Inter Integrated Circuit (IIC)
(I2C Protocol Basics)

- I2C (Inter connected IC)

- Developed by PHILIPS SC. (low cost and simple interface, easy to use, used for simple interconnect)

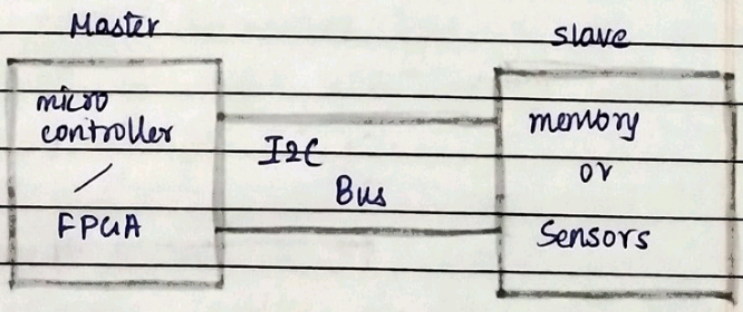
- Used to send data between microcontrollers and peripherals like EEPROM, RTC, SENSORS, SD CARD, LCD, RFID CARD MODULE, consumer electronics.

- It is a serial communication protocol

- Max speed 5 Mbps.

- Important bus used in consumer electronics.

eg:



- It is a simple bus having 2 wires for data communication.

- SCL : clock signal line (serial clock used for data communications)
- SDA : data signal line (Serial Data)

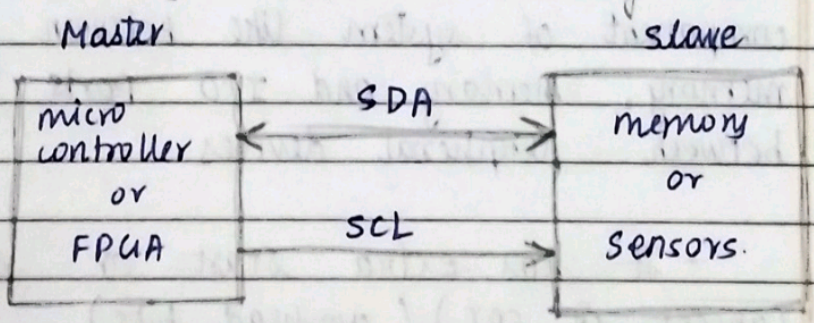
- It is half duplex. (either master or slave can send data at same time through SDA line). Either read or write operation possible via bus at a single time.

- Transmitter : A device that sends data to the bus. It can either put data on the bus of its own or in response to a request from another devices.

- Receiver : A device that recieves data from bus.

- Master : The one which initializes the transfer, generates clock signal, terminates the transfer.

half duplex → one way communication
full duplex → two way communication



⟷ Master can put data on SDA line or slave can put data.

⟶ Only master can generate the clock.

- It's normally used for short distance communication (communication within same circuit board).

- It supports multi master means here many devices can access the I2C bus at a given time with arbitration process (it decides which devices act as master in multi master system). Slaves takes instruction from master

- Adding new slave is easy. Add slave device without adding a new slave select line unlike SPI. It is the advantage.

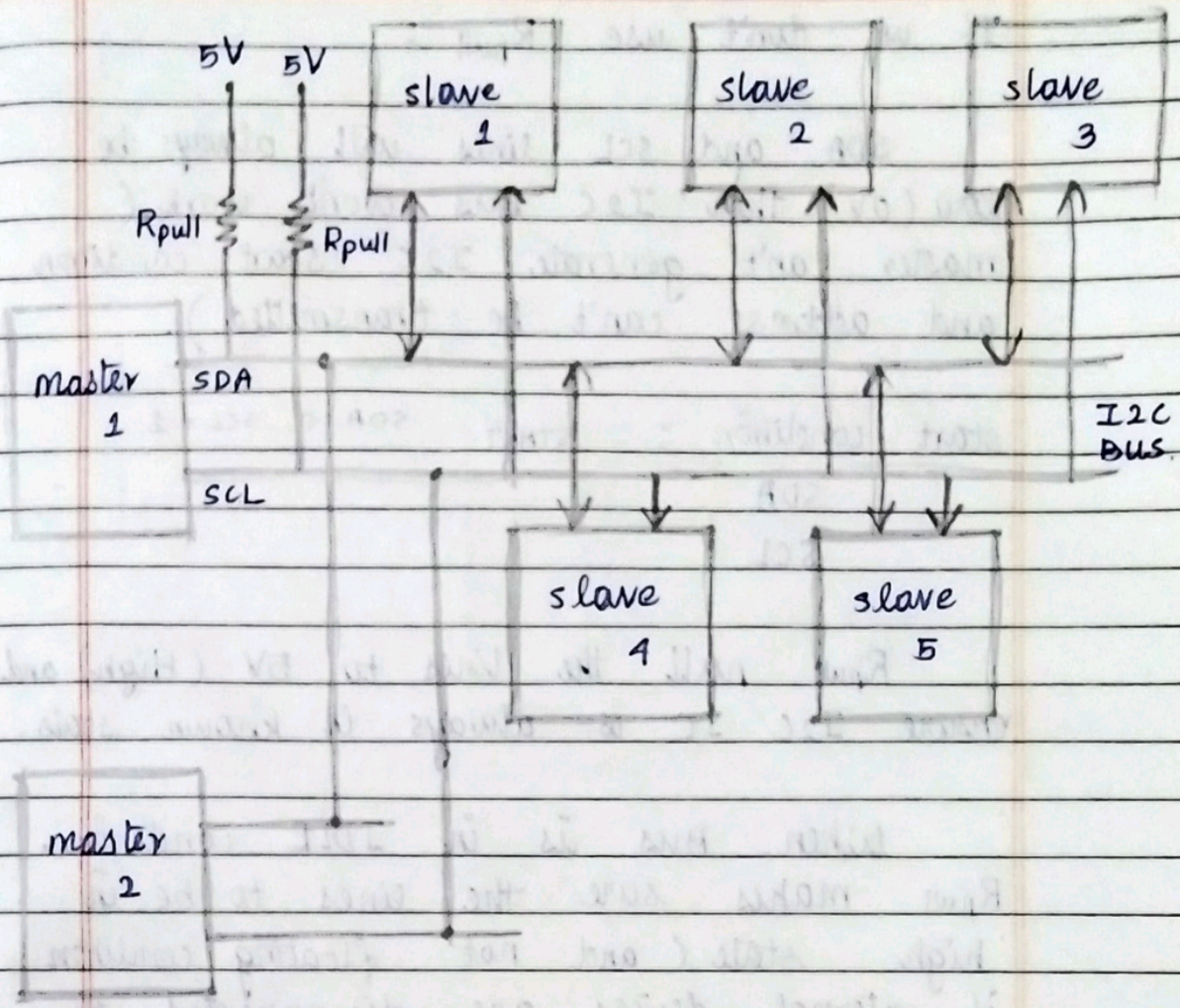
- I2C is slower than SPI (5Mbps, 10 Mbps)
- I2C uses a pull up resistor
- It preferred where large amounts of high speed data is not transferred.

- This bus is used to connect component of system like between CPU and memory, memory and I/O ports or between peripheral devices.

- It has extra start or stop bit (absent in SPI) (overhead bits).

$R_{pull} \Rightarrow$ 1.8 kΩ, 4.7 kΩ, 10 kΩ } commonly used values for pull up resistor

In a multi-master system, at a time usually only one master access the slave.

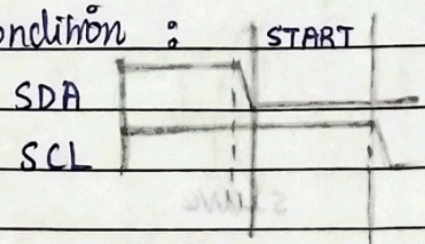


Block Diagram for I2C protocol
- Multi-master and 5 slaves.

If we don't use R_{pull} :

SDA and SCL lines will always be low (0V) then I2C bus won't work (master can't generate I2C 'start condition and address can't be transmitted).

start condition :

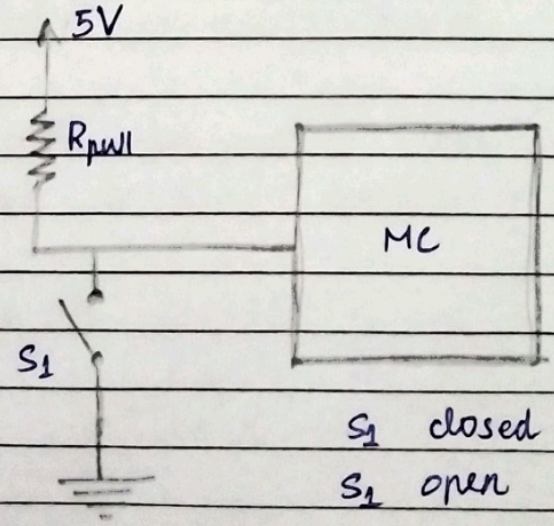


SDA=0, SCL=1.

R_{pull} pull the lines to 5V (High) and ensure I2C IC is always in known state.

When Bus is in IDLE condition, R_{pull} makes sure the lines to be in high state (and not floating condition if external devices are disconnected or high 'z' is introduced.

It prevents undefined state at input.



S_1 closed → gnd

S_1 open → undefined state.

Documentation for a Basic I^{2}C Master Controller

Generated by AI Assistant

December 9, 2025

Abstract

This document provides the technical specification and implementation details for a basic I^{2}C (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 I^{2}C Protocol

The I^{2}C 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 I^{2}C 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 $50\,\text{MHz}$ (period $T_{CLK} = 20\,\text{ns}$), the resulting SCL frequency (f_{SCL}) is calculated as:

$$f_{SCL} = \frac{f_{CLK}}{2 \times \texttt{SCL\_DIV\_COUNT}} = \frac{50\,\text{MHz}}{2 \times 250} = 100\,\text{kHz}$$

This corresponds to the I^2C 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 I^2C Write Transaction

State	Value	Action / Description
`IDLE`	0	Wait for transaction start. SCL/SDA are high.
`START`	1	Generates the **START Condition** (SDA **1** $\rightarrow$ **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** $\rightarrow$ **1** while SCL is **1**).

2.3.1 Transaction Flow

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

2.4 Verilog Code: I2C Master Controller

```
module I2C_1 (
    input  wire        CLK,
    input  wire        RST,
    input  wire [7:0]  data_in,
    input  wire [6:0]  addr_in,
    inout  wire        SDA,
    output reg         SCL
);

reg i2c_sda;
reg i2c_sda_en;      // 0 = drive SDA, 1 = release (Z)
assign SDA = i2c_sda_en ? 1'bz : i2c_sda;

// State + counters
reg [7:0] state;
reg [3:0] count;
reg [7:0] tx_data;

```

```
localparam IDLE      = 0;
localparam START     = 1;
localparam ADDR      = 2;
localparam ACK_WAIT1 = 3;
localparam DATA      = 4;
localparam ACK_WAIT2 = 5;
localparam STOP      = 6;

// Clock division for SCL
localparam SCL_DIV_COUNT = 250;
reg [8:0] scl_count;

always @(posedge CLK) begin
    if (RST) begin
        scl_count <= 0;
        SCL <= 1;
    end else begin
        if (state != IDLE) begin
            if (scl_count < SCL_DIV_COUNT - 1)
                scl_count <= scl_count + 1;
            else begin
                SCL <= ~SCL;
                scl_count <= 0;
            end
        end else begin
            SCL <= 1;
            scl_count <= 0;
        end
    end
end

always @(posedge CLK) begin
    if (RST) begin
        state       <= IDLE;
        i2c_sda     <= 1;
        i2c_sda_en  <= 1;
        count       <= 0;
        tx_data     <= 0;
    end else begin

        if (scl_count == 0) begin
            case(state)

            IDLE: begin
                i2c_sda_en <= 0;
                i2c_sda    <= 1;
                if (SCL == 1) state <= START;
            end

            START: begin
                if (SCL == 1) begin
                    i2c_sda    <= 0;
                    i2c_sda_en <= 0;
                    tx_data    <= {addr_in, 1'b0};
                    count      <= 7;
                    state      <= ADDR;
                end
            end

            ADDR: begin
                i2c_sda_en <= 0; // Drive
                if (SCL == 0) begin
                    i2c_sda    <= tx_data[count];
                end else begin
                    if (count == 0) begin
                        state <= ACK_WAIT1;
                        count <= 7;
                    end
                    else count <= count - 1;
                end
            end

            ACK_WAIT1: begin
```

```
                if (SCL == 0) begin
                    i2c_sda_en <= 1;
                end
                else begin
                    tx_data <= data_in;
                    count   <= 7;
                    state   <= DATA;
                end
            end

            DATA: begin
                i2c_sda_en <= 0; // Drive
                if (SCL == 0) begin
                    i2c_sda   <= tx_data[count];
                end else begin
                    if (count == 0) begin
                        state <= ACK_WAIT2;
                        count <= 7;
                    end
                    else count <= count - 1;
                }
            end

            ACK_WAIT2: begin
                if (SCL == 0) begin
                    i2c_sda_en <= 1;
                end else begin
                    state <= STOP;
                end
            end

            STOP: begin
                if (SCL == 0) begin
                    i2c_sda_en <= 0;
                    i2c_sda    <= 0;
                end else begin
                    i2c_sda    <= 1;
                    i2c_sda_en <= 0;
                    state      <= IDLE;
                end
            end

            default: state <= IDLE;

            endcase
        end
    end
end

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**

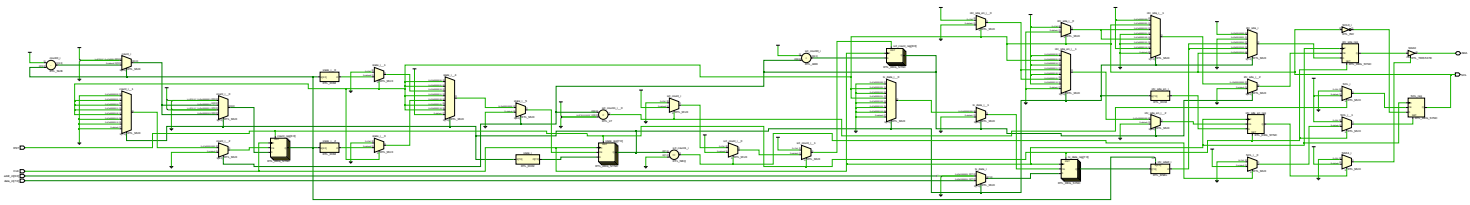
The simulation runs long enough (**192000** ns) to cover the full I^2C transaction at the 100 kHz speed.

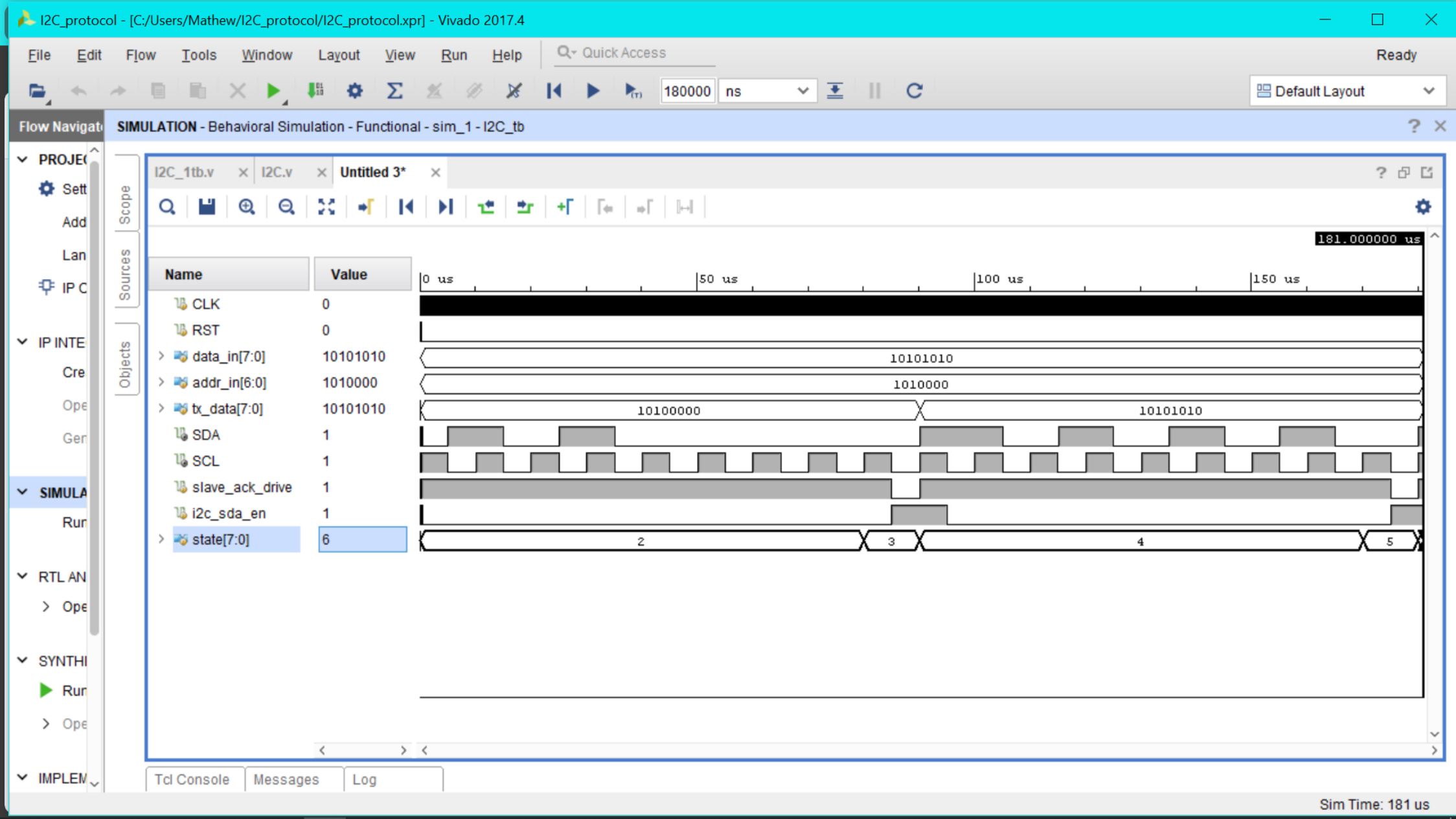
3.3 Verilog Code: I2C Testbench

```
`timescale 1ns / 1ps

module I2C_tb;

reg       CLK;
reg       RST;
reg [7:0] data_in;
reg [6:0] addr_in;

wire SDA;
wire SCL;

reg slave_ack_drive;

// 1. Model Pull-up Resistor for SDA
// This primitive ensures SDA defaults to '1' (high) when released.
pullup (SDA);

assign SDA = (slave_ack_drive == 1'b0) ? 1'b0 : 1'bz;

always @ (posedge CLK) begin
    if (RST) begin
        slave_ack_drive <= 1'b1;
    end
    else begin
        if (dut.state == 3 || dut.state == 5) begin // ACK_WAIT1 or ACK_WAIT2
            if (SCL == 0) begin
                slave_ack_drive <= 1'b0;
            end else begin
                slave_ack_drive <= 1'b1;
            end
        end
        else begin
            slave_ack_drive <= 1'b1;
        end
    end
end


// 3. Instantiate the Device Under Test (DUT)
I2C_1 dut (
    .CLK     (CLK),
    .RST     (RST),
    .data_in (data_in),
    .addr_in (addr_in),
    .SDA     (SDA),
    .SCL     (SCL)
);

// 4. Clock Generator (20 ns period for 50 MHz CLK)
always #10 CLK = ~CLK;

```

```
initial begin
    CLK = 0;
    RST = 1;

    data_in = 8'hAA;
    addr_in = 7'h50;

    $display("====== Simulation starts at %0t ns ======", $time);
    $display("Target Write: Addr=7'h%2h, Data=8'h%2h", addr_in, data_in);

    #100;
    RST = 0;

    #192000;

    $display("====== Simulation ends at %0t ns ======", $time);
    $finish;
end

initial
$monitor("Time=%0t ns | State=%0d | SCL=%0b | SDA=%0b | Master_SDA_En=%0b | Slave_ACK_Drive=%0b",
          $time, dut.state, SCL, SDA, dut.i2c_sda_en, slave_ack_drive);

endmodule
```

Listing 2: I2C Master Testbench (I2C_1tb.v)




I2C_protocol - [C:/Users/Mathew/I2C_protocol/I2C_protocol.xpr] - Vivado 2017.4
File Edit Flow Tools Window Layout View Run Help
Quick Access
Ready
180000 ns
Default Layout
SIMULATION - Behavioral Simulation - Functional - sim_1 - I2C_tb
I2C_1tb.v
I2C.v
Untitled 3*
181.000000 us
Name
Value
0 us
50 us
100 us
150 us
CLK 0
RST 0
data_in[7:0] 10101010
addr_in[6:0] 1010000
tx_data[7:0] 10101010
SDA 1
SCL 1
slave_ack_drive 1
i2c_sda_en 1
state[7:0] 6
10101010
1010000
10100000
10101010
2
3
4
5
Tcl Console
Messages
Log
Sim Time: 181 us