CENTRE FOR WIRELESS SYSTEM DESIGN (C-WiSD)

&

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

REPORT

SUBMITTED BY:

ARAVIND D

FOR THE PROJECT WORK EXECUTED DURING THE FOUR WEEKS

(20.05.2024 TO 14.05.2024)

INTERNSHIP PROGRAM TITLED

SEMICONDUCTOR IC DESIGN AND TESTING

(I2C Protocol – Implementation in Digital ASIC)

IN

CENTRE FOR WIRELESS SYSTEM DESIGN (C-WiSD)

ANNA UNIVERSITY CHENNAI : CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this Report titled **I2C - Implementation in Digital ASIC** is the bonafide work of Aravind D and Vignesh M who carried out the project work as part of their Summer Internship program titled "**Semiconductor IC Design and Testing"** at Centre for Wireless System Design (C-WiSD) during the period 13.06.2024 to 26.06.2024.

Dr. J. Dhurga Devi
Project Co-ordinator &
Associate Professor,
Department of Electronics and
Communication Engineering
College of Engineering, Guindy
Anna University, Chennai -600 025

Dr. M. Meenakshi
Director,
Centre for Wireless System Design,
College of Engineering, Guindy
Anna University, Chennai -600 025

ABSTRACT

This project focuses on implementing the I2C (Inter Integrated Circuit) protocol in a single Verilog module, incorporating both master and slave functionalities. The primary objective was to create a versatile design that can be deployed as an integrated circuit (IC) or on a Field-Programmable Gate Array (FPGA). The project involved developing the Verilog module and simulating the design in Xilinx Vivado to verify waveform generation. Following successful signal waveform simulation, the ASIC flow was executed using Cadence's Genus and Innovus tools, encompassing synthesis, floorplanning, placement, routing, clock tree synthesis (CTS), and generating the GDSII file.

The major results included accurate waveform generation in the simulation phase and a comprehensive ASIC flow process, resulting in a fully functional GDSII file. This project demonstrates the feasibility and efficiency of integrating I2C protocol designs into both FPGA and ASIC platforms, contributing valuable insights and methodologies to the field of VLSI design and peripheral I/O interfaces. The successful implementation highlights the potential for creating adaptable and robust communication interfaces in modern electronic systems.

INTRODUCTION

Efficient and reliable communication between different components is crucial in modern electronic systems. Peripheral I/O interfaces play a significant role in facilitating this communication by providing standardized protocols for data exchange. Among these protocols, the Inter Integrated Circuit (I2C) stands out due to its simplicity, speed, and widespread adoption. I2Cis commonly used for short-distance communication in embedded systems, connecting microcontrollers to peripherals like sensors, displays, and other ICs.

Objective:

The primary objective of this project was to implement the I2C protocol in a single Verilog code that integrates both master and slave functionalities. This dual functionality allows the design to be versatile and adaptable for various applications. The project aimed to:

- Develop a Verilog code that implements I2C master and slave modules.
- Simulate the design in Xilinx Vivado to verify correct operation and waveform generation.
- Execute the ASIC flow using Cadence's Genus and Innovus tools to transition the design from synthesis to GDSII file generation.

Significance:

Implementing the I2C protocol in a unified Verilog code offers several advantages. It simplifies the design process, reduces development time, and enhances the flexibility of the interface. By ensuring that the design can be implemented in both FPGA and ASIC platforms, this project provides a robust solution for integrating I2Ccommunication in various hardware environments. The ability to switch between FPGA and ASIC implementations without significant modifications further underscores the adaptability and scalability of the design.

By achieving these objectives, the project not only demonstrates the practicality of I2Cprotocol implementation but also provides a valuable reference for future developments in peripheral I/O interfaces within the VLSI design field.

METHODOLOGY

In this section, we detail the methodology used in the project, including the research, design, and implementation processes. The project involved developing the I2C protocol in Verilog, simulating the design in Xilinx Vivado, and performing the ASIC flow using Cadence's Genus and Innovus tools. A flowchart of the ASIC design flow, which outlines the key stages from synthesis to GDSII file generation, is provided below.

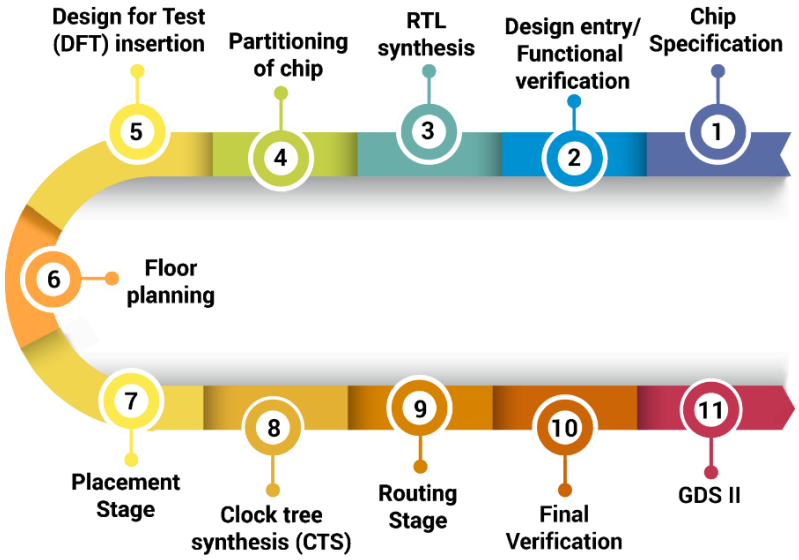

Figure 1: ASIC Design flow

Research and Understanding:

The initial phase of the project involved extensive research to understand the I2C protocol and its requirements. Key activities included:

- **Literature Review:** Reviewing technical documents, datasheets, and academic papers to gain a thorough understanding of the I2Cprotocol, its signals (SDA, SCL) and typical use cases.
- **Studying Verilog HDL:** Refreshing knowledge on Verilog hardware description language to ensure efficient coding practices.
- **Exploring Design Tools:** Familiarizing with Xilinx Vivado for simulation and Cadence's Genus and Innovus for the ASIC design flow.

Verilog Code Design:

The next phase involved designing the I2C master and slave modules in Verilog. This included defining the necessary signals, creating state machines, and writing the HDL code.

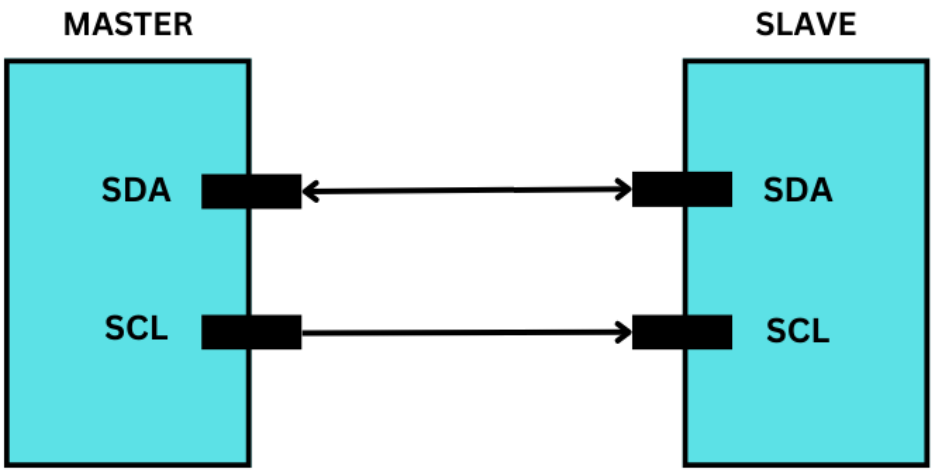

Figure 2: Block diagram of I2C protocol

Signal Definitions:

- SCL (Serial Clock Line): Clock signal generated by the master to synchronize data transmission.
- SDA (Serial Data Line): It is responsible for transmitting and receiving data between the master and slave devices on the I2C bus.

State Machine Design: Developing a state machine for the I2C master and slave modules. The state machine included states such as IDLE, START, ADDRESS, WRITE, READ, STOP and RE-START to manage data transmission and reception.

HDL Implementation: Writing the Verilog code for the I2Cmaster and slave modules, ensuring proper timing and signal integrity.

ASIC Flow:

After completing the Verilog code, the design was simulated in Xilinx Vivado to verify its functionality and generate waveforms. The waveform generation helped verify the proper functioning of I2C communication between the master and slave modules. The verified design was then taken through the ASIC flow using Cadence's Genus and Innovus tools.

The ASIC design flow involved several stages, each crucial for transitioning the design from HDL to a physical chip. A flowchart of the ASIC design flow is included in this section. The key stages are:

Synthesis: Converting the Verilog code into a gate-level netlist using Cadence Genus.

Floorplanning: Defining the layout of the chip and the placement of major functional blocks.

Placement: Placing the standard cells according to the floor plan.

Clock Tree Synthesis (CTS): Designing the clock distribution network to ensure proper timing across the chip.

Routing: Connecting the standard cells with metal wires.

Signoff: Performing final checks and optimizations.

GDSII File Generation: Creating the final layout file for fabrication.

By following this comprehensive methodology, we ensured that the I2C protocol implementation was robust, efficient, and ready for both FPGA simulation and ASIC fabrication.

RESULT

Waveform Simulation(VIVADO):

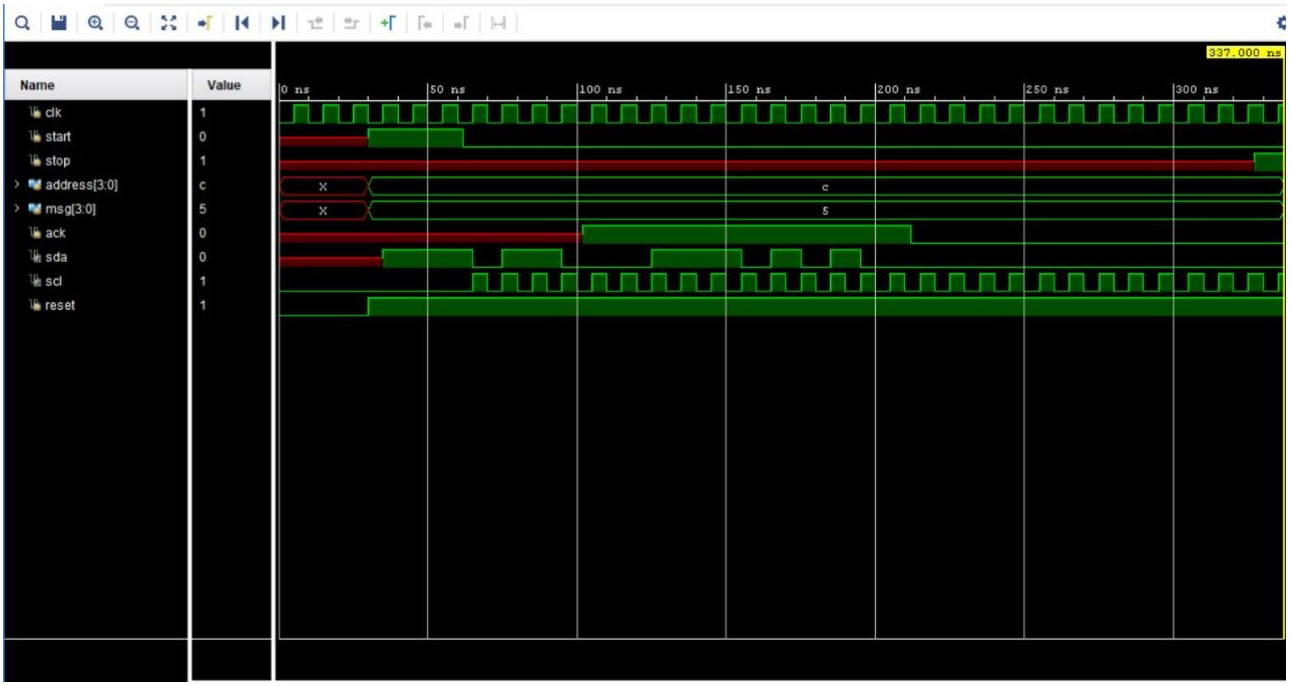

Figure 3: Behavioral simulation of master in I2C

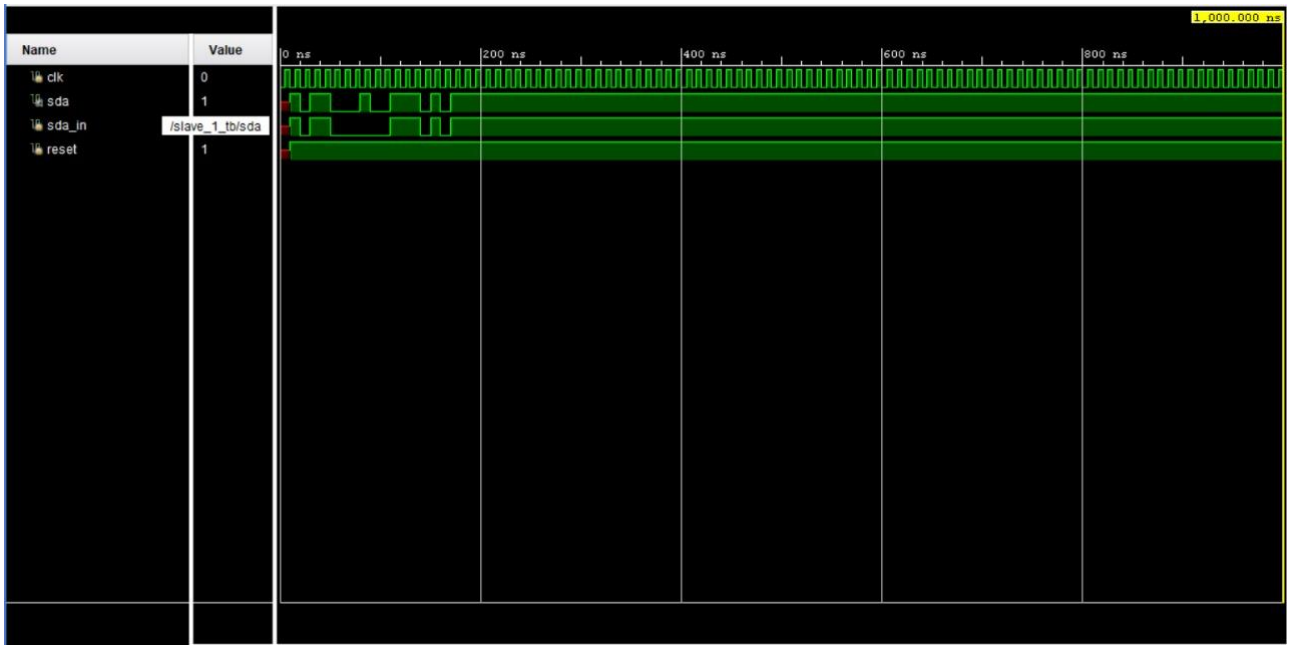

Figure 4: Behavioral simulation of slave in I2C

Before performing ASIC realization, the operation of I2C was verified in FPGA. Logic functionality was verified in simulation using VIVADO simulator. The testbench inputs and the obtained output are explained below. Master and clock frequency used in slave and master are for the purpose of verifying I2C master and slave logic function, data was used as constant. The above figure shows the digital waveforms of the I/O signals. The address Bit of slave from the master reaches slave and the slave acknowledges the address and sends the ACK signal. Then based on the Read/Write signal the master send/receives data to/from the slave. The waveform explains the input/output interface using the I2Cprotocol.

ASIC Flow(Genus and Innovus tool):

We used Cadence's Genus and Innovus tool to get our project into the Asic Flow. After the synthesis process, the design was imported into Cadence – Innovus tool. The Design has been imported after adding pad files to the RTL source file. Figure 4 shows the imported design after synthesis (Floor Plan)

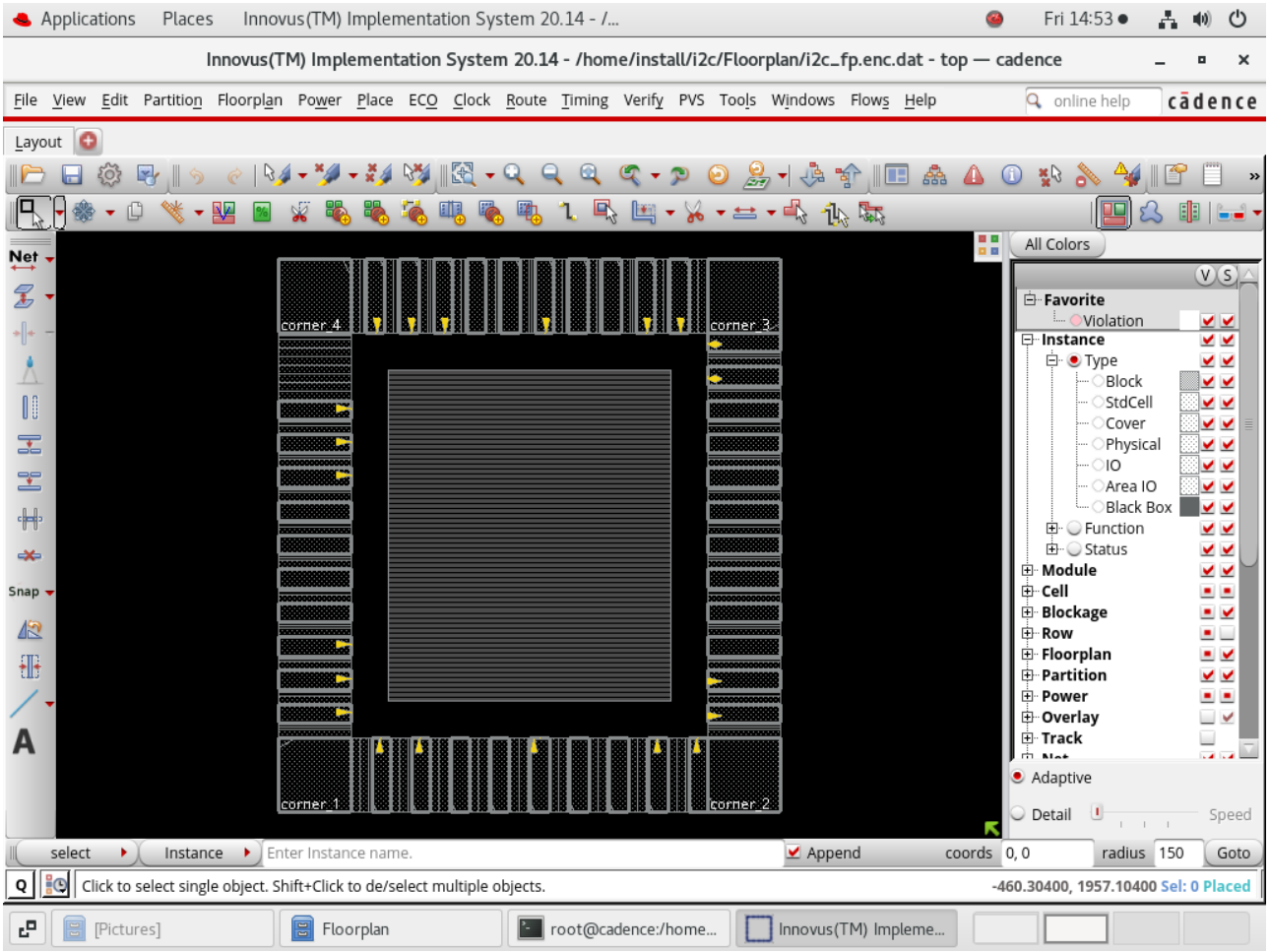

Figure 5: Floor planning

The size and dimensions of the chips are designed based on our requirements and constraints, then the I/O filler cells are added to fill the gap between the IP cores. Then the power ring and stripes were added to the design. The special routing is done to connect VDD_CORE and VSS_CORE. Then the standard cells were placed in the design according to our requirements. Routingof VDD_CORE and VSS_CORE tracks are shown in the Figure 5.

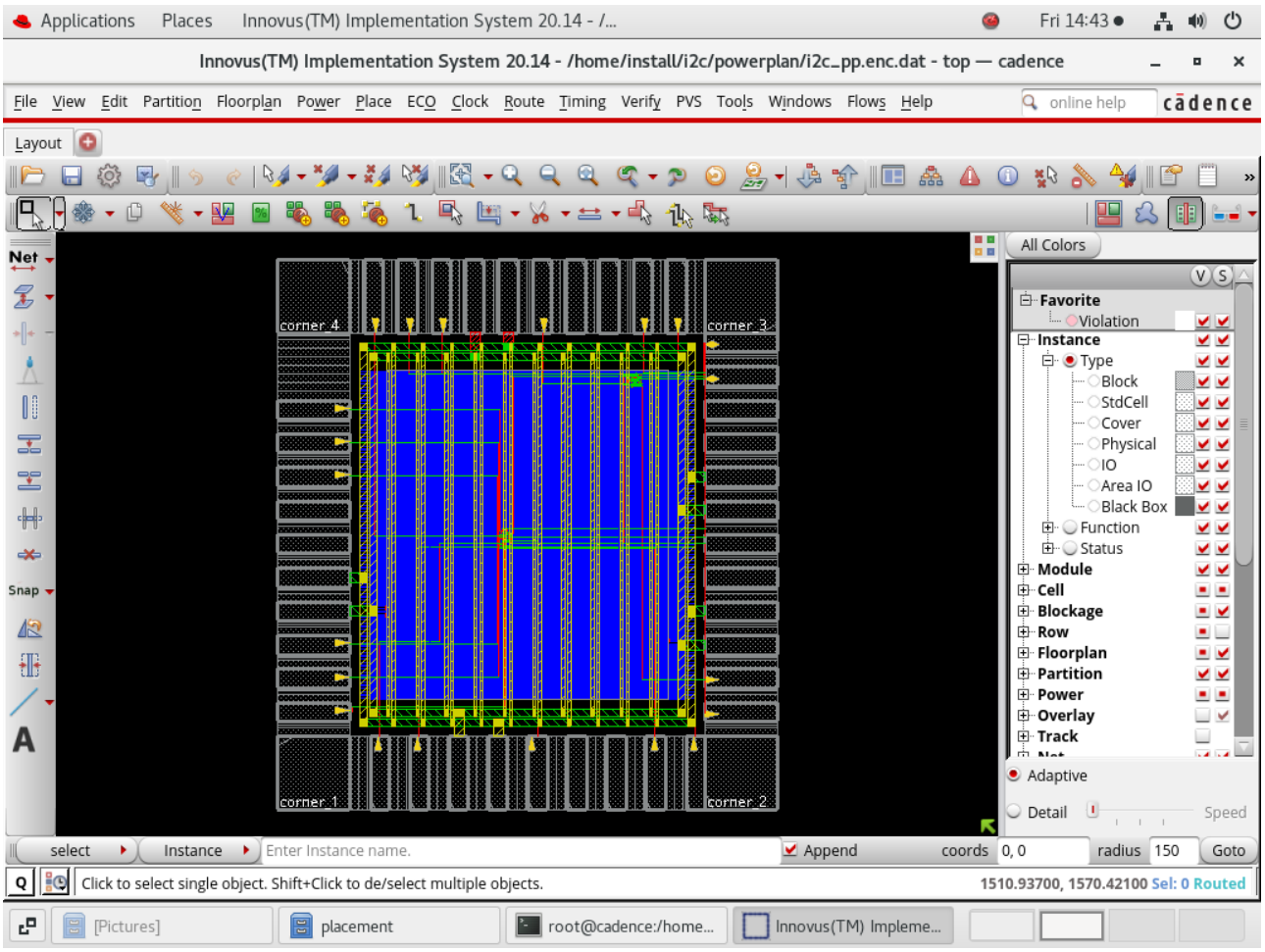

Figure 6: Power planning and Placement

After completing the placement process, the next step involves performing clock tree synthesis. During this process, the clock signals from the standard cells are connected to the IP cores within the design. The clock signals can be effectively routed to the IP cores.

CONCLUSION

The project successfully implemented the I2Cprotocol using a single Verilog code that integrates both master and slave functionalities. This design was simulated using Xilinx Vivado to ensure correct operation and waveform generation, and further transitioned through the ASIC flow using Cadence's Genus and Innovus tools, culminating in the generation of a GDSII file.

Key Achievements:

1. **Unified I2CDesign**: The project achieved the development of a versatile I2Cmaster-slave module in Verilog, suitable for both FPGA and ASIC implementations.
2. **Successful Simulation**: The design was validated through simulation in Xilinx Vivado, where waveform analysis confirmed the correct functioning of I2Ccommunication.
3. **Comprehensive ASIC Flow**: The project meticulously followed the ASIC design flow, from synthesis to GDSII file generation, ensuring that the design met all necessary specifications for physical chip fabrication.

Future Works:

- **I2C Protocol Integration**: Expanding the design to include I2C protocol support for more versatile peripheral communication.
- **Hardware Testing**: Implementing the design on an actual FPGA board to validate real-world performance and interaction with peripheral devices.
- **Optimization**: Further optimizing the design for power, area, and performance to meet specific application requirements.

In summary, this project showcases the practical implementation of the I2Cprotocol in both FPGA and ASIC environments, providing a strong foundation for further research and development in peripheral I/O interfaces within the VLSIdesign domain.

APPENDIX

Design of I2C:

VERILOG CODE FOR MASTER:

```
module master(
    output wire sda,
    output wire scl,
    output reg enable,
    input reset,
    input clk,
    input start,
    input [3:0] address,
    input rw,
    input stop,
    input ack,
    input [3:0] msg
    );
    reg [3:0] state;
    reg sdao;
    //wire temp;
    reg [3:0] count;
    reg temp1;
    reg out;
    assign scl= (!reset) ? 0 :((start==0 && temp1) ? clk : 1'b0);
    //assign temp=sdao;
    assign sda=sdao;

    always @(posedge clk )begin
   case(state)
   0: begin
   if(reset)
   begin
    state<=1;
    temp1<=0;
    sdao<=1;
    enable<=0;
    count<=5;
    end
   else if(clk)
    state<=0;
   end
   1: begin
   if (start==0) begin
       count<=4;
       sdao<=0;
       state<=2;
       temp1<=1;
   end
   else state<=1;
   end
```

```
2: begin
if(count==0)begin
state<=3;
enable<=1;
end
else begin
sdao<=address[count-1];
count<=count-1;
state<=2;
end
end
3: begin
if (ack) begin
state<=4;
end
else begin
state<=3;
end
end
4: begin
if(rw==0) begin
count<=4;
enable<=0;
sdao<=0;
state<=5;
end
else begin
state<=6;
sdao<=1;
enable<=1;
end
end
5: begin
if(count==0) begin
state<=7;
sdao<=0;
end
else begin
sdao<=msg[count-1];
count<=count-1;
end
end
6: begin
out<=ack;
state<=7;
end
7: begin
if(stop) begin
sdao<=0;
state<=0;
end
end
default: state<=0;
endcase
end
```

VERILOG CODE FOR SLAVE:

```
module slave(
input wire sda,
output wire scl,
input reset,
input [3:0] msg,
output reg enable,
output reg ack,
input clk
);
reg [3:0] state1=0;
reg [3:0] count;
reg  [3:0] address=4'b1100;
assign scl= (reset) ?  clk : 1'bz;
reg temp;
reg a;
reg out;
always @(negedge clk)begin
case(state1)
0:begin
if(reset) begin
ack<=0;
temp<=0;
a<=0;
enable<=1;
state1<=1;
count<=4;
end
else state1<=0;
end
1: begin
if(sda==0) state1<=2;
else state1<=1;
end
2: begin
if(count==0) begin
state1<=3;
enable<=0;
a<=1;
end
else begin
if(sda==address[count-1]) begin
count=count-1;
state1<=2;
end
else begin
count=count-1;
state1<=2;
temp<=1;
end
end
end
3:begin
if(temp==0)begin
ack<=1;
a<=0;
state1<=4;
end
```

```
else begin
ack<=0;
a<=0;
state1<=2;
end
end
4: begin
if(sda) begin
state1<=5;
count<=4;
enable<=0;
end
else begin
state1<=6;
enable<=1;
end
end
5: begin
if(count==0) begin
state1<=0;
end
else begin
ack<=msg[count-1];
count=count-1;
state1<=5;
end
end
6: begin
enable<=1;
a<=0;
out<=sda;
state1<=0;
end
endcase
end
endmodule
```

TESTBENCH FOR MASTER:

```
module master_1_tb();
   reg clk;
   reg start;
   reg stop;
   reg [3:0] address;
   reg  [3:0] msg;
   reg ack;
   wire sda;
   wire scl;
   reg reset;
   master_1 uut(.clk(clk),.start(start),.stop(stop),.ack(ack),.sda(sda),.msg(msg),.address(address),.reset(reset),.scl(scl));
   always #5 clk=~clk;
   initial begin
   clk=1'b0;
   reset=0;
   #30 reset=1;
   address<=4'b1100;
   msg<=4'b0101;
   start=1;
   #32start=0;
   #40 ack=1;
   #110 ack=0;
```

```
    #115 stop=1;

    #10;
    $finish;
    end
endmodule
```

TESTBENCH FOR SLAVE:

```
module slave_1_tb();
reg clk;
wire sda;
reg sda_in;
reg reset;
slave_1 uut(.clk(clk),.sda(sda),.sda_in(sda_in),.reset(reset));
always #5 clk=~clk;
initial begin
clk=1'b0;
#10 reset=1'b0;
reset=1'b1;
sda_in=1'b1;
#10sda_in=1'b0;
#10sda_in=1'b1;
#10 sda_in=1'b1;
#10 sda_in=1'b0;
#10 sda_in=1'b0;
#30;
#10 sda_in=1'b0;
#10 sda_in=1'b1;
#10 sda_in=1'b1;
#10 sda_in=1'b1;
#10 sda_in=1'b0;
#10 sda_in=1'b1;
#10 sda_in=1'b0;
#10 sda_in=1'b1;
end
endmodule
```