

Documentation for a Basic I²C Master Controller

Generated by AI Assistant

December 9, 2025

Abstract

This document provides the technical specification and implementation details for a basic I²C (Inter-Integrated Circuit) master controller, implemented in Verilog. The controller is designed to handle a standard single-byte write transaction, including the START condition, addressing, data transfer, acknowledgment (ACK) handling, and the final STOP condition. It also includes the logic for generating the serial clock (SCL) signal.

1 Introduction to the I²C Protocol

The I²C bus is a two-wire, bi-directional serial bus widely used for connecting low-speed peripheral integrated circuits to a microprocessor or microcontroller. The two wires are:

- **SCL** (Serial Clock): The clock signal, controlled by the master device.
- **SDA** (Serial Data): The bi-directional data line.

Both lines are open-drain (or open-collector) and require external pull-up resistors to ensure the bus is high when idle. Data transfer occurs when the SCL is low, and data is sampled when SCL is high. [Image of I2C bus connection diagram]

2 I²C Master Controller Design (I2C.v)

The I2C_1 module implements the master state machine and clock generation necessary to perform a single write transaction to a slave device.

2.1 Module Ports and I/O

The module interface is defined as follows:

Table 1: I2C_1 Module Port Descriptions

Port Name	Direction	Width	Description
CLK	Input	1	System Clock (High Frequency)
RST	Input	1	Asynchronous Reset (Active High)
data_in	Input	8	8-bit Data to be written
addr_in	Input	7	7-bit Slave Address
SDA	Inout	1	Bi-directional Serial Data Line
SCL	Output	1	Serial Clock Line

2.1.1 SDA Driver Logic

The SDA line is shared and must be driven using an open-drain style. The master logic models this using a tristate buffer:

```
assign SDA = i2c_sda_en ? 1'bz : i2c_sda;
```

- `i2c_sda_en (1'b1)` releases the line (`1'bz`) for floating or for the slave to drive (e.g., during an ACK cycle).
- `i2c_sda_en (1'b0)` drives the line with the value of `i2c_sda`.

2.2 SCL Clock Generation

The SCL clock is generated by dividing the high-frequency system clock (`CLK`). The master operates in a way that SCL is high for `SCL_DIV_COUNT` clock cycles and low for the same number of cycles, creating a 50% duty cycle.

The division factor is set by:

```
localparam SCL_DIV_COUNT = 250;
```

Assuming a system clock frequency (f_{CLK}) of 50 MHz (period $T_{CLK} = 20$ ns), the resulting SCL frequency (f_{SCL}) is calculated as:

$$f_{SCL} = \frac{f_{CLK}}{2 \times SCL_DIV_COUNT} = \frac{50 \text{ MHz}}{2 \times 250} = 100 \text{ kHz}$$

This corresponds to the I²C Standard Mode speed.

2.3 State Machine Description

The transaction is controlled by a Finite State Machine (FSM) implemented in the `always @(posedge CLK)` block, transitioning on the falling edge of the divided SCL clock (when `scl_count = 0`).

Table 2: FSM States for I²C Write Transaction

State	Value	Action / Description
IDLE	0	Wait for transaction start. SCL/SDA are high.
START	1	Generates the START Condition (SDA 1 → 0 while SCL is 1).
ADDR	2	Transmits the 7-bit slave address followed by the Write bit (0).
ACK_WAIT1	3	Releases SDA (1'bz) and checks for Slave ACK (SDA 0).
DATA	4	Transmits the 8-bit data byte.
ACK_WAIT2	5	Releases SDA (1'bz) and checks for Slave ACK (SDA 0).
STOP	6	Generates the STOP Condition (SDA 0 → 1 while SCL is 1).

2.3.1 Transaction Flow

The FSM executes a full write operation (**START** → **ADDRESS** + **W** → **ACK** → **DATA** → **ACK** → **STOP**).

2.4 Verilog Code: I2C Master Controller

```

1 module I2C_1 (
2     input wire      CLK,
3     input wire      RST,
4     input wire [7:0] data_in,
5     input wire [6:0] addr_in,
6     inout wire      SDA,
7     output reg      SCL
8 );
9
10 reg i2c_sda;
11 reg i2c_sda_en;    // 0 = drive SDA, 1 = release (Z)
12 assign SDA = i2c_sda_en ? 1'bz : i2c_sda;
13
14 // State + counters
15 reg [7:0] state;
16 reg [3:0] count;
17 reg [7:0] tx_data;
18

```

```

19 localparam IDLE      = 0;
20 localparam START     = 1;
21 localparam ADDR      = 2;
22 localparam ACK_WAIT1 = 3;
23 localparam DATA     = 4;
24 localparam ACK_WAIT2 = 5;
25 localparam STOP      = 6;
26
27 // Clock division for SCL
28 localparam SCL_DIV_COUNT = 250;
29 reg [8:0] scl_count;
30
31 always @(posedge CLK) begin
32     if (RST) begin
33         scl_count <= 0;
34         SCL <= 1;
35     end else begin
36         if (state != IDLE) begin
37             if (scl_count < SCL_DIV_COUNT - 1)
38                 scl_count <= scl_count + 1;
39             else begin
40                 SCL <= ~SCL;
41                 scl_count <= 0;
42             end
43         end else begin
44             SCL <= 1;
45             scl_count <= 0;
46         end
47     end
48 end
49
50 always @(posedge CLK) begin
51     if (RST) begin
52         state      <= IDLE;
53         i2c_sda     <= 1;
54         i2c_sda_en  <= 1;
55         count       <= 0;
56         tx_data     <= 0;
57     end else begin
58
59         if (scl_count == 0) begin
60             case(state)
61
62                 IDLE: begin
63                     i2c_sda_en <= 0;
64                     i2c_sda    <= 1;
65                     if (SCL == 1) state <= START;
66                 end
67
68                 START: begin
69                     if (SCL == 1) begin
70                         i2c_sda      <= 0;
71                         i2c_sda_en <= 0;
72                         tx_data      <= {addr_in, 1'b0};
73                         count        <= 7;
74                         state        <= ADDR;
75                     end
76                 end
77
78                 ADDR: begin
79                     i2c_sda_en <= 0; // Drive
80                     if (SCL == 0) begin
81                         i2c_sda    <= tx_data[count];
82                     end else begin
83                         if (count == 0) begin
84                             state <= ACK_WAIT1;
85                             count <= 7;
86                         end
87                         else count <= count - 1;
88                     end
89                 end
90
91                 ACK_WAIT1: begin

```

```

92         if (SCL == 0) begin
93             i2c_sda_en <= 1;
94         end
95         else begin
96             tx_data <= data_in;
97             count <= 7;
98             state <= DATA;
99         end
100     end
101
102     DATA: begin
103         i2c_sda_en <= 0; // Drive
104         if (SCL == 0) begin
105             i2c_sda <= tx_data[count];
106         end else begin
107             if (count == 0) begin
108                 state <= ACK_WAIT2;
109                 count <= 7;
110             end
111             else count <= count - 1;
112         }
113     end
114
115     ACK_WAIT2: begin
116         if (SCL == 0) begin
117             i2c_sda_en <= 1;
118         end else begin
119             state <= STOP;
120         end
121     end
122
123     STOP: begin
124         if (SCL == 0) begin
125             i2c_sda_en <= 0;
126             i2c_sda <= 0;
127         end else begin
128             i2c_sda <= 1;
129             i2c_sda_en <= 0;
130             state <= IDLE;
131         end
132     end
133
134     default: state <= IDLE;
135
136     endcase
137 end
138 end
139 end
140
141 endmodule

```

Listing 1: I2C Master Controller (I2C.v)

3 Testbench Implementation (I2C_1tb.v)

The testbench provides the necessary environment to simulate and verify the master controller.

3.1 Key Testbench Components

- **High-Speed Clock:** The CLK is generated with a 20 ns period (50 MHz).
- **SDA Pull-up Model:** The Verilog primitive `pullup(SDA)` is used to model the open-drain nature of the bus, ensuring the line defaults high when released by both master and slave.
- **Simple Slave ACK Model:** A state-dependent model for a slave device is included:
 1. It monitors the master's state.

2. When the master is in ACK_WAIT1 (State 3) or ACK_WAIT2 (State 5), the slave drives the SDA line low to generate an **ACK**.
3. The slave only drives SDA low when SCL is low, ensuring the master sees the **0** during the SCL high phase of the ACK clock cycle.

3.2 Test Scenario

The test scenario is a single write transaction:

- **Slave Address:** `addr_in = 7'h50`
- **Data to Write:** `data_in = 8'hAA`

The simulation runs long enough (**192000 ns**) to cover the full I²C transaction at the 100 kHz speed.

3.3 Verilog Code: I2C Testbench

```

1 `timescale 1ns / 1ps
2
3 module I2C_tb;
4
5     reg        CLK;
6     reg        RST;
7     reg [7:0]  data_in;
8     reg [6:0]  addr_in;
9
10    wire SDA;
11    wire SCL;
12
13    reg slave_ack_drive;
14
15    // 1. Model Pull-up Resistor for SDA
16    // This primitive ensures SDA defaults to '1' (high) when released.
17    pullup (SDA);
18
19    assign SDA = (slave_ack_drive == 1'b0) ? 1'b0 : 1'bz;
20
21    always @ (posedge CLK) begin
22        if (RST) begin
23            slave_ack_drive <= 1'b1;
24        end
25        else begin
26            if (dut.state == 3 || dut.state == 5) begin // ACK_WAIT1 or ACK_WAIT2
27                if (SCL == 0) begin
28                    slave_ack_drive <= 1'b0;
29                end else begin
30                    slave_ack_drive <= 1'b1;
31                end
32            end
33            else begin
34                slave_ack_drive <= 1'b1;
35            end
36        end
37    end
38
39    // 3. Instantiate the Device Under Test (DUT)
40    I2C_1 dut (
41        .CLK      (CLK),
42        .RST      (RST),
43        .data_in  (data_in),
44        .addr_in  (addr_in),
45        .SDA      (SDA),
46        .SCL      (SCL)
47    );
48
49    // 4. Clock Generator (20 ns period for 50 MHz CLK)
50    always #10 CLK = ~CLK;
51
52

```

```

53 initial begin
54     CLK = 0;
55     RST = 1;
56
57     data_in = 8'hAA;
58     addr_in = 7'h50;
59
60     $display("===== Simulation starts at %0t ns ====", $time);
61     $display("Target Write: Addr=7'h%2h, Data=8'h%2h", addr_in, data_in);
62
63     #100;
64     RST = 0;
65
66     #192000;
67
68     $display("===== Simulation ends at %0t ns ====", $time);
69     $finish;
70 end
71
72 initial
73 $monitor("Time=%0t ns | State=%0d | SCL=%0b | SDA=%0b | Master_SDA_En=%0b | Slave_ACK_Drive=%0b",
74         $time, dut.state, SCL, SDA, dut.i2c_sda_en, slave_ack_drive);
75
76 endmodule

```

Listing 2: I2C Master Testbench (I2C_1tb.v)