i2C Protocol Simulation Using Xilinx Vivado Design Suite

Bristo C J
B.Tech Electronics and Communication Engineering
College of Engineering Trivandrum

Table of Contents

1.1 Problem Statement....... 2
1.2 Project Objectives....... 2
1.3 Project Overview....... 2
2.1 i2C (Inter Integrated Circuit).......3
2.1.1 UART Protocol :....... 3
2.1.2 SPI Protocol :....... 3
2.1.3 i2C Protocol :....... 4
2.2 Working of i2C.......4
Message Format:.......4
Stop Condition:.......5
Address Frame:.......5
Read/Write Bit:.......5
ACK/NACK Bit:.......5
Addressing:....... 5
Data Frame:....... 5
2.3 Visualisation of working....... 6
3.1 Finite State Machine.......9
3.2 Code development : Xilinx Vivado.......9
3.2.1 Design file.......9
1) Initial stages :.......10
2) write stages :.......10
3) read stages :....... 10
3.2.2 Simulation file....... 10
3.3 Simulation Output....... 11
4.1 Observations....... 12
4.2 Skills acquired.......12
4.3 Difficulties Faced.......12
5.1_Design File....... 13
5.2 Simulation File....... 18
References.......20

Chapter 1

Abstract

1.1 Problem Statement

The i2C (Inter-Integrated Circuit) protocol is a widely used synchronous serial communication standard for connecting various devices in embedded systems. The objective of this project is to design and simulate an I2C protocol using Verilog in Xilinx Vivado.

1.2 Project Objectives

1. Verilog Implementation:
    a. Provide a well-documented Verilog code that represents the I2C protocol.
    b. Include comments to explain the functionality of module and sections.
2. Simulation Results:
    a. Demonstrate the correct execution of the I2C protocol in Vivado simulation using a testbench.
3. Documentation:
    a. Prepare a comprehensive project report .
    b. Include simulation waveforms with relevant observation.

1.3 Project Overview

The simulation project aims to create a visual simulation of the I2C protocol. The key aspects of the project include:

- Simulate the communication flow between the master and slave devices.
- Develop a mechanism for correct addressing of devices on the I2C bus.
- Simulate the transmission of data between the master and slave, considering start and stop conditions.
- Ensure Clock synchronisation between the master and slave devices controlling the external clock frequency.

This project is an opportunity to gain a deep understanding of the I2C protocol and enhance simulation skills using Vivado. Successful completion will demonstrate proficiency in Verilog, Vivado simulation tools, and an understanding of serial communication protocols.

Chapter 2

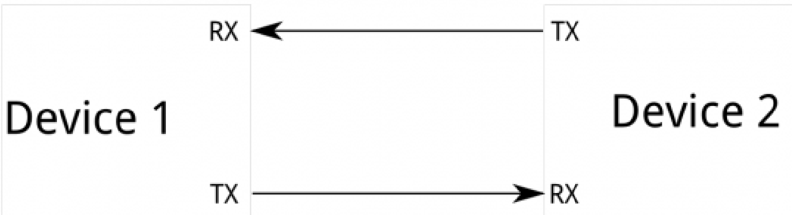
Introduction

2.1 i2C (Inter Integrated Circuit)

The Inter-Integrated Circuit (I2C) Protocol is a protocol intended to allow multiple "peripheral" digital integrated circuits ("chips") to communicate with one or more "controller" chips.

2.1.1 UART Protocol :

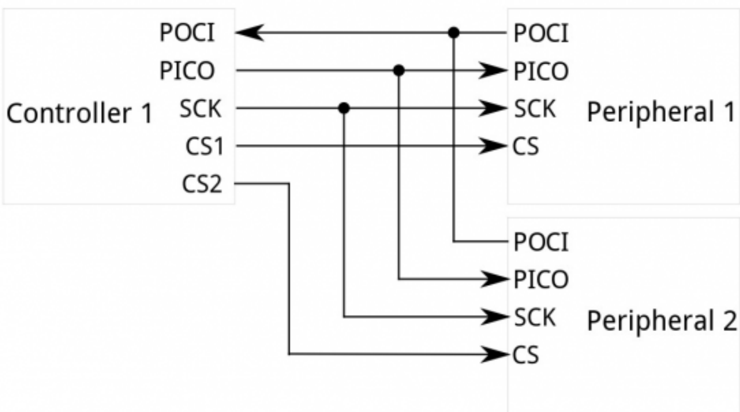
UART serial ports are asynchronous (no clock data is transmitted), devices using them must agree ahead of time on a data rate. The two devices must also have clocks that are close to the same rate, and will remain so--excessive differences between clock rates on either end will cause garbled data



2.1.2 SPI Protocol :

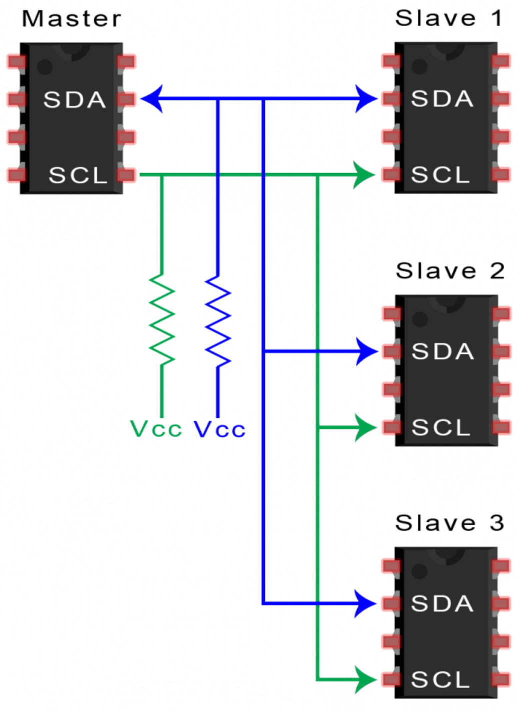
The most obvious drawback of SPI is the number of pins required. Connecting a single controller to a single peripheral with an SPI bus requires four lines; each additional peripheral device requires one additional chip select I/O pin on the controller.

The rapid proliferation of pin connections makes it undesirable in situations where lots of devices must be connected to one controller. Also, the large number of connections for each device can make routing signals more difficult in tight PCB layout situations.



2.1.3 i2C Protocol :

i2C combines the best features of SPI and UARTs. With i2C, you can connect multiple slaves to a single master (like SPI) and you can have multiple masters controlling single, or multiple slaves. This is really useful when you want to have more than one microcontroller logging data to a single memory card or displaying text to a single LCD.



i2C is a standard protocol that is widely supported by a variety of integrated circuits, microcontrollers, and other devices. This broad support makes it a popular choice for communication between different components in electronic systems.

2.2 Working of i2C

With i2C, data is transferred in messages. Messages are broken up into frames of data. Each message has an address frame that contains the binary address of the slave, and one or more data frames that contain the data being transmitted. The message also includes start and stop conditions, read/write bits, and ACK/NACK bits between each data frame:

Message Format:

START	7 or 10 bits Address Frame	READ/ WRITE	8 bits Data Frame	ACK/ NACK	8 bits Data Frame	ACK/ NACK	START

Start Condition:
The SDA line switches from a high voltage level to a low voltage level before the SCL line switches from high to low.

Stop Condition:
The SDA line switches from a low voltage level to a high voltage level after the SCL line switches from low to high.

Address Frame:
A 7 or 10 bit sequence unique to each slave that identifies the slave when the master wants to talk to it.

Read/Write Bit:
A single bit specifying whether the master is sending data to the slave (low voltage level) or requesting data from it (high voltage level).

ACK/NACK Bit:
Each frame in a message is followed by an acknowledge/no-acknowledge bit. If an address frame or data frame was successfully received, an ACK bit is returned to the sender from the receiving device.

Addressing:
The master sends the address of the slave it wants to communicate with to every slave connected to it. Each slave then compares the address sent from the master to its own address. If the address matches, it sends a low voltage ACK bit back to the master.
If the address doesn't match, the slave does nothing and the SDA line remains high.

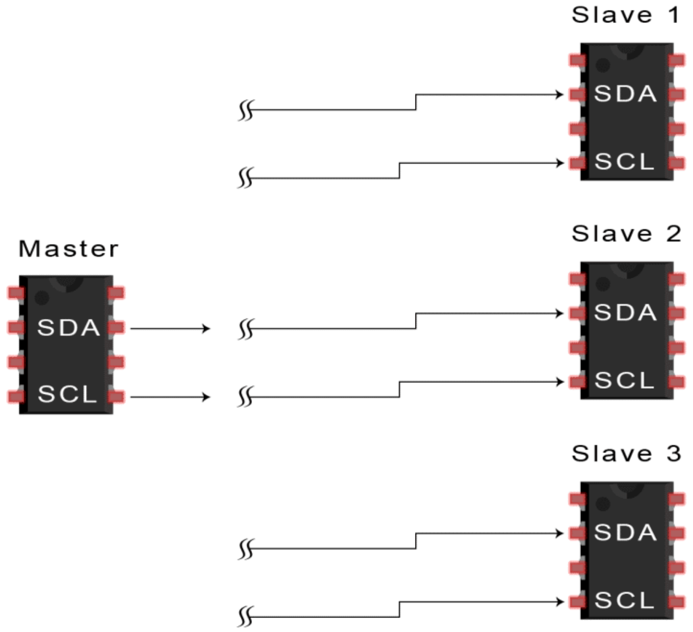
(In code ack pin is high and external inorder for the user to control it during simulation)

The address frame includes a single bit at the end that informs the slave whether the master wants to write data to it or receive data from it. If the master wants to send data to the slave, the read/write bit is a low voltage level. If the master is requesting data from the slave, the bit is a high voltage level

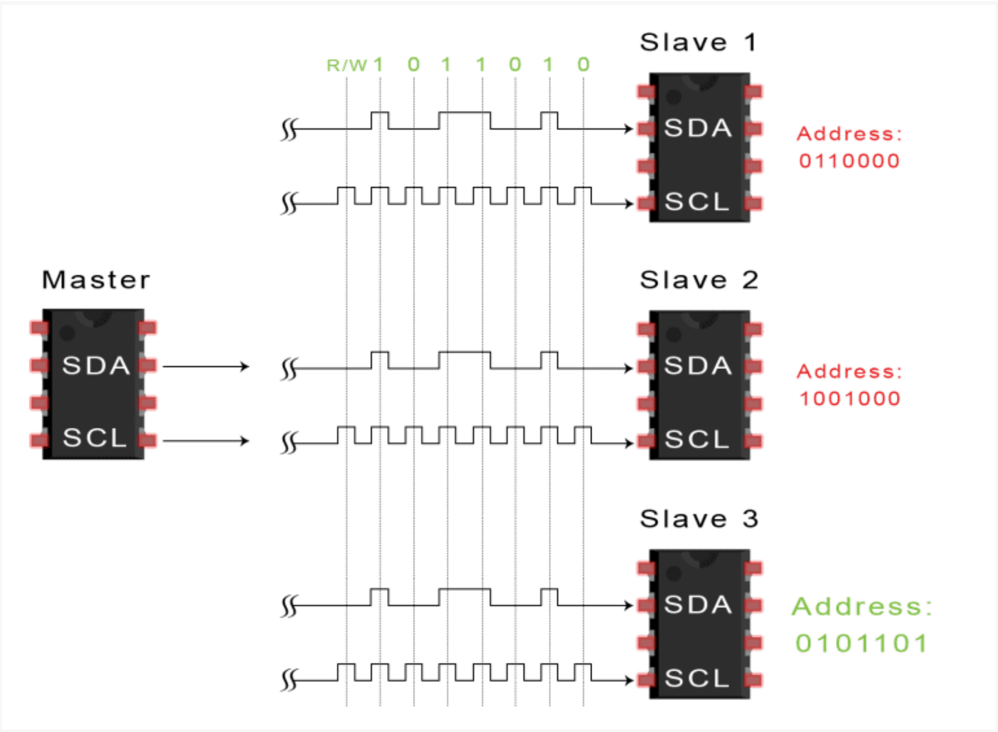
Data Frame:
After the master detects the ACK bit from the slave, the first data frame is ready to be sent.
The data frame is always 8 bits long, and sent with the most significant bit first. Each data frame is immediately followed by an ACK/NACK bit to verify that the frame has been received successfully.

2.3 Visualisation of working

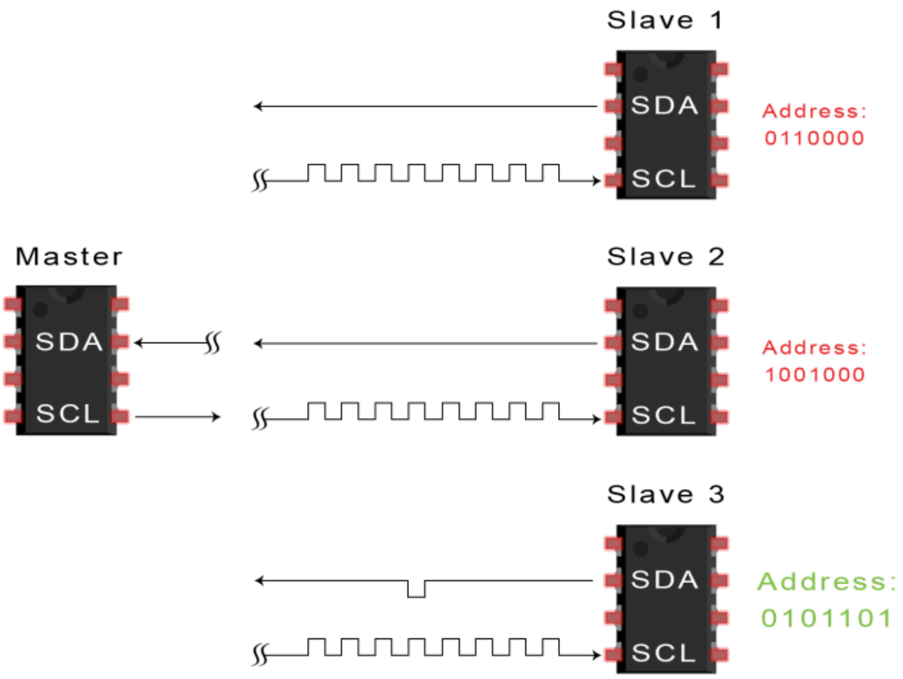
The master sends the start condition to every connected slave by switching the SDA line from a high voltage level to a low voltage level before switching the SCL line from high to low



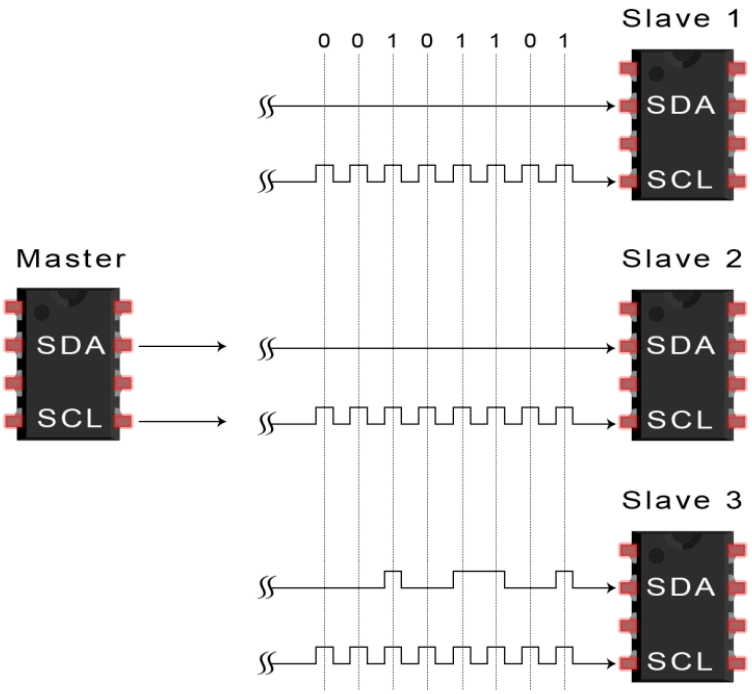
The master sends each slave the 7 or 10 bit address of the slave it wants to communicate with, along with the read/write bit



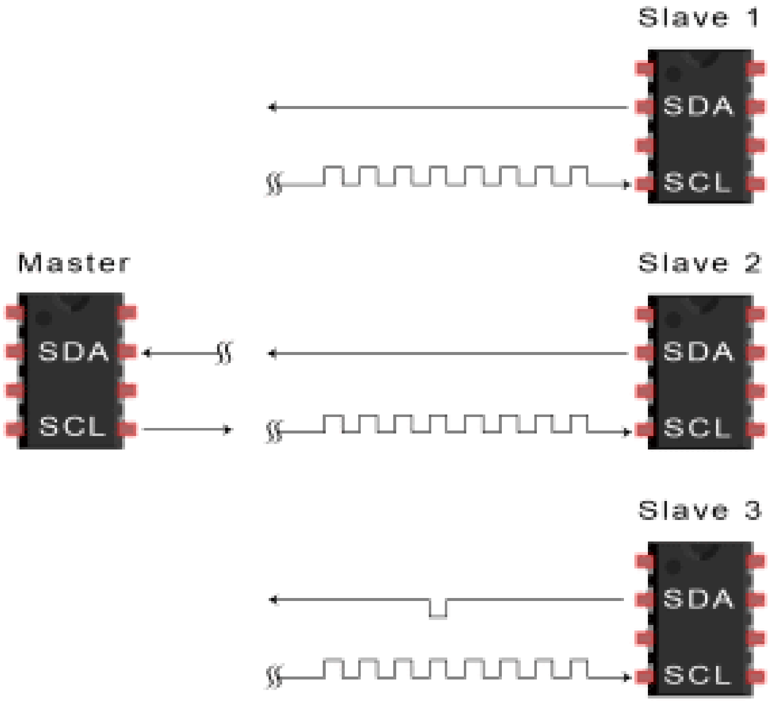
Each slave compares the address sent from the master to its own address. If the address matches, the slave returns an ACK bit by pulling the SDA line low for one bit. If the address from the master does not match the slave's own address, the slave leaves the SDA line high.



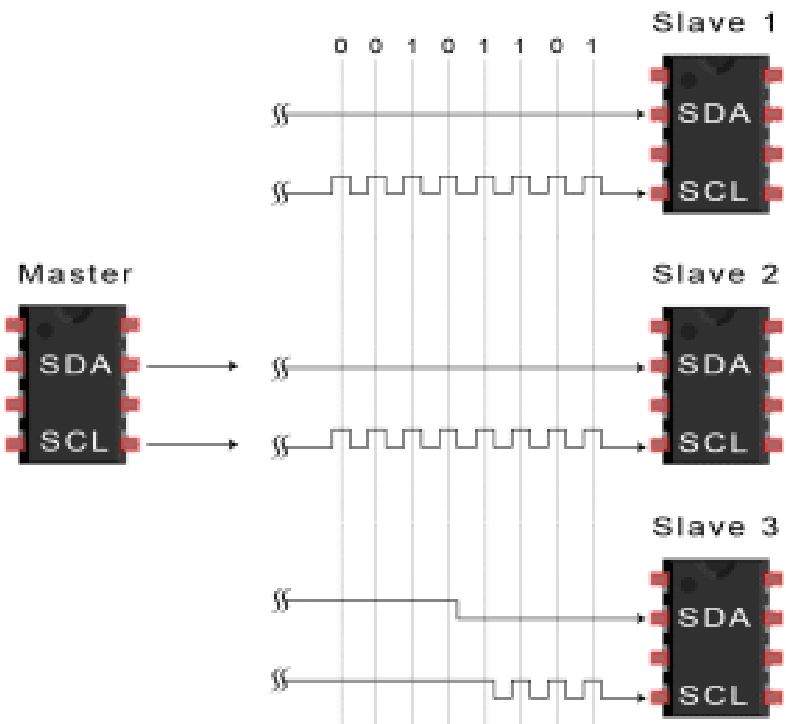
The master sends or receives the data frame:



After each data frame has been transferred, the receiving device returns another ACK bit to the sender to acknowledge successful receipt of the frame:



To stop the data transmission, the master sends a stop condition to the slave by switching SCL high before switching SDA high:

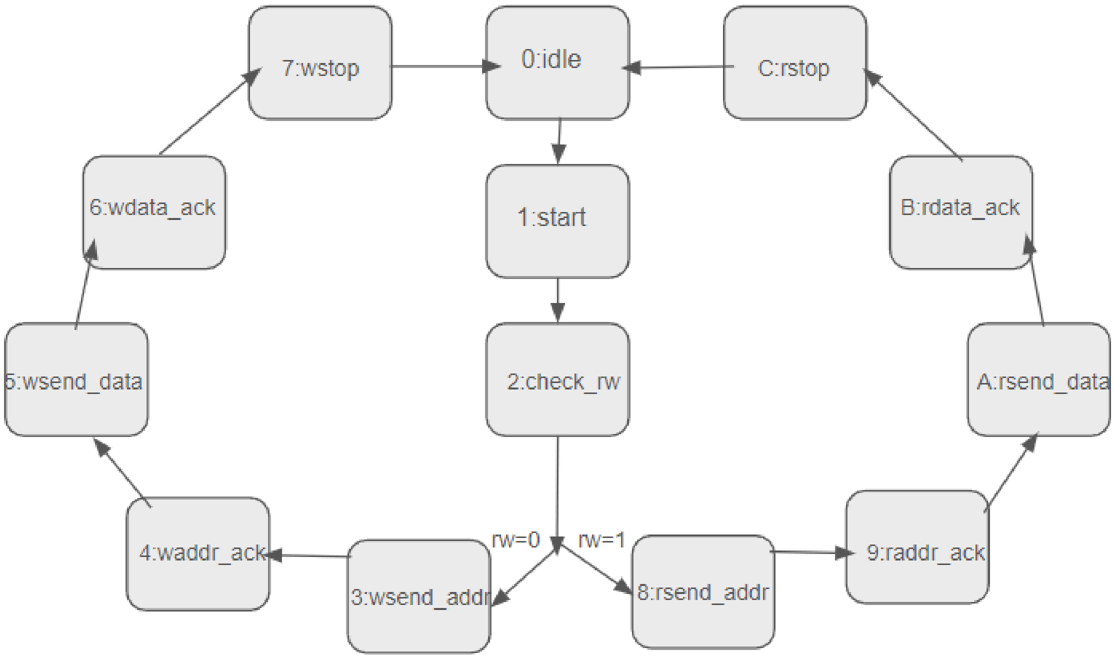


Chapter 3

Design and Simulation

3.1 Finite State Machine

The FSM of this simulation includes 13 states which can follow 2 paths depending whether the data is to be written to the slave or read from the slave. Upon completion on any one of it, it loops back to the idle condition.



3.2 Code development : Xilinx Vivado

3.2.1 Design file

The design file is a Verilog file which contains a single module which consists of the input,output and inout pins which are required to execute the simulation as per the i2C protocol.

There are 2 Signals which determine the flow of transmission:

- SCL : Downclocked Clock signal [FPGA clock _ 100MHz,Required clock _ 400KHz]
- SDA : Data signal

1) Initial stages :

Idle : SDA and SCL are initialised to high.

start : SDA is 0, SCL is kept High.This calls for the START condition.addrt is formed by combining {addr : rw}.

check_rw : rw signal is checked . If rw = 0 :write state , rw = 1 :read state.

2) write stages :

wsend_addr : Address transfer through SDA from addrt.

waddr_ack : Acknowledgement of Address transfer.

wsend_data : Data transfer to SDA from wdata.

wdata_ack : Acknowledgement of Data transfer,SDA = 0,SCL = 1.

wstop : SDA is 1,SCL is kept High.Done=1.This calls for the STOP condition

3) read stages :

rsend_addr : Address transfer through SDA from addrt.

raddr_ack : Acknowledgement of Address transfer.

rsend_data : Data transfer from SDA to rdata.

rdata_ack : Acknowledgement of Data transfer,SDA = 0,SCL = 1.

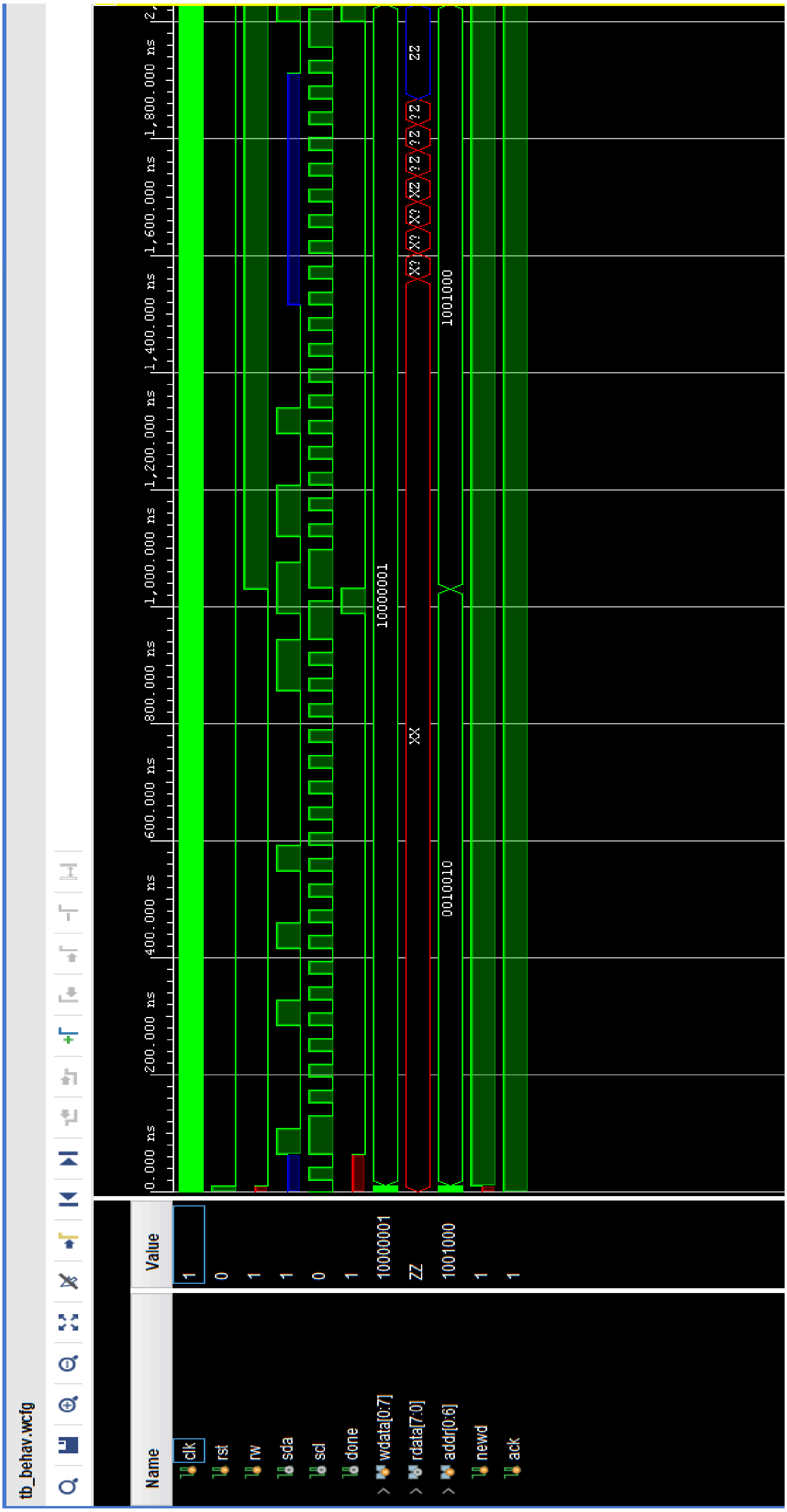
rstop : SDA is 1,SCL is kept High.Done=1.This calls for the STOP condition.

We use a sda_en register inorder to control SDA. If kept high SDA = sdat, else SDA = high impedance state

3.2.2 Simulation file

Testbench of the project is written in such a way as to input random addresses and data to ensure the design file is executing irrespectively of the values on said registers. Testbench is used inorder to input stimulus to the design module without manual labour. It also helps in verification of the results and to observe the output waveforms.

3.3 Simulation Output



Chapter 4

Conclusion

4.1 Observations

The proper working of the signals SDA and SCL is observed .The address and data are transmitted through the SDA as per the clock cycles.

During the first phase of execution the rw (read/write) signal is low which indicates it's in write state. Then the SDA line follows the address and then the data to be written alongside the acknowledgement which is kept high for ease of execution.

In the second phase the rw (read/write) signal is high which indicates it's in read state. The SDA line still transmits the address while during the data transmission region it shows high impedance as memory is not provided to the module.

4.2 Skills acquired

The project was focused on gaining knowledge about the design and simulation sources in Verilog and how to execute them.During the process I acquired a good knowledge in :

- i2C protocol.
- Usage of Vivado for implementing Verilog codes.
- Writing Testbenches.
- Analysing Simulation results.

Along with skills like proper usage of internet to obtain information regarding the topics,preparation of reports etc:

4.3 Difficulties Faced

The few difficulties faced were:

- Analysing the required inputs , outputs , and in-program registers
- FSM preparation
- Test bench timing were done by trial and error

Chapter 5

Code

5.1_Design File

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: College of Engineering Trivandrum
// Student: Bristo C J
//
// Create Date: 10.11.2023 00:45:07
// Design Name: i2C_protocol_1.1
// Module Name: main
// Project Name: i2C protocol
// Target Devices: none
// Tool Versions: 2021.1
// Description: i2C Protocol simulation using verilog
//////////////////////////////////////////////////////////////////////////////////
//
module main
  (
 input clk,rst,ack,rw,scl,newd,  //newd = 1:whenever a new data is incoming
 inout sda,  //in-write, out-read
 input [7:0] wdata,   //8 bit write data
 input [6:0] addr,  //7 bit address of slave
 output reg [7:0] rdata,  //8 bit read data
 output reg done // done indicator
  );

 reg sda_en = 0; //1(write):sda=dat 0(read):sda=High Impedance
 reg sclt, sdat, donet;  //temporary in-program usage
 reg [7:0] rdatat;  //read data temp storage
 reg [7:0] addrt;  //8-bit  7-bit addr : 1-bit r/w

 reg [3:0] state;  //13 states

 parameter
 idle = 0,  //initial stage
 start = 1,  //start operation
 check_rw = 2,  //check rw signal
 wsend_addr = 3,  //send address for write
 waddr_ack = 4,  //write address acknowledgment
 wsend_data = 5,  //send data for write
```

```
 wdata_ack = 6,  //write data acknowledgment
 wstop = 7,  //stop write
 rsend_addr = 8,  //send address for read
 raddr_ack = 9,  //read address acknowledgment
 rsend_data = 10,  //send data for read
 rdata_ack = 11,  //read data acknowledgment
 rstop = 12 ;  //stop read

 reg sclk_wr = 0; //Actual slower clock (except when
start-writing,stop-writing,stop-reading)
 integer i,count = 0;

  //Slower clock generation
  always@(posedge clk)
    begin
      if(count <= 9)
        begin
           count <= count + 1;
        end
      else
         begin
           count  <= 0;
           sclk_wr  <= ~sclk_wr;
         end
    end


  //FSM
  always@(posedge sclk_wr, posedge rst)
    begin
      if(rst == 1'b1)
         begin
           sclt  <= 1'b0;
           sdat  <= 1'b0;
           donet <= 1'b0;
         end
       else begin
         case(state)
           idle :
           begin
              sdat <= 1'b0;
              done <= 1'b0;
              sda_en  <= 1'b1;
              sclt <= 1'b1;
              sdat <= 1'b1;
             if(newd == 1'b1)
                state  <= start;
```

```
     else
        state  <= idle;
   end

    start:
    begin
      sdat  <= 1'b0;
      sclt  <= 1'b1;
      state <= check_rw;
      addrt <= {addr,rw};
    end

    check_rw: begin //addr remain same for both write and read
      if(rw)
         begin
         state <= rsend_addr;
         sdat <= addrt[0];
         i <= 1;
         end
       else
         begin
         state <= wsend_addr;
         sdat <= addrt[0];
         i <= 1;
         end
    end

 //write state

   wsend_addr : begin
              if(i <= 7) begin //7 bit address
              sdat  <= addrt[i];
              i <= i + 1;
              end
              else
                begin
                  i <= 0;
                  state <= waddr_ack;
                end
            end


   waddr_ack : begin
     if(ack) begin
       state <= wsend_data;
       sdat  <= wdata[0];
       i <= i + 1;
```

```
      end
    else
      state <= waddr_ack;
  end

wsend_data : begin
  if(i <= 7) begin
     i     <= i + 1;
     sdat  <= wdata[i];
  end
  else begin
     i     <= 0;
     state <= wdata_ack;
  end
end

 wdata_ack : begin
    if(ack) begin
      state <= wstop;
      sdat <= 1'b0;
      sclt <= 1'b1;
      end
    else begin
      state <= wdata_ack;
    end
   end



wstop: begin
     sdat  <=  1'b1;
     state <=  idle;
     done <=  1'b1;
end

//read state


 rsend_addr : begin
           if(i <= 7) begin
            sdat  <= addrt[i];
            i <= i + 1;
            end
            else
              begin
                i <= 0;
                state <= raddr_ack;
```

```
               end
           end


  raddr_ack : begin
    if(ack) begin
      state  <= rsend_data;
      sda_en <= 1'b0;
    end
    else
      state <= raddr_ack;
  end

rsend_data : begin
          if(i <= 7) begin
                i <= i + 1;
                state <= rsend_data;
                rdata[i] <= sda;
             end
             else
               begin
                 i <= 0;
                 sda_en <= 1'b1;
                 state <= rdata_ack;
               end
end

rdata_ack : begin
    if(ack) begin
      state <= rstop;
      sdat <= 1'b0;
      sclt <= 1'b1;
      end
    else begin
      state <= rdata_ack;
    end
   end


rstop: begin
     sdat  <=  1'b1;
     state <=  idle;
     done  <=  1'b1;
     end
```

```
        default : state <= idle;
        endcase
        end
  end

 assign scl = (( state == start) || ( state == wstop) || ( state == rstop)) ?
sclt : sclk_wr;
 assign sda = (sda_en == 1'b1) ? sdat : 1'bz;
endmodule
```

5.2 Simulation File

```
`timescale 1ns / 1ps
/////////////////////////////////////////////////////////////////////////////////
// Company: College of Engineering Trivandrum
// Student: Bristo C J
//
// Create Date: 10.11.2023 00:45:07
// Design Name: i2C_protocol_1.1
// Module Name: tb
// Project Name: i2C protocol
// Target Devices: none
// Tool Versions: 2021.1
// Description: i2C Protocol simulation using verilog
/////////////////////////////////////////////////////////////////////////////////
//
module tb();
reg clk=1'b0,rst,ack=1'b1,rw,newd;
wire sda,scl,done;
reg [7:0] wdata;
wire [7:0] rdata;
reg [6:0] addr;

//reset gen
task reset;
    begin
        rst=1'b1;
        #10
        rst=1'b0;
    end
endtask

//write function
task write;
```

```
    begin
        newd = 1'b1;
        addr = $random;
        rw = 1'b0;
        wdata = $random;
        #1000;
    end
endtask

//read function
task read;
    begin
        newd = 1'b1;
        rw = 1'b1;
        addr = $random;
        #1000;
    end
endtask



//instantiating module
main main1(clk,rst,ack,rw,scl,newd,sda,wdata,addr,rdata,done);

initial begin
clk=1'b0;
ack=1'b1;
end

//clk gen
always #1 clk=~clk;

initial begin
    reset();
    write();
    #20
    read();
    $finish();
end

initial begin
$monitor("sda = %b,scl = %b",sda,scl);
end

endmodule
```

References

1. Basics of the I2C Communication Protocol

2. Verilog for an FPGA Engineer with Xilinx Vivado Design Suite | Udemy