I2C Master Chip Design for Data Communication

Ketan J. Raut[1*], Minal Deshmukh[1], Yash Nalle[1], Satyam Agrawal[1], Vedant Tangadpalliwar[1], and Vaibhav Pavnaskar[2]

[1]*Department of Electronics & Telecommunication Engineering, Vishwakarma Institute of Information Technology, Pune, India.*

[2]*Microchip, Cork, Ireland.*

**E-mail: ketan.raut@viit.ac.in*

Abstract

Real-time data communication is a crucial aspect of various modern technological applications such as Industrial Control Systems, Data Acquisition systems, Networking Equipment, Consumer Electronics and Automotive. The Inter-Integrated Circuit (I2C) protocol stands as a prominent solution for these applications enabling the seamless transmission of critical information between integrated circuits.

This paper focuses on the development of a high-performance I2C (Inter-Integrated Circuit) master chip using the advanced SKY130 Process Design Kit (PDK) collaboratively created by Skywater Technologies and Google. The paper leverages the capabilities of OpenLane, an open-source digital ASIC design flow, for a seamless and efficient chip design process.

The I2C master chip serves as a critical component in digital communication systems, facilitating reliable and efficient data exchange between integrated circuits. The SKY130 PDK, known for its state-of-the-art technology and process nodes, provides a solid foundation for achieving optimal performance, power efficiency, and scalability in the design.

The paper workflow involves leveraging the OpenLane flow, which automates various stages of the ASIC design process, including synthesis, place and route, and physical verification.

Keywords: I2C protocol, VLSI Design, SKY130 PDK, OpenLane

1. Introduction

In the rapidly evolving landscape of semiconductor design, the demand for efficient and reliable digital communication interfaces has become increasingly pronounced. One such vital component in this domain is the I2C (Inter-Integrated Circuit) master chip, a cornerstone for seamless data exchange between integrated circuits. This paper embarks on the journey of designing and implementing an I2C master chip, leveraging the innovative SKY130 Process Design Kit (PDK) collaboratively developed by Skywater Technologies and Google. The integration of this cutting-edge PDK is seamlessly facilitated through the utilization of the OpenLane flow, an open-source digital ASIC design framework.

The I2C protocol, renowned for its simplicity and versatility, plays a pivotal role in enabling communication between diverse digital components in embedded systems, sensors, and other integrated circuits. The SKY130 PDK, characterized by its advanced process nodes and technological advancements, provides an ideal platform for harnessing the full potential of the I2C master chip design. The collaboration between Skywater Technologies and Google in creating the SKY130 PDK represents a convergence of industry expertise and technological prowess, setting the stage for groundbreaking advancements in chip design.

Integral to this paper is the incorporation of OpenLane, an open-source digital ASIC design flow that streamlines and automates various stages of the chip design process. From synthesis and place-and-route to physical verification, OpenLane offers a comprehensive and efficient approach, enhancing the design workflow and minimizing time-to-market. The amalgamation of SKY130 PDK and OpenLane not only facilitates the design process but also opens avenues for innovation and collaboration within the broader open-source ASIC design community. This paper aims not only to deliver a functional and high-performance I2C master chip but also to showcase the seamless integration of advanced PDKs with open-source design flows.

2. I2C Bus Specifications

The I2C Controller Bus represents a two-wire, bidirectional serial bus facilitating efficient data transmission over short distances among numerous devices. It proves particularly supportive in communicating with various intermittent, slow on-board peripheral devices, while demanding minimal hardware resources. Serving as a straightforward, low-bandwidth, and short-distance protocol, I2C simplifies the interconnection of multiple devices due to its integrated addressing system [1-2]. Figure-1 illustrates the two I2C signals: serial data (SDA) and serial Clock (SCL). In this configuration, the master, responsible for initiating transactions, regulates the clock signal, whereas the recipient of the command is designated as a slave [1-2]. While the I2C protocol accommodates multiple masters, the standard practice typically involves a single master and one or more slaves on the bus. Regular I2C devices support operation up to 100Kbps, while fast-mode devices can handle speeds of up to 400Kbps, aligning with the increasing bandwidth demands in contemporary multimedia applications [2].

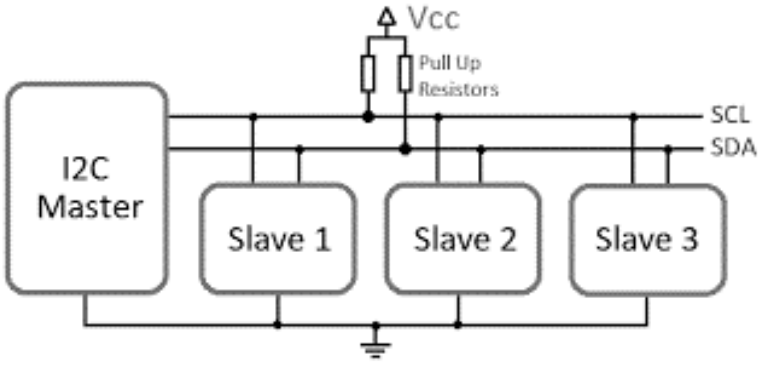

Fig. 2.1: I2C bus configuration using masters and slaves

3. I2C Operation

The I2C bus is a standard bidirectional interface which uses a controller, known as the master, to communicate with slave devices. For data transmission, a slave is required to await the master's specific address [4-5]. Each I2C device is allocated a distinct device address, distinguishing it from others on the same bus. Configuration of many slave devices is essential at the beginning to define their behavior, typically facilitated when the master accesses the unique register maps within the slave [3-4]. These maps contain individual register addresses where data is stored, written, or read. The physical I2C interface includes the serial clock (SCL) and serial data (SDA) lines, both connected to VCC via a pull-up resistor whose value is determined by the capacitance on the I2C lines. Data transfer initiation is permissible solely during idle bus conditions, identified by both SDA and SCL lines being high following a STOP condition [3-5]. The general sequence of steps for master to establish communication with a slave device is as follows [5-7]:

In the case where the master intends to transmit data to a slave:

- The master transmitter initiates communication with a START condition and addresses the slave-receiver.
- Subsequently, the master transmitter transmits data to the slave-receiver.
- Finally, the master transmitter concludes the transfer using a STOP condition.

Alternatively, if the master intends to receive or read data from a slave:

- The master receiver begins the process with a START condition and addresses the slave-transmitter.
- Next, the master-receiver specifies the desired register to read from the slave transmitter.
- Following this, the master receiver receives the data from the slave transmitter.

The entire process is concluded by the master receiver with a STOP condition.

4. Methodology

4.1 Front-end design

The module in the code serves as a design blueprint for implementing the I2C interface in digital systems. It includes input and output ports that represent various signals required for I2C communication [11].

The different parts and functionalities of the presented module are:

4.1.1 Clock Generation

The module incorporates a clock generation block responsible for producing two essential signals: qclk and half. qclk serves as the primary clock signal used for synchronization, ensuring that data is transmitted and received at the correct timing. The half

signal represents the half-cycle of the clock and aids in dividing the clock cycle into precise intervals, essential for the I2C protocol's timing requirements [2].

4.1.2 Data Transmission and Reception

Within the module, data transmission and reception are facilitated by the TX_REG and RX_REG registers [5]. TX_REG acts as a data buffer, storing information meant for transmission over the I2C bus. To populate the TX_REG register, the din input port is utilized. On the receiving end, the module includes an RX_REG register that collects and retains data received from slave devices [6]. This received data is then made available through the receive output port.

4.1.3 State Machine

The module employs a state machine approach to manage the various states of the I2C communication protocol. A dedicated state variable keeps track of the current state and transitions to other states based on specific conditions and clock edges. The state machine is instrumental in orchestrating different aspects of I2C communication, including the initiation of start and stop conditions, addressing, data transmission, and the handling of acknowledgments.

4.1.4 I2C States

The module defines distinct states of the I2C protocol by utilizing parameters such as S0, S1, S2, and so on [10-12] each state represents a specific operation or condition within the I2C communication sequence. For instance, S0 typically denotes the idle state, while S1 signifies the initiation of the start condition, and subsequent states correspond to various phases of the I2C communication process [10-12].

4.1.5 Data Transmission

In the context of data transmission, the module implements the necessary logic for transmitting data over the I2C bus [1, 4, 5]. This encompasses the transmission of the slave address, individual data bits, and the generation of clock pulses to synchronize the communication [8]. The sda_bus output port takes charge of transmitting data, while the scl output port generates the required clock signal, ensuring data is exchanged in a synchronous manner [1-2].

4.1.6 Data Reception and Acknowledgement

The module also excels in data reception from slave devices. It is equipped to receive data bit by bit, acknowledge the received data, and store it efficiently within the receive register (RX_REG) [5-6]. Furthermore, the code exhibits the capability to generate appropriate acknowledgment signals (y) in response to the data received. This acknowledgement mechanism is pivotal in confirming successful data reception [5, 6, 8].

4.1.7 Control Signals

The module incorporates a set of control signals, including cmd (command), rst (reset), and slave (slave mode) [5]. These signals are vital in enabling the master device to initiate

specific operations, reset the I2C module when necessary, and select the slave mode for communication [5, 8]. The control signals offer flexibility and control over the behavior of the I2C module, making it adaptable to various operational scenarios and requirements.

4.1.8. Clock Division

In described module, the qclk signal is used as the clock signal for the module as a Quarter Clock of the main clock. The division of qclk is achieved by utilizing a binary counter represented by the q variable, which is a 4-bit register [4].

The division of qclk is based on the clock edges in the always @(posedge qclk) block. Inside this block, the q register is incremented by 1 on each positive clock edge using the statement q = q + 1'b1; this effectively divides the frequency of qclk by 2 since the counter increments at each positive edge.

The division of qclk into two separate signals as shown in Figure 4.1, half and clk, is done using the assignments assign half = q[0]; and assign clk = q[1];. Here, half represents the least significant bit (LSB) of the q register, and clk represents the second least significant bit (the next bit after half). By using half and clk signals in different parts of the code, specific operations can be synchronized to either every positive edge or every second positive edge of qclk. This allows for precise timing control and coordination of different processes within the module.

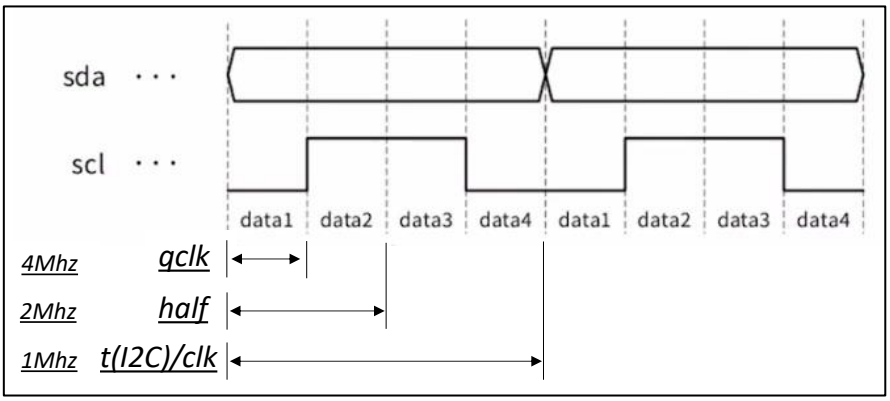

Fig. 4.1: Clock division timing diagram

4.1.9 Data Transmission

The Verilog code for the data transmission is divided into namely 4 states or stages. Namely data1, data2, data3, data4 [6]. The code in each of these stages is to be executed once to transfer a single bit. Thus, to transmit a data of 8 bits each of these stages are to be executed 8 times. Since, a standard approach of bit transmission states that each bit should be transmitted in one clock cycle of the main i2c clock, each of these stages (data1, data2, data3, data4) are to be executed at 1/4th of the clock cycle of the i2c clock which was achieved by us in the clock dividing section mentioned above. Making four data stages makes the transmission more efficient by pushing the data on sda line so that it gets carry forwarded. Whenever transmission of a single bit takes place (i.e., all the four states are executed) the control of the program goes

to another stage in which the indexing of the transmit register is decremented by one and as soon as the indexing reaches the value of zero the control goes to the next state of the Verilog code else the looping continues and the four states are executed. This transmission logic has applications internally in the i2c controller for the purposes including the data transmission, slave address transmission, slave register address transmission [5], etc. as shown in Figure 4.3. Each of the data stages contains the similar code for transmitting data on the sda line at 1/4th clock pulse of the i2c clock time period.

4.1.10 Data Reception

Unlike the transmit logic, the receive logic mainly consists of only one state or stage which enhances the data reception in the i2c module [7, 9]. The sole purpose of the data reception logic is to get the data bit by bit (i.e., each bit at once) from the sda line to the receive register [1-3]. The standard approach of the data reception is similar to that of the data transmission i.e., one bit should be received at each clock cycle of the i2c controller clock [5].

Since we have designed only a single stage for the data reception, the receiving state can be operated on the edges of the main i2c controller clock [7, 10]. The code for the receive purpose is also similar to that which we have viewed in the transmit section above, the difference lies between the no. of stages and the clock at which both operate. Diving into the actual content of the code, the code consists of the logic which writes the sda line into the nth index of the receive register and increments the value of the index by one every time the state gets executed (initial value of the index being zero) [7, 10]. As soon as the index value reaches a certain count (8 in our case), the control of the program goes to the next state and the value of the indexing variable is set to zero again, which facilitates in starting the data reception again from the initial index. This reception logic finds its applications internally in the i2c module for data reception, receiving the acknowledgement, etc. from the sda line.

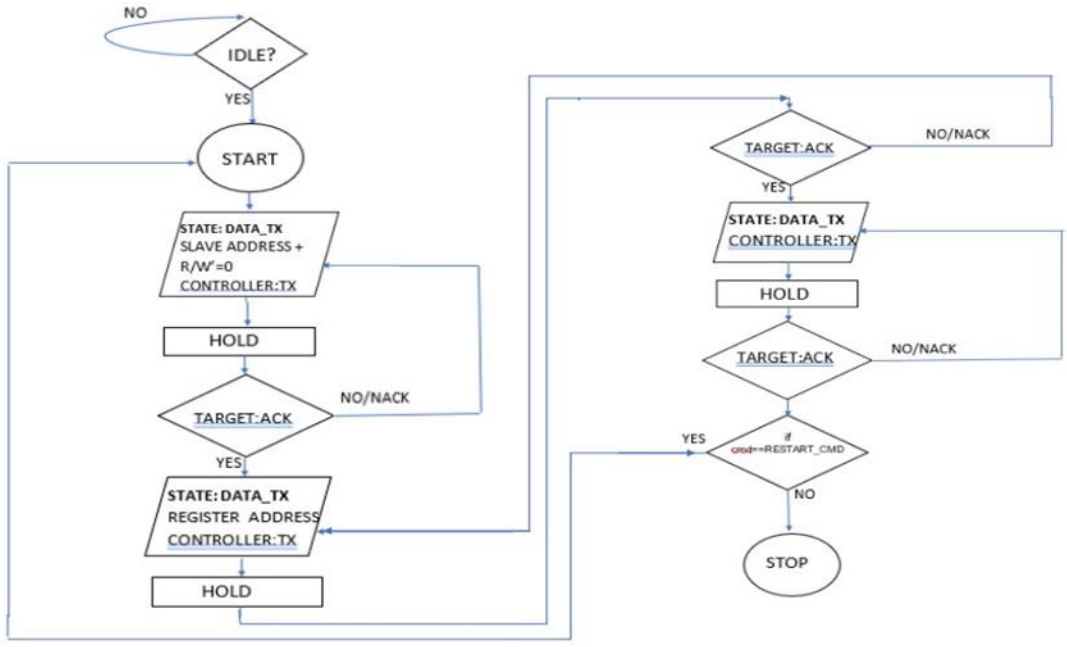

Fig. 4.2: Flowchart of Master Write in slave

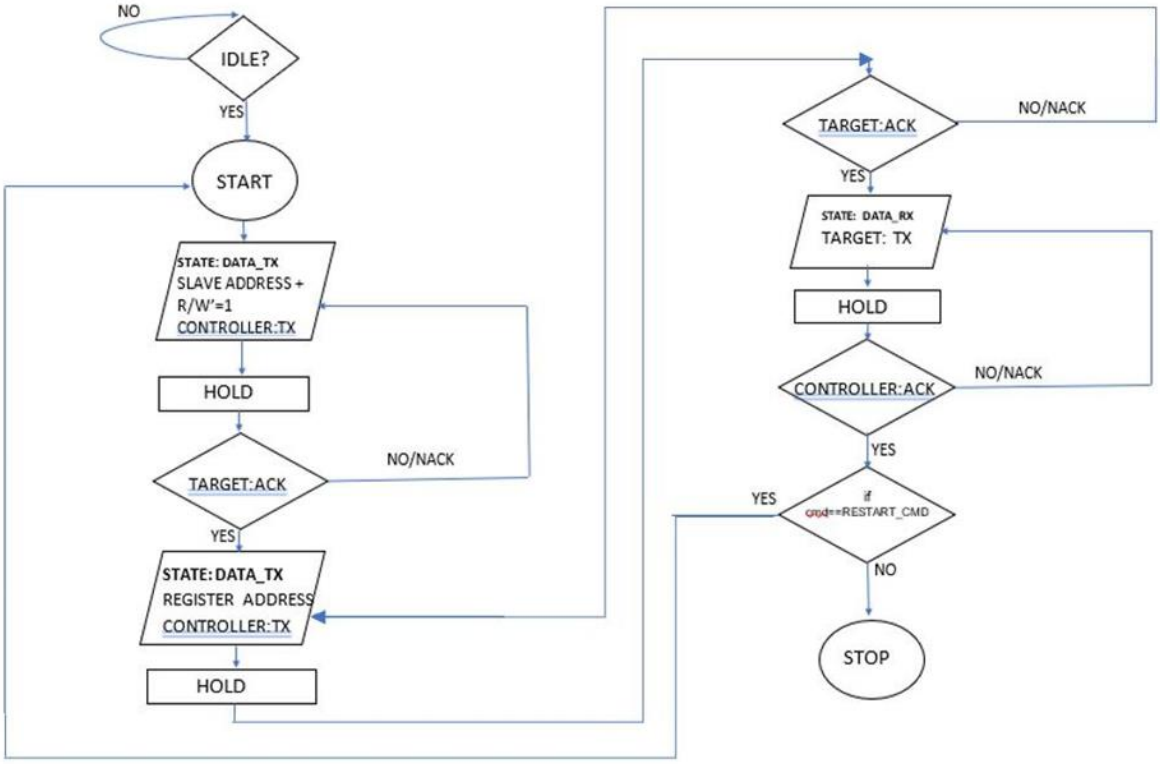

Fig. 4.3: Flowchart of Master Read from Slave

4.2 Back-end Design

The complete openlane flow has been followed to complete the backend process of the design. OpenLane is an open-source digital ASIC (Application-Specific Integrated Circuit) design flow that automates the process of designing and implementing digital integrated circuits. The design flow in OpenLane consists of several key blocks, each responsible for a specific stage in the overall process. Here's a detailed explanation of the OpenLane design flow block wise:

- Design Input (Configurations):
  - This block involves specifying the design parameters and configurations. Users provide input files that define the design specifications, technology information, and constraints.
- Synthesis:
  - In this block, the RTL (Register Transfer Level) code of the digital design is synthesized into a gate-level netlist. The synthesis process optimizes the design for area, power, and timing based on the specified constraints.
- Floorplanning:
  - Floorplanning involves placing the macroblocks and defining the core and periphery areas of the chip. It also includes power planning and placement of clock distribution network.
- Placement:

- This block assigns physical locations to each logic cell in the design. The goal is to minimize wirelength and optimize for timing, power, and congestion.

➢ Clock Tree Synthesis (CTS):
- CTS involves the synthesis of a clock distribution network to ensure that the clock signal reaches all parts of the chip with minimal skew and power consumption.

➢ Routing:
- Routing is the process of determining the physical paths of wires that connect the logic cells according to the placed design. This step aims to meet timing and congestion constraints.

➢ Fill Insertion:
- The design is filled with dummy metal to meet manufacturing requirements such as density rules and manufacturing constraints.

➢ Signoff:
- Signoff involves running various checks to ensure that the design meets the specified requirements and adheres to the manufacturing rules. This includes checks for timing, power, and physical design rules.

➢ GDSII Generation:
- GDSII (Graphic Data System II) is the final output file that contains the physical layout of the chip. This file is used for manufacturing.

➢ EDA Tool Integration:
- Throughout the design flow, various EDA (Electronic Design Automation) tools are integrated to perform specific tasks at each stage. OpenLane orchestrates these tools to automate the entire process.

It is important to note that OpenLane is continuously evolving, and the specific tools and versions may change over time. Users can customize the design flow parameters based on their specific requirements and constraints. The flexibility and openness of OpenLane make it a powerful tool for digital ASIC design.

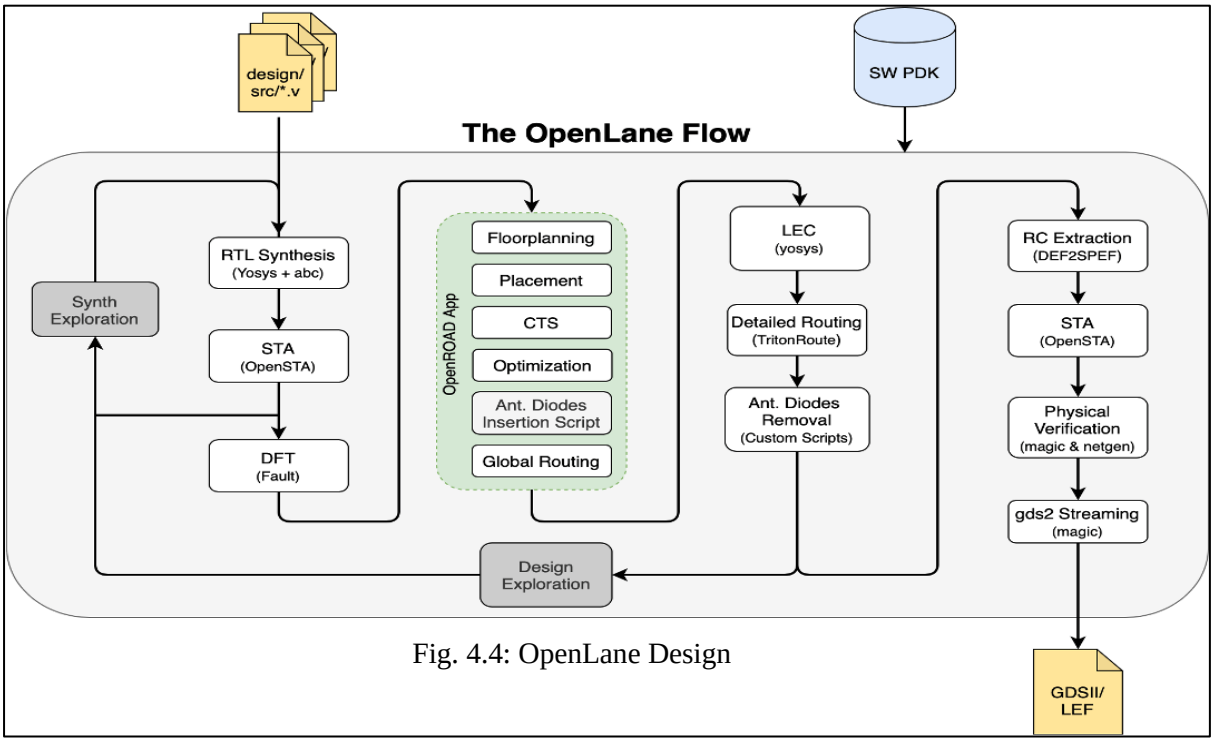

Fig. 4.4: OpenLane Design

5. Results

5.1 RTL Top View (Frontend):

Following is the top view RTL sematic obtained after performing synthesis is Xillinx-ISE.

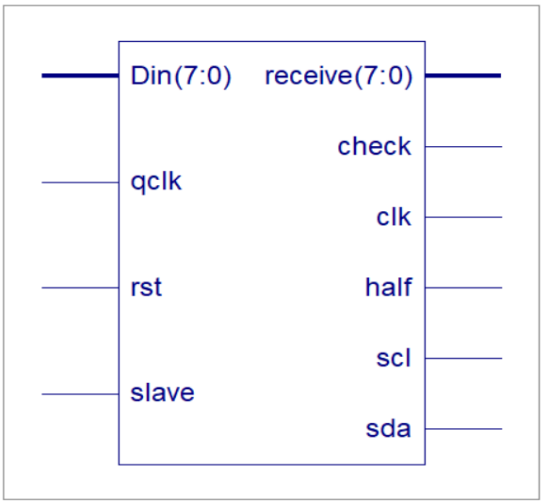

Fig. 5.1: RTL top view of I2C Architecture

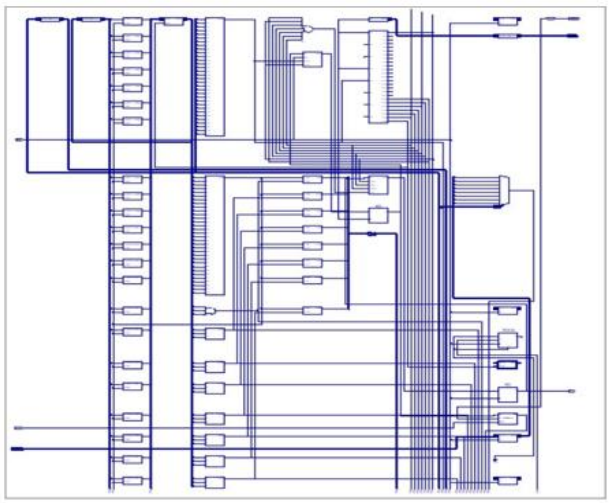

Fig. 5.2: Detailed RTL

5.2 Simulation Output Waveform (Frontend)

5.2.1 Master Write in Slave

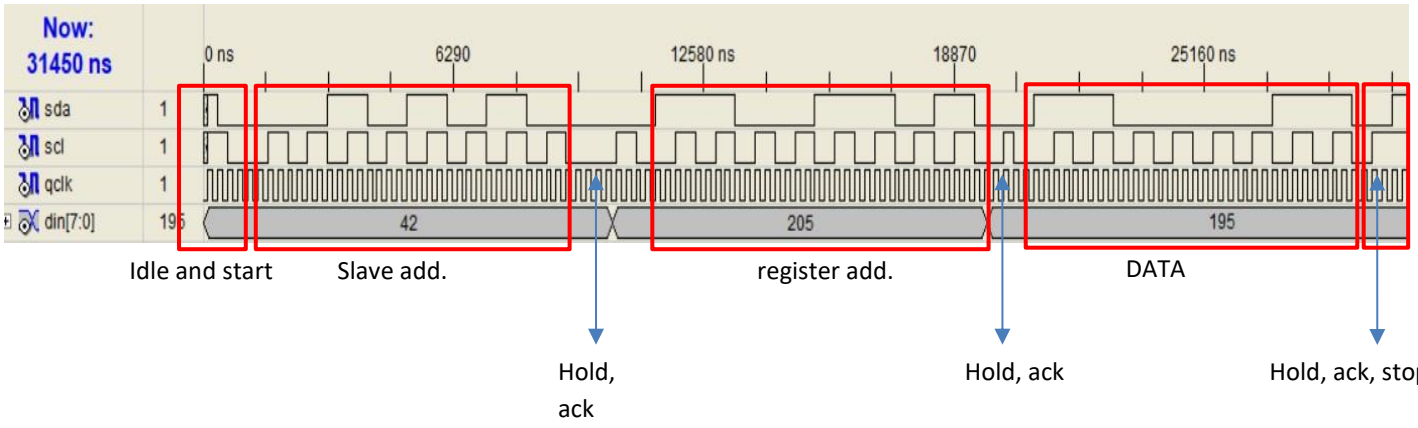

Fig. 5.3.: Master writes in slave waveform xillinx ISE

5.2.2 Master Read from Slave

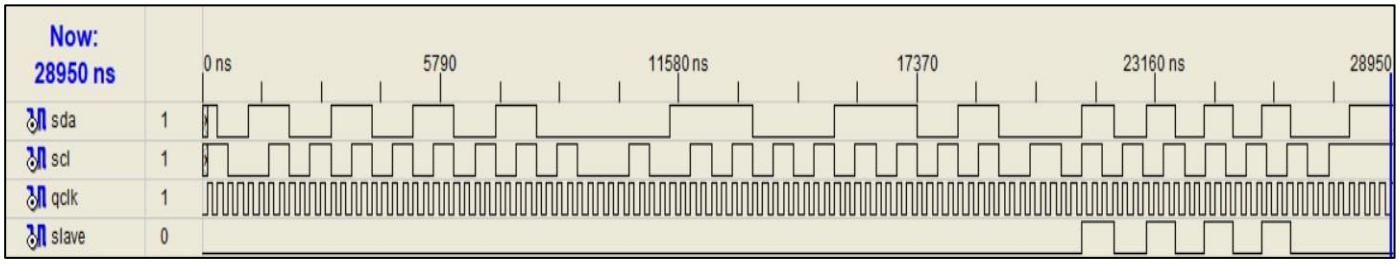

Fig. 5.4: Master read from slave waveform xillinx ISE

5.3 GDS File (Backend):

5.3.1 Output in Magic-

Following is the output obtained when GDS file is viewed in magic layout tool using skywater 130 standard cells.

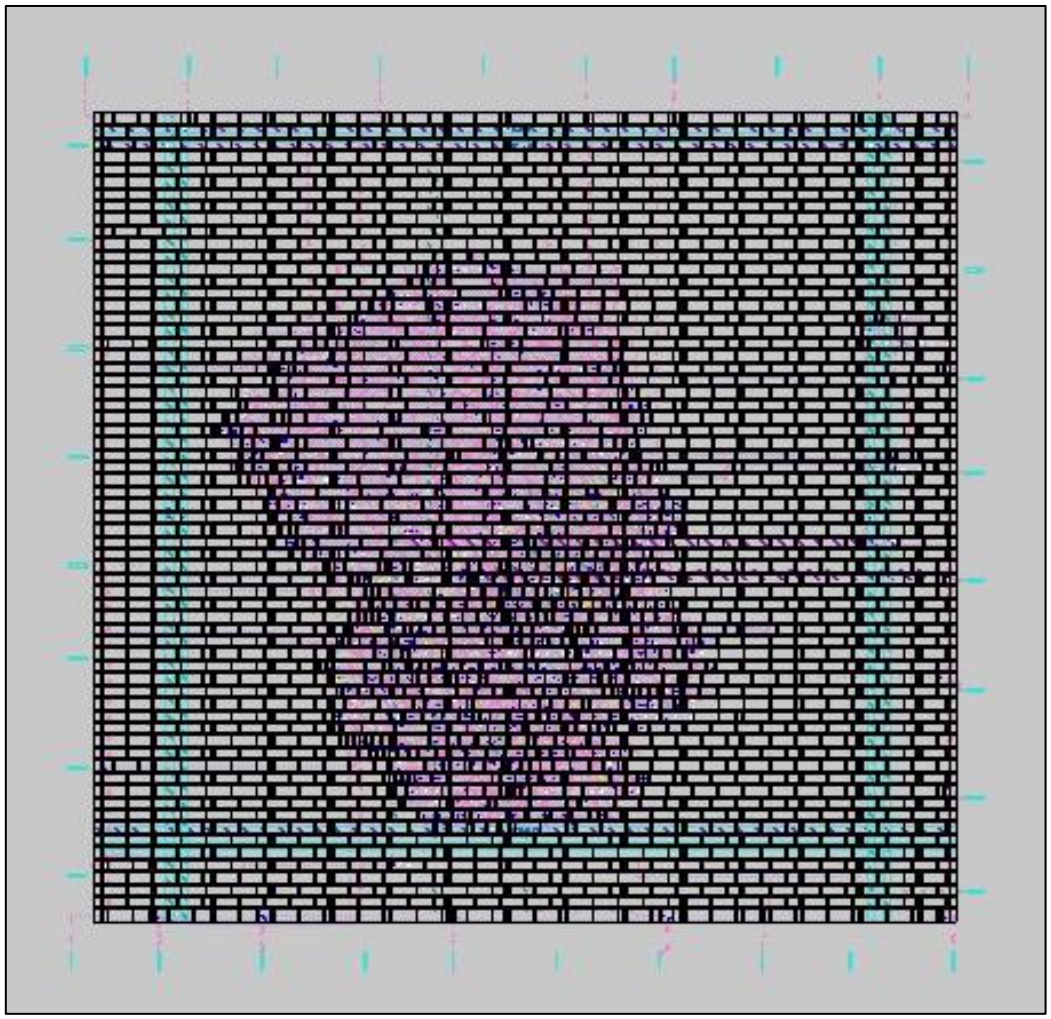

Fig. 5.5: GDS file in Magic Layout Tool

5.3.2 Output in Klayout

Following is the output obtained when GDS file is viewed in klayout tool.

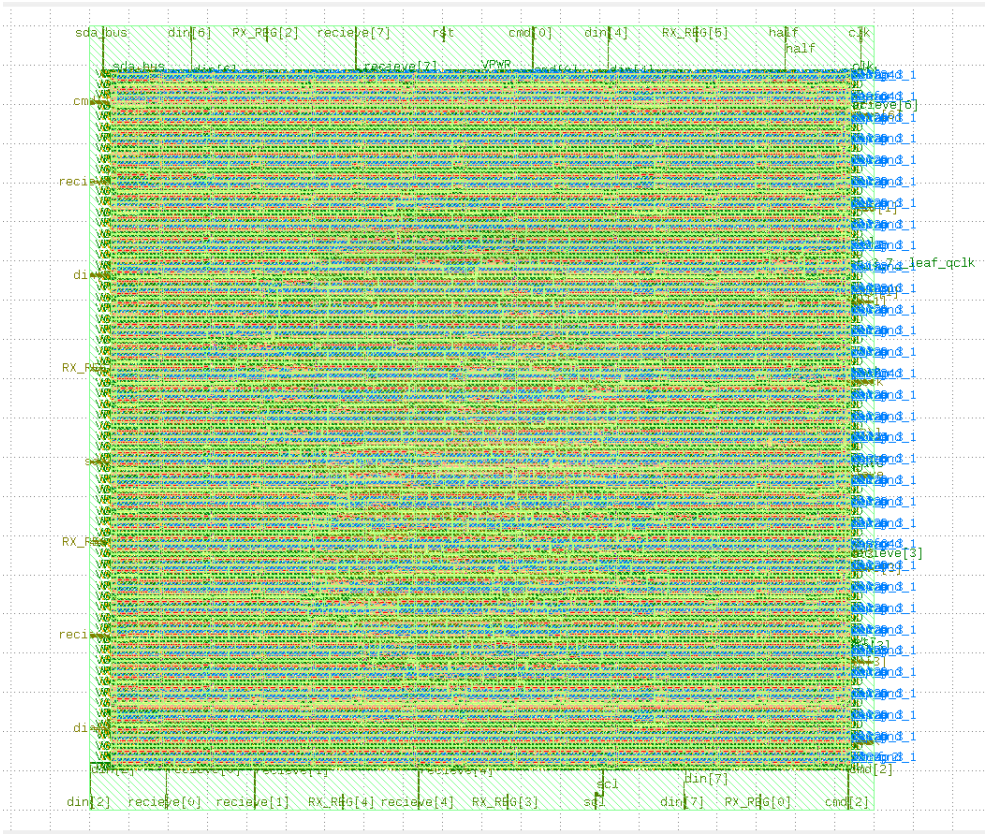

Fig. 5.6: GDS file in KLayout Tool

5.4. Summary of Reports and Logs

5.4.1 Synthesis Log

Tabe 5.1: Synthesis Log Report

Number of wires:	420
Number of wire bits:	809
Number of public wires:	22
Number of public wire bits:	245
Number of cells:	753
Chip Area	6398.25

5.4.2 Metrics CSV

Table 5.2 Metrics Report

Die Area(mm^2)	0.04
Number of Cell/mm^2	19125
Peak Memory Usage (MB)	580.52
Synth cell count	609
Number of AND Gates:	23
Number of NAND Gates:	43
Number of NOR Gates:	66
Number of XOR Gates:	112
Number of XNOR Gates:	10
Number of MUX:	47
Number of Decap Cells:	1880
Core Area (um^2):	3344.48

6. Conclusion

This paper presents a methodology for the design of I2C chip, including writing the Verilog code for the design from scratch to following the Openlane flow and generating the GDS file. After complete verification and testing the GDS file can be sent to the fab for completion of further steps involving the manufacturing of an IC. The design offers an efficient and reliable solution for I2C communication, and it can serve as a foundation for more complex I2C systems or be integrated into larger digital designs.

The proposed design has a space for future scope and can be further extended by incorporating a greater number of slaves and masters to enhance the communication and incorporating mix signal technologies with secured communication protocols to achieve efficient, secured and reliable communication.

References

[1] *Philips Semiconductors; The I2C Bus specification, version 2.1, 2000.*

[2] *NXP Semiconductors; UM10204 I2C-bus specification and user manual, Rev. 7.0, 2001.*

[3] *Philips Semiconductor; I2C Bus Specification, 2010.*

[4] *NXP Semiconductors; I2C-bus specification and user manual, version 7. 2001.*

[5] *Valdez, J.; Becker, J.; Understanding I2C Bus, 2015, Texas Instruments.*

[6] *Sutherland S.; Verilog® HDL Quick Reference Guide. 2001, Sutherland HDL Inc.*

[7] *Leens, F.; An Introduction to I2C and SPI Protocols. IEEE Instrumentation & Measurement Magazine, 2009, 12, 8-13.*

[8] *Megalingam, R.K.; Varghese, J.M.; Anil, S.A.; Distance estimation and direction finding using I2C protocol for an auto navigation platform, 2016 International Conference on VLSI Systems, Architectures, Technology and Applications (VLSI-SATA), 2016, 1.*

[9] *Palnitkar, S.; Verilog HDL: A Guide to Digital Design and Synthesis. Prentice Hall PTR, 1996, Second Edition.*

[10] *IEEE Standard for Verilog Hardware Description Language, IEEE Std. 1364-2005, 2006, 1-590.*

[11] *Mehto, P.K.; Mishra, P.; Lal, S.; Design and Implementation for Interfacing Two Integrated Devices Using I2C Bus, IJIRCCE, 2013, 2, 3.*

[12] *VSD - Physical Design Flow, Udemy.*

[13] *VSD Intern - Mixed Signal Physical Design Flow with Sky130, Udemy.*

[14] *The OpenROAD paper online: https://github.com/The-OpenROAD-Paper/OpenLane.*