```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    12:47:59 06/25/2021
// Design Name:
// Module Name:    I2C_slave
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module I2C_Slave(
   inout sda,
   inout scl
   );

   localparam ADDRESS = 7'b1000100;

   localparam READ_ADDR = 0;
   localparam SEND_ACK = 1;
   localparam READ_DATA = 2;
   localparam WRITE_DATA = 3;
   localparam SEND_ACK2 = 4;

   reg [7:0] addr;
   reg [7:0] counter;
   reg [7:0] state = 0;
   reg [7:0] data_in = 0;
   reg [7:0] data_out = 8'b11001100;
   reg sda_out = 0;
   reg sda_in = 0;
   reg start = 0;
   reg write_enable = 0;

   assign sda = (write_enable == 1) ? sda_out : 'bz;

   always @(negedge sda) begin
      if ((start == 0) && (scl == 1)) begin
         start <= 1;
         counter <= 7;
      end
   end

   always @(posedge sda) begin
      if ((start == 1) && (scl == 1)) begin
         state <= READ_ADDR;
         start <= 0;
         write_enable <= 0;
      end
   end

   always @(posedge scl) begin
      if (start == 1) begin
         case(state)
            READ_ADDR: begin
               addr[counter] <= sda;
               if(counter == 0) state <= SEND_ACK;
               else counter <= counter - 1;
            end

            SEND_ACK: begin
               if(addr[7:1] == ADDRESS) begin
                  counter <= 7;
                  if(addr[0] == 0) begin
                     state <= READ_DATA;
                  end
                  else state <= WRITE_DATA;
               end
            end

            READ_DATA: begin
               data_in[counter] <= sda;
               if(counter == 0) begin
                  state <= SEND_ACK2;
               end else counter <= counter - 1;
            end

            SEND_ACK2: begin
               state <= READ_ADDR;
            end

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

         endcase
      end
   end

   always @(negedge scl) begin
      case(state)

         READ_ADDR: begin
            write_enable <= 0;
         end

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

         READ_DATA: begin
            write_enable <= 0;
         end

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

         SEND_ACK2: begin
            sda_out <= 0;
            write_enable <= 1;
         end
      endcase
   end
endmodule

```