;
            end

            READ_ACK2: begin
               if ((i2c_sda == 0) && (enable == 1)) state <= IDLE;
               else state <= STOP;
            end

            READ_DATA: begin
               //data_out[counter] <= i2c_sda;
               if (counter == 0) state <= WRITE_ACK;
               else counter <= counter - 1;
            end

            WRITE_ACK: begin
               state <= STOP;
            end

            STOP: begin
               state <= IDLE;
            end
         endcase
      end
   end

   always @(negedge i2c_clk, posedge rst) begin
      if(rst == 1) begin
         write_enable <= 1;
         sda_out <= 1;
      end else begin
         case(state)

            START: begin
               write_enable <= 1;
               sda_out <= 0;
            end

            ADDRESS: begin
               sda_out <= saved_addr[counter];
            end

            READ_ACK: begin
               write_enable <= 0;
            end

            WRITE_DATA: begin
               write_enable <= 1;
               sda_out <= saved_data[counter];
            end

```

```
                WRITE_ACK: begin
                    write_enable <= 1;
                    sda_out <= 0;
                end

                READ_DATA: begin
                    write_enable <= 0;
                end

                STOP: begin
                    write_enable <= 1;
                    sda_out <= 1;
                end
            endcase
        end
    end

endmodule

```