TEC de Monterrey®

DEL SISTEMA TECNOLÓGICO DE MONTERREY

TE3059 Embedded Systems

FIRST PARTIAL PROJECT

Hardware Specifications and Design

SEPTEMBER 20, 2020

Revision History

Rev. #	Date	Changes By	Description
0.1	09/09/20	Vanessa Jaime	Made a draft of the block diagrams.
0.2	09/11/20	Isaac Ipenza	Made a draft of the state machine.
0.3	09/13/20	Vanessa Jaime	Completed state machine.
0.4	09/14/20	Gabriela Hernandez	Star coding i2c.
0.4	09/16/20	Isaac Ipenza	Start document and completed introduction and specifications.
0.5	09/19/20	Isaac Ipenza	Finished document.
1.0	09/19/20	Gabriela Hernandez	Finished coding i2c.

Table of Contents

Table of Figures

1.0 Introduction

I2C is a two-wire, bi-directional serial bus that provides a simple and efficient method of data exchange between devices. It is most suitable for applications requiring occasional communication over a short distance between many devices. The I2C standard is a tru multi-master bus including collision detection and arbitration that prevents data corruption if two or more masters attempt to control the bus simultaneously.

The general goal of this project is to design and implement up to simulation stage a I2C master controller using our previous knowledge of Verilog HDL, this simulation should have it's own test bench code that will be based on a I2C slave.

The complete project consists of a I2C master verilog code, all the verilog codes needed to make it work, the testbench code for the project with a .do code that allows us to test it faster and finally the specifications document.

1.1 Scope

In this document you will find all the specifications of the project, and most importantly all the design documentation and how this was implemented, here you can read how step by step we carry out the project until we reach the results the specifications ask us. The document contains the different diagrams needed to understand how this project works, and how it is connected internally.

1.2 Intended Audience

This document is focused on reaching an audience that has an interest in knowing more about the I2C controller, and how it can be implemented using Verilog HDL.

1.3 Terminology

Acronym	Description
ADC	Analog to Digital Converter
DAC	Digital to Analog Converter
DDR	Double Data Rate
DI	Data Invalid
DMA	Direct Memory Access
DSP	Digital Signal Processing
EMIO	Extended Multiplexed I/O
FPGA	Field Programmable Gate Array
GEM	Gigabit Ethernet MAC
GMII	Gigabit Media Independent Interface
GUI	Graphical User Interface
HDL	Hardware Description Language
HLS	High Level Synthesis
I2C	Inter-Integrated Circuit (Pronounced I-squared-C)
LEA	Low Energy Analog
LED	Light Emitting Diode

LS	Light Source
LSB	Least Significant Bit
LVDS	Low-Voltage Differential Signaling
MAC	Medium Access Control
Mbps	Mega Bits per Second
MII	Media-Independent Interface
MMCM	Mixed-Mode Clock Manager
MMS	Manufacture Messaging Specification
MSB	Most Significant Bit
NCO	Numerically Controlled Oscillator
OTR	Out of Range
PC	Personal Computer
PD	Photo Detector
PHY	Physical Layer
PL	Programmable Logic
PLL	Phase Locked Loop
PPS	Pulse Per Second
PS	Processing System
RMII	Reduced Media Independent Interface
Rx	Reception
SFP	Small Form-Factor Pluggable Transceptor
SoC	System on Chip
SoW	Statement of Work
Tx	Transmission

Table 1 Acronyms

2.0 Project Requirements

I2C is a serial communication protocol to help data to be transferred bit by bit along a single wire. I2C is synchronous so the output of bits is synchronized to the sampling of bits by a clock signal shared by the master and the slave. Data is transferred in messages which are broken up into frames of data. Each message has a start condition, a stop condition, address frame (which is a unique sequence that identifies the slave to the master), read/write bit and ACK/NACK Bit.

Initially the master will come in master transmit mode and start sending a start bit and after that the 7-bit address of the slave device it wishes to communicate which is followed by a single bit (the 8th bit) representing whether it wishes to write(0) to or read(1) from the slave device. If any slave with the sent address exists in the bus line then it will respond with an ACK bit [1] (active low for acknowledged) for that address sequence. The master then stays in either transmit or receive mode (as per the read/write bit it sent) while slave continues with the complementary mode (receive or transmit, respectively). The address and the data bytes are sent as MSB (most significant bit) first.

2.1 Block diagram

High-level block diagram of the system requirements and description of the system component

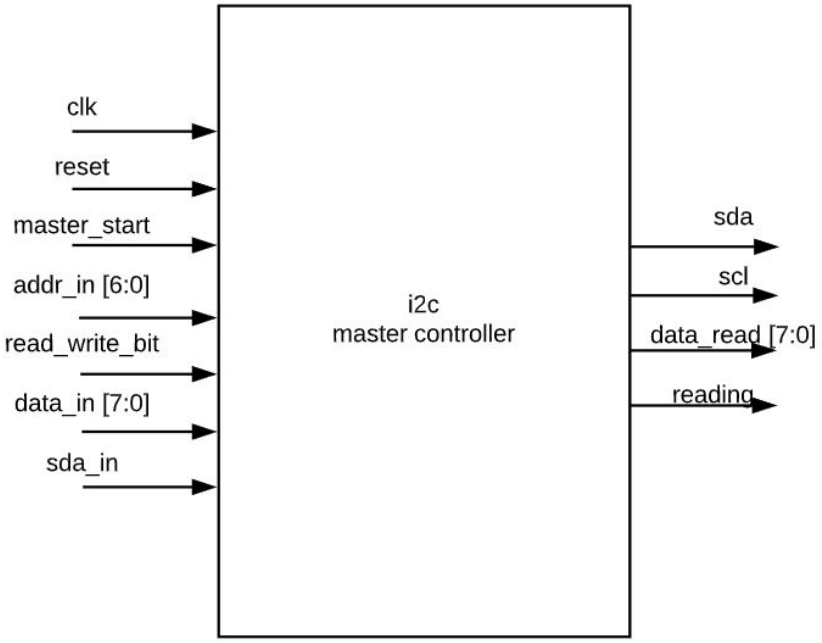

Figure 1 - I2C High Level Diagram

2.2 Overall functionality

The I2C protocol comprises a set of conditions to establish communication between the devices. These include:

1. Start condition
2. Sending of the slave address along with read/write information by the master
3. Acknowledgement by the slave device
4. Data transmission between the master and the slave device
5. Stop condition

3.0 Design Description

Our design is made up of seven inputs and 4 outputs which are described below:

Type	Name	Description
Input	clk	Main clock of the system
	reset	Reset signal that sets main signals to 0 whenever it's equal to 1
	master_start	Start signal for the fsm to start it's course. It's expected to come from the slave. Master part of the name it's to avoid being confused with the start state inside the fsm.
	addr_in	Address of the slave from which we want to read from or write to. Size of 7 bits.
	read_write_bit	This is a one sized signal bit which will tell us if we want to read from a slave or write to (0 - write / 1 - read).
	data_in	Data we want to write to the previously specified address. This value is not required whenever we want to read from the slave. Size of 8 bits.
	sda_in	Data the controller is supposed to get from the slave 1 bit at a time.
Output	sda	The master controller will try to send to the slave one bit at a time following the data_in signal.
	scl	Serial clock line. It's value will follow several signals differently from the main clock, which will perpetually run.
	data_read	This signal will show the data that was read from the slave on its final size of 8 bits.
	reading	This signal was included in our project for simulation purposes. It'll only become 1 when the fsm is going through the read state.

3.1 Intricacies of the I2C controller

The main intricacies we found while designing and implementing this controller were determining how the scl signal, which is also a clock but not the main one, should behave. It was something that took lots of steps and try and error. We ended up by adding extra states just to control this signal so it'd be easy for the rest of the signals to set as well, before going into the next step.

Another intricacy during the development of this controller was the sda signal, which is supposed to act as an input as well as an output. Although there's indeed a type signal that works as this inside the Verilog environment which is called inout, we discovered it was more complex than wished to use this type so we ended up choosing to add another input called sda_in which acts as the signal sda would be carrying from the slave and leave the signal sda as the main output.

3.2 Hardware architecture at signal level

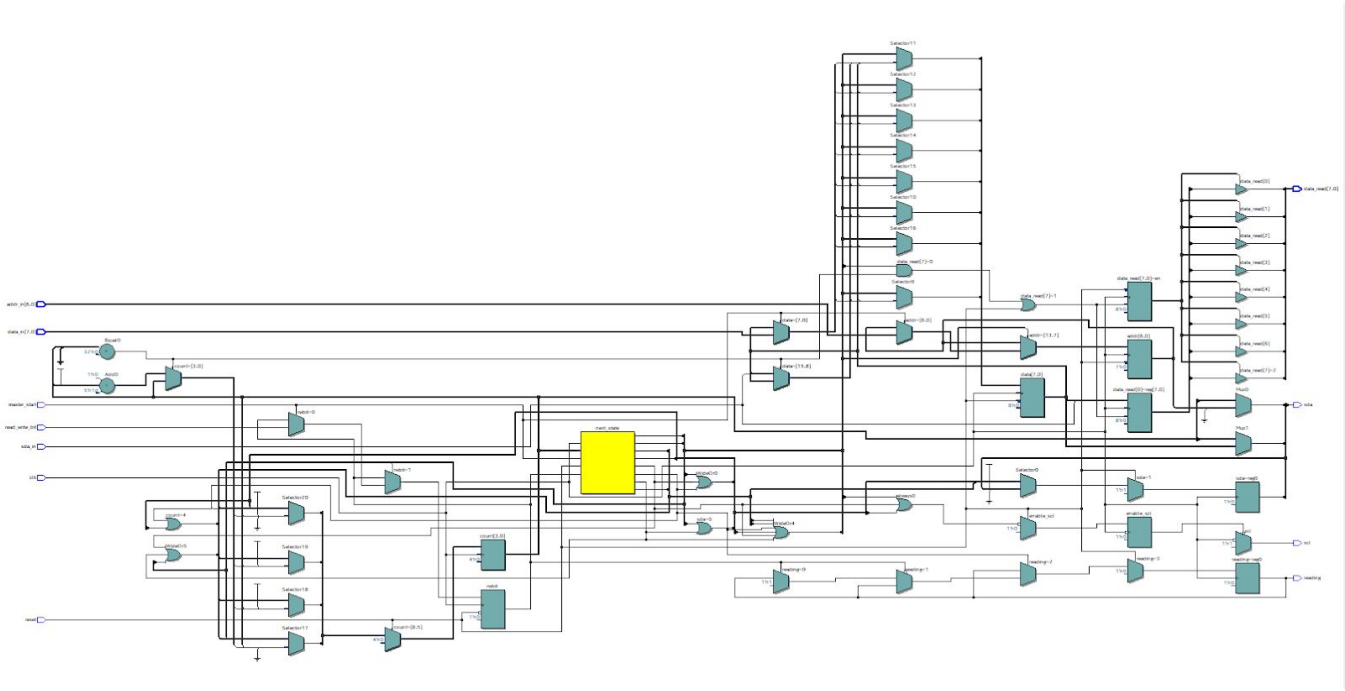

Figure 2 - I2C Hardware at signal level

3.3 FSM controllers

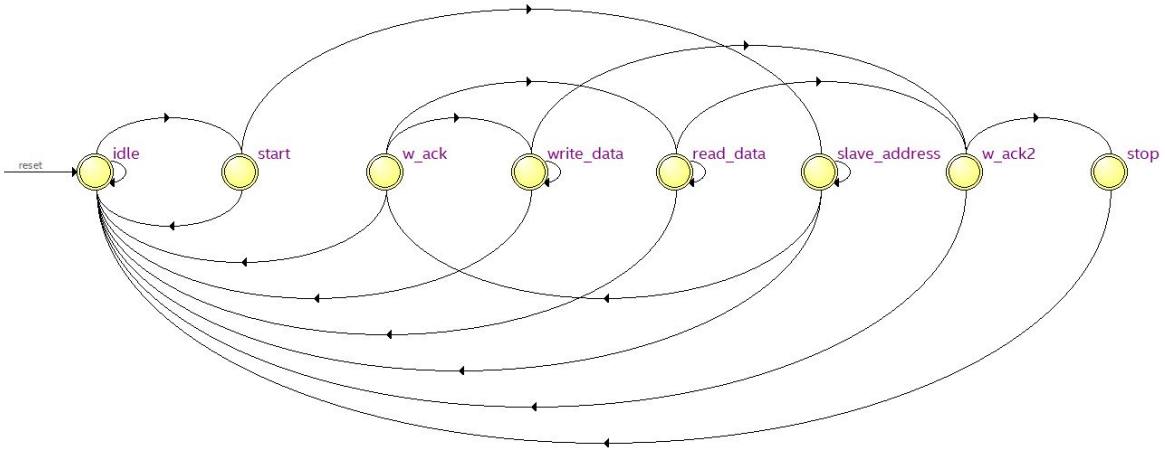

Figure 3 - I2C High FSM

4.0 Test and Test Results

4.1 Test Strategy

This project is tested with the code that generates the clk signal and that will be giving the necessary values to the reset and read_write_bit signals. One important thing to be noticed in this testbench is that we're emulating how a slave would act since there are no real ones that will send data back. That's why a function is implemented, and an internal variable named i and rd_data are also included.

```
always @ (negedge clk)
        if(read_write_bit==1 && reading==1)begin
            sda_in = rd_data[i];
            if(i==0) begin
                sda_in = rd_data[0];

            end
            else begin
                i = i-1;
            end
        end
        else begin
            i = 4'b0111;
        end
```

The previous function is implemented in order to simulate a slave sending back data to the i2c master controller. We take the signal reading implemented just for simulation purposes, which comes from the read state. Inside the fsm in our main module. We then proceed to send the data 1 bit by bit through the sda_in which is intended to emulate a bidirectional main signal sda which in this case we implemented separately (one as an output, one as an /input). We send the data to our controller and it gives us back the signal data_read in the end.

4.2 Test Results

In this project we did a test bench in order to test it. This test bench is called i2c_tb.v. In this test bench we enter some important inputs as reset and master_start in order to simulate our multiplier. We also set the clk to ticks every certain time. There are three inputs addr_in, data_in and read_write_bit. The outputs to be check are: data_read and reading.

4.2.1 Synthesis results

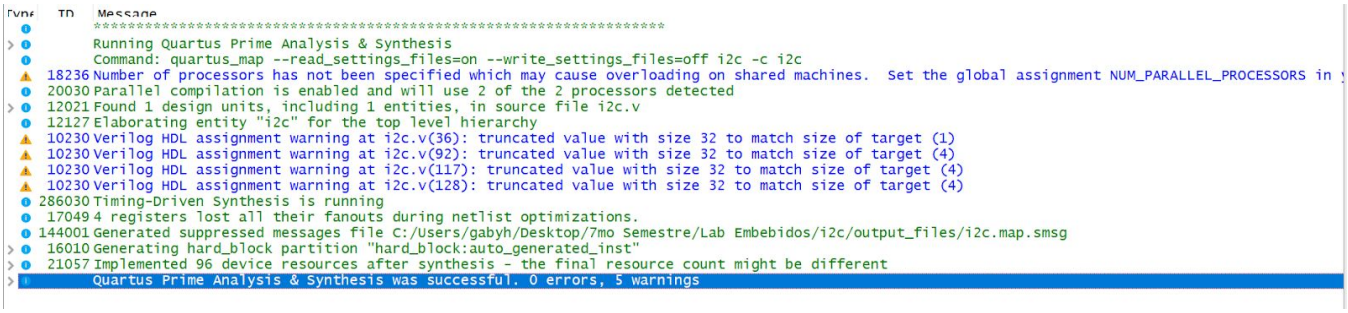

Figure 4 - I2C code Synthesis result

4.2.2 Simulation results

```
reset=1;
sda_in=0;
#10;
master_start=1'b1;
reset=0;
addr_in=7'h52;
data_in=8'haa;
read_write_bit = 0;
```

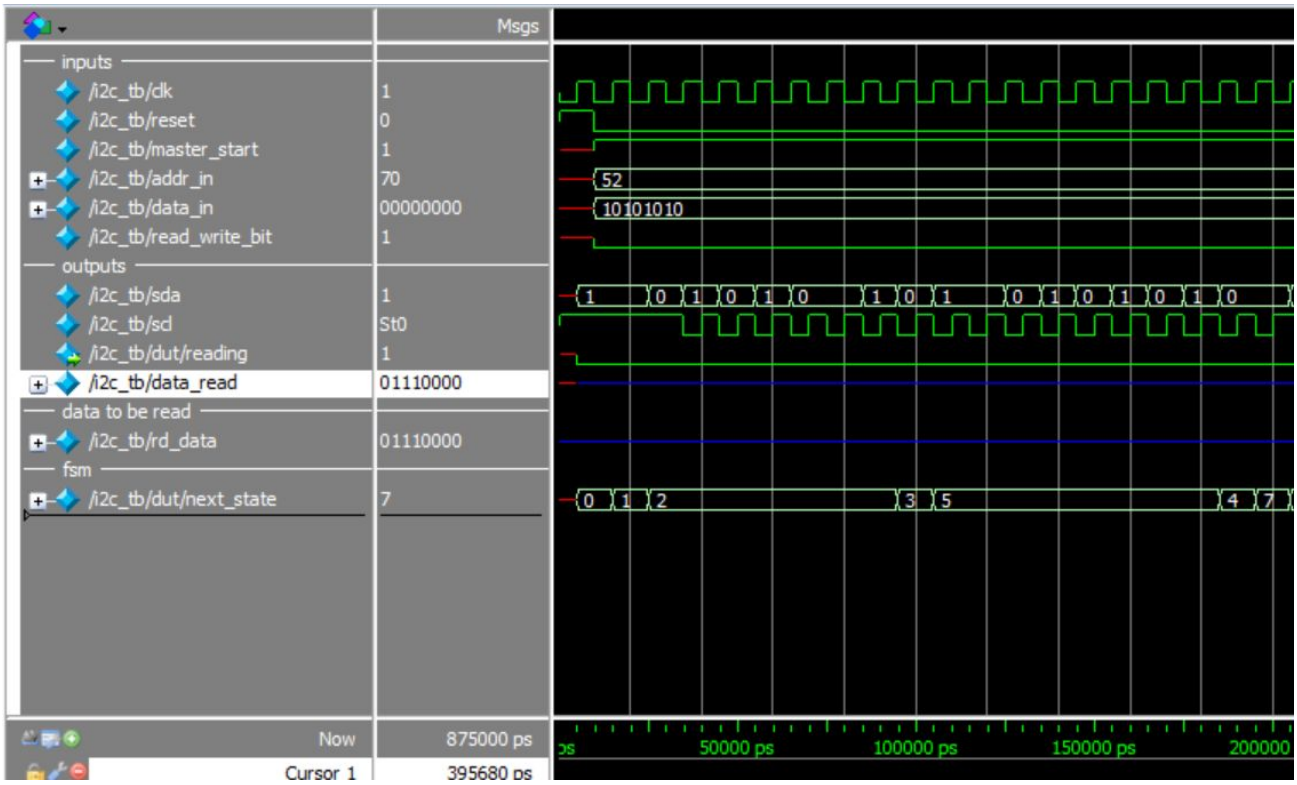

Figure 5 - I2C Simulation result 1

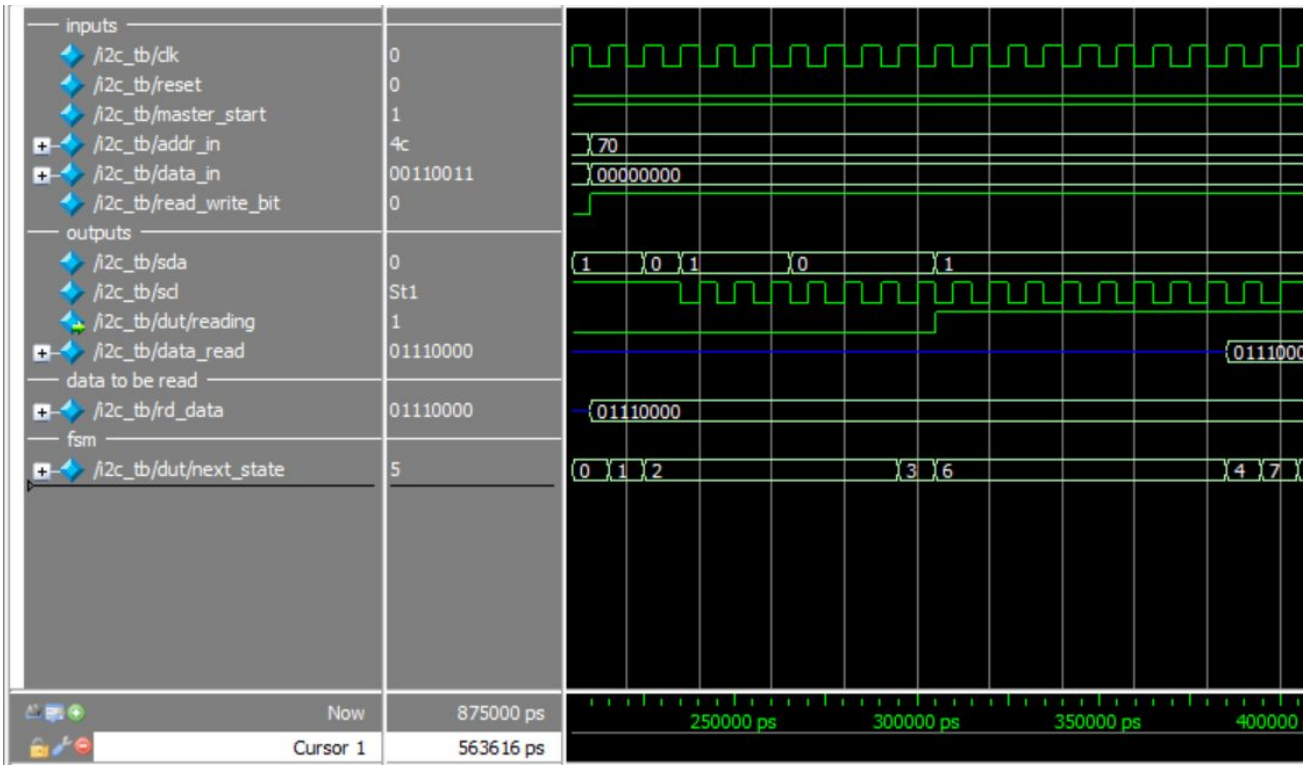

Figure 6 - I2C Simulation result 2

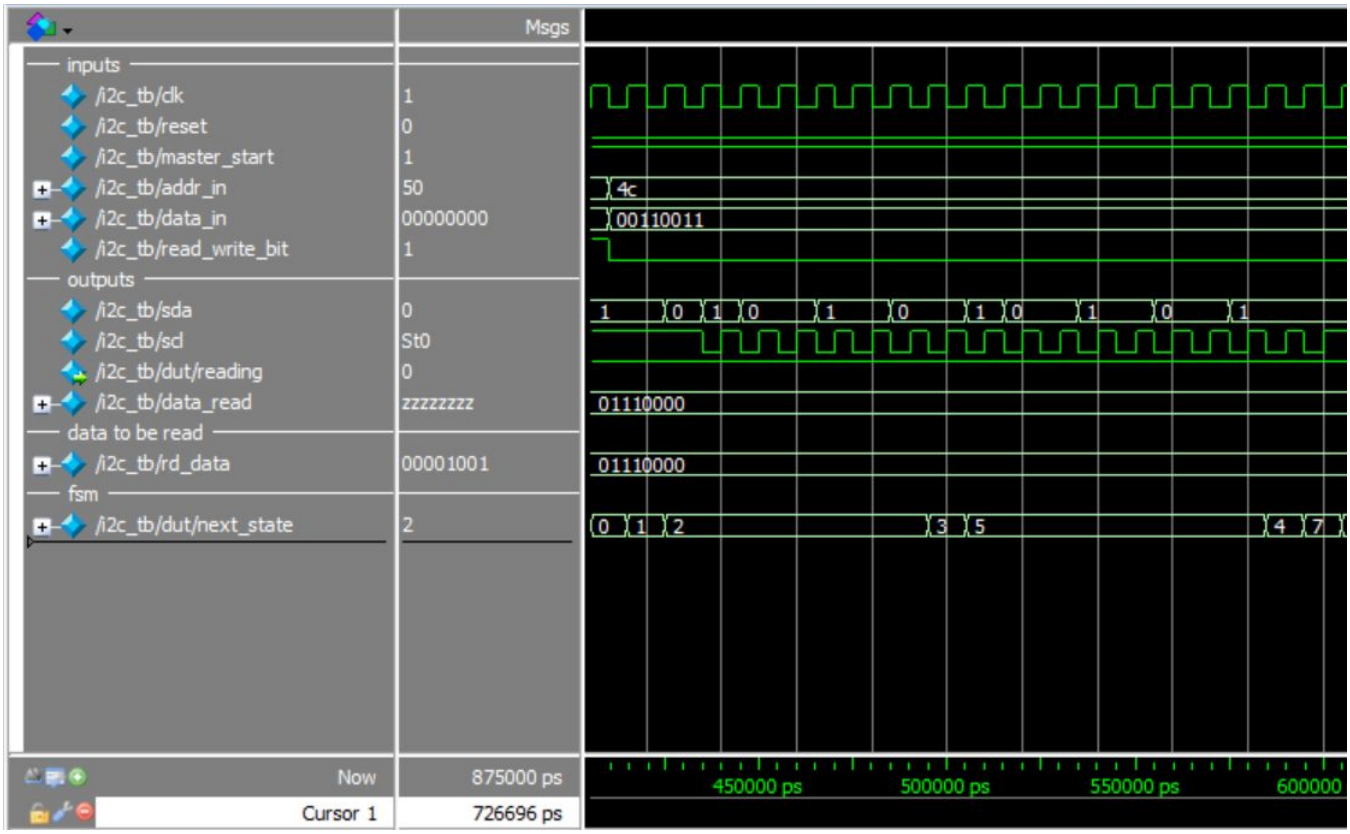

Figure 7 - I2C Simulation result 3

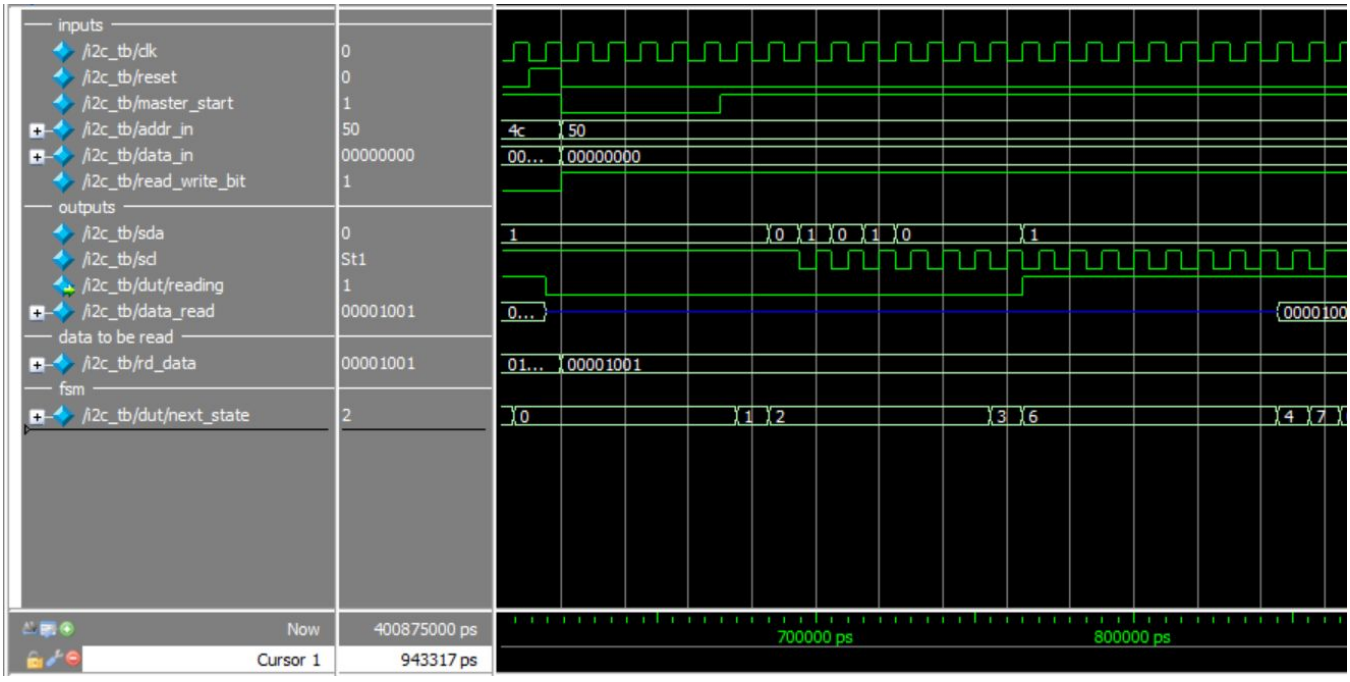

Figure 8 - I2C Simulation result 4