Complete ASIC Flow of I2C communication protocol

ADVANCED DIGITAL DESIGN

Final Project

CND 211

Marwa Moaz	V23009897
Shrouq El Shaal	V23010529
Fagr Ahmed	V23010586
Mohamed Yasser	V23010495

Center of Nano electronics and Devices (CND) – AUC

May 2024

***Abstract*— ASIC Implementation of I2C Master core has been proposed in this project. I2C is one the most prominent protocols used in on chip communication among sub-systems. The generic design of I2C master core has ample features to incorporate vast varieties of application and I2C standards. The generic design is slow, congested and requires high power. It’s rare to utilize all the features of generic design fully in a single application or system. Hence, a modified ASIC design with specific less features but with better timing, low power requirement and less area overhead, has been proposed in this project. This design is specifically apt for digital systems which have serial bus interface requirement for on board communication. The entire custom ASIC implementation of proposed design has been done in Synopsys Tool chain with 90nm technology using standard cell library.

1. INTRODUCTION

Communication protocols for data interchange are usually incorporated on digital systems using an on-chip master controller sub-system. The protocols used for communication are generally divided into two broad categories: Parallel and Serial. Parallel buses for all the interfaces are not a good trade-off between cost, time, power and performance. Alternate serial buses are much more efficient in on-board data communication between different subsystems on an IC or SOC.

Most of the peripherals on modern ASIC & SOC designs use serial communication buses for data transfers between processors or between processor and peripherals. Few examples of Serial communication protocols are UART, CAN, USB, SPI and Inter IC. The USB, SPI and UARTS can communicate from one device to only one other at a time, i.e. not more than two devices can be connected with each other at a time.

Whereas USB has a problem of using a multiplexer for communication with other devices which makes it bulky. Only I2C and CAN protocol uses software addressing but only I2C is convenient to design for ICs because CAN is specifically designed for automobiles. The objective of this paper is to develop a Firm IP core of I2C Master Bus Controller for ASIC and other applications like SOCs, wherever there is a requirement of compact & small customized I2C Protocol Master Bus controller module for communication between on-board components.

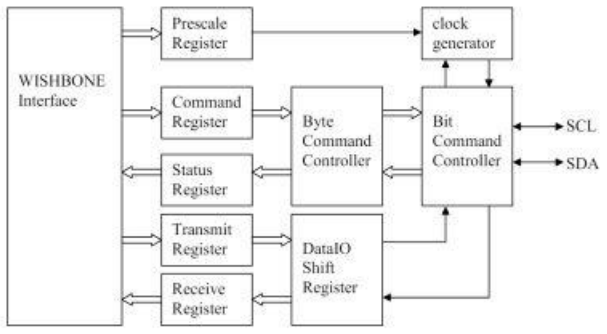


Figure 1: I2C Master Block Diagram

2. I2C

I2C, or Inter-Integrated Circuit, is a synchronous, multi-master, multi-slave, packet-switched, single-ended, serial communication protocol invented by Philips Semiconductor (now NXP Semiconductors) in the 1980s. It's widely used for interfacing sensors, EEPROMs, real-time clocks (RTCs), LCDs, and other integrated circuits in embedded systems.

- **The I2C specification**

The electrical and protocol characteristics of the I2C bus. Here's a summary of the key aspects of the I2C specification:

Physical Layer:

Two-Wire Interface: The I2C bus consists of two lines: Serial Data (SDA) and Serial Clock (SCL).

Open-Drain Architecture: Both SDA and SCL lines are open-drain, meaning they can pull the line low but cannot drive it high. Pull-up resistors are used to pull the lines high when they are not being actively driven low by a device.

Voltage Levels: The voltage levels on the bus typically conform to either 5V or 3.3V standards, depending on the specific implementation and the voltage requirements of the connected devices.

Logical Layer:

Bit Transfer: Data transfer on the bus occurs one bit at a time, synchronized to the clock signal.

Start and Stop Conditions: Communication begins with a Start condition (SDA transitions from high to low while SCL is high) and ends with a Stop condition (SDA transitions from low to high while SCL is high).

Addressing: Each device on the bus has a unique 7-bit or 10-bit address. The master device addresses the slave it wants to communicate with by sending its address along with the desired read or write operation.

Acknowledgment: After receiving each byte of data, the receiver sends an acknowledgment bit (ACK) to indicate successful receipt.

Data Transfer:

Byte Format: Data transfer occurs in packets, with each packet consisting of an address byte followed by one or more data bytes.

Read and Write Operations: The master device initiates communication by sending a Start condition followed by the address of the slave device and the desired read or write operation. Read operations are preceded by a repeated Start condition.

Clock Stretching: Slave devices can hold the SCL line low to pause the master's clock signal, allowing them to process data at their own pace.

Speed Modes:

I2C supports multiple speed modes, including Standard Mode (100 kbit/s), Fast Mode (400 kbit/s), Fast Mode Plus (1 Mbit/s), and High-Speed Mode (3.4 Mbit/s). The specific speed mode used depends on the capabilities of the devices on the bus and the requirements of the system.

Error Handling:

The I2C specification defines mechanisms for error detection and recovery, including NACK (not acknowledge) signaling and arbitration in multi-master configurations.

Thus, the discarding of every 8th bit of the key produces a 56-bit key from the original 64-bit key.

DES is based on the two fundamental attributes of cryptography: substitution (also called confusion) and transposition (also called diffusion). DES consists of 16 steps, each of which is called a round. Each round performs the steps of substitution and transposition. Let us now discuss the broad-level steps in DES.

3. System functionality

The I2C system functionality encompasses a range of operations and capabilities that enable devices to communicate with each other on an I2C bus. Here's a breakdown of the key functionalities:

- **Clock Generator**

A circuit or component within the master device of an I2C system that produces the clock signal (SCL) used to synchronize data transfer between master and slave devices on the I2C bus. The clock generator determines the frequency and timing characteristics of the clock signal, ensuring reliable communication between devices. It is essential for coordinating the transmission and reception of data bits, enabling efficient and synchronized operation of the I2C bus.

- **Byte Command Controller**

A module or component within an I2C-based system that oversees the generation, interpretation, and execution of byte-level commands exchanged between the master and slave devices on the I2C bus. The Byte Command Controller facilitates the communication protocol by handling commands such as read, write, address selection, and data transfer at the byte level. It ensures proper formatting, addressing, and synchronization of commands to facilitate seamless interaction between devices connected to the I2C bus. Additionally, it may incorporate logic for error detection, command validation, and bus arbitration in multi-master environments to ensure reliable and efficient operation of the I2C communication protocol.

- **Bit Command Controller**

Component or module within an I2C system that manages individual bit-level commands or operations during data transmission. Unlike the typical byte-level command controller, this theoretical controller could handle finer-grained control over individual bits within a byte or across multiple bytes. Its responsibilities might include bit-wise manipulation, bitwise logical operations, or other low-level command execution. Such a controller, if it existed, would likely operate in conjunction with the standard I2C protocol, providing enhanced functionality or customization options for specific applications requiring fine-grained control over data transmission. However, it's essential to note that this concept might not align with standard I2C terminology and practices, and its actual implementation and depend on specific system requirements and design choices.

- **Data IO Shift Register**

The DataIO Shift Register contains the data associated with the current transfer. During a read action, data is shifted in from the SDA line. After a byte has been read the contents are copied into the Receive Register. During a write action, the Transmit Register's contents are copied into the DataIO Shift Register and are then transmitted onto the SDA line.

4. ASIC Flow

In integrated circuit design, physical design (ASIC design) is a step in the standard design cycle which follows the circuit design. At this step, circuit representations of the components (devices and interconnects) of the design are converted into geometric representations of shapes which, when manufactured in the corresponding layers of materials, will ensure the required functioning of the components. This geometric representation is called integrated circuit layout. This step is usually split into several sub-steps, which include both design and verification and validation of the layout.

Modern day Integrated Circuit (IC) design is split up into Front-end Design using HDLs and Back-end Design or Physical Design. The inputs to physical design are (i) a netlist, (ii) library information on the basic devices in the design, and (iii) a technology file containing the manufacturing constraints. Physical design is usually concluded by Layout Post Processing, in which amendments and additions to the chip layout are performed. This is followed by the Fabrication or Manufacturing Process where designs are transferred onto silicon dies which are then packaged into ICs.

Each of the phases mentioned above has design flows associated with them. These design flows lay down the process and guidelines/framework for that phase. The physical design flow uses the technology libraries that are provided by the fabrication houses. These technology files provide information regarding the type of silicon wafer used, the standard-cells used, the layout rules (like DRC in VLSI), etc.

The main steps in the ASIC physical design flow are:

- Design Netlist (after synthesis)
- Floor planning (with Power Planning)
- Partitioning
- Placement
- Clock-tree Synthesis (CTS)
- Routing
- Physical Verification
- Layout Post Processing with Mask Data Generation

We performed full ASIC flow on the I2C design, starting from synthesis, then PNR and signoff.

1-Synthesis:

In the synthesis step, we read the text file, read the constraints that are the same as the file provided by the project, decide the link path and target library then compile the design and create reports for area and timing as well as writing the SDC file for usage in the PNR step.

Timing passed with zero violations. A photo of synthesis output is shown below and a photo of passing timing as well.

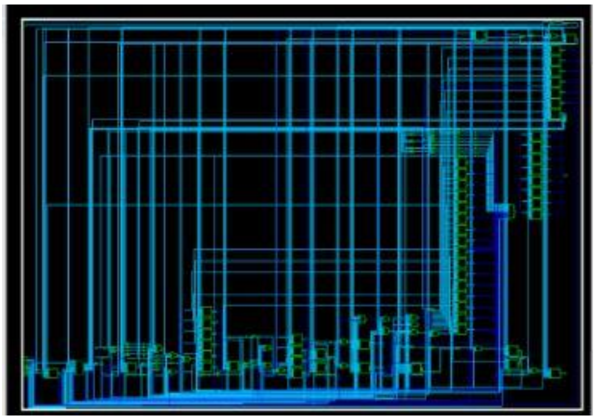
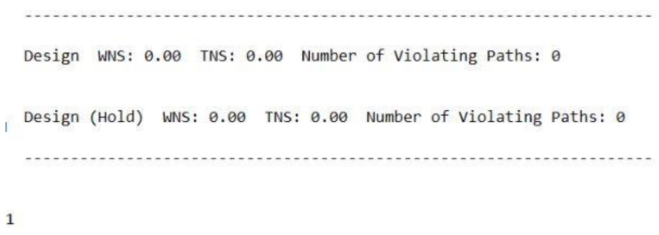
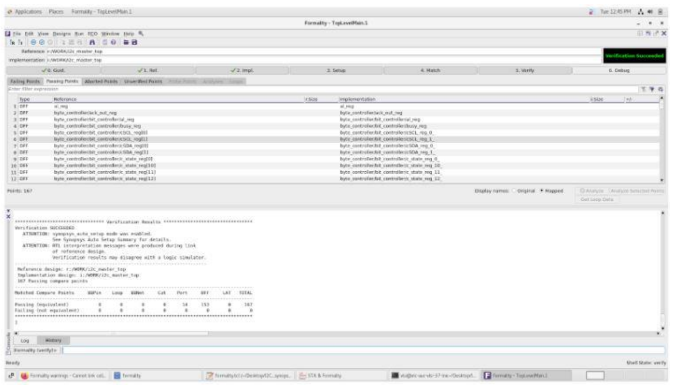


Figure 1: Schematic


```
------------------------------------------------------------------------

Design  WNS: 0.00  TNS: 0.00  Number of Violating Paths: 0

Design (Hold)  WNS: 0.00  TNS: 0.00  Number of Violating Paths: 0

------------------------------------------------------------------------

1
```
Figure 2: timing violations found

2-Formality:

Formality is done to compare post synthesis netlist with pre synthesis netlist and formality succeeded as in the below figure.


```
****************************** Verification Results ******************************
Verification SUCCEEDED
   ATTENTION: synopsys_auto_setup mode was enabled.
              See Synopsys Auto Setup Summary for details.
   ATTENTION: RTL interpretation messages were produced during link
              of reference design.
              Verification results may disagree with a logic simulator.
-----------------------------------------------------------------------------
 Reference design: r:/WORK/i2c_master_top
 Implementation design: i:/WORK/i2c_master_top
 167 Passing compare points
------------------------------------------------------------------------------
Matched Compare Points     BBPin    Loop   BBNet     Cut    Port     DFF     LAT   TOTAL
------------------------------------------------------------------------------
Passing (equivalent)           0       0       0       0      14     153       0     167
Failing (not equivalent)       0       0       0       0       0       0       0       0
**********************************************************************************
1
```
Figure 3: Formality

3-PNR

NDM creation:

An NDM is created using the library manager to be used in the next steps of PNR.

LEF of an AND gate is shown below.

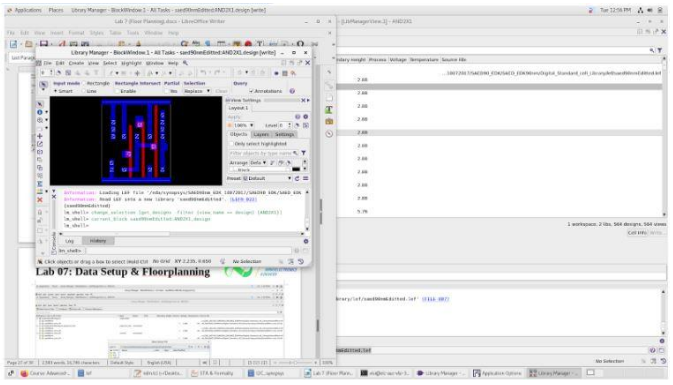


Figure 4: NDM

• Floorplan:

We created a library for the design and read parasitic and SDC files. We defined the default routing direction for the metal layers, initialized the floorplan and placed the pins.

Photos of the initial floor plan, floorplan with pavement without pins and the floorplan with pins placement are shown below.

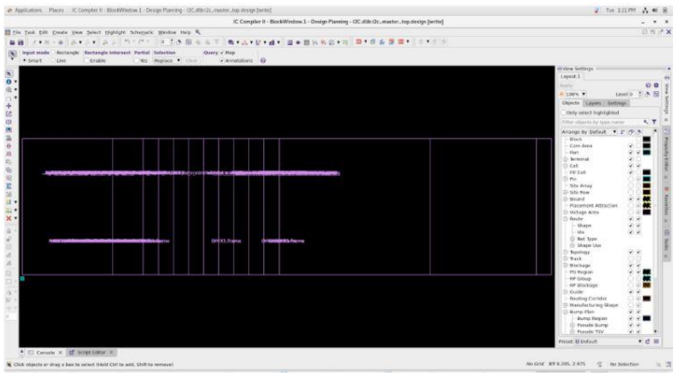
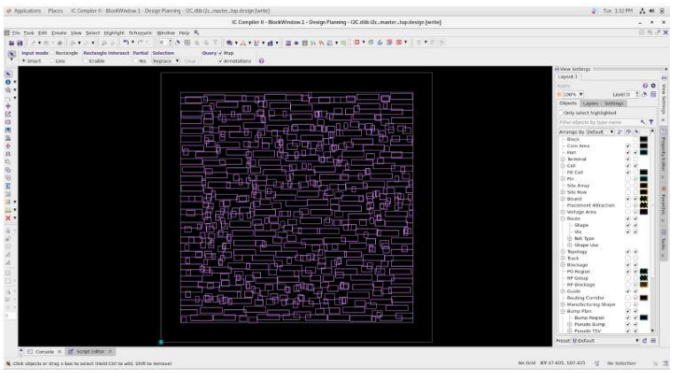
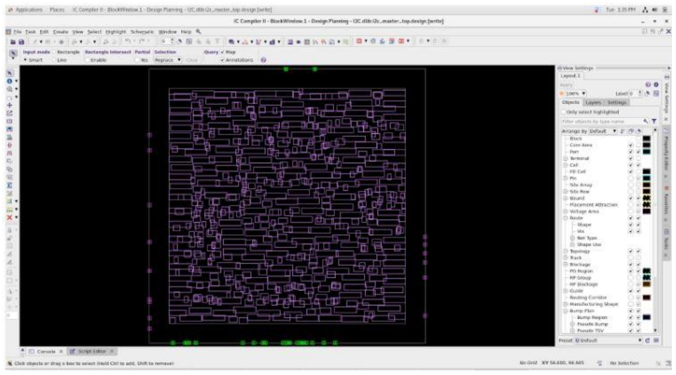




Figure 5: FloorPlanning View

• Powerplan:

For power planning we copied the block of floorplan to a new block named powerplan. We created power and ground nets and connected them to their respective pins. We created the power ring and power mesh and defined the corner cells. A photo of the power plan is shown below.

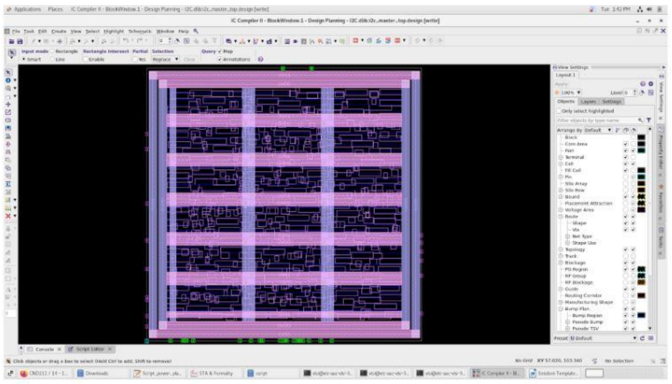


Figure 6: Power Planning View

• Placement:

A new block for placement is created which starts from the last step of the power plan. We set the parasitic parameters and place the cells and then check for legality and report timing and it had no violations. The photo below shows the design after placement.

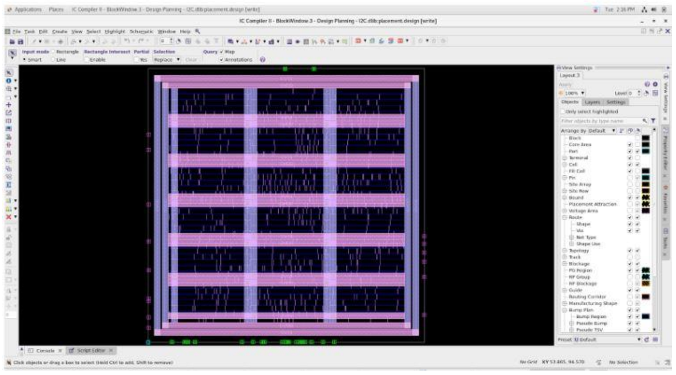


Figure 7: Placement View

- **Clock Tree Synthesis (CTS):**

CTS was performed. We set the non-default routing layers for CTS creation and set the driving cell for the clock tree, then build the tree and save the block. The photo below shows the design with the clock tree.

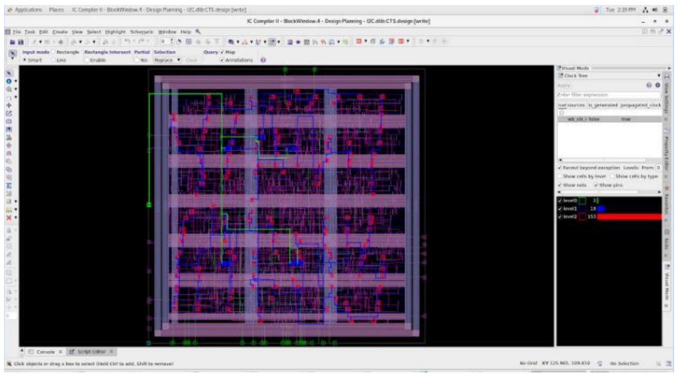


Figure 8: Clock Tree

- **Routing:**

A new block for routing was created. The antenna tcl file is run, then routing is performed. We created the filler cells and routed the power and ground nets to the design then removed the filler cells with violations. Then we wrote the output netlist, DEF file and SDC file. Then, we wrote the GDS file and saved the design. A photo of routing is shown below.

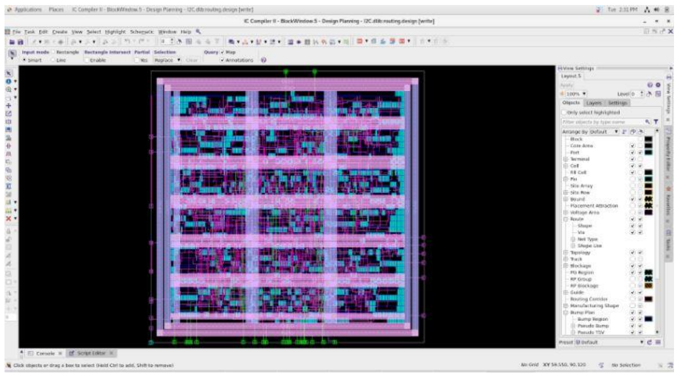


Figure 9: Routed Chip

.

.

5. Conclusion

In summary, the completion of our digital IC design project stands as a significant achievement in our journey as a students in the CND course. This endeavor has been a rich learning experience, providing hands-on exposure to ASIC design principles and methodologies.

Throughout the project lifecycle, from conceptualization to realization, we encountered and surmounted diverse challenges, each contributing to our growth and understanding. Refining specifications, crafting RTL designs, navigating synthesis and verification processes, and meticulously laying out the ASIC all demanded dedication and skill.

Beyond mastering technical aspects, this project fostered the development of problem-solving abilities and adeptness with design tools. Collaboration with peers and guidance from mentors further enriched our learning, highlighting the value of teamwork and mentorship in complex endeavors.

Looking forward, the knowledge gained from this project serves as a robust foundation for future endeavors in digital IC design and related fields. It equips us to tackle forthcoming challenges with confidence and innovation.

In reflection, I am grateful for the opportunity to delve into the captivating realm of digital IC design. The experience has been both rewarding and enlightening, and I eagerly anticipate applying this newfound expertise in future pursuits.